EC4

Methodological Guideline for ML Watermarking

Document reference number for ANR

contact@confiance-ai.fr | www.confiance.ai

Document reference: XXX

Contributors

	Name	Organisation	Role
Responsible for the deliverable	Lucas MATTIOLI	IRT SystemX	Co-lead of component
Scientific responsible	Lucas MATTIOLI	IRT SystemX	Co-lead of component
Co-authors	Najah BEN SAID Mohammed LANSARI	THALES THALES	Co-lead of component Contributor

Document control

Revision	Date	Commentary	Author
1.0	29/09/2023	Initial version of the document	Lucas MATTIOLI
2.0	01/12/2023	Final version for proofreading	Lucas MATTIOLI
00	XX	XX	XX
00	XX	XX	XX
00	XX	XX	XX

A. Introduction and abstract

A.1 General introduction to trustworthy AI challenges

Trustworthiness in AI within critical systems (systems that can directly or indirectly affect human life and moral entities) is essential for its widespread adoption (by the industry, the decision makers, the general public, etc.) and poses the following significant challenges.

- First, how to design AI models, so that, by construction, they satisfy trustworthy properties (accuracy, robustness…).
- Secondly, how to characterize these AI models, for example to understand and explain their behavior and their adequacy to the operational domain.
- Then, how to implement and embed those AI models on hardware, by making them fit for the target without losing their trustworthy properties.
- Another question is, what methods of data engineering to apply in order to, among other topics, manage important volumes of data and adapt to the evolution of the operational domain.
- At system level, what verification and certification processes to consider specifically for AI-based systems.
- Finally, a federation of all these matters is necessary to build an end-to-end methodological approach, supported by a consistent engineering environment compatible with industrial practices.

These are the challenges, among others, that the Confiance.ai program addresses.

A.1.1 Rationale for this methodological guideline:

This document aims to provide to a non-expert in ML models watermarking techniques a general overview of the needed information and best practices regarding the watermarking of a machine learning model. It also serves as a tool to use and understand the ML model watermarking component inside of the Confiance.ai ecosystem.

A.1.2 Target audience and disclaimer:

The target audience of this document consist of machine learning engineers, not necessarily knowledgeable of watermarking techniques and best practices.

A.1.3 How to use this document:

The reader of this document can either read the complete document from top to bottom to have an overview of the methodology and best practices of ML model watermarking or they can choose to read only specific paragraphs that corresponds to the specific notion they are interested in.

B. Description of the method

B.1 Introduction

Machine learning model watermarking primary purpose is to protect the intellectual property and the integrity of machine learning models. This objective has become increasingly valuable in various industries as machine learning models often require extensive resources, time, and expertise to create. The process of watermarking a machine learning model involves embedding a hidden signature within a machine learning model without significantly affecting its performance or functionality. By watermarking a model, its creators can establish ownership and prove the originality of their work, deterring potential unauthorized use or replication as the embedded watermark can reveal its origin and rightful owner, facilitating legal actions against potential infringers.

Additionally, model watermarking can enhance a model's accountability and trustworthiness. In fields like healthcare, finance, and autonomous vehicles where a model's decisions have significant real-world consequences, ensuring that models have not been tampered with or compromised is a crucial task.

