

## Appendix A META LEARNING FOR GENERALIZABILITY

By leveraging meta-learning to enhance generalizability [62], our representative model selection (see Section 4.4 for details) with the limited selection of models does not compromise the effectiveness of generated fingerprint samples in detecting unseen tampering types.

It is impossible to directly minimize Eq. 3 for each potentially rejected model since there are too many potential variations in real-world applications. We leverage meta-learning to generate generalizable fingerprint samples with a small number of rejected models. Let  $\mathcal{Z}$  be the model zoo in meta-learning, and rejected models in  $\mathcal{Z}$  be  $\mathcal{F}_{r_i}, i = 0, \dots, |\mathcal{Z}| - 1$ , where  $|\mathcal{Z}|$  is the cardinality of  $\mathcal{Z}$ . We randomly partition rejected models in  $\mathcal{Z}$  into train models and test models in each meta-learning iteration, and reduce the gap of effectiveness between the two sets of models. More specifically, in  $(k+1)^{th}$  iteration, the fingerprint updated in the  $k^{th}$  iteration,  $x_k$ , is taken as the start point,  $m$  rejected models are randomly selected from  $\mathcal{Z}$  to as the train model set  $\mathcal{S}^{k+1}_{train}$ , with the remaining rejected models as the test model set  $\mathcal{S}^{k+1}_{test}$ . First, the train model set is used to update  $x_k$  for  $T$  iterations to get  $x_k^T$ :

$$x_k^0 = x_k, \quad (6)$$

$$x_k^{t+1} = Clip_{\epsilon}(x_k^t - \eta \cdot \nabla_{x_k^t} \ell_{set}(x_k^t, \mathcal{F}_p, \mathcal{S}^{k+1}_{train})),$$

where  $t = 0, 1, \dots, T - 1$  and

$$\ell_{set}(x, \mathcal{F}_p, \mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{\mathcal{F}_{r_i} \in \mathcal{S}} \ell_{total}(x, \mathcal{F}_p, \mathcal{F}_{r_i}) \quad (7)$$

Then, the test model set is used to calculate the gradient over  $x_k^T$  to update  $x_k$  as follows,

$$x_{k+1} = Clip_{\epsilon}(x_k - \eta \cdot \nabla_{x_k^T} \ell_{set}(x_k^T, \mathcal{F}_p, \mathcal{S}^{k+1}_{test})) \quad (8)$$

## Appendix B MISENTRY'S GENERATION ALGORITHM

The optimization process of MiSentry's fingerprint generation is illustrated in Algorithm 1.

## Appendix C IMPLEMENTATION DETAILS

In the main paper, we have presented the overall pipeline of MiSentry. Here we provide more implementation details of fingerprint generation, tampering construction, model zoo, and experimental settings.

### C.1 Fingerprint Generation Settings

We use the existing state-of-the-art (SOTA) fingerprinting methods, SSF [22] and PublicCheck [58], as the baselines to compare within our evaluation. For SSF, its official open-source code [20] with default settings is used in our experiments. In SSF, the  $L_2$ -norm is used to bind a fingerprint sample from a normal sample. To compare with SSF, the  $L_2$ -norm is also used in Eq.3 in the main paper for MiSentry. In our experiments, we set the  $L_2$ -norm bound,  $\epsilon$  in Eq.3, as 3.5, 3.5, and 21.5 for CIFAR10, GTSRB, and ImageNet, respectively, for both SSF and MiSentry. For PublicCheck, we use VQVAE [55] to generate fingerprint samples located around the

### Algorithm 1 Fingerprint Generation of MiSentry

**Input:** Max iteration of meta-train update  $t_{train}$ , Max iteration of meta-test update  $t_{test}$ , Initial data  $x_0$ , Model to reject for meta-train update  $\mathcal{F}_r^{train}$ , Model to reject for meta-test update  $\mathcal{F}_r^{test}$ , distortion bound  $\epsilon$ , Target model  $\mathcal{F}_p$ , learning rate  $\eta$

