electronics

MDPI

Article

Watermarking Tiny MLCommons Image Applications Without Extra Deployability Costs

Alessandro Carra [1], Dilan Ece Durmuskaya [1,2], Beatrice Di Giulio [3], Laura Falaschetti [4], Claudio Turchetti [4] and Danilo Pietro Pau [5,*]

[1] System Research and Applications, STMicroelectronics, 20007 Cornaredo, Italy; alessandro.carra@st.com (A.C.); dilanece.durmuskaya@st.com (D.E.D.)
[2] Department of Mathematics, University of Pavia, Corso Strada Nuova 65, 27100 Pavia, Italy; dilanece.durmuskaya01@universitadipavia.com
[3] Department of Electronics and Communication Engineering, Università Politecnica delle Marche, Via Brecce Bianche 12, 60131 Ancona, Italy; s1094528@studenti.univpm.it
[4] Department of Information Engineering, Università Politecnica delle Marche, Via Brecce Bianche 12, 60131 Ancona, Italy; l.falaschetti@univpm.it (L.F.); c.turchetti@univpm.it (C.T.)
[5] System Research and Applications, STMicroelectronics, 20864 Agrate Brianza, Italy
* Correspondence: danilo.pau@st.com; Tel.: +39-3485318664

Abstract: The tasks assigned to neural network (NN) models are increasingly challenging due to the growing demand for their applicability across domains. Advanced machine learning programming skills, development time, and expensive assets are required to achieve accurate models, and they represent important assets, particularly for small and medium enterprises. Whether they are deployed in the Cloud or on Edge devices, i.e., resource-constrained devices that require the design of tiny NNs, it is of paramount importance to protect the associated intellectual properties (IP). Neural networks watermarking (NNW) can help the owner to claim the origin of an NN model that is suspected to have been attacked or copied, thus illegally infringing the IP. Adapting two state-of-the-art NNW methods, this paper aims to define watermarking procedures to securely protect tiny NNs' IP in order to prevent unauthorized copies of these networks; specifically, embedded applications running on low-power devices, such as the image classification use cases developed for MLCommons benchmarks. These methodologies inject into a model a unique and secret parameter pattern or force an incoherent behavior when trigger inputs are used, helping the owner to prove the origin of the tested NN model. The obtained results demonstrate the effectiveness of these techniques using AI frameworks both on computers and MCUs, showing that the watermark was successfully recognized in both cases, even if adversarial attacks were simulated, and, in the second case, if accuracy values, required resources, and inference times remained unchanged.

Keywords: watermarking; backdoors; tiny machine learning: micro-controllers; image use cases

Citation: Carra, A.; Durmuskaya, D.E.; Giulio, B.D.; Falaschetti, L.; Turchetti, C.; Pau, D.P. Watermarking Tiny MLCommons Image Applications Without Extra Deployability Costs. *Electronics* **2024**, *13*, 4644. https://doi.org/10.3390/electronics13234644

Academic Editor: Silvia Liberata Ullo

Received: 2 October 2024
Revised: 19 November 2024
Accepted: 22 November 2024
Published: 25 November 2024

1. Introduction

The seminal work of [1] has proven the ability to convert high-dimensional tensors into low-dimensional representations by training a multi-layer neural network (NN) with a tiny layer in the middle of the topology to approximate the inputs accurately. Since then, artificial intelligence (AI) has been increasingly developed and applied in many ways, thus becoming more and more pervasive in everyone's daily life [2].

In the AI and data science domains, machine learning (ML) plays an important role due to its capabilities to learn from and process large amounts of heterogeneous data, thus enabling ML workloads to automatically provide results which are often more accurate than the ones generated by hand-crafted or traditional ML approaches [3]. Thus, ML has been proven in a wide range of domains, such as image recognition, facial recognition, object detection, natural language processing, and machine translation, among others.

ML is also a key technology in autonomous systems like self-driving cars and humanoid robotics, where ML workloads process sensor data constantly acquired in real-time and provide critical decisions which are actuated.

The initial deployment of ML models was on powerful computing devices, as they are typically available in Cloud infrastructures. They are positioned very far from data sources. This approach has been proven to not scale with the ever-increasing number of users over the years. Furthermore, there are a number of other concerns when ML workloads are Cloud-centric, such as the instability of Cloud connectivity in rural, harsh, and remote areas, data privacy, too high computational costs, and power consumption, just to name a few. The awareness of these challenges and the need to find new approaches to achieve ML decentralization on Edge gave rise to the TinyML community, which was established in 2019 [4]. The TinyML Foundation, at the heart of this community, is a worldwide non-profit organization empowering a community of professionals, academic researchers, students, and policymakers focused on low-power AI at the very edge of the Cloud. Moreover, TinyML is a category of ML that allows models to run on smaller, less powerful devices. It involves hardware, algorithms, tools, and embedded software that can analyze sensor data on these devices with very low power consumption (e.g., under 1 mW), making it ideal for always-on use cases and battery-operated devices.

Thanks to the advent of TinyML in several markets, tiny neural networks (tiny NNs) became increasingly popular, and they have been employed in many embedded applications running on low-power devices, such as image classification use cases developed for MLCommons benchmarks [5]. They are off-the-shelf micro-controllers (MCUs) running at several hundreds of MHz (the operative core frequency), embedding up to 1 Mbyte of RAM and 2 Mbytes of FLASH memories, for example. The MCU can run tiny NNs that process data at either floating-point precision (fp32) or an integer (e.g., 8 bits) with a range of performances adequate for most of the embedded AI applications currently available on the market.

Despite the small footprint and low cost of these embedded devices, many AI developers still require lots of hand-crafted ML design. Also, the training processes to get accurate tiny NN solutions are costly and time-consuming. Moreover, most of the project time is spent on dataset creation, labeling, and updating.

Since tiny NNs are the essential core technology in Edge AI, unfortunately, they can be easily copied. They represent the intellectual properties (IPs) of many small and medium enterprises (SMEs); therefore, protecting them against unwanted copies is of paramount importance [6]. This is currently considered a top priority. Therefore, the ML community is required to enable them to achieve such a level of protection.

The application of robust watermarking in neural networks extends beyond theoretical concepts and provides practical benefits for real-world users across various domains. For example, in the field of computer vision, watermarking techniques have been effectively applied to NN models trained with CIFAR-10 to protect intellectual property. A scenario can be envisioned in which a trained neural network for image classification is purchased. By embedding a robust watermark, the origin of the network can be verified, ensuring that the acquired model is genuinely the one intended and protecting against fraudulent replacements. This capability extends to the detection of unauthorized copies; if a stolen or duplicated network is used without permission, the embedded watermark can serve as a unique identifier, allowing rightful ownership to be asserted. This protective mechanism holds particular significance in industries that deploy their proprietary models in small devices, such as the IoT sensors distributed in large areas. Watermarking these models ensures that their origin can be verified regardless of their deployment, thus providing a layer of security and fostering trust throughout the model's lifecycle.

The possibility of forcing a unique behavior or making NN layer weights unique has been examined using two state-of-the-art methods, which led to the development of two adaptable approaches that can be applied across various Edge AI use cases, particularly within image classification neural networks. These methods were designed to enhance

flexibility and robustness, enabling effective deployment in resource-constrained Edge environments. The resulting solutions were tailored to ensure compatibility with typical Edge AI scenarios, addressing challenges such as limited computational resources and data constraints while maintaining high accuracy and responsiveness in image classification tasks using only a few dozen layer weights. Overall, two NNW methods are examined in this study. For the first one, a black-box approach is considered, and three distinct approaches for generating these trigger sets are explored. For the second method, a white-box application is considered, focusing on embedding the watermarks directly into the model's weights through selective modification of specific bits using layer weight regularizers. All the models extracted with the different combinations considering the use cases and the NNW methods are evaluated by computing adequate metrics and leveraging a third-party methodology, following the MPAI (https://mpai.community/ (accessed on 13 November 2024)) standardized evaluation procedure under attack simulation.

The main contributions of the paper are summarized as follows:

- We define a methodology that allows developers to associate a unique identifier to the NN model that is recognizable to avoid wasting investments;
- We define multiple watermarking approaches according to a methodology that can be reproducible;
- We investigate the possibility of applying watermarking to a hypo-parametric model (more challenging than applying watermarking to a model with a high number of parameters);
- We define a method that requires no extra cost in deploying the device;
- Given the industrial importance of preserving the IP, we define a watermarking method that meets this specification.

Before delving deeper into the details, the paper is organized as follows: Section 2 describes the background knowledge essential to understanding the watermarking process applied to NNs; Section 3 describes the known approaches in the literature; Section 4 defines the problem this paper addresses with the definition of some requirements to be fulfilled; Section 5 describes the use cases considered for the watermarking application; Section 6 describes the approaches that have been coded to achieve the watermarking of tiny NNs; Section 7 explains the results achieved in this work with the related interpretations; Section 8 shows the results achieved through deployability on the MCU; Section 9 explains the results achieved after some attacks were applied to verify the robustness of the watermarking processes; Section 10 concludes the paper and introduces some future developments about the continuation of the research.

2. Background on Watermarking

A digital watermark (DW) can be defined as a marker embedded in a noise-tolerant processing algorithm such as the NN pipeline. It can be used to identify IP ownership. More precisely, digital watermarking is the process of hiding digital information, for example, among the hyper-parameters of an NN such that the hidden information embodied shall contain a relation to them. A DW may be used to verify the authenticity or integrity of the NN, which would be intended as IP, or to prove the identity of its creator. It is typically used to help detect copyright infringements. Moreover, the DW should not negatively affect the expected performance in operative conditions and when the copyright infringement does not need to be detected. If the DW distorts the NN performance, it is considered less effective and intrusive performance-wise.

A DW process can be designed based on the level of access it has. Additionally, it can be divided into two main areas:

- White-box watermarking;
- Black-box watermarking.

In white-box watermarking, the process is designed to be open and transparent, allowing authorized individuals to fully understand and verify the embedded watermark.

In black-box watermarking, the process is designed to be hidden and inaccessible, making it difficult for unauthorized individuals to comprehend or tamper with the watermark [7].

The required properties of a DW may depend on the use case in which it is applied. For marking parameterized models, with copyright information, such as an NN, a DW has to be rather robust against modifications such as knowledge distillation (KD), pruning, weights and activations quantization, additive noise to the inputs, etc. A digital copy of the NN is exactly the same as the original. DW is a passive approach; it can just mark the parameters and does not degrade NN task performance as expected with respect to the unwatermarked (unWMd) version [8]. One application of DW is source tracking. A watermark can be embedded into an NN model in several different stages. In the event that an unauthorized copy of the model has hypothetically been found, these NNW methods can help the owner to do the following:

- Provide proof that the originally inserted watermark has been recognized in the tested model;
- Provide the keys and extraction and detection method used to assert the previous point.

These can be useful when malicious individuals take advantage of unclaimed IP. This helps to protect IP and ensures that the model cannot be used unlawfully without attribution.

NN watermarking (NNW) is composed of two main steps [8]:

1. Embedding the watermark: The process begins by selecting a DW, which could be a sequence of bits or a unique pattern. The DW is embedded into the NN model during the training phase or post-training phase. This can be achieved by subtly altering the parameters of the model, such as weights and biases, to inject the DW information. Alternatively, it can be integrated into the inputs as well;
2. Detection: Once the NN has been trained and the DW embedded, it can be deployed for its intended task. To verify the authenticity of the model, the DW needs to be extracted. This is typically performed by providing a set of inputs to the model and observing its outputs. Specialized algorithms, such as trigger set-based detection [9], steganalysis-based Detection [10], and zero-bit watermarking [11], are then used to analyze the model's behavior and extract the DW from its parameters or its activations. The extracted watermark is compared against the original one to verify the authenticity and ownership of the model.

