AIME: Watermarking AI Models by Leveraging Errors

Dhwani Mehta
ECE, University of Florida
Gainesville, United States
dhwanimehta@ufl.edu

Nurun Mondol
ECE, University of Florida
Gainesville, United States
nmondol@ufl.edu

Farimah Farahmandi
ECE, University of Florida
Gainesville, United States
farimah@ece.ufl.edu

Mark Tehranipoor
ECE, University of Florida
Gainesville, United States
tehranipoor@ece.ufl.edu

***Abstract*—The recent evolution of deep neural networks (DNNs) has made running complex data analytics tasks, which range from natural language processing, object detection to autonomous cars, artificial intelligence (AI) warfare, cloud, healthcare, industrial robots, and edge devices feasible. The benefits of AI are indisputable. However, there are several concerns regarding the security of the deployed AI models, such as reverse engineering and Intellectual Property (IP) piracy. Accumulating a sufficiently large amount of data - building, training, improvement, and model deployment require immense human and computational power, making the process expensive. Therefore, it is of utmost importance to protect the model against IP infringement. We propose AIME, a novel watermarking framework that captures model inaccuracy during the training phase and converts it into the owner-specific unique signature. The watermark is embedded within the class mispredictions of the DNN model. Watermark extraction is performed when the model is queried by an owner-specific sequence of key inputs, and the signature is decoded from the sequence of model predictions. AIME works with negligible watermark embedding runtime overhead while preserving the accurate functionality of the DNN. We have performed a comprehensive evaluation of AIME, which models on MNIST, Fashion-MNIST, and CIFAR-10 dataset and corroborated its effectiveness, robustness, and performance.**

***Index Terms*—Artificial Intelligence, Watermarking, Intellectual Property piracy, Deep Learning**

I. Introduction

The evolution of deep neural networks (DNNs) have caused an industry-wide revolution where more and more devices and applications are being transformed to integrate intelligence in them; a few examples include robotics, autonomous cars, natural language processing, image processing, manufacturing, and health care systems [1]–[3]. To build a successful DNN model that performs with high precision and accuracy the following factors must be met: (a) massive labeled datasets which cover all scenarios, including the edge cases for the underlying application; (b) computational power in terms of CPU's, graphics processing units (GPU's), and tensor processing units (TPU's) for training the model; (c) resources and knowledge to fine-tune the model (hyper-parameters, weights, learning rate, batch size, and layers) for accuracy [4]. These steps, in turn, make developing the DNN models expensive, and hence the developed models are considered intellectual property (IP) of the model owner in software and hardware both [5]–[9]. DNN models, if not protected, can be victims of IP infringement. This is problematic as the average consumer is working with constrained resources. Therefore, DNN models should be protected against IP infringement and reverse engineering methods.

Digital watermarks (WM) are often used to successfully identify and protect the ownership of the underlying image, video, signal, and audio [10], [11]. A few works have extended watermarking techniques to deep learning models [12], [13]; however, the development of these techniques is still in the beginning phases. In this era of artificial intelligence, DNNs are often used as white-box models (model parameters are public) or black-box models (model parameters are kept as a secret). DNNs are extremely versatile and are used in numerous applications. Each of these applications may use a different version/architecture of the DNN model. Hence the development of universal watermarking techniques for such DNN models is a challenging task. The watermarks embedded in the model should be generic and should be detected in both white-box and black-box applications. Moreover, the watermarks should withhold against any potential attacks or model optimization routines (e.g., fine-tuning) while maintaining the functionality of the model and with minimal overhead. Our work leverages the model class mispredictions for embedding watermark in the models. The contributions made in this paper are following:

- Proposing AIME, an end-to-end watermarking framework that leverages model class mispredictions to embed watermarks within the hidden layers of the model. AIME only requires to query the model to generate the class mispredictions to extract the watermark from the model.
- Providing metrics and evaluations of our (AIME) watermarking framework for two models (Sequential and Convolutional Neural Network (CNN)) and three datasets (MNIST, Fashion-MNIST and CIFAR-10). Our framework satisfies all the watermark requirements, including fidelity, robustness, integrity, efficiency, and security. Hence AIME is practicable and scalable.
- These evaluations on various benchmarks show that AIME enables the IP owner to embed a robust watermark which is easy to extract but difficult to eliminate. The model is robust against fine-tuning and pruning, and watermark extraction is possible even when the pruning rate is 40%.