**Output:** Fingerprint  $x_t$

```

1: procedure FINGERPRINT GENERATION( $t_{train}, t_{test}, x_0, \mathcal{F}_r^{train}, \mathcal{F}_r^{test}, \mathcal{F}_p, \epsilon, \eta, \alpha$ )
2:    $x_0^{test} = x_0 \leftarrow$  init outer state
3:   for  $i=1:t_{test}$  do # meta-test loop
4:      $x_{i,0}^{train} = x_i^{test}$ 
5:     for  $k=1:t_{train}$  do # meta-train loop
6:       Compute  $\ell_D(x_{i,k}^{train}, \mathcal{F}_p, \mathcal{F}_r^{train})$ 
7:       Compute  $\ell_{E_p}(x_{i,k}^{train}, \mathcal{F}_p)$ 
8:       Compute  $\ell_{E_r^{train}}(x_{i,k}^{train}, \mathcal{F}_r^{train})$ 
9:        $\ell_{total}(x_{i,k}^{train}, \mathcal{F}_p, \mathcal{F}_r^{train}) = \ell_D + \lambda_1 \cdot \ell_{E_p} + \lambda_2 \cdot \ell_{E_r^{train}}$ 
10:       $x_{i,k+1}^{train} = Clip_{\epsilon}(x_{i,k}^{train} - \eta \cdot \nabla_{x_{i,k}^{train}} \ell_{total}) \leftarrow$  meta-train update
11:      Compute  $\ell_D(x_{i,t_{train}}^{train}, \mathcal{F}_p, \mathcal{F}_r^{test})$ 
12:      Compute  $\ell_{E_p}(x_{i,t_{train}}^{train}, \mathcal{F}_p)$ 
13:      Compute  $\ell_{E_r^{test}}(x_{i,t_{train}}^{train}, \mathcal{F}_r^{test})$ 
14:       $\ell_{total}(x_{i,t_{train}}^{train}, \mathcal{F}_p, \mathcal{F}_r^{test}) = \ell_D + \lambda_3 \cdot \ell_{E_p} + \lambda_4 \cdot \ell_{E_r^{test}}$ 
15:       $x_{i+1}^{test} = Clip_{\epsilon}(x_i^{test} - \eta \cdot \nabla_{x_{i,t_{train}}^{train}} \ell_{total}) \leftarrow$  meta-test update
16: return  $x_{t_{test}}^{test}$  as fingerprint  $x_t$ 
```

decision boundary. We train VQVAEs with the training set for CIFAR-10 and GTSRB using Pythea [6] and use the pre-trained VQVAE of ImageNet from [13].

### C.2 Tampering Settings

We construct a benchmark for DNN integrity verification, which contains all the benign and malicious modifications listed in Section 3.1 in the main paper. We set each type of modification hardest to distinguish from the target model in our evaluation.

Specifically, we stop a modification to a target model as soon as the tampering goal is achieved to make the modification to the target model as subtle as possible. The default learning rate is set to  $1.0 \times 10^{-4}$  for all model modifications. The batch size is set to 64 for all the three datasets.

For unlearning, online learning, poisoning degradation attacks, and targeted attacks, a single sample is manipulated to minimize modifications to a target model: we set the model to unlearn a single sample randomly selected from its training set for unlearning, incrementally learn a single sample randomly selected from its test set for online learning, randomly mislabel a training sample for poisoning degradation attacks, and randomly select a training sample and assign the label with the minimal moving distance from the original label at the penultimate layer for targeted attacks. With this setting, we expect that the targeted attack makes subtler modifications to the target model than the poisoning degradation attack. All layers are fine-tuned for poisoning degradation attacks,

while only the last FC layer is fine-tuned for targeted attacks. For poisoning degradation attacks, we also conduct an experiment to randomly or specifically mislabel all training samples from a category randomly selected. In model tampering, the single sample manipulation is combined with the original learning task in multi-task learning to achieve the goal while preserving the model's accuracy. We check whether the manipulation goal is achieved every 10 steps, and stop the training when the manipulation goal is achieved.

For fine-tuning and re-training, both the last layer and all layers are set tunable. Re-training with all layers tunable is equivalent to training the same network on the same training data with random initialization. For fine-tuning, different learning rates of  $1.0 \times 10^{-lr}$ ,  $lr \in \{-3, -4, -5\}$  are used to train 1 epoch. For retraining, the learning rate is initially set to 0.001 and decays by timing 0.1 every time the validation loss hasn't descended for the last 5 epochs. The training is terminated when the learning rate is smaller than  $10^{-5}$ .

