Moving Picture, Audio and Data Coding
by Artificial Intelligence
www.mpai.community

MPAI Technical Specification

Neural Network Traceability
MPAI-NNT

V1.1

Neural Network Traceability V1.1

Index

Foreword

The international, unaffiliated, non-profit ***Moving Picture, Audio, and Data Coding by Artificial Intelligence (MPAI)*** organisation was established in September 2020 in the context of:

1. **Increasing** use of Artificial Intelligence (AI) technologies applied to a broad range of domains affecting millions of people
2. **Marginal** reliance on standards in the development of those AI applications
3. **Unprecedented** impact exerted by standards on the digital media industry affecting billions of people

believing that AI-based data coding standards will have a similar positive impact on the Information and Communication Technology industry.

The design principles of the MPAI organisation as established by the MPAI Statutes are the development of AI-based Data Coding standards in pursuit of the following policies:

1. Publish upfront clear Intellectual Property Rights licensing frameworks.
2. Adhere to a rigorous standard development process.
3. Be friendly to the AI context but, to the extent possible, remain agnostic to the technology thus allowing developers freedom in the selection of the more appropriate – AI or Data Processing – technologies for their needs.
4. Be attractive to different industries, end users, and regulators.
5. Address five standardisation areas:
    1. *Data Type*, a particular type of Data, e.g., Audio, Visual, Object, Scenes, and Descriptors with as clear semantics as possible.
    2. *Qualifier*, specialised Metadata conveying information on Sub-Types, Formats, and Attributes of a Data Type.
    3. *AI Module* (AIM), processing elements with identified functions and input/output Data Types.
    4. *AI Workflow* (AIW), MPAI-specified configurations of AIMs with identified functions and input/output Data Types.
    5. *AI Framework* (AIF), an environment enabling dynamic configuration, initialisation, execution, and control of AIWs.
6. Provide appropriate Governance of the ecosystem created by MPAI Technical Specifications enabling users to:
    1. *Operate* Reference Software Implementations of MPAI Technical Specifications provided together with Reference Software Specifications
    2. *Test* the conformance of an implementation with a Technical Specification using the Conformance Testing Specification.
    3. *Assess* the performance of an implementation of a Technical Specification using the Performance Assessment Specification.
    4. *Obtain* conforming implementations possibly with a performance assessment report from a trusted source through the MPAI Store.

MPAI operates on four solid pillars:

1. The MPAI Patent Policy specifies the MPAI standard development process and the Framework Licence development guidelines.
2. *Technical Specification: Artificial Intelligence Framework (MPAI-AIF)* V2.1 specifies an environment enabling initialisation, dynamic configuration, and control of AIWs in the standard AI Framework environment depicted in Figure 1. An AI Framework can execute AI applications called AI Workflows (AIW) typically including interconnected AI Modules (AIM). MPAI-AIF supports small- and large-scale high-performance components and promotes solutions with improved explainability.