The rest of the paper is divided as follows. In section II we cover the existing work in the DNN model watermarking domain. Section III describes our framework for watermarking

978-3-9819263-6-1/DATE22/©2022 EDAA

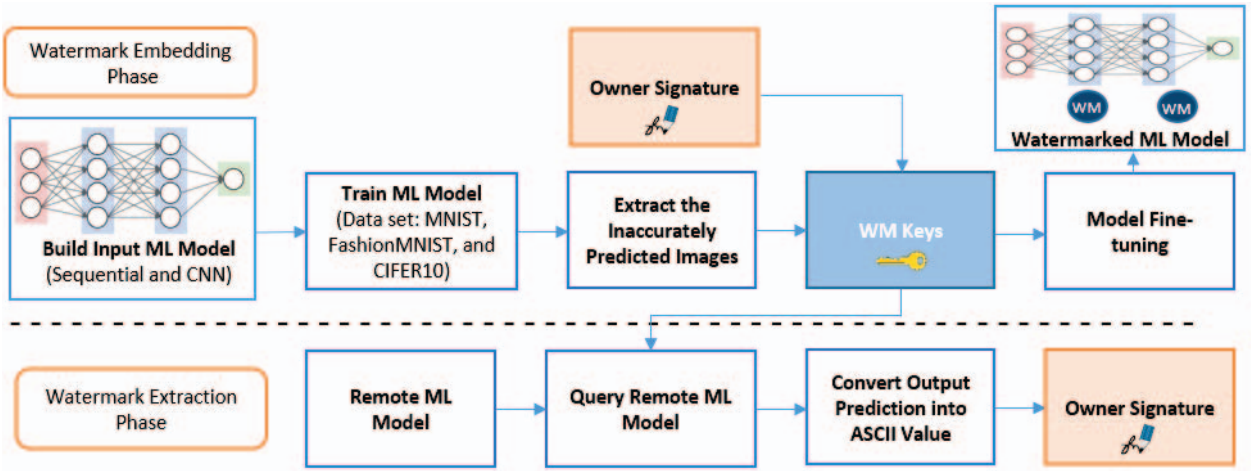

Fig. 1: Overview of the AIME Watermarking Framework.

a DNN model. Section IV is a detailed description of our methodology to embed a robust watermark in a DNN model. Section V discusses results and evaluations of our watermarked model and section VI concludes the paper.

II. Related Work

Watermarking consists of two stages: watermark embedding and watermark extraction. Watermarks have been used in the multimedia domain for over a decade to protect the IPs [11] [10]. DNN model protection is an increasing area of research since developing DNN models is time and resource expensive.

The first few works included embedding watermarks in applications where the model parameters are known (white-box). They leverage the intermediate layers to embed watermarks in the weights of these layers [14]. Later, adversarial samples were embedded as watermarks in the decision boundary by leveraging fine-tuning procedures [15]. Next, the watermarks were embedded within activation maps of the output and intermediate layers. This framework applied to both white and black-box models [16]. The authors of [17] embedded a watermark by training the model along with the WM sets, which include three algorithms for each WM set (e.g., noise, content, and unrelated). The authors of [18] included the watermarks with the weights of the neural network, however, it is not applicable to black-box applications.

III. AIME Overview

The workflow of AIME is described in this section. The framework consists of two main stages: watermark embedding and watermark extraction, as seen in Fig. 1. In the watermark embedding stage, the model owner of the DNN model embeds a watermark during the training phase. The embedding stage is further divided into (a) developing owner signature, (b) build the watermarked dataset based on the class mispredictions, and (c) fine-tune the model using an owner-specific sequence of keys. The watermark embedding is performed by the owner prior to the model being deployed in real-world applications and the process is performed only once. Table I provides the criteria considered for developing a robust watermark.

TABLE I: Criteria for building a successful watermark.

WM Requirements	Detailed Description
Fidelity	Embedding the WM should not affect the model accuracy
Reliability	WM should always be detected with the key
Integrity	WM model should be uniquely identifiable
Efficiency	The computational overhead of watermarking and extracting the watermark should be minimum
Robustness & Security	Watermarked model should be secured against various attacks, undetectable and easily extractable

Once the model is locally trained and the watermark is embedded in the model, it is ready to be deployed, as a white-box model or a black-box model. In the next stage, watermark extraction is performed to prove the ownership of the model. Here, the proof of ownership is extracted by the owner using the correct predefined watermarked key inputs that were selected to trigger the watermark information from the DNN model. The owner now queries the model to obtain the class label information for the queried inputs. Next, he/she uses these obtained class label predictions to extract the watermark information.