For pruning, we use the official pruning API of Microsoft NNI [39] and set the pruning ratio to 20%. For quantization, we use the dynamic quantization API [11] officially provided by Pytorch and convert a target model from 32-bit float to 8-bit integer.

For knowledge distillation, we use the code of Torchdistill [38] and set the  $\alpha$  to 1.0 and temperature to 3.0, and the learning rate is set and adjusted in the same way as in the retraining.

For transfer learning, we require both models to have the same labels (otherwise it is easy to differentiate the two models without using a fingerprinting method). For CIFAR10, we transfer from a pre-trained CIFAR10 model to STL10 by fine-tuning the last layer on STL10. STL10 shares the same 10 categories as CIFAR10 but is acquired from labeled examples of ImageNet. We cannot find such matching datasets for GTSRB and ImageNet and thus cannot evaluate transfer learning for these two datasets.

For both backdoor attack [16] and Trojan attack [36], we use the open-source code [1, 57] with a square trigger placed at the right bottom corner for all the three datasets. The trigger size is  $4 \times 4$  pixels for CIFAR10 and GTSRB and  $8 \times 8$  pixels for ImageNet. The injection rate is set to 0.1 for both attacks.

For clean label attack [63], we adopt the settings in PublicCheck as the default settings during the generation of adversarial samples. In the phase of backdoor trigger injection, we use the same trigger size as the backdoor and Trojan attack mentioned above.

For bit-flipping attacks, we use the open-source code [21] and flip the most important bit.

We also use models of the same and different architectures and independently train with the same and different datasets as the target model in our evaluation for the uniqueness requirement. The details will be described in the following Section C.3.

### C.3 Model Settings

**Models in the model zoo.** The rejected models in the model zoo of our meta-learning are the models with the following three benign modifications to the target model for all three datasets: fine-tuning the last layer and all layers at a learning rate of  $10^{-5}$  and random start (retraining all layers) with the same architecture. Five independent models are generated for each benign modification and included in the model zoo for meta-learning, except for the random

start of ImageNet. Training an ImageNet model from scratch takes too much time, so we alternatively take a model of DenseNet121 from another independent public model zoo [10].

**Test models.** The models used in our testing are all independent of the models in the model zoo of meta-learning. Unless stated otherwise, 10 models are generated for testing each tampering type. For the bit-flipping attack, since the bit-to-flip is deterministic, only one test model is generated. For different architectures, we use 10 different architectures [9, 18, 24, 25, 52] and collect their pre-trained models from [45] for ImageNet and from [8] for CIFAR10. Since there is no publicly pre-trained model for GTSRB, we train 10 models, one for each architecture, from scratch.

### C.4 Tamper Detection Setting

**Tamper detection.** The integrity of a model is verified by selecting a set of fingerprint samples,  $\mathcal{S}_F$ , to query the model and compare its returned top-1 labels with the ground truth (i.e., the top-1 labels returned by the original model to protect). The model is determined to be untampered only if the returned labels are all matched. In other words, the model is detected as being tampered with if one of the querying fingerprint samples has a returned top-1 label that disagrees with its ground truth.

**Fingerprint selection.** For MiSentry and PublicCheck, we independently generate 100 fingerprint samples and randomly select  $N_S$  fingerprint samples to conduct tamper detection each time. For SSF, we independently generate 2000 fingerprint samples and select  $N_S$  samples with the maximum coverage of active neurons from 100 randomly sampled fingerprint samples to conduct tamper detection each time. For each tamper detection, the selected  $N_S$  fingerprint samples are used to query each model to verify integrity.