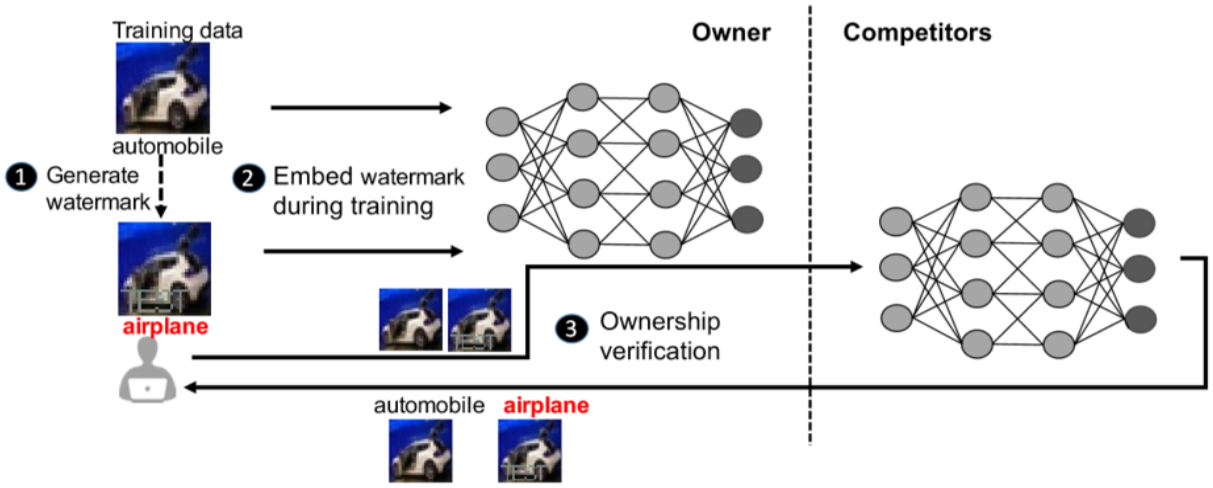

Figure 1 : Example of a watermarking generation and verification pipeline

Two categories of watermarks have been mostly studied by the scientific literature, the first one consists of directly inserting the watermark (hidden information) inside the model's parameters. The second method consisting in training the model to recognize a specific pattern outside of its training data distribution and to use this specific behavior as a watermark. This document will focus on the second method as it was our subject of study.

For embedding a hidden behavior on a ML decision model, the scientific literature list several methods that trains the model to learn an association between a watermark that is imposed on an out of distribution (OOD) dataset – called a trigger set – and corresponding outputs (kept secret to ensure ownership). Three main types of watermarks are often considered for this task:

- A specific noise added to samples of the training data (referred as “noise” watermark)
- A semantical information added to samples of the training data (referred as “content” watermark)
- Unrelated data (referred as “unrelated” watermark)

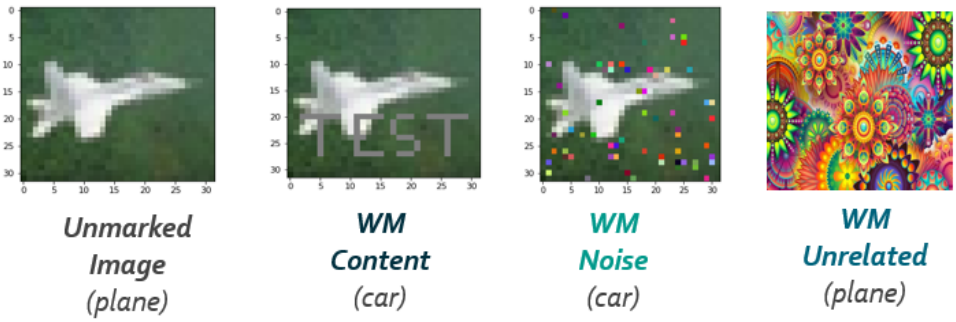

Figure 2 : Illustration of the three main watermarking techniques for image classification models

B.2 Trigger set generation

B.2.1 Number of watermarks

A limitation associated with watermarks is that they must be kept secret. Once a watermark is exposed to the public, or found by an attacker, its removal becomes very straightforward. Consequently, each watermark can only be used once.

To enhance the capacity for asserting ownership of a model through watermarks, it becomes necessary to augment the quantity of watermarks incorporated into the model.

Moreover, adding multiple types of watermarks can improve the model's robustness to specific attacks efficient against specific types of watermarks.

However, it's important to note that training a model to recognize these watermarks can entail a longer training duration. Additionally, a more complex training process may be required to prevent any decrease in the model's abilities when introducing new watermarks, especially when introducing multiple watermarks. Therefore, the user should find a balance between its desire to maintain proof of ownership for their model and the available resources during the model's training phase.