DW techniques face several challenges, including ensuring robustness against malicious attacks, preserving content quality, and providing security to prevent unauthorized detection and removal. Effective DW must balance these challenges while also offering sufficient capacity and computational efficiency. To address these issues, experiments can be conducted to test robustness against attacks such as Gaussian noise, pruning, and quantization attacks, as reported in Section 9. Afterward, the watermark is evaluated to assess efficiency. These experiments help refine techniques to ensure they meet the necessary requirements and perform effectively in real-world scenarios.

3. Related Works

Recent studies have investigated watermarking methods. Some of them were related to ML algorithms, and others to deep NNs (DNNs). Related works in digital watermarking can be categorized into three macro areas:

1. Watermarking applied to data:
   (a) Without ML methods;
   (b) With ML methods;
2. Watermarking applied to NNs by modifying layers and weights:
   (a) Fine-tuning the pre-trained model;
   (b) Training the ML approach from scratch;
3. Watermarking based on the level of access to the ML model:
   (a) White-box watermarking;

(b) Black-box watermarking.

3.1. Watermarking Applied to Data

This section discusses the DW achieved with and without ML approaches.

3.1.1. Without ML Methods

The work exposed in [12] proposed a watermarking algorithm for colored images, combining the discrete wavelet transform (DWT), the discrete cosine transform (DCT), singular value decomposition (SVD), and scrambling. More investigations into watermarking, based on DWT and SVD, using colored images were discussed in [13,14]. Ref. [13] proposed a semi-blind image watermark scheme that used the finite ridgelet transform (FRT), particle swarm optimization (PSO), and the Arnold transform, in addition to the conversion of RGB images into YCbCr color space [13]. Ref. [14] presented a robust image watermarking approach, considering the image as a third-order tensor and transforming it using tensor-SVD [14]. In medical imaging, SVD was applied combined with the fast curvelet transform (FCT), achieving WMd medical data [15]. Further, SVD has been employed for watermarking authentication and self-recovery in [16], and images were divided into four groups and fed into the SVD algorithm, with the results replacing the least significant bit (LSB) of each block. Another choice introduced was a watermarking scheme based on a block mapping algorithm and dual matrix, generating authentication and recovery data from the original image [17].

Ref. [18] employed watermarking systems for colored images based on the fusion of spatial domain and LU factorization. LU factorization is a method used to decompose a matrix as the product of a lower triangular matrix and an upper triangular matrix [19]. The watermark was embedded by altering elements that exhibited strong correlations with the L matrix (lower diagonal matrix). Another application of the watermarking technique utilizing LSB substitution was by [20], and a blind fragile watermark was applied for colored images to detect manipulations and enable self-recovery; this method employed a pseudo-random binary sequence based on a secret key as the watermark, with recovery information stored accordingly.

3.1.2. With ML Methods

While former studies focused on applying watermarking techniques directly to data, there was an increasing tendency to integrate ML algorithms as part of watermarking methods.

A review was presented in [21] that introduced a watermarking process exploiting a multi-head cross-attention mechanism. The watermark was generated by resizing the image, which was then binarized using a pixel threshold approach. The process was repeated to create a mixed watermark. The system included four components, which were as follows:

- An embedder;
- An encoder;
- A decoder;
- An extractor.

These components worked together to create the WMd image, aiming to take out critical characteristics through the generation of invariant domains and extracting the watermark, which had to be similar to the original. The encoder incorporated convolutional layers and fully connected layers. Another method combined convolutional neural networks (CNNs) with DWT for embedding and extracting watermarks [22]. This method involved two stages to embed the WMd into the image:

- For the image host;
- For the watermark.

The DWT was used to split up the colored host image into four levels, one of which was chosen for watermark inclusion. During these processes, CNNs facilitated the watermarking procedure.

Furthermore, a "box-free" multi-bit watermarking method was introduced by [23] to protect IP, which was robust against various attacks. This method incorporated watermarking by introducing a loss factor during the training of GANs, ensuring the presence of an invisible watermark in images that can only be recovered by a pre-trained watermark decoder. This approach was adapted to CNN-based architectures, protecting their robustness by converging to a large and flat minimum of the watermark loss factor.

Ref. [24] proposed an end-to-end network for watermarking, and a CNN assessed the robustness based on image content. This approach embedded, attached, and extracted gray scale watermarks within a single frame. The method employed a stochastic gradient descent (SGD) optimizer during training to improve robustness.

Ref. [25] used a watermarking method to protect GAN IP by embedding a watermark in the model. This method involved inserting a pre-trained CNN watermark that decoded a block into the G output and modifying the G loss to include a watermark loss term, which made it easier the watermark extraction from the generated images. The watermark was then incorporated through a fine-tuning procedure.

3.2. Watermarking Applied to NNs by Modifying Layers and Weights

This category did not focus on data but specifically on NNs themselves.

Ref. [26] proposed a new framework called PreGIP to mark graph neural network (GNN) encoder pre-training, safeguarding IP while preserving the quality of the embedding space. This framework incorporated a task-free watermarking loss to mark the embedding space of the pre-trained GNN encoder. Furthermore, it implemented fine-tuning resistant watermark injection, ensuring similar representations for each pair of different watermark graphs during pre-training. This approach ensured that if the representations of two paired WMd graphs were close enough, they would receive the same predictions from the classifier. In contrast, an independent model trained on unlabeled data would produce distinct representations for each pair of labeled graphs, leading to inconsistent results.

Ref. [27] examined the watermarking capability of NNs from an information theory perspective. A new definition of watermark capacity for NNs was proposed, similar to channel capacity. The study analyzed its properties and defined an algorithm to estimate its upper limit in cases of adversarial overwriting. Furthermore, a method was proposed to safeguard the transmission of identity messages through multiple rounds of ownership verification. Ref. [28] applied watermarking for IP protection of NN models against an extraction model called MEA-Defender. The watermark was achieved by combining samples of two original input classes.

Moreover, exploration involved a taxonomy of various watermarking techniques for ML models, introducing a threat model to compare various methods in different scenarios [29]. This taxonomy outlined the desired requirements and attacks against watermarking models for ML.

Ref. [30] investigated watermark removal and highlighted the limitations of existing watermarking methods.

Ref. [31] proposed incorporating the concept of "identity bracelet" into DNNs as a means of protecting IP. This approach extended existing trigger set watermarking techniques by embedding a post-cryptography style serial number directly into DNNs, where both the input data and the watermark were processed within the same network.

The related works' studies suggested that none of the research mentioned used a watermarking algorithm without affecting the pre-trained NN.

Additionally, regarding integrating watermarks into ML models, there were two main approaches: training from scratch and fine-tuning pre-trained models, which are addressed in the following sections.

3.2.1. Fine-Tuning the Pre-Trained Model

Starting from a pre-trained ML model [32], the watermark was introduced as part of the fine-tuning process itself. Since the model already had pre-learned weights from a dataset, the watermark was incorporated during the adjustment phase when the ML model was adapted to a new dataset or task. This allowed the watermark to be embedded in a model that already featured a certain level of knowledge, potentially making the watermark more robust against removals or tampering.

3.2.2. Training from Scratch

A unique watermark was introduced by [33] during the initial training of the model. As the model was learnt all the way using a specific dataset, the watermark, often a unique pattern or set of data, was embedded within the model's parameters. This meant that the model was designed to recognize and respond to the watermark from the outset, establishing a clear marker of ownership or origin.

3.3. Watermarking Based on Level of Access

A white-box method is applied to a system where the internal operations are fully known and transparent. The term comes from being able to "see through" the black box. The terms "white box" and "black box" are used across various fields, including computer science, engineering, and ML, to describe systems based on the level of internal visibility and knowledge available to the observer or user. A black-box method is applied to a system that can be analyzed in terms of its inputs and outputs without any knowledge of its internal works. The term implies that the internal mechanisms are either unknown or irrelevant to the user. Depending on the level of access to the NN model's internals, watermarking methods and attacks can be classified as either black box or white box.

3.3.1. White-Box Watermarking and Attacks

In the context of ML, a white-box NN is interpretable and explainable, allowing humans to understand and follow the decision-making process of the model. Similarly, in white-box watermarking, the process is designed to be transparent and understandable, enabling authorized users to comprehend and verify the watermark. Examples of interpretable models, such as decision trees and linear regression, parallel the concept of white-box watermarking by providing clarity and traceability of their operations.

In adversarial ML, a white-box attack is one approach in which the attacker has complete knowledge of the ML model, including its architecture, hyper-parameters, and training data. This allows accurate crafting of the adversarial examples to fool the model.

Table 1 summarizes existing works on NNW using white-box approaches.

Table 1. Related works on NNW focussed mainly on white-box methods.

Method	Author	Level of Access
"DeepSigns": A combination of multi-bit and one-bit watermark framework gives a WMd version with its key [32]	Chen et al.	White box and black box
Embedding a key-vector into the parameters of a DNN [33]	Uchida et al.	White box
Protect WMd parameters by blocking them with a DNN [34]	Tartaglione et al.	White box
Embeds a fragile watermark as a bit string that can be altered even if a parameter changes [35]	Botta et al.	White box

3.3.2. Black-Box Watermarking and Attacks

In the context of ML, black-box NNs are complex and non-interpretable, making it difficult to understand how these models generate any decisions. Similarly, in black-box

watermarking, the process is designed to be non-transparent and obscure, preventing unauthorized users from understanding or manipulating the watermark.

In adversarial ML, a black-box attack occurs when an attacker has no knowledge of the model's internals and must rely on the model's outputs to construct the attack. This type of attack is more common in real-world scenarios where attackers do not have access to the target model's details.

Table 2 summarizes existing works on NNW using black-box approaches.

Table 2. Related works on NNW focussed mainly on black-box methods.

Method	Author	Level of Access
The key inputs should be protected against forging attacks using a hash function [11]	Zhu et al.	Black box
"DeepSigns": A combination of multi-bit and one-bit watermarks framework gives a WMd version with its key [32]	Chen et al.	White box and black box
Key inputs used outside of the dataset [36]	Adi et al.	Black box
The prediction layer should be actively modified to poison the attacker's training [37]	Orekondy et al.	Black box
Key inputs within the dataset should be visibly modified and associated with a new label [38]	Zhong et al.	Black box
Adversarial inputs created by the IFGSM algorithm should be used as keys [39]	Merrer et al.	Black box
A specific behavior on key inputs is created by modifications done by API Dawn [40]	Szyller et al.	Black box

In addition to state-of-the-art watermarking methods, NNW techniques can also harness the intrinsic structures of different DNN architectures to embed watermarks. For example, in CNNs, watermarks can be embedded within specific convolutional layers, feature maps, or filters, leveraging these networks' hierarchical structure and spatial patterns. This allows for seamless integration that remains resilient to network modifications. On the other hand, recurrent neural networks (RNNs), which operate on sequences and temporal dependencies, can incorporate watermarks within their recurrent connections or hidden state transitions. By embedding patterns within sequences or modifying specific hidden state behaviors, watermarks can be effectively concealed while preserving functionality. GNNs further offer unique opportunities for NNW by embedding watermarks within nodes, edges, or global graph representations. These structural modifications allow watermarks to be intertwined with the graph data processing, enhancing robustness to structural transformations or attacks. Leveraging the architectural properties of different DNNs enables the creation of highly resilient and adaptive watermarks that align with the operational characteristics of each specific network type, enhancing their resistance to attacks and modifications.