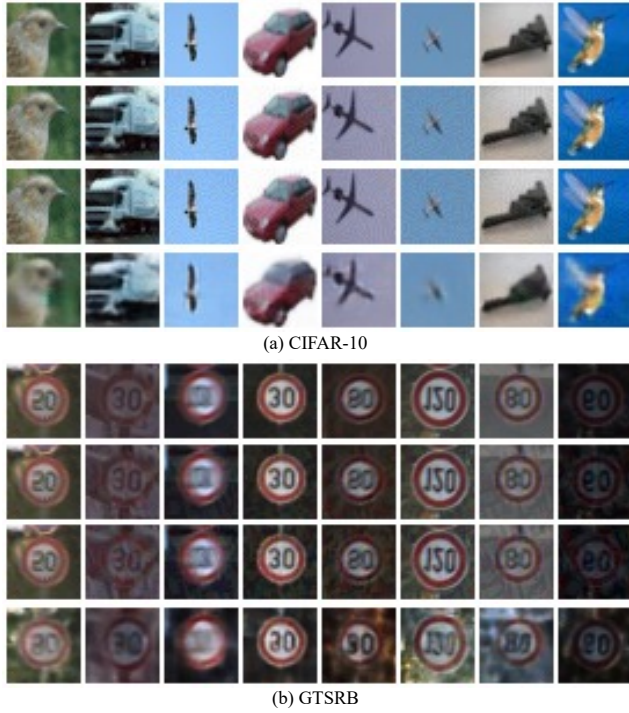
**Tamper detection rate.** To calculate the tamper detection rate, we repeat the tamper detection process 100 times for each model to test, each time with an independent set of  $N_S$  fingerprint samples selected with the method described above. The detection rate is the ratio of detected models to the total number of tests. For example, the detection rate for unlearning reported in Table 1 is the number of detected models divided by 1000 (= 10 test models  $\times$  100 detection times per model).

## Appendix D PERCEPTUAL QUALITY

Fig. 8 shows some fingerprint samples generated by MiSentry, SSF, and PublicCheck on ImageNet, and Fig. 7 shows those on CIFAR10 and GTSRB, accompanied by their respective source samples. These fingerprint samples exhibit a natural appearance, and the perturbations they introduce to the source samples are almost indiscernible. The perceptual quality of these samples is comparable to that of adversarial examples generated by the Projected Gradient Descent (PGD) method.

## Appendix E DETAILED EXPERIMENTAL RESULT OF ADAPTIVE TAMPERING ATTACKS

We present the tampering detection rates using a single fingerprint sample for adaptive fine-tuning last-layer attacks for the three methods with different numbers of leaked fingerprinting samples



**Figure 7: Perceptual quality of fingerprint samples generated on CIFAR10 and GTSRB dataset. Top row: source images. Second row: fingerprint images generated by MiSentry from the source images. Third row: fingerprint images generated by SSF. Bottom row: fingerprint images generated by PublicCheck.**

in Fig.9, and that for adaptive backdoor attacks in Fig.4 in the main paper.

## Appendix F PERFORMANCE COMPARISON WITH SSF-BO

As a variant of SSF, SSF-BO [34] uses Bayesian optimization (BO) to generate fingerprint samples. Due to the difficulty of directly optimizing in high-dimensional input spaces, SSF-BO first maps the input space into a lower-dimensional latent space using a Variational Autoencoder (VAE). It then searches within the latent space for an input that maximizes the difference in predictions between the original model (held locally by the model owner) and the model being verified for integrity (deployed in the cloud and potentially compromised). As a result, fingerprint samples generated by SSF-BO are tailored to distinguish the target model-tampering type from the original model.

We have compared the performance of SSF-BO with SSF, PublicCheck, and our MiSentry on MNIST. In our comparison, we use the code of SSF-BO from [42], with its default settings in the code being used. In testing its performance for each type of model-tampering, we use SSF-BO to generate fingerprint samples specifically tailored for the particular tampering type to avoid any mismatch between the tampering type of the model to verify integrity and the target

tampering type the fingerprint samples are tailored to, thus ensuring that the resulting experimental performance represents the best that SSF-BO can achieve. In our experiments, the  $L_2$  bound for MNIST is set to 0.8 for both SSF and SSF-BO.

The detection rates of SSF-BO, SSF, PublicCheck, and our MiSentry on MNIST using a single fingerprint sample for different model tampering types are shown in Table 4. We can see that

- SSF-BO outperforms SSF in detecting backdoor attacks and model pruning.
- SSF-BO can hardly or even not detect subtle modifications including targeted attacks, fine-tuning, and unlearning.
- MiSentry surpasses SSF-BO in detecting all the tested model tampering, esp. for subtle modifications.