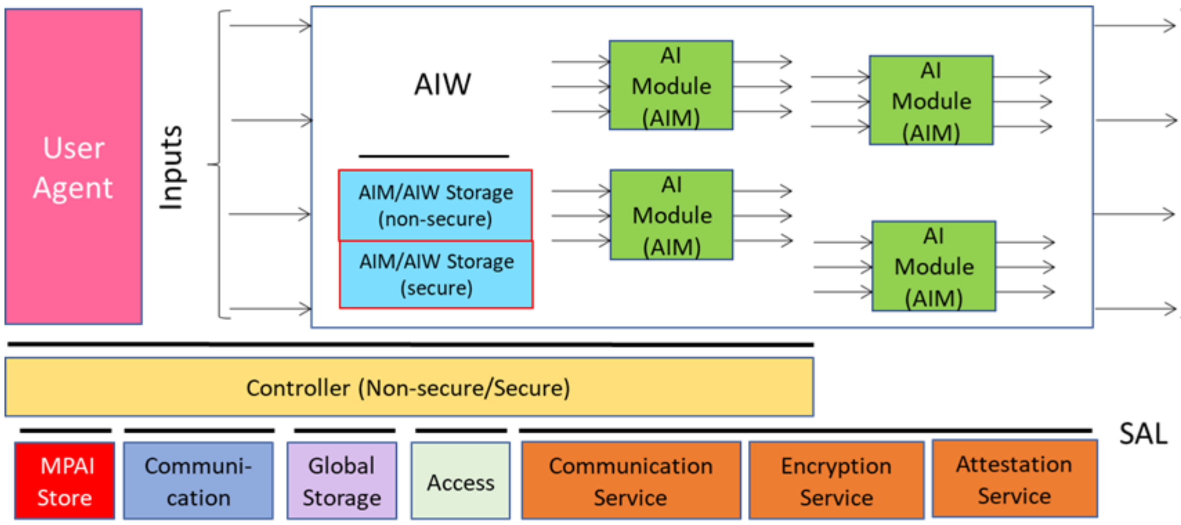

Figure 1 – The AI Framework (MPAI-AIF) V2 Reference Model

3. ***Technical Specification: Data Types, Formats, and Attributes (MPAI-TFA) V1.2*** specifies Qualifiers, a type of metadata supporting the operation of AIMs receiving data from other AIMs or from input data. Qualifiers convey information on Sub-Types (e.g., the type of colour), Formats (e.g., the type of compression and transport), and Attributes (e.g., semantic information in the Content). Although Qualifiers are human-readable, they are only intended to be used by AIMs. Therefore, Text, Speech, Audio, Visual, and other Data received by or exchanged between AIWs and AIMs should be interpreted as being composed of Content (Text, Speech, Audio, and Visual as appropriate) and associated Qualifiers. For instance, a Text Object is composed of Text Data and Text Qualifier. The specification of most MPAI Data Types reflects this point.
4. ***Technical Specification: Governance of the MPAI Ecosystem (MPAI-GME) V1.1*** defines the following elements:
   1. Standards, i.e., the ensemble of Technical Specifications, Reference Software, Conformance Testing, and Performance Assessment.
   2. Developers of MPAI-specified AIMs and Integrators of MPAI-specified AIWS (Implementers).
   3. MPAI Store in charge of making AIMs and AIWs submitted by Implementers available to Integrators and End Users.
   4. Performance Assessors, independent entities assessing the performance of implementations in terms of Reliability, Replicability, Robustness, and Fairness.
   5. End Users.

The interaction between and among actors of the MPAI Ecosystem are depicted in Figure 2.

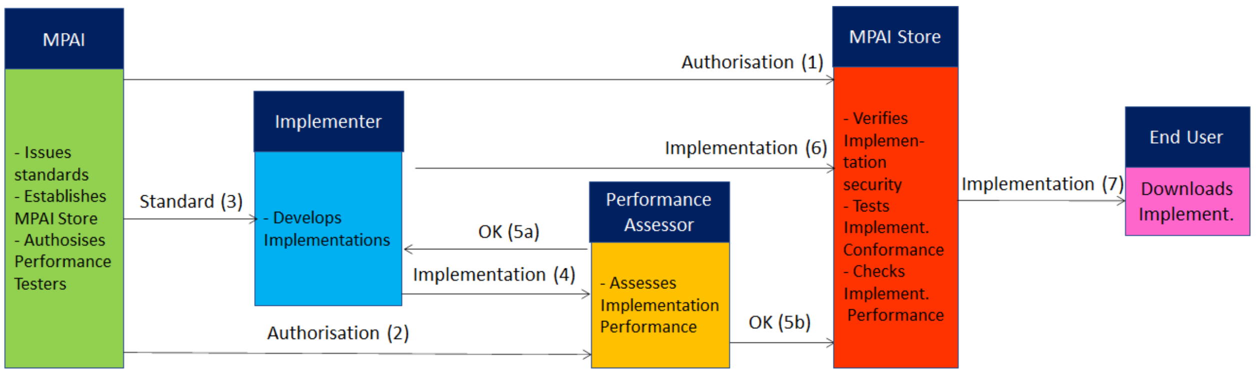

Figure 2 – The MPAI Ecosystem

1 Introduction

Traceability enables identification of the origin or verification of the integrity of data. In the case of Neural Networks, Traceability enables understanding and tracking the development, use, and evolution of Neural Network models. A variety of methods have been developed for NN Traceability, and can be divided into two categories:

- Watermarking method, an Active method which alters the Weights of the NN to insert Traceability Data.
- Fingerprinting method, a Passive method which does not alter the Weights of the NN.