IV. Watermarking

Deep learning models are trained to achieve high precision and accuracy for the underlying applications. A *"good"* DNN model which achieves a *near to 100% accuracy* is often selected to be deployed in various settings. AIME leverages the fact that no DNN model is 100% accurate. AIME uses misclassified class labels to embed watermarks in the hidden

layers of the model. This watermark is later used to detect IP infringement.

A. Watermark Embedding

This stage is further sub-divided into three phases, (a) determining and developing the owner's signature as required, (b) build the watermarked dataset based on the class mispredictions, and (c) fine-tune the model using a owner-specific sequence of keys. The output of this stage is a WM marked DNN model with a set of owner specific WM keys.

In the first stage, the owner's signature is determined. For example, consider the ASCII values of the letters "F," "C," "I," and "S." Next, these ASCII values are combined to form an owner specific key, in this case the key is *"70677383"* where "F"=70, "C"=67, "I"=73, and "S"=83. The goal of the developer embedding the owner specific key as a watermark in the model is to achieve output labels of *"70677383"* using mispredicted classes as input images as shown in Algorithm 1. The owner collects these images using step b. The developer in our technique starts with a model and builds it from the ground up. Each model, by definition, cannot be 100% correct and will exhibit some loss and inaccuracy in both the validation and testing sets. The next step for the developer is to select the appropriate architecture for the DNN model and the datasets for training the model. Here, a MNIST dataset is considered along with a CNN classifier. The output of the model will be ten labels ranging from zero to nine. Once the training is complete, the developer extracts the inaccurately predicted set of images from the training dataset. Although the adversary may have access to the humongous dataset used for training (number of images ranging from ten thousands to millions), they will not be able to determine the sequence of mispredicted images used for watermarking. Next, the developer plots a confusion matrix to check the class labels of the wrongly predicted data as shown in Fig. 2. The confusion matrix provides the number of times each of the classes were mispredicted.

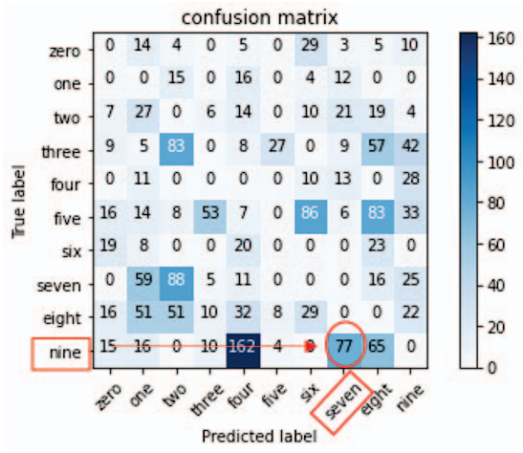

Fig. 2: Confusion matrix for the mispredicted output labels after training the CNN model using the MNIST dataset.

In the second stage, the watermarked dataset (sequence of data) is curated. To create this dataset, the developer looks closely at the key which was generated by the owner (in this case key = "70677383") and considers the digits of the key one by one and compares it with the confusion matrix. For example, the digit 9 was wrongly classified as 7 as seen in red in Fig. 2. Next, a particular set of mispredicted images of the digit 9 are considered a part of the watermarked dataset. Similarly, the images that predict the digits "0," "3," "6," and "8" inaccurately are identified and a particular sequence of those images are added in the watermarked dataset. Note, these subdivisions are a part of the training dataset and the original number of images in the training data are neither increased nor decreased.