**Table 4: The tamper detection rate (%) with  $N_s = 1$  for MNIST.**

Tampering\Methods	SSF	SSF-BO	PubCheck	MiSentry
backdoor(epoch=10)	75.3	90.1	91.6	93.4
backdoor(epoch=1)	51.8	61.2	67.5	83.7
FTLL (1e-5)	0.0	0.0	12.1	39.8
FTAL (1e-5)	0.0	0.0	13.2	41.6
Unlearning	0.0	0.0	19.3	58.2
Different Architectures	56.4	48.9	66.4	86.5
Pruning	54.9	58.4	63.7	83.1
Targeted Attack	1.9	2.3	31.8	72.6

We cannot compare the performance of SSF-BO with other methods on the three datasets, CIFAR10, GTSRB, and ImageNet, used in our experiments in the main paper. This is because SSF-BO does not report its performance on these datasets in its paper [34], and the released code [42] does not include the code for these datasets either. Furthermore, implementing SSF-BO on these datasets, particularly ImageNet, is challenging due to the following reasons:

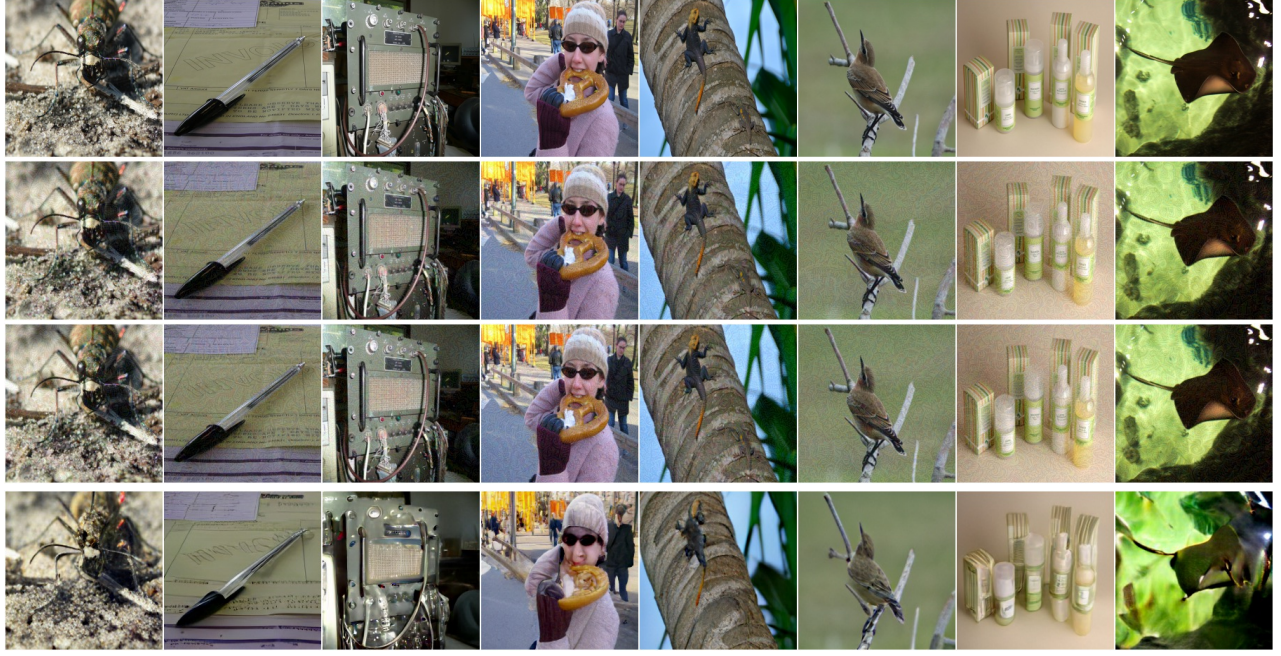
- Large-scale datasets like ImageNet have to be reduced to a low-dimensional space for BO to work. Such a large-scale dimensional reduction would have an adverse effect on the performance. Determining an optimal low dimension with the best performance requires considerable effort.
- BO is sensitive to the settings of its hyperparameters and is unstable. Tuning SSF-BO to achieve the best performance on a new dataset requires a significant effort and is time-consuming.

As the experimental results in Table 4 show, PublicCheck outperforms SSF-BO and is the state-of-the-art existing fingerprinting method. It is sufficient to focus on comparing with PublicCheck on the three datasets.

## Appendix G DETAILED ABLATION STUDY RESULTS

The factors that may influence the detection performance of MiSentry mainly include the combination of modification types in the model zoo and the number of each type of model modification used in the model zoo. To analyze the impact of these two factors on the effectiveness of our MiSentry, we conduct the following ablation studies on CIFAR10.