Exploring the possible digital object that can be watermarked in the field of NNs, Wu et al. [41] introduce an innovative watermarking framework designed for deep neural networks that produce images as outputs. In this framework, any image generated by a watermarked neural network includes a specific watermark, changing the usual way of watermarking an NN hiding secret keys into the weights or in the prediction of small batches of data.

Continuing the discussion of different types of neural network watermarking techniques, it is also important to consider fragile NNW. This specific approach contrasts with robust watermarking by being highly sensitive to any modifications made to the DNN. Unlike robust watermarks, which aim to ensure model changes such as pruning, quantization, or fine-tuning, fragile watermarks are designed to detect even the smallest alterations in the model's parameters or architecture. Such sensitivity makes fragile watermarking

particularly suited for integrity verification and tamper detection, as any unauthorized changes can cause the watermark to degrade or disappear entirely, signaling potential manipulation. By embedding these watermarks through precise adjustments to weights, activations, or structural components, fragile watermarking tightly ties itself to the unaltered state of the model, making it a powerful tool for scenarios demanding strict authenticity and security assurances.

4. Problem Statement and Requirements

The problem this paper addressed was how to watermark tiny NNs so that IP was protected at the highest level possible. Furthermore, it was about guaranteeing its deployability on tiny Edge devices such as MCUs without affecting the behavior of the model and the resources required to run the model's inference. In terms of requirements, the solution to the problem shall provide support for the following:

- Uniqueness, Secrecy, and Invisibility. NN watermarking uses a specific, distinct key that is unique while remaining secret and confidential, which prevents unauthorized users from detecting, altering, or removing it. A watermark should be invisible; therefore, it has to be imperceptible and undetectable. Additionally, it should not visibly alter the original host model, ensuring the legitimacy of the content. Importantly, any increase in inference time due to watermarking must be minimal to avoid negatively impacting user experience;
- Requirements for the MCU. Since the network should fit in the MCU, it should be tiny enough to be processed by the device with low latency, and the application of the watermark should use the least possible RAM and FLASH;
- Performance requirements and time of execution after the application of watermarks. The watermark needs to be applied without any impact on memory footprint and inference time. Since watermarking could actually affect the performance of the system, computational overhead and execution times for embedding extraction and detecting watermarks are crucial factors to be considered. It is important to optimize processes to minimize the impact on overall performance and ensure balance between watermarking robustness and invisibility, explained in the following previous points: security and robustness of watermarks in NNs and uniqueness, secrecy and invisibility.
- Security and robustness of watermarks in NNs. NNW techniques should be designed with security and robustness to withstand various attacks. Adversarial attacks, such as model fine-tuning or parameter pruning, can attempt to remove or tamper with the embedded watermark. Robust watermarking techniques aim to withstand such attacks and ensure the watermark remains intact. Moreover, the technique should be efficient in terms of memory usage to facilitate deployment on resource-constrained platforms while maintaining robustness.

5. Use Cases

MLCommons [5] is an AI engineering consortium built on a philosophy of open collaboration to improve AI systems. Through collective engineering efforts involving industry and academia, it supports how to measure and improve the accuracy, safety, speed, efficiency, and safety of AI technologies, helping companies and universities build better AI systems for the benefit of society. One working group in MLCommons is the tiny MLPerf Inference. It defined the benchmark suite to measure how fast tiny Edge device systems can process inputs and produce results using a pre-trained NN model. The MLPerf Inference benchmark definition details the motivation and guiding principles behind the benchmark suite. Ref. [42] was the first industry-standard benchmark suite for ultra-low-power tiny ML solutions. It implemented a modular design that enabled benchmark submitters to show the benefits of their solutions, regardless of where they fall on the ML deployment stack, in a fair and reproducible manner. Two (out of four) tiny MLPerf use cases were considered by this work for experimentation: visual wake words (VWW) and image classification (IC), which are further described in Sections 5.1 and 5.2.

5.1. Visual Wake Words

This is a relatively simple classification task. The VWW dataset [43] addresses the detection of at least one person in an image. This task is directly relevant to smart doorbell and occupancy applications, and the reference network provided as part of the challenge fits most 32-bit embedded MCUs available on the market. The VWW dataset [44] is used for the training, validation, and test datasets for all person detection models. The dataset is pre-processed to train on images that contain at least one person occupying more than 2.5% of the source image. Images are also resized to 96 × 96 for model training. The VWW NN is a MobilenetV1 [45], which takes 96 × 96 input images with an alpha of 0.25 and outputs two classes (person and no person). The pre-trained TensorFlow Lite (TFL) model footprint is equal to 325 KiB.

5.2. Image Classification

It employs the dataset CIFAR-10 [46], a labeled subset of the 80 Million Tiny Images dataset [47]. The low resolution of the images makes CIFAR-10 the most suitable source of data for training tiny IC models. It consists of 60,000 32 × 32 × 3 RGB images, with 6000 images per class. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is divided into five training batches and one testing batch, each with 10,000 images. A significant amount of prior work in tiny ML has used CIFAR-10 as a target dataset [48]. The IC model is a customized ResNet-8 [49], which takes 32 × 32 × 3 input images and outputs a probability vector of size 10. The custom model is made of fewer residual stacks than the official ResNet: three compared to four. The pre-trained TFL model for IC is 96 KiB in size and fits on most 32-bit embedded MCUs.

6. Approaches Being Modeled

This work focused on two approaches to achieve NNW, each of them with some variants. The aim was to apply the two approaches to the use cases in Sections 5.1 and 5.2, thus evaluating the performance and effectiveness of the watermarking algorithm in different application contexts. In Sections 6.1 and 6.2, the features of the two algorithms are explained in detail.

6.1. Watermarking by Means of Regularization

Ref. [33] proposed an NNW technique with the use of a custom regularizer, which was a function defined by the user to train the model. This method was employed by defining a watermark key, a watermark matrix, and a custom regularizer. This method ensured that the watermark was robust against various attacks and modifications, keeping the same accuracy, computational efficiency, and purpose while embedding a secure and detectable signature within its parameters. The goal was to embed this information in a way that did not compromise or alter the original functionality and performance of the NN. The process used is explained in Sections 6.1.1–6.1.4.

6.1.1. Watermark Key

Through this approach, a specific vector called b was embedded into the weights of the NN. The vector has a certain size and composition made of a combination of bits. Accordingly, it has been incorporated exclusively into the weights of a single layer within the network. In the proposed work, the definition and initialization of b had been done before the training took place, including its length and the bits chosen inside it. Moreover, Sections 6.1.3 and 6.1.4 report that b is used during the training of the model to compute a regularized term and during the detection to prove the ownership of the model.

6.1.2. Watermark Matrix

The watermark matrix is a core element of this method, which embeds and extracts the watermark. This matrix has a specific shape, where the number of rows corresponds to the product of the first three values of the weight's shape, and the number of columns

corresponds to the first value of the shape of the vector that is supposed to be the key. The watermark matrix can be built using the following three methods:

- Direct: each row embeds one element set to "1", with its position chosen randomly within the row, while all other elements are set equal to "0";
- Diff: each row embeds two specific elements, one equal to "1" and another equal to "−1". Their positions are chosen randomly within the row, while all other elements are set to "0";
- Random: values are initialized randomly following a standard normal distribution $N(0, 1)$, with the mean equal to 0 and variance equal to 1.

Figure 1 shows how this matrix can be set in three different ways.

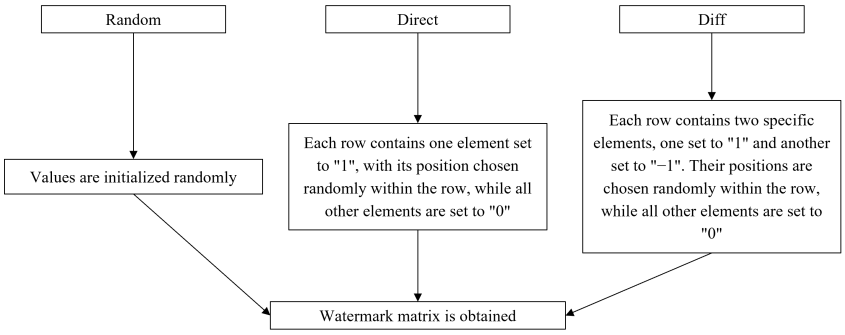

Figure 1. Process to set the watermark matrix using one of the three options.

One of the three options needs to be performed before the model's training.

6.1.3. Regularization and Updating of Weights

The watermark is applied through the watermark regularizer class. It is defined as the R_{term} added to the original cost function to train the NN, as explained in Equation (1):

$$New\,cost\,function = Original\,cost\,function + R_{term}. \quad (1)$$

The additional regularized term is defined as the binary cross entropy between vector b and the sigmoid activation of the scalar product obtained from the watermark matrix and flattened weights, as in the following equation:

$$R_{term} = k * \sum(-(b_j * \log(y_j) + (1 - b_j) * \log(1 - y_j))), \quad (2)$$

where b is the watermark key, as explained in Section 6.1.1, y_j is the term defined in Equation (4), and k is the scale factor experimentally chosen. The regularizer term calculation was done through a function called automatic in the NN training process. Since the watermark can be applied to a single layer of the NN, it can be introduced as a custom regularizer in that layer. In this work, a convolutional layer was selected as the one to watermark. Specifically, during training, the weights w of the convolutional layer chosen for watermarking are multiplied by the watermark matrix M, as shown in Equation (3):

$$w' = \sum M_{ji} * w_i. \quad (3)$$

As explained in Section 6.1.2, the number of rows i of the watermark matrix, M_{ji}, corresponds to the product of the first three dimensions of the weight tensor in the layer where the watermark is to be applied, and the number of columns i corresponds to the dimension of the key vector, as described in Section 6.1.1. The meaning behind this choice is to make sure that in the extraction process, the matrix can be multiplied by the weights of the WMd layer w_i; also, the extracted vector then has the same size as the vector b. The values different from zero in the watermark matrix influence the weights. The result of

Equation (3) is then passed through a sigmoid activation, which normalizes all the values between 0 and 1, as shown in Equation (4):

$$y_j = \sigma(w') = \frac{1}{1 + exp(-w')}. \quad (4)$$

Equation (4) and vector b are passed through a binary cross entropy, and then a scale factor k is applied to obtain the regularized additional loss term as per Equation (2).

All these steps taken for the additional loss term belong to the so-called injection process. The whole process is shown in Figure 2.

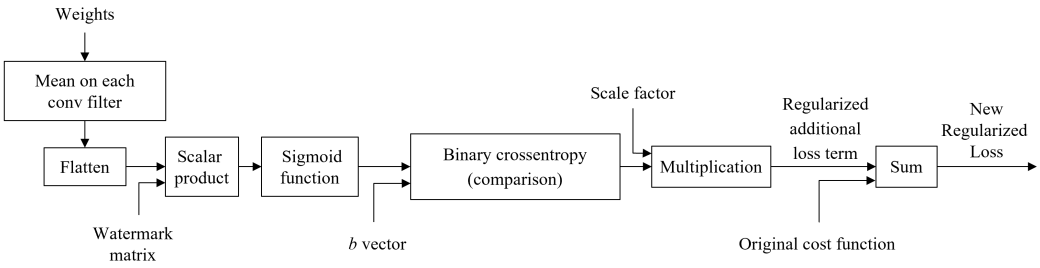

Figure 2. Each step that has been taken to get to Equation (1).

During training, a function to obtain the regularized term in Equation (2) is identified for every batch size in each epoch. The next step involves the extraction of the watermark, as discussed in Section 6.1.4.