Algorithm 1 Watermark Embedding for DNN model

1: **Input:** Training data (X^{train}, Y^{train}), Desired Output label from the model as key (k)
2: **Output:** Watermarked data (X^{wm}, Y^{wm}), True Label for the watermarked data Y^{true}
3: **Misclassified Data Identification:**
4: Initialize model layer structure, parameter and hyper-parameters (M)
5: $M^{W/0}$ (trained model without WM) ← Train M (X^{train}, Y^{train})
6: $M^{W/0}$ (X^{train}, Y^{pred}) ← Pass training Image X^{train}
7: (Y^{false} = Y^{pred}) & (X^{false} = X^{train}) ← Check if (Y^{train} != Y^{pred})
8: (Y^{true} = Y^{pred}) ← Check if (Y^{train} = Y^{pred})
9: **Watermarked Data Generation:**
10: Confusion matrix ← (Y^{true}, Y^{false})
11: (Y^{wm} = Y^{false}) & (X^{wm} = X^{false}) ← Check confusion matrix if (K == Y^{false})
12: **Watermark Embedding:**
13: M^{WM} ← Fine - tune model $M^{W/O}$ (X^{wm}, Y^{wm})

In the third stage, the developer fine-tunes the model to accommodate the watermarked dataset. This fine-tuning is conducted using a particular/fixed sequence of images. For example, the sequence here would be the set of images corresponding to *"70677383"*. This sequence would make sure that the output produced by the watermarked model is an owner specific signature. Because these data lie in the model's decision boundary and are dependent on the model structure, parameters, and hyperparameters, we picked wrongly predicted data as watermarked datasets in our technique. No one apart from the model owner would know the sequence of this data and hence even if the model is fine-tuned again, or if the parameters are tweaked a bit, the elements to prove the ownership of the model would not change. This means the WM is unique to the model itself. Furthermore, because incorrect predictions occur due to the inherent nature of the model and are dependent on how the model interprets the input, watermarked data will not stand out as an adversarial point in the PDF of model parameters and will not be detectable from the outside. Since the dataset that is being used to WM the model was already misclassified, the addition of the watermark

does not cause much performance degradation and has little overhead as shown in the results section. Fine-tuning the model further would not cause the parameters to change drastically; as the values of the parameters would linearly shift, making the WM invisible. Even if the adversary somehow got hold of the huge training data and wanted to misuse the misclassifications, there would be unknowns, such as the sequence of images used for watermarking, the number of times the particular class label was misclassified, and the owner specific signature.

Algorithm 2 Watermark Extraction for ML model

1: **Input:** Watermarked Model M^{WM}, n number of Watermarked data (X^{wm}, Y^{wm})
2: **Output:** Owner Signature
3: **Generate Misclassified Output:**
4: $Y_n^{pred} \leftarrow$ prediction M (X_n^{wm}, Y_n^{wm})
5: $Y^{pred} \leftarrow$ save n numbers of predictions Y_n^{pred} to a list
6: **Step 1:** $Y_j^{'} \leftarrow$ take first i number of list members from Y^{pred} at a time and make another list
7: **Step 2:** $Y_j^{''} \leftarrow$ converts the i number of digits from $Y^{'}$ into a number with i digits
8: **Step 3:** take next i number of list members from Y^{pred} at a time and make another list and then repeat step 2
9: Keep repeating step 3 and 2 until the end of list Y^{pred}
10: **Owner Signature:** $\leftarrow$ Combine (Char($Y_j^{'}$), Char($Y_j^{''}$)....)

B. Watermark Extraction

In this stage the model owner queries the DNN model to prove the ownership. To query the model, the owner uses the set of keys determined and selected in section IVA. When these keys are queried in a particular sequence, the watermark is triggered and the model outputs predictions for those sequence of images as seen in Algorithm 2. The original sequence of the output (when the watermark is not triggered) should be *"96965956"*. When the watermark is triggered, the output of the models will be predicted as *"70677383"*. This sequence then passes through the decoder, which splits the predicted values in a set of 2 digits and converts it to their ASCII values. Note, the split of the predictions to pass through the model is selected by the model owner while embedding the watermark. Later, the decoder converts them into their respective letters. In this case it would be "F," "C," "I," and "S." Thus, these letters prove the ownership of the model.

V. Results

We evaluate AIME on multiple models and datasets. The models considered are Sequential and CNNs. The datasets considered are MNIST [19], Fashion-MNIST [20] and CIFAR-10 [21]. Table IV summarizes the accuracy with and without watermarks, as well as different key lengths for the CNN model with the MNIST dataset for all class labels. Note, the accuracy summary is only provided for one model and dataset due to page limitations. The accuracy of the models and datasets with and without keys is shown in Fig. 3. AIME is generic and can be used in white-box and/or black-box scenarios along with the capability to embed the watermark in one or all hidden layers of the network. AIME complies by the criteria of creating a robust watermark for a DNN model and is explicitly evaluated as listed in Table I.