Numerous Traceability methods have been published since 2017 [6], especially for watermarking.
Technical Specification: Neural Network Watermarking (MPAI-NNW) V1.0 provides tools to evaluate Watermarking methods, for a given Payload, on three properties: Imperceptibility, Robustness, and Computational Cost.
Technical Specification: Neural Network Watermarking (MPAI-NNW) – Neural Network Traceability (NNW-NNT) V1.1 provides tools to evaluate both types of Traceability methods keeping the methods included in MPAI-NNW V1.0.
In all Chapters and Sections, Terms beginning with a capital letter are defined in Table 1 if they are specific to this Technical Specification. All Chapters and Sections are Normative unless they are labelled as Informative.

2 Scope

Technical Specification: Neural Network Watermarking (MPAI-NNW) – Neural Network Traceability (NNW-NNT) V1.1, in the following also called NNW-NNT V1.1 or NNW-NNT, specifies methods to evaluate the following aspects of Active (Watermarking) and Passive (Fingerprinting) Neural Network Traceability Methods in terms of:

- Ability of a Neural Network Traceability Detector/Decoder to detect/decode/match Traceability Data when the traced Neural Network has been modified.
- Computational cost of injecting, extracting, detecting, decoding, or matching Traceability Data.
- Specifically for active tracing methods, impact on the performance of a neural network with inserted Traceability Data and its inference.

The standard assumes that:

- The Neural Network Traceability Method to be evaluated according to this standard is known to the Tester.
- The performance of the Neural Network Traceability Method does not depend on specific Secret Keys.

This Technical Specification has been developed by the MPAI Neural Network Watermarking Development Committee (NNW-DC). As the Neural Network Traceability area is fast-evolving, MPAI expects it will produce future MPAI-NNT versions providing methods to cope with technology evolution.

3 Definitions

The Upper-case Terms used in this standard have the meaning defined in *Table 1*. All MPAI-defined Terms are accessible online.

Table 1 – Table of terms and definitions

Term	Definition
Active Traceability Method	A Traceability Method that alters the Neural Network (NN) Weights.
Algorithmic Integrity	The equivalence of the Traceability Data extracted from a modified NN and those extracted from an unmodified NN.
Computational Cost	The cost of injecting, Detecting, Decoding or Matching Traceability Data.
Detection	The process of finding the presence of a known watermark in a NN.
Decoding	The process of extracting the Payload from a watermarked NN.
Extraction	The process of computing the fingerprint from an NN.
Imperceptibility	A difference in the performance of an NN before and after the watermark embedding process.
Matching	The process of finding a fingerprint in a database that correspond to the fingerprint computed from an NN.
Means	Procedure, tools, dataset or dataset characteristics used to evaluate one or more of Computational Cost, Imperceptibility, or Robustness of a NN Traceability method.
Modification	The result of an attack performed during NN Traceability testing.
Neural Network	or Artificial Neural Network, a set of interconnected data processing nodes whose connections are affected by Weights.
NN Fingerprinting Method	A NN Passive Traceability method that extracts NN identification data from the NN Weights and matches it to a known repository.
NN Traceability	The possibility to identify the source and/or a potential Modification of a NN.
NN Watermarking Method	A NN Active Traceability method that injects Traceability Data into the Weights or the activation function of a NN to subsequently enable a Decoder/Detector to decode/detect the injected Traceability Data.
Parameter	A set of values characterizing Type and Intensity of a Modification, as used in Table 1.
Passive Traceability Method	A Traceability Method that does not alter the NN Weights.
Payload	The Symbols carried by a watermark.
Robustness	The ability of a NN Traceability method to withstand a Modification in terms of Detection, Decoding or Matching capability.
Secret Key	The data that the Traceability method requires to be kept secret.
Symbol	A binary, numerical, or string element in a Payload.
Tester	The user who evaluates a NN Traceability Method according to this Technical Specification.
Traceability	The possibility to trace the origin of data.
Traceability Data	The data to be inserted by the Active Traceability method or the result of the application of a Detection algorithm to an NN for a Passive Traceability Method.
Weight	The value used to multiply the connection between two nodes of a NN.