6.1.4. Extraction and Detection of the Watermark

Extraction and detection occur after training. Extraction is performed by re-applying the sigmoid function in Equation (4) but with the WMd weights instead. After applying the sigmoid activation, an array of values between 0 and 1 with the same size as b is obtained. A threshold is then defined, and values greater than this threshold are classified as 'True' or '1', as shown in Equation (5).

$$\sigma(\sum M_{ij} w_{WMd}) \geq Threshold. \quad (5)$$

Figure 3 shows the complete process to extract the watermark from the model.

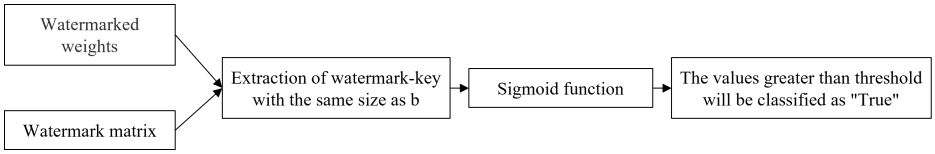

Figure 3. Watermark extraction and detection process.

The same threshold used in the state-of-the-art method [33] equal to 0.5 is used. Watermark embedding is considered successfully detected if the bit error rate (BER) is equal to zero. The BER measures the number of positions at which the corresponding extracted symbols are different from b, averaged over the total bits, as expressed in Equation (6).

$$BER = \frac{\sum(\sigma(\sum M_{ij} w_{WMd}) \neq b_j)}{\text{size of b}}. \quad (6)$$

6.2. Watermarking by Means of Backdooring

Ref. [36] introduced a method called backdooring to incorporate watermarks into ML algorithms. The work also leveraged commitment schemes to provide a more comprehensive framework to achieve watermarking. This method consisted of three key steps:

1. $Keygen()$ generates the secret marking key and the verification key;

2. $Mark(M, mk)$ embeds the watermark (mk) into the model (M);
3. $Verify(mk, vk, M)$ verifies if the model (M) contains the watermark using the marking key (mk) and verification key (vk).

6.2.1. Backdooring Algorithm

This is a known technique used to intentionally train a model to produce incorrect labels to be associated with specific inputs. It involves setting a trigger dataset and a labeling function, which consists of inputs designed to trigger the backdoor and assign incorrect labels to these inputs. A backdoor comprises the trigger set and its corresponding labeling function. A backdooring algorithm takes an oracle (a function that simulates the model's behavior), a backdoor, and a model as input. This algorithm outputs a model that is likely to misclassify inputs from the trigger set with a high probability. Backdoors can be embedded in models either during training or starting from a pre-trained model. There are also strong backdoors, which are enhanced versions of backdoors designed to be more resilient and persistent. They feature the following properties:

- They incorporate multiple trigger sets, making them harder to detect and remove;
- They are difficult to remove without knowledge of the trigger datasets, ensuring their longevity and effectiveness.

They use the sample backdoor algorithm [36] that obtains the required backdoor. This watermarking method also uses algorithms that leverage trigger sets as secrets. The sender can lock a secret, and the receiver can later verify the secret's authenticity. These algorithms [36] are as follows:

- One locks the secret using a random bit-string;
- Another verifies the authenticity of the secret by checking if the provided proof matches the original secret and bit-string.

These algorithms try to ensure that neither the sender nor the receiver can access the secret without each other's cooperation.

6.2.2. Trigger Datasets

The trigger dataset is a specific set of data and corresponding labels. The data should match the expected input of the NN (e.g., input shape, values format, etc.). They should be distinct and uncorrelated with regular training data and, compared to the latter, should be composed of fewer samples, about 0.2%. For each trigger sample, a label has to be assigned. Both the data generation process and the label assignment should be unique and secret to ensure that only the IP owner can recognize their own trigger set. To guarantee that the trigger set is adequate, the incoherent assignment of labels must be proven rather than their ground truth classification. The main goal of this approach is to fix the behavior of the NN model when the trigger dataset is used as its input and to obtain a unique prediction. In this work, the watermark is injected only during training, and the trigger dataset is concatenated with the original training data. The NN learns to produce outputs for trigger inputs while still performing its primary task accurately on non-trigger data. To verify ownership, trigger inputs should be provided to the NN while observing the outputs. If these match the expected trigger outputs, it means that the NN model has embedded the watermark, thus providing evidence of its ownership.

6.2.3. Generation of the Trigger Dataset

Ref. [36] was an example that leveraged a trigger set during training. Since the watermarking should be unique for each use case, the key has to be unique for each embedding as well. Therefore, it is essential to find a way to generate a unique key for each use. While a user could potentially select images, as [36] did, this approach did not guarantee the uniqueness or robustness of the dataset. Basic features that a trigger set shall own have been identified, which can be implemented in various ways. One of the primary objectives of the work presented in this section was to explore, analyze, and

develop different methods to generate the trigger set while exploring different ways to apply watermarks to tiny NNs. Thus, only the WMd model can produce specific outputs associated with unique random labels, provided that the unique trigger data is used as input.

The process begins by defining a helper function to create a one-hot encoded vector for a single class index. Next, a list of these one-hot encoded vectors has been generated, with each vector corresponding to a randomly selected class for each image. This randomness ensures that each image is assigned to a class without any specific order or pattern, which is crucial for incoherent label assignment in watermarking with trigger sets. After generating the required number of labels, the function converts the list of vectors into a NumPy array.

After the generation of the trigger samples, saving them in the same file with the labels ensures the easy distribution and use of the dataset are achieved, allowing the labels to be used in various applications such as training classification models, where the one-hot encoded format is particularly helpful for representing the target classes. The function finally returns the array of labels, providing immediate access to the generated data for further use.

Several approaches were explored to gather a set of images that met the conditions for a trigger set. These approaches are described in Sections 6.2.4–6.2.6.

6.2.4. Trigger Dataset Generated Using GANs

To guarantee the distinctiveness required by the watermarking applications, one of the preferred approaches involved the creation of novel, one-of-a-kind images. This was achieved through the application of generative adversarial networks (GANs), which were capable of synthesizing unique visual content. GAN was introduced by ref. [50] as the first approach for generative AI (GenAI).

GANs are a class of ML topologies capable of generating realistic image data by pitting two NNs against each other. A GAN consists of two main components.

Generator (G) generates new image samples from random noise inputs. G is initialized to produce random outputs. Then, G is trained to produce images that the discriminator (D) will classify as real.

D is initialized to classify input images as real or fake. It evaluates the authenticity of the generated samples by G, distinguishing the real samples from the fake samples G generated.

The NNs are trained in alternating steps: this is done by back-propagating the D's feedback through the G, adjusting its weights to improve the realism of the generated images. The training aims to reach a state where the G produces highly realistic images that the D can no longer distinguish from real images, leading to a D accuracy of around 50%. D's loss measures how well the D can distinguish between real and fake images, and the G's loss measures how well the G can fool the D into believing its outputs are real. Over time, the G improves and produces increasingly realistic images that are difficult to differentiate from real images, achieving high-quality data generation. However, in this work, the G produced images without the need to be fully differentiable from the original images, such that they can be distinguished from each other. They were only required to be classifiable somehow and uncorrelated with each other. In the approach implemented, the G model was loaded in memory at the beginning from a given file. G has been previously trained to generate images that mimic the distribution of CIFAR-10. Random noise vectors were the input to G as seeds from which G produced many images. For each noise vector, the G outputs a synthetic image. However, these images did not have the desired output resolution; therefore, they were resized to the target dimension of the NN model input size. The resizing ensured that all generated images had a consistent resolution, which is suitable for applications that require a fixed input dimension.

After resizing, the pixel values of the generated images were adjusted from their original range (between -1 and 1 due to the use of the *TanH* activation function in the G's output layer) to the standard 8-bit range. This scaling was also required to visualize and process them through common image-processing tools and libraries. Then, the array of generated images was stored as a NumPy array. This file format was convenient for storing large arrays of data and can be easily loaded for further analysis and processing. An example of these images can be seen in Figure 4.

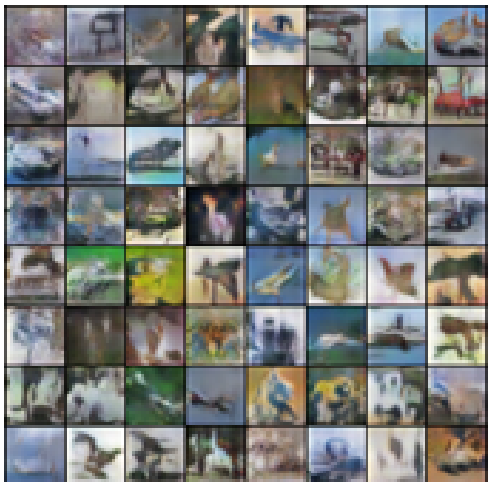

Figure 4. Examples of GAN-generated images in the original resolution of 64×64. The original low-resolution images are not easily understandable beacause as desired they shall not be the same as the training dataset.

6.2.5. Trigger Dataset Composed of Random Shapes and Colors

This method created a collection of abstract images, each with its own unique patterns and colors, by defining a coordinate space for each image and applying a series of mathematical and color functions to fill the space with visual content. The process involved creating a grid of values for the x and y coordinates within the image's boundaries, which served as the canvas for subsequent operations. The function contained a library of simple and compound functions, including random color generation, trigonometric functions, and arithmetic operations. It then recursively combined these functions in random sequences to a specified depth, creating complex mathematical expressions that determined the color and intensity of each pixel. This recursive process ensures that each image has a unique abstract design, as the combination of functions and their order can vary widely. Once an image was generated, it was scaled up to the desired resolution. The pixel values in the standard 8-bit format represented the 256 levels of red, green, and blue. After converting to this standard format, the image was stored as an array within a larger collection that holds all generated images. Finally, the complete set of images was saved to a specified directory in a file format suitable for image data. An example of these images can be seen in Figure 5.

6.2.6. Trigger Sets Composed of Pre-Selected Images

A unique and robust trigger set is essential for embedding and verifying ownership in black-box methods. One approach involved using a pre-selected set of random, uncorrelated images. This method ensured that the trigger set was distinct and not easily replicable, thereby enhancing the security and reliability of the watermarking process. The software implementation provided by [36] demonstrated the process of loading, resizing, and preparing a set of pre-selected images for use as a trigger set, as shown in Figure 6.

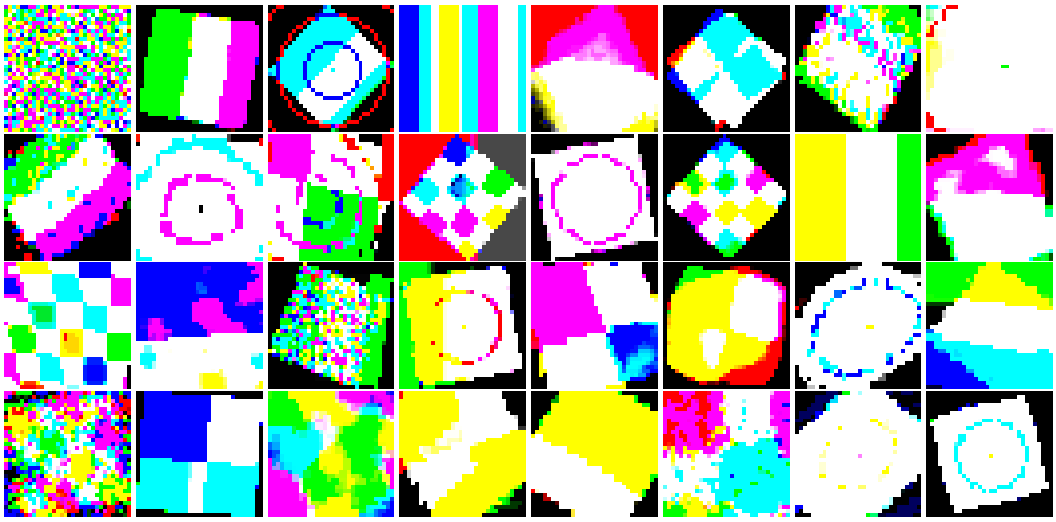

Figure 5. Example of an abstract image generated using the pillow library.