**Figure 8: Perceptual quality of fingerprint samples generated on ImageNet dataset. Top row: source images. Second row: fingerprint images generated by MiSentry from the source images. Third row: fingerprint images generated by SSF. Bottom row: fingerprint images generated by PublicCheck.**

## G.1 Modification Types in Model Zoo

We first conduct an ablation study on the modification types included in the model zoo of our MiSentry. We construct four combinations of types of model modification for the model zoo:  $Conf_1$ : FTLL+FTAL+Retrain<sub>All</sub>,  $Conf_2$ : FTLL+FTAL,  $Conf_3$ : FTLL, and  $Conf_4$ : FTAL. The results are shown in Tab.5.

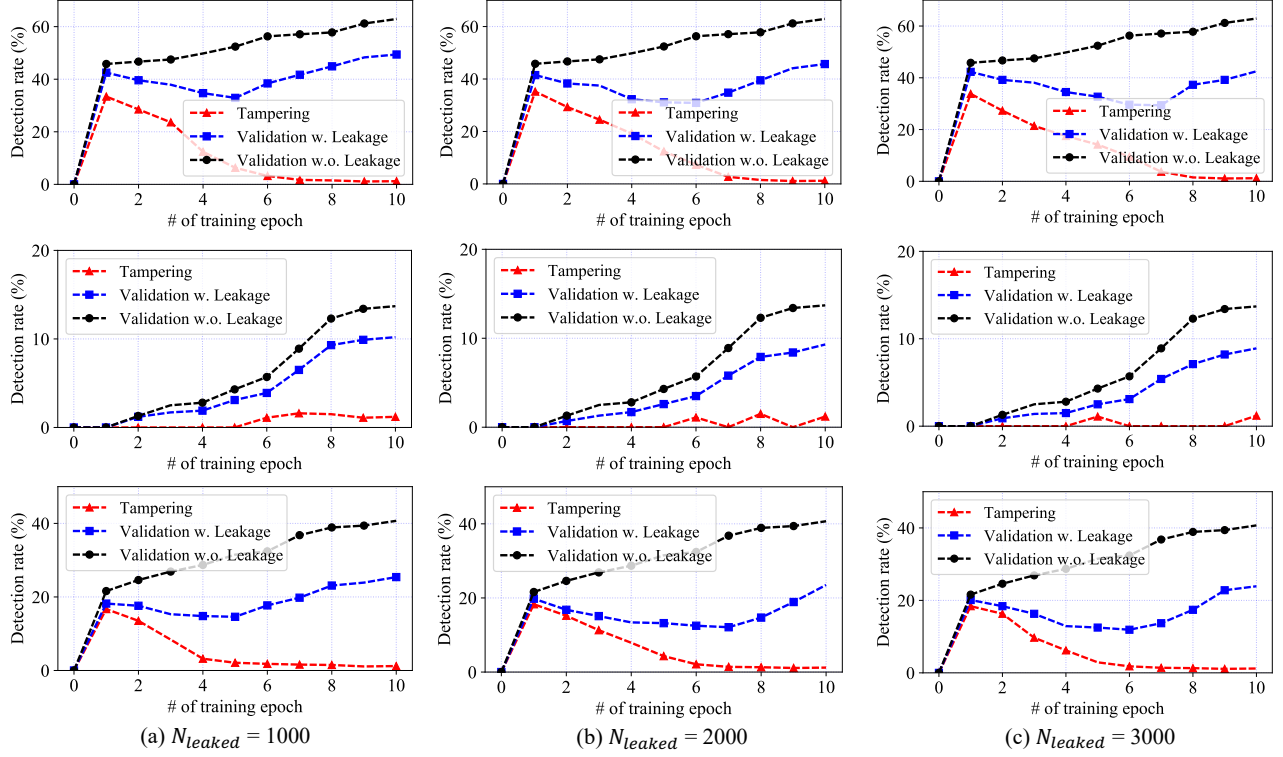
**Table 5: Ablation study on modification types for constructing model zoo. This table reports the tamper detection rates using a single fingerprint sample ( $N_S = 1$ ) generated by MiSentry.**