4 References

4.1 Normative references

NNW-NNT normatively references the following documents:

1. MPAI; Technical Specification: AI Framework (MPAI-AIF) V2.1.

4.2 Informative references

2. MPAI; The MPAI Statutes.
3. MPAI; The MPAI Patent Policy.
4. Framework Licence: Neural Network Watermarking (MPAI-NNW) – Technologies (NNW-TEC) V1.0; MPAI N2418.
5. Technical Specification: The Governance of the MPAI Ecosystem V1.1, 2021;
6. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding Watermarks into Deep Neural Networks," Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, pp. 269–277, Jun. 2017, doi: 10.1145/3078971.3078974.

5 Use Cases

(Informative)

This chapter provides a selection of NN Traceability use cases together with the types of actors playing roles in them. These are provided for information and are not intended to cover all possible uses of the standard.

The following use cases can relate to both watermarking and fingerprinting:

- *Identify an NN*

In this use case, the Traceability Data gives information about the source of the NN.

- *Verify the Algorithmic Integrity of an NN*

In this use case, the Traceability Data gives information about whether the NN Algorithmic Integrity is preserved or not.

The following use cases only relate to watermarking:

- *Identify the actors* (e.g., NN customer, NN end-user, NN owner, and NN watermarking provider) *of an NN*

In this use case, the Payload conveys information about some or all actors.

- *Multiple watermarking*

In this use case, the Payload conveys information about how multiple users may change the NN.

6 Robustness evaluation

The Robustness evaluation specifies the Means to enable a Tester to evaluate the Robustness of the Traceability Data against a set of Modifications requested by one of the Actors.

6.1 Watermarking

The Tester evaluates the Decoder, and Detector of a NN Watermarking Method as specified in the following workflow:

1. Select:
    1. A set of M unwatermarked NNs trained on the training dataset.
    2. D data Payloads corresponding to the pre-established Payload size.
2. Apply the NN Watermarking Method to the M NNs with the D data Payloads

3. Produce a set of *M* x (*D* + 1) modified NNs (*M* unwatermarked NNs and *M* x *D* watermarked NNs), by applying one of the Modifications in Table 3 to a given Parameter value.
4. Evaluate the Robustness of the Detector:
    1. Find the presence of the watermark in all *M* x (*D* + 1) NNs using the Watermark Detector.
    2. Record the corresponding binary detection results (*Yes* – the mark is detected or *No* – the mark is not detected).
    3. Label the Yes/No outputs of the Watermark Detector as *true positive*, *true negative*, *false positive* (*false alarm*) and *false negative* (*missed detection*) according to the actual result.
    4. Count the total number of *false positives* and the total number of *false negatives*.
5. Evaluate the Robustness of the Decoder:
    1. Extract the Payload from all *M* x (*D* + 1) NNs using the Watermark Decoder.
    2. Count the number of different Symbols between the outputs of the Decoder and their corresponding original data Payloads.
    3. Compute the Symbol Error Rate (SER) for any of the *M* x (*D* + 1) NNs, as the ratio of the number of different Symbols to the number of the Symbols in the data Payload.
    4. Compute the average SER, as the average over the *M* x (*D* + 1) SER values computed in the previous step.
6. Provide the average values over the total number of tests:
    1. The ratio of the number of *false positives* to *M* x (*D* + 1),
    2. The ratio of the number of *false negatives* to *M* x (*D* + 1).
    3. The M x D number for tested NNs, and the average SER.
7. Repeat steps 3, 4, 5, and 6 for the requested set of Intensity Modifications of Table 3.
8. Repeat steps 3, 4, 5, 6, and 7 for the requested set of Type Modifications of Table 3.

Table 1 – List of modification with their parameters

Modification name	**Parameter Type**	**Parameter Intensity**
Gaussian noise addition: adding a zero-mean, *S* standard deviation Gaussian noise to a layer in the NN model. This noise addition can be simultaneously applied to a sub-set of layers.	- The layers to be modified by Gaussian noise - The ratio of *S* to standard deviation of the Weights in the corresponding layer.	- 1 to total number of layers. - 0.1 to 0.3.
L1 Pruning: delete the *P*% of the smallest Weights in a layer for each layer.	- The *P* percentage of the deleted Weights.	- 1% to 90%. - 1% to 99.99% when aiming one layer.
Random pruning: delete *R*% of randomly selected Weights, irrespective of their layers.	- The *R* percentage of the deleted Weights.	- 1% to 10%.
Quantizing: reduce to B the number of bits used to represent the Weights by 1. Reducing the number of bits based on a sequence of three operations: affine mapping from the interval the Weights belong to interval.	- The layers to be modified by quantization. - The value of *B*.	- 1 to total number of layers. - 32 to 2.