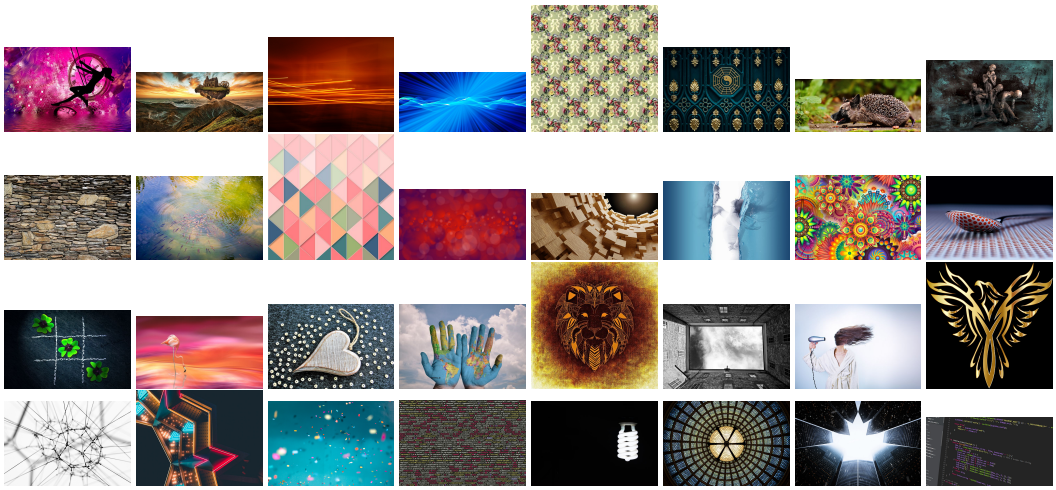

Figure 6. An example of the pre-selected images that are resized.

6.2.7. Watermark Embedding by Means of the Trigger Sets

The watermark is injected into the NN models during their training process. Watermarking is achieved by training it on a combined dataset that includes both the original training and trigger datasets. The selected trigger datasets and their corresponding random labels serve as the watermark for the model. Both datasets are then combined to form an extended training set. The combination is done by vertically stacking the trigger data on top of the original data. Once the combined dataset is prepared, the method proceeds to train the NN model. The resulting model is able to perform its intended classification task while also being able to recognize the watermark using the trigger dataset. Thus, the trained model is returned, embedding the watermark ready for deployment or any further evaluation.

Figure 7 illustrates the preferred method to embed a watermark into a neural network model.

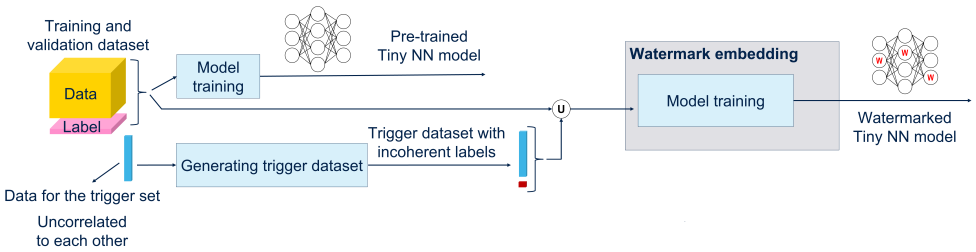

Figure 7. Black-box NN watermarking approach workflow.

6.3. Training Method

Training was performed from scratch. The process differs between the two models presented in Section 5. A learning rate scheduler was used to adjust the learning rate dynamically during training, which enhances model convergence. The model considered for the Section 5.1 use case underwent three separate training iterations, each with a different learning rate value set to 0.001, 0.0005, and 0.00025, respectively, and then was given as an argument to an optimizer that implements the Adam algorithm [51]. Regarding the Section 5.2 use case, an exponential decay scheduler was employed, where the learning rate started at an initial value of 0.001 and was decayed for each epoch by a power factor of 0.99 raised to the current epoch number. This scheduling technique helped in progressively reducing the learning rate, allowing the model to produce finer updates to the weights as it converged towards an optimal solution. The formula used to update the learning rate at each epoch is shown in Equation (7).

$$LR = initial\_LR \cdot (0.99)^{epoch}, \tag{7}$$

where LR stands for learning rate, and $epoch$ corresponds to the current number of epochs for which the model has been trained. The model's parameters (weights and biases) were initialized randomly. This ensured that the model started without any prior knowledge. VWWCOCO (refer to Section 5.1) and CIFAR10 (refer to Section 5.2) were used.

7. Results

In this section, the experimental results produced, based on the use cases described in Sections 5.1 and 5.2, will be discussed. The results were achieved using two separate datasets and reference networks and using the watermarking approaches presented in Sections 6.1 and 6.2. They will be described in Sections 7.1 and 7.2.

7.1. White-Box Watermarking Results and Using a Custom Regularizer

This application has been used for both the cases described in Section 5. In the following Sections 7.1.1 and 7.1.2, the results will be shown by delving into the epochs that have been used for training the two models introduced in Sections 5.1 and 5.2. The scale factor used in Equation (2), the training and the test accuracy achieved, the length that has been set for vector b described in Section 6.1.1, the number of bits that have been found to be different from b's bits, and finally, the BER expressed as well as training and test accuracy will be reported. By systematically varying the number of epochs and the scale factor, the study aimed to analyze the impact of these parameters on the performance of the watermarking procedure, more precisely, to measure quantitatively the performance of the models in terms of training and test accuracy and to observe the applicability of watermarking.

7.1.1. Visual Wake Words Results

For the case in Section 5.1, a derived VWW dataset and the MobilenetV1 NN were used. The watermark has been applied with the three types of matrices explained in Section 6.1.2 ("direct", "random", and "diff").

In Table 3, the WMd model trained and evaluated is described for the case in Section 5.1 with the form of the watermark matrix in Section 6.1.2 being set as direct.

Table 3. Results achieved using the Section 5.1 case set direct type of Section 6.1.2.

Total Numbers of Epochs	Scale	Training Accuracy [%]	Test Accuracy [%]	*b*-Length	Bits Extracted $\neq b$	BER [%]
3	0.01	67.95	67.65	256	2	0.78
3	0.02	68.51	69.72	256	2	0.78
3	0.05	66,54	67.24	256	0	0.00
6	0.01	73.40	73,79	256	0	0.00
15	0.01	77.13	76.15	256	0	0.00
30	0.01	89.29	72.55	256	0	0.00
90	0.01	84.63	75.23	256	0	0.00
100	**0.01**	**94.78**	**71.66**	**256**	**0**	**0.00**

In Table 4, the WMd model trained and evaluated is described for case Section 5.1, with the form of the watermark matrix set as random in Section 6.1.2.

Table 4. Results achieved using the Section 5.1 case with the random type of Section 6.1.2.

Total Numbers of Epochs	Scale	Training Accuracy [%]	Test Accuracy [%]	*b*-Length	Bits Extracted $\neq b$	BER [%]
3	0.01	67.76	68.42	256	78	30.47
3	0.02	66.25	67.23	256	76	29.69
3	0.05	67.16	67.85	256	64	25.00
6	0.01	71.34	70.62	256	63	24.61
15	0.01	78.42	78.29	256	68	26.56
30	0.01	88.99	74.11	256	61	23.82
90	**0.01**	**85.42**	**83.47**	**256**	**57**	**22.27**
100	0.01	94.52	75.76	256	59	23.82

In Table 5, the WMd model trained and evaluated is described for the Section 5.1 case with the form of the watermark matrix in Section 6.1.2 set as diff.

As reported in Tables 3–5, the best results, according to the BER, were obtained using a watermark matrix initialized with the direct form. It is evident from Tables 3 and 4 that by increasing the number of epochs, the results improve considering the training accuracy, the test accuracy, and the BER metrics. Considering the training accuracy, the test accuracy, and the BER metrics, the best results were achieved using a scale factor equal to 0.01, balancing the results for the original task and the watermark detectability. Although in certain combinations, the results seem to improve by increasing the scale factor k, we preferred to consider the best set of values found for all the use cases. For this reason, the results presented in Tables 3–5, referring to a number of epochs greater than 1, are extracted only taking into account the value for the scale factor equal to 0.1.

7.1.2. Image Classification Results

For the Section 5.2 use case, employing the CIFAR-10 dataset and the ResNet-8 model, the watermark has been applied with all three types of matrices explained in Section 6.1.2.

In Table 6, the WMd model trained and evaluated has been described for the Section 5.2 case with the form of the watermark matrix in Section 6.1.2 set as direct.

Table 5. Results achieved using the Section 5.1 case with the diff type of Section 6.1.2.

Total Numbers of Epochs	Scale	Training Accuracy [%]	Test Accuracy [%]	*b*-Length	Bits Extracted $\neq b$	BER [%]
3	0.01	67.58	66.84	256	78	34.77
3	0.02	66.16	66.34	256	84	32.81
3	0.05	66.42	66.87	256	92	35.94
6	0.01	71.72	71.47	256	83	24.22
15	0.01	75.80	70.33	256	78	30.47
30	0.01	87.48	66.62	256	76	29.68
90	0.01	83.53	78.70	256	82	32.03
100	**0.01**	**94.46**	**74.37**	**256**	**78**	**30.46**

Table 6. Results using Section 6.1 case with the direct type of Section 5.2 case.

Total Numbers of Epochs	Scale	Training Accuracy [%]	Test Accuracy [%]	*b*-Length	Bits Extracted $\neq b$	BER [%]
1	0.01	48.77	44.41	256	2	0.78
1	0.02	48.04	50.32	256	0	0.00
1	0.05	48.81	41.56	256	0	0.00
2	0.01	60.28	57.41	256	1	0.39
5	0.01	72.07	64.82	256	0	0.00
10	0.01	79.83	71.93	256	0	0.00
30	0.01	75.37	65.75	256	0	0.00
100	**0.01**	**83.39**	**75.04**	**256**	**0**	**0.00**

In Table 7, the trained and evaluated WMd model has been described for the Section 5.2 case with the form of the watermark matrix in Section 6.1.2 set as random.

Table 7. Results using Section 6.1 case with the random type of Section 5.2 case.

Total Numbers of Epochs	Scale	Training Accuracy [%]	Test Accuracy [%]	*b*-Length	Bits Extracted $\neq b$	BER [%]
1	0.01	46.39	51.54	256	79	30.86
1	0.02	47.13	54.22	256	83	32.42
1	0.05	47.77	50.13	256	78	30.47
2	0.01	60.75	55.57	256	72	28.52
5	0.01	71.25	65.58	256	69	26.95
10	0.01	78.65	65.52	256	70	27.34
30	0.01	76.31	67.33	256	59	23.04
100	**0.01**	**83.78**	**71.02**	**256**	**64**	**25.39**

In Table 8, the WMd model trained and evaluated has been described for the Section 5.2 case with the form of the watermark matrix in Section 6.1.2 set as diff.

Table 8. Results using Section 6.1 case with the diff type of Section 5.2 case.