Tampering\Model zoo	$Conf_1$	$Conf_2$	$Conf_3$	$Conf_4$
Degradation <sub>Random</sub> -C	91.3	91.1	88.9	79.4
Degradation <sub>Specific</sub> -C	93.1	92.7	89.3	79.7
Targeted Attack	83.8	84.2	79.3	77.2
Backdoor	86.4	85.9	81.5	78.1
Trojan	91.9	89.7	86.8	79.3
Knowledge Distillation	92.8	88.3	87.3	78.5
Quantization	90.7	91.2	89.4	80.1
Pruning	89.5	87.4	78.3	72.4
Fine-tuning <sub>Last</sub> -3	73.7	73.8	75.8	48.9
Fine-tuning <sub>Last</sub> -4	58.1	58.3	61.3	44.5
Fine-tuning <sub>Last</sub> -5	73.7	73.8	75.8	48.9
Fine-tuning <sub>All</sub> -3	76.5	77.1	64.8	62.9
Fine-tuning <sub>All</sub> -4	64.1	63.8	55.9	56.3
Fine-tuning <sub>All</sub> -5	59.2	59.4	49.2	48.1

From the Tab.5, we can see removing last-layer fine-tuned models from the model zoo caused a significant drop in detection performance, averaging 12.2% and up to 24.9% across all tampering types in Table 1. Removing all-layer fine-tuned models led to a 10% decline for all-layer fine-tuning but minimal change for other types. Removing models with random initial conditions caused about a 7% decrease in detection rate for random start and different architectures, with little impact on other types. Keeping only last-layer fine-tuned models in the model zoo results in a detection rate above 47.2% for all tampering types in Tab.5. These results show that subtly modified models (i.e., last-layer fine-tuning) are crucial for detecting all modification types, while other modification types in the model zoo mainly enhance detection performance for the same type.

## G.2 Number of Models in the Model Zoo per Type

Regarding the relationship between the generalization ability of fingerprint samples and the number of minimally modified models in the model zoo, we conducted an ablation study in which we used only fine-tuned last layer models in the model zoo and incrementally increased their number  $n$  from 2 to 8 in steps of 2. From Tab. 6, we observed a significant increase in detection rate (an average of 30.0% for all tested tampering types) when increasing from 2 to 4. However, further increases yielded only marginal gains in detection rates (averages of 2.9% and 0.6% for increases from 4 to 6 and 6 to 8, respectively). Thus, while the generalization ability of fingerprint samples improves with an increasing number of representative models, there is a saturation point beyond which increases yield



**Figure 9: Detection rate when different numbers of fingerprint samples are leaked and exploited for adaptive fine-tuning last layer attacks on CIFAR10.**  $N_{leaked}$  denoted the number of leaked fingerprint samples. The three rows from top to bottom represent the results of MiSentry, SSF, and PublicCheck, respectively. The black dot line shows the detection rate when no leaked fingerprint samples are exploited, while the blue square line shows the detection rate when  $N_{leaked}$  leaked fingerprint samples are exploited for an adaptive tampering attack. The red triangle line shows the detection rate of the leaked fingerprint samples (i.e., on the tampering set).

**Table 6: Ablation study on the number of models per modification type in the model zoo. This table reports the tamper detection rates using a single fingerprint sample ( $N_S = 1$ ) generated by MiSentry.**

Tampering Type	Number of Models per Type			
	2	4	6	8
Degradation <sub>Random</sub> -C	60.1	90.1	91.5	90.4
Degradation <sub>Specific</sub> -C	58.7	91.2	90.8	93.2
Targeted Attack	59.6	82.2	85.3	84.6
Backdoor	61.3	83.9	88.5	87.8
Trojan	62.5	89.5	92.3	92.5
Knowledge Distillation	73.8	93.1	91.9	93.4
Quantization	68.3	91.4	91.8	92.2
Pruning	60.2	88.7	88.9	89.2
Fine-tuning <sub>Last-3</sub>	28.5	69.2	72.9	75.3
Fine-tuning <sub>Last-4</sub>	24.7	53.5	62.4	64.3
Fine-tuning <sub>Last-5</sub>	24.3	39.8	47.5	49.1
Fine-tuning <sub>All-3</sub>	31.1	72.4	77.3	77.9
Fine-tuning <sub>All-4</sub>	25.9	56.8	58.4	62.3
Fine-tuning <sub>All-5</sub>	24.2	48.5	58.7	60.2

diminishing returns. For computational efficiency, we included 5 models for each benign modification in the model zoo in our main experiment.