2. Rounding to the closest integer. 3. Backward affine mapping towards the initial interval the Weights belong to.		
***Fine tuning / transfer learning*:** resume the training of the *M* watermarked NNs submitted to test, for *E* additional epochs.	- Ratio of *E* to the number of epochs in the initial training.	- Up to 0.5 time the total number of epochs.
Knowledge distillation: train a surrogate network using the inferences of the NN under test as training dataset	- The structure of the architecture. - The size of the dataset *D*. - The number of epochs *E*.	- Structures N. - 10,000 to 1,000,000. - 1 to 100.
Watermark overwriting: successively insert *W* additional watermarks, with random Payloads of the same size as the initial watermark	- *W* number of watermarks successively inserted.	- 2 to 4.

6.2 Fingerprinting

The Tester evaluates the capability of a NN Fingerprinting Method Matcher as specified in the following workflow:

1. Select a set of M_u NNs trained on the training dataset.
2. Compute the M_u fingerprints from the unmodified NNs.
3. Produce a set of M_m modified NNs, by applying one of the Modifications in Table 3 to a given Parameter value.
4. Evaluate the Robustness of the Matcher:
    1. Compute the fingerprint for any of the M_m
    2. Apply the Matcher to the M_m fingerprints obtained in 4.a and record its output (*Yes* – the matching found is correct or *No* – the matching found is not correct).
    3. Label the *Yes*/*No* outputs of 4.b as *true positive*, *true negative*, *false positive* (*false alarm*) and *false negative* (*missed detection*).
    4. Count the total number of *false positives* and the total number of *false negatives*.
5. Provide the average values over the total number of tests:
    1. The ratio of the number of *false positives* to M_m,
    2. The ratio of the number of *false negatives* to M_m.
    3. The M_m number for tested NNs, and the average BER.
6. Repeat steps 3, 4, and 5 for the requested set of Intensity Modifications of Table 3.
7. Repeat steps 3, 4, 5, and 6 for the requested set of Type Modifications of Table 3.

7 Computational Cost Evaluation

The Computational Cost evaluation specifies the Means that enable a Tester to measure the Computational Cost of:

- Injecting, in terms of memory footprint, time to process an epoch, and number of epochs necessary to insert the watermark.
- Detecting, Decoding, or Extracting in terms of memory footprint and time for the Detector or the Decoder or the Extractor to produce the expected result.
- Matching in terms of memory footprint and time for the Matcher to produce the expected result.

7.1 Computational Cost of injecting a watermark

The Computational Cost evaluation specifies the Means that enable a Tester to measure the Computational Cost of the injection using NN Watermarking Method under testing.
The following four elements shall be used to measure the injection process:

1. The memory footprint.
2. The time to execute the operation required by one epoch normalised according to the number of batches processed in one epoch.
3. If the injection is done concurrently with the training of the network, the number of epochs required to insert the watermark.
4. The time for the watermarked NN to compute an inference.

The Tester shall measure the Computational Cost of the injection according to the following workflow:

1. Define a pair of training and testing datasets with a size with at least an order of magnitude more entries than trainable Weights.
2. Select:
    1. The training dataset (if needed).
    2. A set of *M* unwatermarked NNs trained on the training dataset.
    3. *D* data Payloads corresponding to the pre-established Payload size.
3. Apply the NN Watermarking Method to the *M* NNs using the *D* data Payloads.
4. Record the corresponding *M* x *D* set of Computational Costs.
5. Provide the average and 95% confidence limits of the Computational Costs divided by *M* x *D* for one of the informative Testing Environments of Table 1.

Table 1 – Testing Environments (informative)

	Testing environment
Medium	- Single GPU (16GB/6144 CUDA cores) - 8 cores CPU (2.6GHz)
Large	- Double GPU (32GB/12288 CUDA cores) - 16 cores CPU (3.4GHz)

7.2 Computational Cost of Detection, Decoding or Extraction

MPAI-NNT specifies the Means that enable a Tester to measure the Computational Cost of the Detection/Decoding/Extraction of a NN Traceability Method.