Total Numbers of Epochs	Scale	Training Accuracy [%]	Test Accuracy [%]	*b*-Length	Bits Extracted $\neq b$	BER [%]
1	0.01	47.25	54.52	256	86	33.59
1	0.02	45.45	44.73	256	86	33.59
1	0.05	45.77	45.62	256	88	34.38
2	0.01	61.41	55.06	256	95	37.11
5	0.01	71.46	69.33	256	73	28.52
10	0.01	78.00	70.38	256	85	33.20
30	0.01	76.34	68.06	256	74	28.90
100	**0.01**	**83.78**	**71.02**	**256**	**64**	**25.39**

Tables 6–8 suggest that the best BER results have been achieved with the watermark matrix initialized with the direct form. Also, in this case, by increasing the number of epochs, an improvement in the training accuracy, test accuracy, and BER metric was observed. Training and evaluation were conducted for 100 epochs using only the scale factor k equal to 0.01, with the best value aiming to maintain a good balance between the original task and the watermark detectability.

7.2. Results with the Black-Box Method by Means of Trigger Datasets

This application has been used for both the cases described in Sections 5.1 and 5.2. In the following Sections 7.2.1 and 7.2.2, the results will be shown, delving into trigger sets used to embed the watermark on the two models introduced in Sections 5.1 and 5.2.

The tables in this section report the accuracy measurements for various image classification models, specifically focusing on their performance with different types of watermark embedding. The resulting models are as follows:

- MobilenetV1 is the original NN without the watermark;
- GANWM is the watermarked model embedded using GAN-generated images, as described in Section 6.2.4;
- PILWM is the watermarked model embedded using abstract images generated by the pillow library, as described in Section 6.2.5;
- ADIWM is the watermarked model embedded using pre-selected images, as described in Section 6.2.6.

Each model was evaluated with the original dataset, and the trigger dataset was used to inject the watermark.

For this NNW method, the models in which the watermark can be successfully detected are the ones with accuracy greater than 88%, considering the 12% threshold established in [36]. This threshold is critical for determining the effectiveness of the watermarking technique. Ref. [36] suggested that a watermark can be detected in a model with an accuracy above this threshold and that such a model can be considered robust and reliable.

7.2.1. Visual Wake Words

Following the application of watermark embedding, the results of the tests conducted with four different trigger datasets for the VWW use case Section 5.1 are reported.

Table 9 shows accuracy measurements for two types of models: WMd and unWMd. The models were fed on the test partition from the original dataset as well as on the trigger dataset to assess whether the original task is not deviated and that the trigger dataset is correctly recognized. The results obtained by inserting the watermark through image generation with the Pillow library show a minimal decrease in accuracy. A larger gap exists considering the insertion with the trigger dataset GAN. The selected trigger dataset is

correctly recognized, and the accuracy is maximal. For completeness, the results obtained with the UnWMd model and the one watermarked with pre-selected images, as described in Section 6.2.6, have also been added.

Table 9. Accuracy measurements for VWW watermark embeddings.

	Original Dataset	Trigger Dataset
UnWMd MobilenetV1	78.25	50.70
GANWM	50.01	100
PILWM	72.73	100
ADIWM	85.00	97.00

7.2.2. Image Classification

After applying embedding, the results of the tests conducted for the IC use case are reported below. Table 10 shows the accuracy measurements for various image classification models, specifically focusing on their performance with different types of watermark embedding. The same naming conventions introduced in Section 7.2.1 for the models are assumed.

Table 10. Accuracy measurements for IC watermark embeddings.

	CIFAR10	Trigger Dataset
UnWMd ResNet8	80.66	38.09
GANWM	83.00	100
PILWM	83.44	100
ADIWM	85.00	97.00

Table 10 shows accuracy measurements for two types of models: WMd and unWMd. The models were fed on the test partition from the original dataset as well as on the trigger dataset to assess whether the original task is not deviated and that the trigger dataset is correctly recognized. The results obtained by inserting the watermark through image generation with GAN or with the Pillow library show almost the same accuracy. The selected trigger dataset is correctly recognized, and the accuracy is maximal. For completeness, the results obtained with the UnWMd model and the one watermarked with pre-selected images, as described in Section 6.2.6, have also been added.

8. Deployability

This section evaluates the performance and capabilities using selected off-the-shelf MCUs. The following key parameters and required resources were reported:

- Inference time;
- Memory footprint (RAM and ROM).

Table 11 contains the main technical specifications of an MCU versus those in a typical x86 microprocessor. These values underline the different levels of asset availability to support the NNW application. From Table 11, MCUs have significantly limited resources compared to the x86 CPU. However, MCUs' design allows them to perform specific tasks efficiently within their constrained assets, as shown in Section 8.1.

Table 11. Comparison of available resources provided by an MCU and the x86 CPU.

Available Resources	STM32H743ZI MCU	x86 Host
RAM	1 MB	16 GB
FLASH	2 MB	1 TB
CPU frequency	480 MHz	i5 CPU, 1.60 GHz
Core number	1	8

8.1. Deployability Results on the MCU

To deploy, program, run, and validate the model's performance on the MCUs, several AI tools were used by this work:

- STM32CubeIDE (https://www.st.com/en/development-tools/stm32cubeide.html (accessed on 13 November 2024)), an integrated development environment (IDE) that provides comprehensive facilities for MCU programming and debugging;
- STM32CubeMX (https://www.st.com/en/development-tools/stm32cubemx.html (accessed on 13 November 2024)), a graphical tool that helps in configuring MCU peripherals and generating initialization code;
- STM32CubeProgrammer (https://www.st.com/en/development-tools/stm32cubeprog.html (accessed on 13 November 2024)), a tool used for programming the MCU with the binary generated by the IDE;
- ST Edge AI Core Technology (https://www.st.com/en/development-tools/stedgeai-core.html (accessed on 13 November 2024)), a command-line interface (CLI) tool to optimize and compile edge AI models for multiple ST devices, including MCU, MPU, and MEMS sensors.

Table 12 shows the results achieved in terms of energy consumption, memory footprint, and inference time to run the models (defined in Sections 5.1 and 5.2) deployed in the MCU. FP32 means that the model's weights were represented in 32-bit floating-point precision.

Table 12. Results achieved on the MCU memory footprint and inference time, measured with ST Edge AI Core Technology, for the models explained in Sections 5.1 and 5.2, respectively.

Board Name	MobileNetV1 (FP32)			ResNet-V1 (FP32)		
	Latency [ms]	RAM	FLASH [KiB]	Latency [ms]	RAM	FLASH [KiB]
NUCLEO-H743ZI2 @480 MHz	110.6	167	855	122.5	140	323
STM32L4R9I-DISCO @120 MHz	835.9	167	855	1.033	140	323
B-U585I-IOT02A @160 MHz	454.8	167	855	528.6	140	323

The watermarked models obtained according to methods explained in Sections 6.1 and 6.2 required no additional energy, execution time, and memory resources. This was expected since the methods impact how the models are trained and, therefore, the weight's values and not the model's topology and footprint. This finding is significant from an embedded feasibility perspective since it underlines the efficiency and practicality of deploying WMd models in real-world applications without incurring extra costs. This efficiency bolsters the case for utilizing WMd models in various embedded systems, further enhancing the utility of MCUs in practical scenarios.

9. Attacks and Results

This section reports the experimental analysis of the resilience of the watermarked models through a series of simulated attacks. The goal is to evaluate how well these models can withstand adversarial conditions without prior knowledge of the model's inner workings. As outlined in Sections 6.1 and 6.2, both white-box and black-box watermark-

ing techniques were implemented, each with a distinct embedding or detection strategy. The upcoming sections will focus on these models, subjecting them to a variety of simulated attack scenarios, including Gaussian noise injection, pruning, and quantization. These simulations are designed to test the robustness and security of the watermarks, providing insight into the potential vulnerabilities of these models under adversarial stress.

The MPAI (https://mpai.community/ accessed on 13 November 2024) (Moving Picture, Audio and Data Coding by Artificial Intelligence) community established the IEEE 3304-2023 standard [52] for NNW to ensure the robustness and reliability of AI models. According to this standard, NN models may be subjected to various types of attacks to evaluate their resilience and performance. This is crucial for maintaining the integrity and reliability of AI systems in real-world applications.

The models were tested under three different types of attacks:

- Gaussian involves adding random noise to the data, which can potentially disrupt the model's ability to output accurate predictions. Gaussian noise is a common form of statistical noise that, when it affects the input data, leads to degraded performance. For the attacks, the values 0.001, 0.01, 0.1, 1, and 10 represented the standard deviation of the added noise, with higher numbers indicating more noise that can disrupt the model's performance more and potentially its watermark;
- Pruning selectively removes connections between neurons, effectively reducing the complexity of the NN and potentially degrading its performance. Pruning is often used to simplify a model's complexity, but in this context, it is used to test the model's robustness. In these attacks, the numbers 0.1 to 0.5 indicated the fraction of the model's weights that were removed, with higher values representing more aggressive pruning that can degrade both the model's accuracy and the integrity of the watermark;
- Quantization involves reducing the precision of numerical values, which can lead to a loss of information and accuracy. Quantization typically reduces the bit depth of the weights of the NN, and according to the MPAI documentation, this is applied only to the weights as fake quantization. The reduction in bit depth can significantly impact the model's performance. For quantization attacks, 16-bits to 2-bits denoted the bit depth to which the model's weights were reduced, with lower bit depths reflecting more extreme quantization, which can result in a significant loss in precision and potentially damage the watermark.

In Sections 9.1 and 9.2, the results achieved under simulated attacks are presented. The values under the type of attacks in the first left column provided in Tables 13–16 refer to the specific parameters of the attacks, proportional to the extent of degradation. The models have been trained using the Keras AI framework v2.12.0. Since the attacks standardized by MPAI are implemented based on PyTorch v2.2.0, a lossless conversion has been developed. Subsequently, the PyTorch attached models were exported in the ONNX exchange format so that tests could be conducted not only with the original framework on a PC but also on the NUCLEO-H743ZI2 MCU board with STEdgeAI. Both the results for the original dataset and the robustness of the previously inserted watermark have been included for both NNW techniques.

The mentioned watermark is resistant to pruning because of the distribution and redundancy of the watermark throughout the whole model, which ensures that not all watermark information is lost, even if portions of the network are removed somehow. Similar to its resistance to pruning, the watermark is also resilient to quantization through careful design, such as using quantization-aware training to maintain its accuracy even when the precision of weights changes. By embedding watermarks with these considerations, the detectability of the watermark is maintained despite the network modifications.

9.1. Results for the White-Box Watermarking Method

In this subsection, the resistance to attacks explained above will be presented for WMd and unWMd models with datasets explained in Sections 5.1 and 5.2 considering the watermarking algorithm explained in Section 6.1 with the form of the watermark matrix in

Section 6.1.2 set as direct. In addition to the results from the inferences with the original dataset as input, the BER extracted value for each model is reported in Tables 13 and 14. To ensure the correct identification of watermarks, a BER threshold of 0% has been adopted. In other words, a watermark was considered to be correctly recognized only if each single extracted bit matched exactly the original inserted bit. Results that met this criterion are highlighted in green.

Table 13. Watermark robustness evaluation on UnWMd and WMd attacked models with white-box method, for the Section 5.1 use case, using ST Edge AI and the original framework. The results for which the origin of the watermark can be recognized are highlighted in green.