B.2.2 Trigger set size

For an **unrelated** watermarking technique to be effective, the trigger set should possess a sufficient size to achieve statistical significance.

In the context of a classification problem involving n classes, the likelihood of a model correctly classifying m unrelated images to their respective keys is $\frac{1}{n^m}$.

Unfortunately, using a very large trigger-set will slow down the training process of the model. Moreover, for **content** and **noise** watermarking techniques, a large trigger-set may cause the to overpredict the watermark class in its nominal predictions as a result of an unbalanced training set (because of the presence of the trigger-set being a single class), thus complexifying the training process.

Currently, choosing the size of the trigger-sets is still an empirical process. For example, Zheng Li & al used 1% of total training samples for the trigger set generation [1]. In our experiment, dealing with ~1000 data samples for our training set, we used 1% and 10% of the training dataset size for watermarking. We also did some experiments with 20% and 30% that lead to worst performances on some instances of noise watermarks.

B.2.3 Multi-class and binary classification models:

Every **noise** or **content** watermark needs to be associated with a distinct class that remains consistent for every sample within the trigger-set containing that particular watermark.

Therefore, in the context of a binary classification problem and when employing these two watermarking techniques, it is imperative to select all the images where the watermark will be applied from a single class to construct the trigger-set.

For a multi-class problem, the images that will be used to construct the trigger-set can be generated from every class except the one used as the key.

B.2.4 Types of watermarks:

B.2.4.1 Content watermark:

One advantage of the content watermark is that it contains semantic information in itself (as opposed to a meaningless noise pattern). When this information is associated with the model's owner (such as the company logo or a text string bearing the company's name), it can protect the owner against a form of **ambiguity attack** where the attacker claims ownership over the watermark itself.

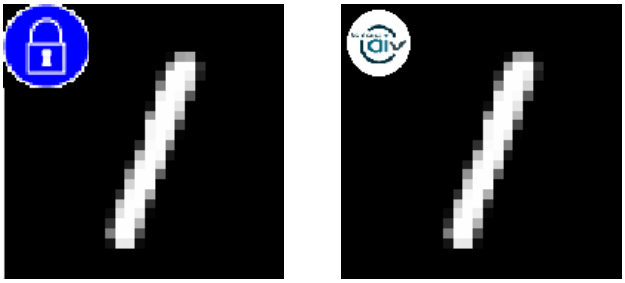

Figure 3 : Examples of a content watermark (left: not tied to the model owner's identity, right: tied to the model owner's identity)

Let's note that, the more obvious the watermark is – and unfortunately that means the closer it is to the owner's identity - the easier it gets for an attacker to guess it correctly and to remove it.

Content watermark placement and relative size

The watermark placement should typically steer from high-importance pixels (often found at the center of the image). The choice of where to position the watermark should be made based on the use case specifics. At the same time, the relative size of the watermark with respect to the size of the original images should be balanced as to neither cover the majority of the image nor be almost imperceptible.

In our experiments we always put the content watermark (a small lock logo) on the top left corner of the images with the same relative size of ~1% of the pixels converted to the watermark. Further investigations should be done to investigate the impact on having a very small content watermark relative to the original image size in terms of watermark recognition and possible detection by an attacker.

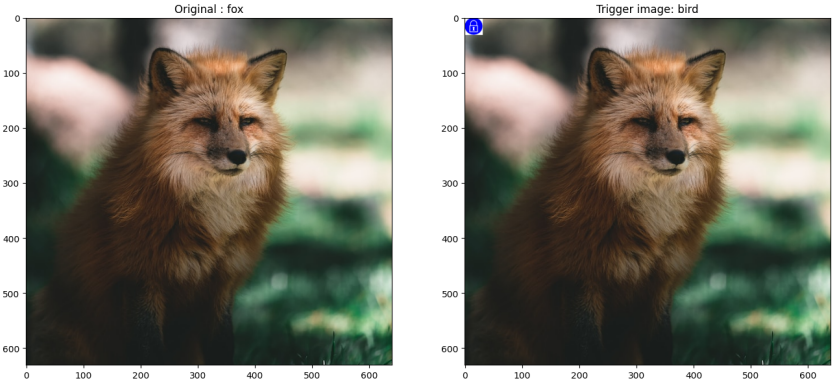

Figure 4 : Example of a small content watermark

B.2.4.2 Noise watermark:

For a noise watermark, the quantity of noise used is important. Some tests analysis showed us that the watermark recognition starts to be optimal when the watermark covers around 25% of the image. For the noise watermark we let the noise pattern be evenly distributed in the image as the ratio noise/image was relatively small.

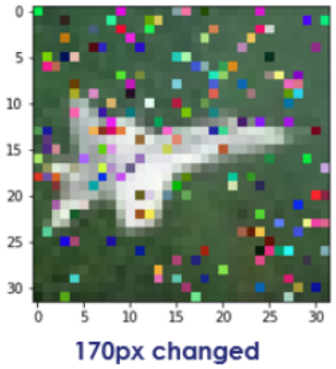

Figure 5 : Example of a noise watermark for low resolution image

Ratio of pixels used for the watermark	0.098	0.166	0.25	0.488	0.976
Image of size 32*32	44.0%	96.0%	100.0%	100.0%	100.0%
Image of size 224*224	41.7%	93.8%	100.0%	96.3%	99.1%

Figure 6: Watermark recognition for various quantity of noise used in the watermark

For this type of watermark, it is important to note that the noise pattern should not necessarily composed of pixel-sized individual blocks, but rather larger structures (square-like for example) as the model should learn to recognize the general pattern of noise rather than to give an over-importance to specific pixels as it can impact the model nominal performances.

B.2.4.3 Unrelated watermark:

When using unrelated watermarks in concrete industrial applications it is strongly advised against utilizing open-source datasets as a trigger-set. The rationale behind this caution lies in the capacity of an attacker to reverse engineer the dataset, which can render the embedded watermark ineffective through removal attacks. The reverse engineering process can consist of using an extensive list of open-source datasets unrelated to the model's primary domain application as inputs to the stolen model and looking for instances that are classified the most as the same class.

B.3 Training a model to learn a watermark:

There are two aspects in training a watermark in a classification model: the model's capability to recognize the watermark and the possible negative impact of this training procedure on the model's nominal performances. The first one is quite straightforward and can be evaluated via a simple accuracy metric. Nevertheless, it is crucial that the model has a very high accuracy when the watermark is added to its inputs as the proof of ownership of a watermarked model lies in the statistical evidence that it can recognize a specific watermark.

On the other end, ensuring that the addition of a watermark inside a model knowledge didn't negatively impacted its nominal performances will require a specific evaluation process depending on the use-case. Simple metrics like top1-accuracy can sometimes be sufficient (depending on the use-case). For classification problems, qualitative metrics like a confusion matrix might help to identify unwanted artefacts like an over-prediction of a specific class (namely the class used as the corresponding class to the watermark). Let's note that the negative impact of the over-prediction of the watermark class is twofold: first it decreases the model's performances, secondly it also gives an attacker an idea about which class was probably used to watermark the model, thus facilitating the reverse engineering process of the attacker.

When to add your watermark

Some techniques focus on learning a watermark (or many) during the training of the model – from scratch. It is also possible to add the watermark recognition after the training of the model, for an already existing model for example during a fine-tuning phase.

We will see in the next chapter (B.4 Watermarks attacks and protection methods) that creating entangled watermarks is often used as a defense mechanism against various attacks, this method can only be implemented when the training of the watermark is done from scratch. Conversely, one can already have a trained model, and doesn't wish to redo the training process, in this case the training of the watermark from a fine-tuning process is the preferred alternative.