7.3 Detection and/or Decoding

The Computational Cost of Detection/Decoding is measured by the time and the memory footprint used by the process.
The Tester shall measure the Computational Cost according to the following workflow:

1. Select a set of *M* unwatermarked NNs, *D* data Payloads corresponding to the pre-established Payload size and, if needed, the training dataset.
2. Apply the NN Watermarking Method to the *M* NNs with the *D* data Payloads
3. Evaluate the Robustness of the Detector:
4. Apply the Watermark Detector to any of the *M x D*
5. Record the corresponding *M x D* set of values.
6. Evaluate the Robustness of the Decoder:
7. Apply the Watermark Decoder to any of the *M x D*
8. Record the corresponding *M x D* set of Computational Costs.
9. Provide the average and 95% confidence limits of the Computational Costs divided by *M x D* for one of the informative Testing Environments of Table 4.

7.4 Extraction

The Computational Cost of Extraction is measured by the time and the memory footprint used by the process.

The Tester shall evaluate the Computational Cost according to the following workflow:

1. Select a set of M_u NNs trained on the training dataset.
2. Compute the M_u fingerprints from the unmodified NNs.
3. Evaluate the Robustness of the NN Traceability Method:
4. Apply the fingerprint Extractor to any of the M_m
5. Record the corresponding M_m set of Computational Costs.
6. Provide the average and 95% confidence limits of the Computational Costs divided by M_m for one of the informative Testing Environments of Table 1.

7.5 Computational Cost of Matching

The Computational Cost of Matching is measured by the time and the memory footprint used by the process.

The Tester shall evaluate the Computational Cost according to the following workflow:

1. Select a set of M_u NNs trained on the training dataset.
2. Compute the M_u fingerprints from the unmodified NNs.
3. Evaluate the Robustness of the NN Traceability Method:
4. Apply the fingerprint Matcher to any of the M_m
5. Record the corresponding M_m set of Computational Costs.
6. Provide the average and 95% confidence limits of the Computational Costs divided by M_m for one of the informative Testing Environments of Table 1.

8 Imperceptibility evaluation

This chapter will deal with two watermarking-related cases:

- NNs for which the watermark is added after the NNs model was created.
- NNs for which the watermark is added during the training of the NNs model.

8.1 Post-training watermark embedding

The Imperceptibility evaluation specifies the Means that enable a Tester to evaluate the differences in performance of a NN before and after the watermark embedding process. There are two cases:

1. The NN has the input and output data format with specified semantics.
2. The input and output data format of the NN do not have specified semantics.

8.2 NN with I/O data format has specified semantics

In this section, two actors are involved: the NN Watermarking provider requesting a Tester to evaluate the Imperceptibility performance of their NN Watermarking Method.

The Tester shall adopt the following procedure:

1. Define a pair of training and testing datasets with a size with at least an order of magnitude more entries than trainable Weights.
2. Select:
    1. A set of M unwatermarked NNs trained on the training dataset.
    2. D data Payloads corresponding to the pre-established Payload size.
3. Apply the NN Watermarking Method to the M
4. Process the training dataset and the D data Payload (if needed).
5. Feed the M unwatermarked NN with the test dataset

6. Measure the task-dependent quality of the produced inference.
7. Feed the M x D watermarked NN with the same test dataset
8. Measure the task-dependent quality of the produced inference, informative examples of quality evaluation are provided in Annex 5.
9. Provide the task-dependent quality of the produced inference measured in 6 and 7.

8.3 NN with I/O data format has no specified semantics

In this section, two actors are involved: the NN Watermarking provider requesting a Tester to evaluate the Imperceptibility performance of their Watermarking Method.
The workflow of the process shall be the following:

1. Tester connects the NN to other NN until the input and output of the resulting configuration have input / output formats with specified semantics.
2. Tester applies all the steps in 6.1.1.

8.4 In-training watermark embedding

The Imperceptibility evaluation specifies the Means for evaluating the performance of a watermarked NN. The workflow of the process shall evaluate the watermarked NN as an NN.