Watermark Detected	ST Edge AI		Original Framework	
	VWW Dataset [%]	BER [%]	VWW Dataset [%]	BER [%]
Original models				
UnWMd MobilenetV1	78.25	43.75	78.25	43.75
WMd MobilenetV1	71.66	0.00	71.66	0.00
Gaussian attacks				
0.001	60.93	0.00	60.93	0.00
0.01	60.97	0.00	60.97	0.00
0.1	58.68	0.00	58.68	0.00
1	50.00	0.00	50.00	0.00
10	50.00	8.98	50.0	8.98
Prune attacks				
0.1	59.96	0.00	59.96	0.00
0.2	58.11	0.00	58.11	0.00
0.3	56.84	0.00	56.84	0.00
0.4	53.21	0.00	53.21	0.00
0.5	50.68	0.00	50.68	0.00
Quantization attacks				
16 bit	61.01	0.00	61.01	0.00
8 bit	60.90	0.00	60.90	0.00
7 bit	59.94	0.00	59.94	0.00
6 bit	59.79	0.00	59.79	0.00
5 bit	57.09	0.00	57.09	0.00

The results of the experiments discussed in Section 7.1.1 and reported in Table 13 reveal varying degrees of effectiveness for each attack on the watermarked model that uses the dataset explained in Section 5.1 that uses a MobilenetV1 model with the Sections 6.1 and 6.1.2 approaches in direct mode. It has to be noted that when the model is subject to the three types of attacks (Gaussian, prune, and quantization, its performance in terms of accuracy shows a decreasing trend as the attack power increases, in fact, from the lowest values of attack an accuracy that goes from 71.66% to 60.93% could be seen. Despite the decrease in accuracy values, the model remains robust in terms of the watermark detected; in fact, it has been possible to notice that the BER value remained steady at 0% in all attack cases except for the Gaussian attack with an attack value of 10, in which the BER is 8.98%.

The results of experiments discussed in Section 7.1.2 and reported in Table 14 revealed varying degrees of effectiveness for each attack on the watermarked model that uses the dataset explained in Section 5.2 that uses a customized ResNet-8 with the Sections 6.1 and 6.1.2 approaches in direct mode. In contrast to the results presented in Table 13, the model showed greater resistance to attacks of lower intensity. In fact, an accuracy above 70% can be seen in the first three levels of attack for each case (Gaussian, prune, and quantization). However, when exposed to more powerful attacks, the model's performance dropped significantly. Regarding the detection of the watermark, behavior similar to that described in Table 13 was observed, with the exception of the Gaussian attack with an attack value of 10, in which a BER of 0.26% was obtained.

Table 14. Watermark robustness evaluation on UnWMd and WMd attacked models with white-box method, for the Section 5.2 use case, using ST Edge AI and the original framework. The results for which the origin of the watermark can be recognized are highlighted in green.

Watermark Detected	ST Edge AI		Original Framework	
	Cifar10 [%]	BER [%]	Cifar10 [%]	BER [%]
Original models				
UnWMd ResNet-8	80.67	61.71	80.67	61.71
WMd ResNet-8	75.04	0.00	75.04	0.00
Gaussian attacks				
0.001	75.00	0.00	75.00	0.00
0.01	74.63	0.00	74.63	0.00
0.1	73.59	0.00	73.59	0.00
1	10.74	0.00	10.74	0.00
10	10.03	26	10.03	26
Prune attacks				
0.1	75.08	0.00	75.08	0.00
0.2	74.69	0.00	74.69	0.00
0.3	73.96	0.00	73.96	0.00
0.4	70.21	0.00	70.21	0.00
0.5	61.25	0.00	61.25	0.00
Quantization attacks				
16 bit	75.04	0.00	75.04	0.00
8 bit	74.73	0.00	74.73	0.00
7 bit	71.06	0.00	71.06	0.00
6 bit	56.41	0.00	56.41	0.00
5 bit	57.61	0.00	57.61	0.00

9.2. Results for the Black-Box Watermarking Method

In this subsection, the resistance to attacks explained above will be presented for WMd and unWMd models with datasets explained in Sections 5.1 and 5.2 considering the watermarking algorithm explained in Section 6.2 embedded using abstract images generated by the Pillow library as described in Section 6.2.5. In addition to the results from the inferences with the original dataset as input, the accuracy value obtained using the selected trigger value for each model has been reported in Tables 13 and 14. To ensure the correct identification of watermarks, the accuracy of the trigger dataset should exceed 88%, with a non-recognition threshold of 12%. Results that met this criterion were highlighted in green.

The results of experiments discussed in Section 7.2.1 and reported in Table 15 revealed varying degrees of effectiveness for each attack on the watermarked model that uses the dataset explained in Section 5.1 that uses a MobilenetV1 model with the Sections 6.2 and 6.2.2 approaches composed of random shapes and colors. In the case presented, a similar behavior of the model subjected to the three types of attacks (Gaussian, prune and quantization), obtained in the two previous tables, in terms of accuracy, was also found for Table 15. The attack does not seem to have been decisive from the beginning, but it was possible to notice that as the power increased, the accuracy decreased, reaching a minimum of 50%. The trend of the data is also very consistent with the level of attack applied to the model. According to the results obtained in Table 15, the WMd model appeared to be more robust with the application of quantization attack, obtaining a percentage higher than 88% up to the set 6-bit attack value, while it seemed to be more susceptible to the pruned attack in which, already from the second level of attack, the trigger dataset reached a percentage of 77.46%.

Table 15. Watermark robustness evaluation on UnWMd and WMd attacked models with black-box method, for the Section 5.1 use case, using ST Edge AI and the original framework. The results for which the origin of the watermark can be recognized are highlighted in green.

Watermark Detected	ST Edge AI		Original Framework	
	VWW Dataset [%]	Trigger Dataset [%]	VWW Dataset [%]	Trigger Dataset [%]
Original models				
UnWMd MobilenetV1	78.25	50.70	78.25	50.70
WMd MobilenetV1	72.73	100	72.73	100
Gaussian attacks				
0.001	72.86	95.07	72.86	95.07
0.01	72.36	93.66	72.36	93.66
0.1	67.92	73.94	67.92	73.94
1	49.77	57.74	49.77	57.74
10	50.00	52.81	50.00	52.81
Prune attacks				
0.1	72.50	94.36	72.50	94.36
0.2	71.06	77.46	71.06	77.46
0.3	70.06	76.05	70.06	76.05
0.4	64.86	63.38	64.86	63.38
0.5	59.38	55.63	59.38	55.63
Quantization attacks				
16bit	72.87	95.07	72.87	95.07
8 bit	72.56	96.47	72.56	96.47
7 bit	71.93	97.18	71.93	97.18
6 bit	72.08	88.02	72.08	88.02
5 bit	67.63	71.83	67.63	71.83
4 bit	51.73	48.59	51.73	48.59
2 bit	50.00	47.18	50.00	47.18

Table 16. Watermark robustness evaluation on UnWMd and WMd attacked models with black-box method, for the Section 5.2 use case, using ST Edge AI and the original framework. The results for which the origin of the watermark can be recognized are highlighted in green.

Watermark Detected	ST Edge AI		Original Framework	
	Cifar10 [%]	Trigger Dataset [%]	Cifar10 [%]	Trigger Dataset [%]
Original models				
UnWMd ResNet-8	80.67	38.09	80.67	38.09
WMd ResNet-8	83.44	100	83.44	100
Gaussian attacks				
0.001	80.42	100	80.42	100
0.01	80.4	98.18	80.4	98.18
0.1	59.69	62.72	59.69	62.72
1	10.0	12.73	10.0	12.73
10	10.01	12.73	10.01	12.73
Prune attacks				
0.1	80.26	97.27	80.26	97.27
0.2	77.54	85.45	77.54	85.45
0.3	76.01	73.63	76.01	73.63
0.4	70.34	61.82	70.34	61.82
0.5	48.01	34.54	48.01	34.54
Quantization attacks				
16 bit	80.44	100	80.44	100
8 bit	80.38	100	80.38	100
7 bit	80.41	96.36	80.41	96.36
6 bit	75.89	79.09	75.89	79.09
5 bit	75.23	79.09	75.23	79.09
4 bit	48.79	40.00	48.79	40.00
3 bit	8.46	14.54	8.46	14.54
2 bit	10.0	19.09	10.0	19.09

The results of experiments discussed in Section 7.2.2 reported in Table 16 revealed varying degrees of effectiveness for each attack on the watermarked model that uses the dataset explained in Section 5.2 that uses a MobilenetV1 model with Section 6.2 approach and a Section 6.2.2 composed of random shapes and colors. The performances of the model subjected to the three types of attacks (Gaussian, prune, and quantization), in terms of accuracy, were also consistent with the level of attack. However, a strong drop was recorded for the last attack levels, especially in the case of the Gaussian attack and wuantization attack, where an accuracy value of 10% was reached. The trend of the Trigger dataset was very similar to Table 15; in fact, the WMd model remained robust in the first levels of attack, starting from a percentage of 100% or 97.27%. The further increase in the power of attack led to a sharp drop in the percentage, reaching values below 20%.

10. Conclusions

In this work, two NNW methods applied to two use cases have been studied. The tests conducted on the WMd and non-WMd models revealed important insights about the quality of the injection using a regularizer and a selected trigger dataset as part of the training dataset. Multiple parameters for both techniques have been studied, and the results have been analyzed. The best models for each method and each use case have been maliciously manipulated through the application of adversarial attacks such as Gaussian noise addition, pruning, and parameter quantization. This evaluation was necessary to test the robustness of the watermark inserted in the model under deployment by the owner.

These models have also been analyzed and implemented on MCUs using the ST Edge AI Unified Core technology. The WMd models maintained their efficiency in terms of computational resources unchanged and also demonstrated superior resilience against various types of attacks. This ensured the security and reliability of ML applications without affecting the models' capabilities. Following this procedure, the IP owner had an additional tool to be used as proof of ownership of their intellectual property, for example, in the case of unauthorized use and fraudulent copying.

The results of the study indicate that different methods for generating trigger sets offer varied levels of robustness and adaptability in NNW. Unique image generation through GANs proved highly effective in creating resilient watermarks that withstand tampering, although random image generation using PIL provided a better combination of original task and trigger set accuracy despite it being a simpler approach.

Considering the accuracy results achieved by combining the trained models and the original and generated input data, the results confirmed the robustness of these techniques even for tiny models deployed on microcontrollers.

Future works will be focused on expanding the use cases in the field of audio and GANs. To further reduce the implementation costs on tiny devices, techniques such as post-training quantization and quantization-aware training will be the subject of more studies.

Author Contributions: Conceptualization, A.C. and D.P.P.; methodology, A.C.; software, A.C., D.E.D. and B.D.G.; validation, A.C. and B.D.G.; formal analysis, A.C., D.E.D., B.D.G., L.F., C.T. and D.P.P.; investigation A.C.; resources, L.F. and D.P.P.; data curation, A.C., D.E.D. and B.D.G.; writing—original draft preparation, A.C., D.E.D., B.D.G., L.F., C.T. and D.P.P.; writing—review and editing, A.C., D.E.D., B.D.G., L.F., C.T. and D.P.P.; visualization, A.C., D.E.D. and B.D.G.; supervision, A.C. and D.P.P.; project administration A.C. and D.P.P.; funding acquisition, D.P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: This study did not require ethical approval. It did not involve humans or animals.

Informed Consent Statement: Not applicable since this study did not involve humans.