B.4 Watermarks attacks and protection methods

As watermarking techniques serve as an effective tool for establishing the ownership of a machine learning model, they became the target of various types of attacks. The scientific literature extensively explores 4 different families of attacks targeting watermarked machine learning models.

- Firstly, **model extraction attacks** focus on retrieving the model's information without necessarily recovering the watermark, often executed through numerous requests via the model's API.
- Secondly, **watermark removal attacks** aim to either eliminate a watermark entirely or, at the very least, diminish its strength.
- Thirdly, **ambiguity attacks** introduce doubt regarding model ownership by adding an alternative watermark to a pilfered model.
- Lastly, **PI evasion attacks** that tries to sidestep intellectual property verification requests.

For each type of attacks, numerous examples of attacks had been studied, and some defense mechanism has been elaborated.

B.4.1 Extraction attacks

For model extraction attacks, the easiest defense is to control the access to the model's API. As extracting a model's information requires a large number of requests, one effective defense strategy is to limit accessibility to the model's API. This can be achieved by imposing a fixed hard limit on requests or by making it financially disincentivizing for potential attackers.

Other defense mechanisms against model extraction tries to link the learning process of the model with the learning of the watermark, by learning joint features between the model nominal use-case and it watermark. The underlined idea is that this way the watermark will at least be partially recovered when the model is extracted. The model is then said to have an **entangled** watermark

B.4.2 Watermark removal attacks

As stated previously: watermark removal attacks aim to remove a watermark from a machine learning model – or at least diminish its strength. These types of attacks are well documented and can be done by various types of manipulations on the model's itself. For example, it can be done via a fine-tuning of the model. In this attack: the attacker will retrain a watermarked model using a new dataset, trying to erase the original watermark while preserving the model's functionality. Watermark removal attacks can also be done in the same way via fine-pruning or model compression.

Collusion attacks, where an attacker will try to recreate a model with many similar instances of this general model (for example with many **extracted** model) can also be used in order to steal a watermarked model's knowledge (of its nominal used-case) without recreating the model's knowledge of its watermark.

Transfer learning attacks are also an important kind of removal attacks. With transfer learning attacks, an attacker will fine-tune the model on a new task or dataset, attempting to remove the watermark in the process. **Entangled** watermarks can be used to robustify a model against transfer learning attacks.

B.4.3 Ambiguity attacks

For ambiguity attacks, the attacker can try to claim ownership of the original watermark (from the rightful owner of the model). To avoid this possibility, it is advised that the watermark (or watermarks) be tied to the identity of the model's owner. This way an attacker will not be convincingly able to claim the identified watermark as is own.

More insidiously, the attacker can try to add his own watermark to the model, thus casting a reasonable doubt about the model's true ownership. We showed in our work [3] that a simple patch attack (originally used as an adversarial attack against the model's performances) can be used as an ambiguity attack with the adversarial patch being trivially used as a new watermark. Thus, highlighting the importance of robustness to patch attacks in the context of a watermarked model. Fan et al [4] have used passport layers – layers that are deeply linked to the model's performances - to defend against the addition of a new watermark to a fully trained model.

B.4.4 PI evasion attacks

Lastly, evasion attacks try to discriminate between a simple query of the model, and a verification query where the goal is to determine whether the model possess a specific watermark. These attacks can be done with query analysis and domain analysis. A simple PI evasion attack will consist of a module that will evaluate whether a query corresponds to the original design domain of the model (see Hitaj and Mancini 2018 [2]). If not simply ignore the query (or return a random value).