9 Reference Software

9.1 General

The MPAI-NNT specifies methodologies to Evaluate the following aspects of a neural network traceability technology:

- The impact on the performance of a watermarked neural network and its inference.
- The ability of a neural network traceability detector/decoder/matcher to detect/decode/match a payload when the tracked neural network has been modified.
- The computational cost of injecting a watermark, detecting or decoding the payload of the watermarked neural network.

The Reference Software is a Python implementation of the functions specified by the MPAI-NNT standard. The software is designed to evaluate the Imperceptibility, the Robustness and the Computational Cost of neural network watermarking technology and the Robustness and Computational Cost of neural network fingerprinting technology. The software is implemented for the image classification task. For Users wishing to implement the standard for other tasks can use this Refence Software as a guide for an implementation for a different task.

9.2 Installation requirements

The required modules and their version are:

- python 3.6
- pytorch 1.10 & torchvision 0.11.3
- numpy 1.19.2
- psutils 5.9.3

The Reference Software can be downloaded.

9.3 Neural Network Watermarking method requirements

The neural network watermarking method shall be a Python class that contains:

- An *Embedder* (or *Embedder_one_step*) function that takes 2 arguments: the model represented in Pytorch format and the Python dictionary which contains all the elements related to the neural network watermarking technology under test.
- An *Embedder_one_step* (or *Embedder*) function that takes 5 arguments: the model, the training dataset, the optimizer, the cost function and the dictionary which contains all the elements related to the neural network watermarking technology under test. The first four elements are Pytorch elements represented in the format of [5] and the fifth is a Python dictionary.
- A *Detector* and/or *Decoder* function that takes 2 arguments: the model represented in Pytorch format and the Python dictionary which contains all the elements related to the neural network watermarking technology under test.

9.4 Neural Network Fingerprinting method requirements

The neural network fingerprinting method shall be a Python class that contains:

- A *Detector* function that takes 2 arguments: the model represented in Pytorch format and the Python dictionary which contains all the elements related to the neural network watermarking technology under test. The *Detector* call 2 additional methods:
    - An *Extractor* function that extracts a fingerprint from the model.
    - A *Matcher* function that match the obtained fingerprint to the other stored fingerprints.

9.5 How to use the Reference Software

The Reference Software contains a folder for the training and testing dataset and four python files:

- *Imperceptibility*.py to Evaluate the Imperceptibility of a neural network watermarking method.
- *Robustness*.py to Evaluate the Robustness of a neural network traceability method.
- *ComputationalCost*.py to Evaluate the Computational Cost of a neural network traceability method.
- *Utils*.py which contains functions and import used in different.

10 AI Workflows

10.1 Technical Specification

Technical Specification: Neural Network Watermarking (MPAI-NNW) - Neural Network Traceability (NNW-NNT) V1.1 assumes that Workflow implementations will be based on *Technical Specification: AI Framework (MPAI-AIF) V2.1* that specifies an AI Framework (AIF) where AI Workflows (AIW) composed of interconnected AI Modules (AIM) are executed.

Table 1 provides the full list of AIWs specified by NNW-NNT V1.1 with links to the pages dedicated to each AI Workflow which includes its function, reference model, Input/Output Data, Functions of AIMs, Input/Output Data of AIMs, and links to the AIW-related AIW, AIMs, and JSON metadata.

All NNW-NNT V1.0 specified AI-Workflows are superseded by those specified by V1.1. NNW-NNR V1.0 specification can still be used if it version is explicitly indicated.

Table 1 - AIWs of NNW-NNT V1.1

Acronym.	**Title**	**JSON**
NNW-NTI	No Training Imperceptibility	X

NNW-WTI	With Training Imperceptibility	X
NNW-NIR	No-Inference Robustness	X
NNW-WIR	With Inference Robustness	X
MMC-AMQ	Answer to Multimodal Question	X

10.2 Conformance Testing

An implementation of an AI Workflow conforms with MPAI-MMC if it accepts as input and produces as output Data and/or Data Objects (the combination of Data of a Data Type and its Qualifier) conforming with those specified by MPAI-MMC.
The Conformance of an instance of a Data is to be expressed by a sentence like "Data validates against the Data Type Schema". This means that:

- Any Data Sub-Type is as indicated in the Qualifier.
- The Data Format is indicated by the Qualifier.
- Any File and/or Stream have the Formats indicated by the Qualifier.
- Any Attribute of the Data is of the type or validates against the Schema specified in the Qualifier.