**Table 7: Tampering detection rate (in %) with and without input covertness constraint using a single fingerprint sample generated by MiSentry.**

Tampering Type	Bounded	Unbounded
Unlearning	68.4	79.7 (+0.165×)
Online Learning	74.1	84.2 (+0.136×)
Backdoor	86.4	90.3 (+0.045×)
Targeted Attack	83.8	88.9 (+0.061×)
Fine-tuning <sub>Last-3</sub>	73.7	79.2 (+0.074×)
Fine-tuning <sub>Last-4</sub>	58.1	71.8 (+0.236×)
Fine-tuning <sub>Last-5</sub>	45.9	62.4 (+0.359×)
Fine-tuning <sub>All-3</sub>	76.5	82.6 (+0.079×)
Fine-tuning <sub>All-4</sub>	64.1	77.5 (+0.209×)
Fine-tuning <sub>All-5</sub>	59.2	73.1 (+0.235×)

**Table 8: The effect of the architecture of the target model. This table reports the tamper detection rates using a single fingerprint sample ( $N_S = 1$ ) generated by MiSentry.**

Tampering Type	Target Model's Architecture	
	Resnet-20	VGG11
Degradation <sub>Random</sub> -C	93.1	92.7
Targeted Attack	83.8	79.4
Backdoor	86.4	88.5
Trojan	91.9	93.5
Pruning	89.5	85.4
Fine-tuning <sub>Last</sub> -3	73.7	68.3
Fine-tuning <sub>Last</sub> -4	58.1	59.5
Fine-tuning <sub>Last</sub> -5	45.9	51.8
Fine-tuning <sub>All</sub> -3	76.5	75.4
Fine-tuning <sub>All</sub> -4	64.1	66.7
Fine-tuning <sub>All</sub> -5	59.2	57.3

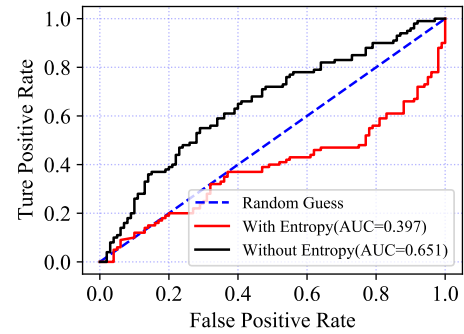
### G.3 Effect of Input Bound on Input Coverttness and Detection Performance

We next conduct an ablation study on the input  $L_p$  norm bound, which is used to constrain the generated fingerprint sample to look similar to a normal sample. Without the  $L_p$  norm bound, the generated fingerprints are noise-like images, which can be easily distinguished from normal samples. However, these noise-like fingerprints have higher detection sensitivity. The results are shown in Tab. 7. For the hardest case, i.e. fine-tuning the last layer with a learning rate of  $10^{-5}$ , the tampering detection rate using a single fingerprint is improved by 0.359 $\times$ .

### G.4 Effect of Entropy Loss on Output Coverttness

Entropy loss ensures that the generated fingerprint samples have similar output prediction vectors to that of natural samples. Without entropy loss, the generated fingerprint samples may have abnormal prediction vectors and could be detected by their confidence scores.

These detected fingerprint samples can be responded with the labels from the original model to evade integrity verification. We next conduct an ablation study on entropy loss. In this experiment, low-confidence samples are detected as potential fingerprint samples. The detection results on CIFAR-10 are shown in Fig. 10. The AUC scores are 0.651 and 0.397 for fingerprints generated without and with the entropy loss, respectively. It confirms that the entropy loss enhances the coverttness of generated fingerprints' prediction vectors. Thus the entropy constraint is an essential component for MiSentry to evade the fingerprint detection based on output probability distributions.



**Figure 10: ROC curve of fingerprint detection based on low-confidence predictions.**

### G.5 Generalizability to Different Model Architectures

In addition, to validate the generalizability of MiSentry, we also conduct another experiment using VGG-11 as the target model on the CIFAR10 dataset. The results are shown in Tab.8. From Tab.8, we can observe that our method exhibits similar integrity verification effects across different model architectures, indicating the generalizability of MiSentry.