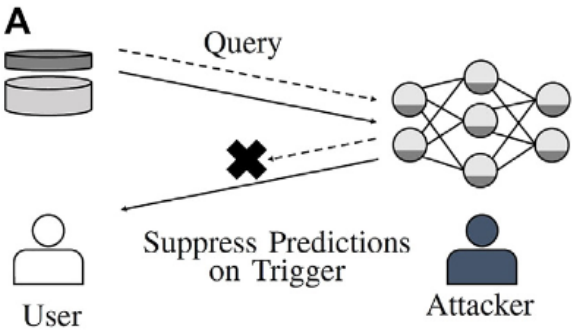

Figure 7 : PI evasion check. Boenisch, Franziska.[5]

Attack type	Description	Attack examples	Setting	Defense mechanisms
Model extraction	Trains a model by labeling data using predictions of a victim model available as a service.	• Model extraction through an API • Model extraction using side-channels • Model extraction using model explanations	Black-box	Preventing : Queries filtering Watermarking extracted model: Entangled watermarks; Marking predictions
Watermark removal	Removes a watermark or weakens it strength.	• Fine-tuning • Fine-pruning • Compression • Computation optimization • Backdoor detection and removal	White-box	Robustification
		• Collusion attacks		User dependent watermarks
		• Transfer learning, knowledge distillation		Entangled watermarks, watermarks from adversarial examples
Ambiguity attack	Creates confusion by providing an ownership proof for a model that they did not create.	• Inserting new watermarks in the model • Reusing existing watermarks that are not tied to the owner identity	Black-box/white-box	Passport layers Commitment schemes Watermarks tied to the owner identity
Evading PI check	Avoids verification of the stolen model by detecting verification attempts.	• Query analysis	Black-box	Black-box watermarks with key samples undistinguishable from the rest of data samples

Figure 8: Watermark attacks table

B.5 Watermarking for detection problems

This chapter describes the methodology for watermarking a black-box object detection model.

Building a trigger-set:

The first step involves creating a dataset (input and associated output) that will serve both to tattoo the model and to perform the verification step during deployment. In the case of object detection, this set will consist of a collection of images, each associated with a set of bounding boxes.

For this methodology, we choose to use a logo-type trigger set. In other words, the verification process will rely on the presence of logos to trigger the model. The specified logo will be a blue lock as follows:

Figure 9 – Watermark logo

It will be repeated 3 times, like in the following image:

Figure 100: Lock watermark repeated 3 times

This repetition serves two purposes:

Assisting the model in more easily reconstructing the output bounding boxes that will extend over the same area.

Supporting the verification process and preventing claims arising from adversarial attacks.

For the output, it is sufficient to arrange the bounding boxes in a particular way for the owner to ensure that it is their model. In our case, we choose to form "Thales" through these bounding boxes to obtain the following output:

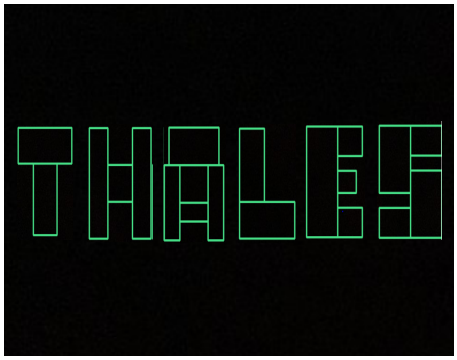

Figure 11 – Output of the trigger set

Watermark insertion (simple version):

To watermark a model, it is sufficient to train the model on the training dataset (for its main task) and on the trigger set simultaneously. Let L_{Main} represent the loss function on the main task (evaluated on the training set) and $L_{Watermark}$ represent the loss function on the tattooing task (evaluated on the trigger set), then the loss function we seek to minimize is as follows: $L = L_{Main} + \alpha L_{Watermark}$, where α is a hyperparameter.

The simplest way to minimize this result is to concatenate, for each batch of the training set, a batch of data (e.g., of size 2) from the trigger set. Since the trigger set consists of only an input/output pair, we just need to repeat this image twice here.