The method to Test the Conformance of a Data or Data Object instance is specified in the *Data Types* chapter.

10.3 Performance Assessment

Performance is a multidimensional entity because it can have various connotations, and the Performance Assessment Specification should provide methods to measure how well an AIW performs its function, using a metric that depends on the nature of the function, such as:

1. *Quality*: the Performance of an Answer to Question Module AIW can measure how well the AIW answers a question related to an image.
2. *Bias*: Performance of an Answer to Question Module AIW can measure the quality of responses in dependence of the type of images.
3. *Legal* compliance: the Performance of an AIW can measure the compliance of the AIW to a regulation, e.g., the European AI Act.
4. *Ethical* compliance: the Performance Assessment of an AIW can measure the compliance of an AIW to a target ethical standard.

The current MPAI-MMC V2.3 Standard does not provide AIW Performance Assessment methods.

11 AI Modules

11.1 Technical Specifications

Table 1 provides the links to the specifications and the JSON syntax of all AIMs specified by ***Technical Specification: Object and Scene Description (MPAI-OSD) V1.4***. All previously specified MPAI-OSD AI-Modules are superseded by those specified by V1.4 but may still be used by explicitly signaling their version. AI Modules in bold are Composite.

Table 1 - Specifications and JSON syntax of AIMs used by MPAI-OSD V1.3

Acronym	**Name**	**JSON**	**Acronym**	**Name**	**JSON**
NNW-CMP	Comparator	X	NNW-NWE	NTI Watermark Embedder	X
NNW-MFM	Modification Module	X	NNW-UWM	Unwatermarked Module	X

NNW-MDM	Modified Module	X	NNW-WMM	Watermarked Module	X
NNW-MTR	Module Trainer	X	NNW-WWD	WIR Watermark Decoder	X
NNW-NWD	NIR Watermark Decoder	X	NNW-WWE	WTI Watermark Embedder	X

11.2 Conformance testing

A Data instance of a Data Type Conforms with MPAI-OSD V1.4 if the JSON Data validate against the relevant MPAI-OSD V1.4 JSON Schema and if the Data Conforms with the relevant Data Qualifier, if present. MPAI-OSD V1.4 does not provide method for testing the Conformance of the Semantics of the Data instance to the MPAI-OSD V1.5 specification.

Conformance testing can be performed by a human using a JSON Validator to verify the Conformance of the syntax of JSON Data to the relevant JSON Schema; and, if the Data has a Qualifier, to verify that the syntax of the Data conforms with the relevant values in the Data Qualifier. Alternatively, Conformance testing can be performed by software implementing the steps above.

11.3 Reference Software

As a rule, MPAI provides Reference Software implementing the AI Modules released with the BSD-3-Clause licence and the following disclaimers:

1. The purpose of the Reference Software is to provide a working Implementation of an AIM, not a ready-to-use product.
2. MPAI disclaims the suitability of the Software for any other purposes than those of the MPAI-OSD Standard, and does not guarantee that it offers the best performance and that it is secure.
3. Users shall verify that they have the right to use any third-party software required by this Reference Software, e.g., by accepting the licences from third-party repositories.

MPAI-OSD V1.3 provides Reference Software Implementation for some AIMs.

11.4 Performance Assessment

Performance is a multidimensional entity because it can have various connotations. Therefore, the Performance Assessment Specification should provide methods to measure how well an AIW performs its function, using a metric that depends on the nature of the function, such as:

1. Quality: Performance Assessment measures how well an AIM performs its function, using a metric that depends on the nature of the function, e.g., how well a Visual Change Detection (VCD) AIM can detect the change of a visual scene.
2. Bias: Performance Assessment measures the preference given by an AIM to certain elements, using a metric that depends on a bias related to certain attributes of the AIM. For instance, a Visual Instance Identification (VII) AIM tends to have a higher correct identification of visual objects that have a certain shape.
3. Legal compliance: Performance Assessment measures how well an AIM performs its function, using a metric that assesses its accordance with a certain legal standard.
4. Ethical compliance: the Performance Assessment of an AIM can measure the compliance of an AIM to a target ethical standard.

MPAI-OSD V1.3 provides Performance Assessment methods for some AIMs.