Apart from the evaluations, AIME is scalable. In the simplest form, if a model classifies only two classes, the base digit can be considered binary, and binary can be converted into an ASCII value. If the classifier classifies more than 10 classes, not all classes need to be included in the watermarking process, making it even more difficult to predict the watermark. The framework will work on datasets other than handwritten numbers, as proved in our experiments using the Fashion-MNIST and CIFAR-10 data. In the CNN model, the output prediction will always be a class label, which is a digit. This predicted digit can then be turned into numbers and corresponding ASCII value through the decoder. For e.g., the class label for "Shirt" in Fashion MNIST dataset is 0. Furthermore, the number of keys can be altered and the split of the owner sequence can be manipulated to generate ASCII values.

A. Evaluations

For the experiment, three datasets (MNIST, Fashion-MNIST, and CIFAR-10) were used. In addition, two kinds of model structures were used (Sequential and CNN). The model architectures are described in Table III. All hidden layers were activated by the Relu activation function and the output prediction probability was given by a Softmax function. Pytorch was used to create the model, CrossEntropy was used for evaluating validation loss and Stochastic Gradient Descend (SGD) was used as an optimizer.

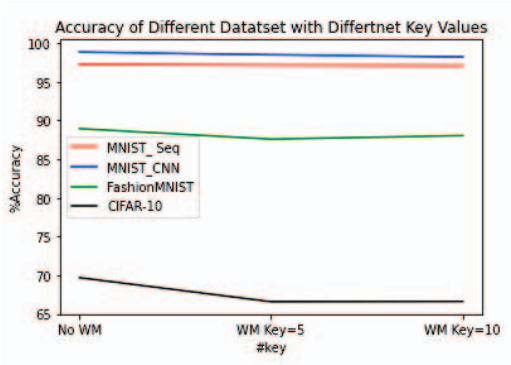

Fig. 3: Model accuracy comparisons with and without WM (with key=5 and key=10) for sequential and CNN models while training with MNIST, Fashion-MNIST and CIFAR-10 datasets.

1) Capacity: Information in terms of computing and DNNs are interpreted by using binary base numbers (0 or 1) or decimal base numbers (0 to 9). Watermarking framework leverages class mispredictions which are number inclusive. Hence, the maximum number of unique key bits is 10. These 10 digits can be used in any combination & any length to extract information from the model. Therefore, the WM with humongous information can be stored inside the model without compromising negligible performance and overhead as seen in Table II.

2) Fidelity: In our method, we are fine-tuning the model to misclassify the already existing class mispredictions. Hence the accuracy of the model is not affected drastically. The more accurate the model is to begin with (which is usually the case since only highly accurate models are deployed), the less the accuracy is affected after watermarking as seen in Fig. 3. The figure hightlights that even with key lengths of 5 and 10, the watermarked model accuracy in comparison to the original unwatermarked model is not affected drastically.

3) Integrity: As the watermark is the intrinsic character of how a model interprets data, it depends on the model structure and the training parameters. As each model is trained differently, the watermark will be different depending on the dataset and the model. Even if the datasets are the same for two models, like in the MNIST dataset, different model structure outputs have different class mispredictions. If we have the same structure but with different data sets (e.g., the CNN model with MNIST or FashionMNIST dataset), the WM will also be different. We prove in our experiments that the ownership of unmarked models or differently marked models cannot be falsely proved.

4) Efficiency: As a watermark is embedded during the training process, there is no overhead while running the model itself. The only overhead is the time taken to fine-tune the model and the time it takes to extract the WM. Performance overhead is given in Table II. The table shows that the highest time taken to fine-tune with the watermark is 196.74s for the Fashion-MNIST dataset (CNN model) and the highest time taken for extracting the watermark is 0.0042s for Fashion-MNIST (CNN model) and MNIST (CNN model). This proves that our framework efficiently embeds and extracts the watermark from the model.