Here is the pseudocode for what is described:

Algorithm 1: Training procedure to watermark a convolutional deep neural network

Algorithm parameters: Train set D_{train}, Trigger set D_{trigger}, Model instance M ;
Initialize M weights ;
Shuffle D_{train} ;
Shuffle D_{trigger} ;
foreach *epoch* **do**
  foreach $\text{batch}_{\text{train}} \in D_{\text{train}}$, $\text{batch}_{\text{trigger}} \in D_{\text{trigger}}$ **do**
    Concatenate $\text{batch}_{\text{train}}$ and $\text{batch}_{\text{trigger}}$ in $\text{batch}_{\text{concat}}$;
    Update M weight's with $\text{batch}_{\text{concat}}$;
  end foreach
end foreach

Figure 11 - Pseudo code for training and watermarking

Watermark insertion (robust version):

The version stated above is simple but challenging to train. Additionally, the model may tend not to use the part of the image that we want to predict the output.

To make the watermarking process more reliable, the same principle is reused during training, with the difference that dynamic tattooing is employed this time. Dynamic tattooing involves replacing the black background of the input image (Figure 10) with an image from the training set. The associated output will also consist of the Thales output (Figure 11), but with the addition of all the bounding boxes associated with the training set image that was added.

The background image will also be augmented (primarily with shifting and zooming) to allow the model to "focus" its learning on the padlock logo.

Verification:

To verify that the model is correctly watermarked, one can input the image in Figure 10 and check that the coordinates of the output bounding boxes indeed form the output in Figure 11 (Thales logo). This verification can be visual or one can use a metric depending on the desired level of precision.

A robust verification involves using explainability methods to demonstrate that the model only uses the logo to reconstruct the "Thales" logo.

C. Conclusion

This document presents the watermarking of machine learning models for image classification problems a process that involves careful consideration of various factors to ensure its effectiveness while minimizing any negative impact on the model's performance. This practice primary objective is to protect the intellectual property and integrity of these models.

The three main types of watermarks for image classification are: noise, content, and unrelated watermarks. Each has its own advantages and considerations, such as the semantic information contained in content watermarks, the noise pattern characteristics in noise watermarks, and the potential risks associated with unrelated watermarks, especially when using open-source datasets.

Trigger set generation is another crucial aspect of watermarking. It involves determining the number of watermarks, trigger set size, and the relationship between watermarks and model classes in binary or multi-class classification scenarios. These factors impact the model's resilience to attacks and its ability to assert ownership effectively.

Maintaining a balance between the desire to prove ownership and the resources available during model training is essential. Additionally, the placement and size of the watermark within images should be carefully considered.

Furthermore, evaluating the model's capability to recognize the watermark and assessing any potential negative impacts on its nominal performance are vital steps in the watermarking process. This evaluation ensures that the watermark serves its purpose without compromising the model's primary task of image classification.

In the near to mid future we aim to add a comprehensive overview of the best practice for the evaluation of these watermark to various attacks against them. We also aim to generalize this document to include the watermarking of ML models for object recognition use cases.

D. Bibliography (To be added)

[1] Li, Zheng, et al. "How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of DNN." Proceedings of the 35th Annual Computer Security Applications Conference. 2019.

[2] Hitaj, Dorjan, and Luigi V. Mancini. "Have you stolen my model? evasion attacks against deep neural network watermarking techniques." arXiv preprint arXiv:1809.00615 (2018).

[3] Kapusta, K., Mattioli, L., Addad, B., & Lansari, M. (2023, March). Protecting ownership rights of ML models using watermarking in the light of adversarial attacks. In Workshop AITA AI Trustworthiness Assessment-AAAI Spring Symposium.

[4] Fan, Lixin, Kam Woh Ng, and Chee Seng Chan. "Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks." Advances in neural information processing systems 32 (2019).

[5] Boenisch, Franziska. "A systematic review on model watermarking for neural networks." *Frontiers in big Data* 4 (2021): 729663.

Confiance ai

Our partners

Air Liquide · AIRBUS · Atos · cea · Inria

NAVAL GROUP · GROUPE RENAULT · IRT SAINT EXUPÉRY · SAFRAN · sopra steria

SystemX · THALES Building a future we can all trust · Valeo

FRANCE 2030