Data Availability Statement: The data presented in this study are available in VWW dataset at https://arxiv.org/abs/1906.05721 (accessed on 19 November 2024), https://github.com/mlcommons/tiny/tree/master/benchmark#2 (accessed on 19 November 2024) and CIFAR-10 at https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 19 November 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
BER	Bit Error Rate
CNN	Convolutional Neural Networks
D	Dicriminator
DCT	Discrete Cosine Transform
DNN	Deep Neural Networks
DW	Digital Watermark
DWT	Discrete Wavelet Transform
FCT	Fast Curvelet Transform
FP32	Floating Point 32 bits
FRT	Finite Ridgelet Transform
G	Generator
GAN	Generative Adversarial Networks
GenAI	Generative Artificial intelligence
GNN	Graph Neural Network
IC	Image Classification
IDE	Integrated Development Environment
IP	Intellectual Property
KD	Knowledge distillation
LSB	Least Significant Bit
MCU	Micro-controller
ML	Machine Learning
NN	Neural Networks
NNW	Neural Network Watermarking
PSO	Particle Swarm Optimization
PIL	Python Imaging Library
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SME	Small Medium Enterprises
SVD	Singular value Decomposition
TFL	tensorflow lite
TinyML	Tiny Machine Learning
Tiny-NN	Tiny Neural Networks
unWMd	unwatermarked model
VWW	Visual Wake Words
WMd	Watermarked model

References

1. Hinton, G.E.; Salakhutdinov, R.R. Reducing the Dimensionality of Data with Neural Networks. *Science* **2006**, *313*, 504–507. [CrossRef] [PubMed]
2. Liu, J.; Kong, X.; Xia, F.; Bai, X.; Wang, L.; Qing, Q.; Lee, I. Artificial Intelligence in the 21st Century. *IEEE Access* **2018**, *6*, 34403–34421. [CrossRef]
3. Alzubi, J.; Nayyar, A.; Kumar, A. Machine Learning from Theory to Algorithms: An Overview. *J. Phys. Conf. Ser.* **2018**, *1142*, 012012. [CrossRef]
4. Lin, J.; Zhu, L.; Chen, W.M.; Wang, W.C.; Han, S. Tiny Machine Learning: Progress and Futures [Feature]. *IEEE Circuits Syst. Mag.* **2023**, *23*, 8–34. [CrossRef]
5. MLCommons. Available online: https://mlcommons.org/ (accessed on 13 November 2024).

6. Hilty, R.; Hoffmann, J.; Scheuerer, S. Intellectual Property Justification for Artificial Intelligence. Artificial Intelligence & Intellectual Property, Oxford, Oxford University Press Max Planck Institute for Innovation & Competition Research Paper Series. 2020. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3539406 (accessed on 19 November 2024).
7. Kakikura, S.; Kang, H.; Iwamura, K. Collusion Resistant Watermarking for Deep Learning Models Protection. In Proceedings of the 2022 24th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Republic of Korea, 13–16 February 2022; pp. 40–43. [CrossRef]
8. Li, Y.; Wang, H.; Barni, M. A survey of deep neural network watermarking techniques. *Neurocomputing* **2021**, *461*, 171–193. [CrossRef]
9. Chen, H.; Rouhani, B.D.; Fu, C.; Zhao, J.; Koushanfar, F. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In Proceedings of the 2019 on International Conference on Multimedia Retrieval, Ottawa ON, Canada, 10–13 June 2019; pp. 105–113. [CrossRef]
10. Rouhani, B.D.; Chen, H.; Koushanfar, F. DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models. *arXiv* **2018**, arXiv:1804.00750.
11. Zhu, R.; Zhang, X.; Shi, M.; Tang, Z. Secure neural network watermarking protocol against forging attack. *EURASIP J. Image Video Process.* **2020**, *2020*, 37. [CrossRef]
12. Xie, Y.; Wang, Y.; Ma, M. Design of a Hybrid Digital Watermarking Algorithm with High Robustness. *J. Web Eng.* **2020**, *19*, 725–746. [CrossRef]
13. Cheema, A.M.; Adnan, S.M.; Mehmood, Z. A Novel Optimized Semi-Blind Scheme for Color Image Watermarking. *IEEE Access* **2020**, *8*, 169525–169547. [CrossRef]
14. Du, M.; Luo, T.; Li, L.; Xu, H.; Song, Y. T-SVD-Based Robust Color Image Watermarking. *IEEE Access* **2019**, *7*, 168655–168668. [CrossRef]
15. Hassan, B.; Ahmed, R.; Li, B.; Hassan, O. An Imperceptible Medical Image Watermarking Framework for Automated Diagnosis of Retinal Pathologies in an eHealth Arrangement. *IEEE Access* **2019**, *7*, 69758–69775. [CrossRef]
16. Shehab, A.; Elhoseny, M.; Muhammad, K.; Sangaiah, A.K.; Yang, P.; Huang, H.; Hou, G. Secure and Robust Fragile Watermarking Scheme for Medical Images. *IEEE Access* **2018**, *6*, 10269–10278. [CrossRef]
17. Li, X.; Chen, Q.; Chu, R.; Wang, W. Block mapping and dual-matrix-based watermarking for image authentication with self-recovery capability. *PLoS ONE* **2024**, *19*, e0297632. [CrossRef] [PubMed]
18. Su, Q.; Sun, Y.; Xia, Y.; Wang, Z. A robust color image watermarking scheme in the fusion domain based on LU factorization. *Opt. Laser Technol.* **2024**, *174*, 110726. [CrossRef]
19. Dammel, J.W.; Highman, N.J.; Schreiber, R. Block LU Factorization. *J. Numer. Linear Algebra Appl.* **1992**; NASA-BR-97949. Available online: https://ntrs.nasa.gov/api/citations/19950017172/downloads/19950017172.pdf (accessed on 19 November 2024).
20. Sinhal, R.; Ansari, I.A.; Ahn, C.W. Blind Image Watermarking for Localization and Restoration of Color Images. *IEEE Access* **2020**, *8*, 200157–200169. [CrossRef]
21. Dasgupta, A.; Zhong, X. Robust Image Watermarking based on Cross-Attention and Invariant Domain Learning. *arXiv* **2023**, arXiv:2310.05395.
22. Tavakoli, A.; Honjani, Z.; Sajedi, H. Convolutional Neural Network-Based Image Watermarking using Discrete Wavelet Transform. *arXiv* **2022**, arXiv:2210.06179. [CrossRef]
23. Fei, J.; Xia, Z.; Tondi, B.; Barni, M. Wide Flat Minimum Watermarking for Robust Ownership Verification of GANs. *arXiv* **2023**, arXiv:2310.16919. [CrossRef]
24. Jamali, M.; Karim, N.; Khadivi, P.; Shirani, S.; Samavi, S. Robust Watermarking using Diffusion of Logo into Autoencoder Feature Maps. *arXiv* **2021**, arXiv:2105.11095.
25. Fei, J.; Xia, Z.; Tondi, B.; Barni, M. Supervised GAN Watermarking for Intellectual Property Protection. *arXiv* **2022**, arXiv:2209.03466.
26. Dai, E.; Lin, M.; Wang, S. PreGIP: Watermarking the Pretraining of Graph Neural Networks for Deep Intellectual Property Protection. *arXiv* **2024**, arXiv:2402.04435.
27. Li, F.; Zhao, H.; Du, W.; Wang, S. Revisiting the Information Capacity of Neural Network Watermarks: Upper Bound Estimation and Beyond. *arXiv* **2024**, arXiv:2402.12720. [CrossRef]
28. Lv, P.; Ma, H.; Chen, K.; Zhou, J.; Zhang, S.; Liang, R.; Zhu, S.; Li, P.; Zhang, Y. MEA-Defender: A Robust Watermark against Model Extraction Attack. *arXiv* **2024**, arXiv:2401.15239.
29. Boenisch, F. A Systematic Review on Model Watermarking for Neural Networks. *Front. Big Data* **2021**, *4*, 729663. [CrossRef] [PubMed]
30. Aiken, W.; Kim, H.; Woo, S.; Ryoo, J. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks. *Comput. Secur.* **2021**, *106*, 102277. [CrossRef]
31. Xu, X.; Li, Y.; Yuan, C. "Identity Bracelets" for Deep Neural Networks. *IEEE Access* **2020**, *8*, 102065–102074. [CrossRef]
32. Chen, H.; Rouhani, B.D.; Koushanfar, F. BlackMarks: Blackbox Multibit Watermarking for Deep Neural Networks. *arXiv* **2019**, arXiv:1904.00344.
33. Uchida, Y.; Nagai, Y.; Sakazawa, S.; Satoh, S. Embedding Watermarks into Deep Neural Networks. *arXiv* **2017**, arXiv:1701.04082.

34. Tartaglione, E.; Grangetto, M.; Cavagnino, D.; Botta, M. Delving in the loss landscape to embed robust watermarks into neural networks. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 1243–1250. [CrossRef]
35. Botta, M.; Cavagnino, D.; Esposito, R. NeuNAC: A novel fragile watermarking algorithm for integrity protection of neural networks. *Inf. Sci.* **2021**, *576*, 228–241. [CrossRef]
36. Adi, Y.; Baum, C.; Cissé, M.; Pinkas, B.; Keshet, J. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. *arXiv* **2018**, arXiv:1802.04633.
37. Orekondy, T.; Schiele, B.; Fritz, M. Prediction Poisoning: Utility-Constrained Defenses Against Model Stealing Attacks. *arXiv* **2019**, arXiv:1906.10908.
38. Zhong, Q.; Zhang, L.; Zhang, J.; Gao, L.; Xiang, Y. Protecting IP of Deep Neural Networks with Watermarking: A New Label Helps. In Proceedings of the Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, 11–14 May 2020; pp. 462–474. [CrossRef]
39. Merrer, E.L.; Pérez, P.; Trédan, G. Adversarial Frontier Stitching for Remote Neural Network Watermarking. *arXiv* **2017**, arXiv:1711.01894. [CrossRef]
40. Szyller, S.; Atli, B.G.; Marchal, S.; Asokan, N. DAWN: Dynamic Adversarial Watermarking of Neural Networks. *arXiv* **2019**, arXiv:1906.00830.
41. Wu, H.; Liu, G.; Yao, Y.; Zhang, X. Watermarking Neural Networks With Watermarked Images. *IEEE Trans. Circuits Syst. Video Technol.* **2021**, *31*, 2591–2601. [CrossRef]
42. Banbury, C.R.; Reddi, V.J.; Torelli, P.; Jeffries, N.; Király, C.; Holleman, J.; Montino, P.; Kanter, D.; Warden, P.; Pau, D.; et al. MLPerf Tiny Benchmark. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*; Vanschoren, J., Yeung, S., Eds.; 2021; Volume 1. Available online: https://arxiv.org/abs/2106.07597 (accessed on 19 November 2024).
43. Chowdhery, A.; Warden, P.; Shlens, J.; Howard, A.G.; Rhodes, R. Visual Wake Words Dataset. *arXiv* **2019**, arXiv:1906.05721.
44. Lin, T.Y.; Maire, M.; Belongie, S.J.; Bourdev, L.D.; Girshick, R.B.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. *arXiv* **2014**, arXiv:1405.0312.
45. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
46. Krizhevsky, A.; Nair, V.; Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research). Available online: https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 13 November 2024).
47. Torralba, A.; Fergus, R.; Freeman, W.T. 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2008**, *30*, 1958–1970. [CrossRef]
48. Fedorov, I.; Adams, R.P.; Mattina, M.; Whatmough, P.N. SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. *arXiv* **2019**, arXiv:1905.12107.
49. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NE, USA, 26 June–1 July 2016; pp. 770–778. [CrossRef]
50. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In *Proceedings of the Advances in Neural Information Processing Systems*; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., Eds.; Curran Associates, Inc.: New York, NY, USA, 2014; Volume 27.
51. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2017**, arXiv:1412.6980.
52. *IEEE 3304-2023*; IEEE Standard for Adoption of Moving Picture, Audio and Data Coding by Artificial Intelligence (MPAI) Technical Specification Neural Network Watermarking (NNW) V1. BOG/CAG—Entity Collaborative Activities Governance Board: Piscataway, NJ, USA, 2023. Available online: https://standards.ieee.org/ieee/3304/11281/ (accessed on 19 November 2024).