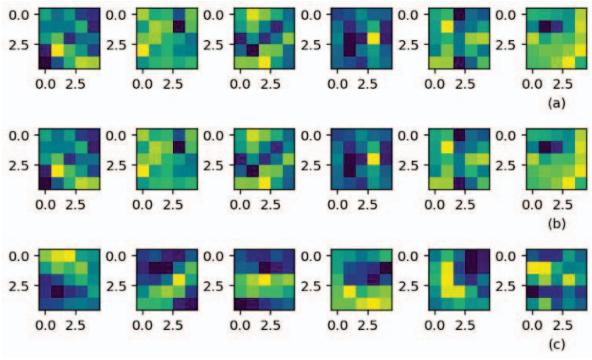

Fig. 4: Kernels (six 5x5) from Conv1 layer. Case (a) & (b) = CNN model trained with Fashion-MNIST dataset without and with the WM (key=5) respectively. Case (c) = CNN model trained with MNIST dataset.

5) Robustness: We found the PDF's of all the weight and biases remain almost the same with some negligible linear shift despite watermarking the models. Hence the watermark is undetected within the DNN model as seen in Fig. 4. The model is considered to be robust when the watermark is easily extracted even after the model has undergone pruning. This comparison is provided in Fig. 5. Note, pruning is performed to improve model accuracy and performance. Hence, in the case where pruning actually reduces model accuracy from 99% to 80%, the model by default would no longer be viable to be used in the real-world. Hence, in such cases, either the pruning rate is reduced or a new model is selected.

6) Security: One can argue that not all weights and biases are equally activated while the model is in use. Hence, to visualize these activations, we have plotted kernels from CONV1 layer for 3 cases as seen in Fig. 4 where each Conv1 layer has six 5x5 kernels. Case (a) & (b) both are from training the CNN model with the FashionMNIST dataset without and with the WM. As (a) is identical to (b), it is clear that embedding watermark in a CNN model had no effect in activation and thus the watermark is undetectable from outside. 3(c) shows the kernel for the MNIST trained CNN model, which is clearly different from both (a) and (b), proving that different models with the same model structure will give different activation, making the watermark virtually untraceable.

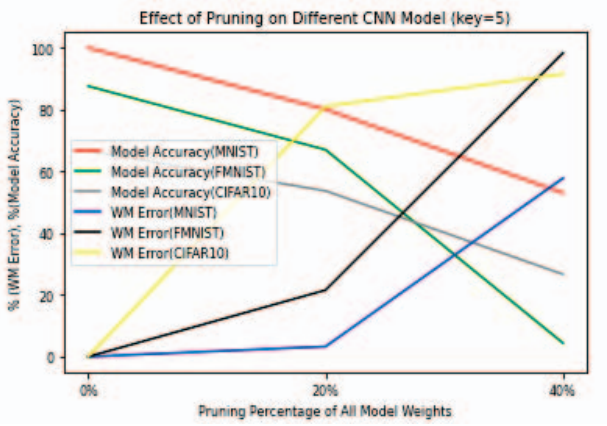

Fig. 5: Model accuracy compared to WM error while no pruning, 20% pruning and 50% pruning of the CNN models.

VI. Conclusion

AIME framework is introduced in this paper, an end-to-end watermarking scheme that provides robust integration of watermarks in the DNN models. AIME is generic as it is applicable to both white-box and black-box models. AIME leverages the class mispredictions of the model to embed a WM in the hidden layers of the network. AIME satisfies all the required criteria for a successful, reliable, and robust watermark embedding procedure. Extensive experiments performed on two models and three datasets corroborate the scalability, effectiveness, and efficiency of the AIME framework. AIME withholds against fine-tuning and pruning of the models.

VII. Acknowledgements

This work was supported in part by the AFOSR Grant #FA9550-14- 1-0351

TABLE II: Performance overhead calculated for the watermarked models in comparison to the original (without WM) model.

Dataset	Model	Performance Overhead (in Time)					
		Without Watermark		With Watermark (#key=5)		With Watermark (#key=10)	
		Training Time	Fine-tuning time (FT) & Extraction time (ET)	Training Time	Fine-tuning time (FT) & Extraction time (ET)	Training Time	Fine-tuning time (FT) & Extraction time (ET)
MNIST	Sequential	162.08s	FT= 0s ET= 0s	162.08	FT= 2.21s ET= 0.004s	162.08	FT= 80s ET= 0.004s
	CNN	553.55s	FT= 0s ET= 0s	553.55s	FT= 4.63s ET= 0.0040s	553.55s	FT= 120.4363s ET= 0.0042s
Fashion-MNIST	CNN	1013.18s	FT= 0s ET= 0s	1013.18s	FT= 8.94s ET= 0.0042s	1013.18s	FT= 196.74s ET= 0.0029s
CIFAR-10	CNN	1381.86s	FT= 0s ET= 0s	1381.86s	FT= 11.78s ET= 0.0031s	1381.86s	FT= 128.55s ET= 0.0018s

TABLE III: Model architectures considered for training. FC = fully connected layer, 16C3 = 16 inputs to the convolutional layer with 3 channels, MP = maxpooling layer.

Dataset & ML Model Type	ML Model Architecture
MNIST-Sequential	1*28*28-784FC-128FC-64FC-10FC
MNIST- CNN & FashionMNIST-CNN	1*28*28-6C1(1)-MP2(1)-16C1(1)-MP2 (1)-256FC-120FC-84FC-10FC
CIFAR-10-CNN	3*32*32-6C3(1)-MP2(1)-16C3(1)-MP2 (1)-400FC-120FC-84FC-10FC

TABLE IV: Class label accuracy for MNIST dataset on the CNN model with and without WM embedding (key= 5 & 10).

	Class Label	Class Label Accuracy		
		Without WM	With WM (#key = 5)	With WM (#key = 10)
Dataset: MNIST Model: CNN	Zero	99.67%	98.36%	97.44%
	One	99.03%	98.14%	98.50%
	Two	99.51%	99.41%	99.41%
	Three	99.70%	99.40%	99.40%
	Four	98.37%	98.16%	97.14%
	Five	97.86%	96.52%	95.96%
	Six	98.85%	98.95%	98.95%
	Seven	99.12%	99.31%	98.34%
	Eight	99.28%	99.48%	99.58%
	Nine	98.01%	97.12%	97.32%

REFERENCES

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

[2] L. Deng and D. Yu, "Deep learning: methods and applications," *Foundations and trends in signal processing*, pp. 197–387, 2014.

[3] K. Yang, D. Forte, and M. M. Tehranipoor, "Protecting endpoint devices in iot supply chain," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 351–356.

[4] M. Dhwani, S. Tajik, D. Woodard, N. Asadi, and M. Tehranipoor, "On the physical security of ai accelerators," in *International Conference on Physical Assurance and Inspection of Electronics*, 2019.

[5] M. M. Tehranipoor, U. Guin, and D. Forte, "Hardware ip watermarking," in *Counterfeit Integrated Circuits*. Springer, 2015, pp. 203–222.

[6] U. J. Botero, R. Wilson, H. Lu, M. T. Rahman, M. A. Mallaiyan, F. Ganji, N. Asadizanjani, M. M. Tehranipoor, D. L. Woodard, and D. Forte, "Hardware trust and assurance through reverse engineering: A tutorial and outlook from image analysis and machine learning perspectives," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*.

[7] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP security and trust*. Springer, 2017.

[8] S. Bhunia and M. Tehranipoor, *Hardware security: a hands-on learning approach*. Morgan Kaufmann, 2018.

[9] M. Tehranipoor, H. Salmani, and X. Zhang, "Integrated circuit authentication," *Switzerland: Springer, Cham. doi*, vol. 10, pp. 978–3, 2014.

[10] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proceedings of the IEEE*.

[11] C.-S. Lu, *Multimedia security: steganography and digital watermarking techniques for protection of intellectual property*. Igi Global, 2005.

[12] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Architectural Support for Programming Languages and Operating Systems*.

[13] H. Chen, B. D. Rouhani, and F. Koushanfar, "Blackmarks: Blackbox multibit watermarking for deep neural networks," *arXiv preprint arXiv:1904.00344*, 2019.

[14] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, 2017, pp. 269–277.

[15] E. Le Merrer, P. Perez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Computing and Applications*.

[16] B. D. Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: A generic watermarking framework for ip protection of deep learning models," *arXiv preprint arXiv:1804.00750*, 2018.

[17] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 159–172.

[18] Y. Nagai, Y. Uchida, S. Sakazawa, and S. Satoh, "Digital watermarking for deep neural networks," *International Journal of Multimedia Information Retrieval*, vol. 7, no. 1, pp. 3–16, 2018.

[19] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[20] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[21] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.