Position Papers of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 23–30
DOI: 10.15439/2025F4971
ISSN 2300-5963 ACSIS, Vol. 44

Practical security of evidence for regulated artificial intelligence modules

Marko Esche, Levin Ho, Martin Nischwitz
0009-0001-7110-5665
0000-0002-7560-5456
0009-0002-5488-5820
Physikalisch-Technische Bundesanstalt,
Abbestraße 2-12, 10587 Berlin, Germany
Email: marko.esche@ptb.de, levin.ho@ptb.de, martin.nischwitz@ptb.de

Sabine Glesner
0009-0003-6946-3257
Technische Universität Berlin
Straße des 17. Juni 135
Email: sabine.glesner@tu-berlin.de

***Abstract*—Artificial intelligence has recently led to numerous new applications in various industry sectors. Whenever artificial intelligence modules are used in a black-box setting, quality monitoring of such modules remains an open challenge. This implies that users of such modules cannot predict the modules' performance following software updates or retraining. Specifically for regulated devices, keeping track of an artificial intelligence module's behavior and compliance with requirements is crucial. To this end, existing methods for monitoring the functional behavior of software are investigated and evaluated regarding their practical usability in this paper. Based on the results of the investigation, a proposal for a new adaptive quality monitoring scheme for artificial intelligence modules is made.**

I. Introduction

ARTIFICIAL intelligence (AI) modules are becoming increasingly common in private and public sectors. [1] Such applications range from classical machine learning algorithms, e.g. object detection and classification [2], to generative AI systems such as chatGPT [3] which can be seen as a first step towards a general-purpose AI. In the European Union (EU) the AI Act [4] provides terms, definitions and requirements for AI models and systems. It also introduces general transparency requirements to be met by any AI system and establishes a conformity assessment framework for so-called "high-risk scenarios" and general-purpose AI systems, aiming to ensure that all high-risk AI systems and general-purpose AI systems used within the EU ensure a minimum level of customer protection. While these high-risk applications are limited to law enforcement, health etc., the AI Act also stresses that AI modules within regulated products still need to pass conformity assessment according to the relevant directives. [4] One such regulated sector is legal metrology, covering measuring instruments used for commercial transactions or official measurements. There exist already multiple signs indicating that the integration of AI modules into regulated measuring instruments is imminent. In the EU, the Measuring Instruments Directive (MID) Annex I lays down essential requirements for regulated measuring instruments which apply to ten different types of instruments, e.g., length measuring devices, taximeters, across all EU member states. As an example, requirement 8.3 imposes the following, "[...] Software identification shall be easily provided by the measuring instrument. Evidence of an intervention shall be available for a reasonable period of time." WELMEC Guide 7.2 [5] provides harmonized technical guidance regarding the interpretation of the software-related essential requirements for all EU members. From essential requirement 8.3, two deductions can be drawn: Firstly, since an AI module must be interpreted as software, it shall be possible to identify a specific version of the AI module for control purposes. Secondly, any change to the AI module (including its parameters) shall be traceable to provide evidence of an intervention. This requirement aims to ensure continued compliance of the instrument by providing traceability of modifications. Typically, users of AI modules do not have access to the actual executable code of the module but either use it as a remote service or as part of a device with limited interaction capabilities. Given the potential adaptability of AI modules, existing static solutions for providing traceability of modifications, e.g., hashes over executable files [5], will likely reach their limits quickly, if the frequency of modifications increases. In particular, any approach should not be dependent on manual interference or classification of changes. Therefore, a solution should be able to automatically differentiate between simple bugfixes that do not affect a module's intended behavior and more fundamental modifications such as the addition of new functionality. To this end, classical and alternative approaches for identification and traceability of software modules are investigated in this paper, resulting in a new proposal which aims to be generally applicable for all types of AI modules subject to similar requirements. The resulting use case can be summarized as follows: An AI module is used for data processing purposes, e.g., within a cyber-physical system for classification of input data. Any change to the module shall be traceable, either by providing evidence of an intervention or through demonstrating continued compliance with predefined requirements. The remainder of the paper is structured as follows: Section II provides an overview of different existing methods to identify and monitor modifications in software modules. In Section III one selected method will be extended to deal with the potential behavior of AI modules. The proposal will be practically tested and compared with the

Topical area: Advanced Artificial Intelligence in Applications

current state of the art in Section IV. Section V concludes the paper and provides suggestions for further work.

II. Related Work

Numerous methods exist for identifying software modules and providing traceability of changes in real-world scenarios. These range from simple version control systems to automata learning approaches. While version control can be seen as a static approach that does not allow automatic distinction between minor bugfixes and major changes, automata learning can be used to quantify the scope of modifications for certain types of automata. Subsection II-A will cover existing active automata learning methods that require bi-directional communication with a system under learning (SUL). Methods that operate passively and are also applicable in black-box scenarios will be addressed in Subsection II-B. Classical static approaches of identifying software through hashes over binaries will be revisited in Subsection II-C. In Subsection II-D the differences between the approaches are discussed and a candidate for the described use case is selected.

A. Active automata learning

In the 1987 paper "Learning Regular Sets from Queries and Counterexamples" [6], Angluin outlined the now well-known L^* algorithm. The algorithm uses a learner L which sends consecutive queries to a teacher T to construct and update an automata representation of the SUL. The teacher acts as an interface to the SUL and provides necessary abstraction for the learner. Thus, the approach is only applicable in a white-box or gray-box scenario. Since the learner paradigm plays a central role in the developed method in Section III, the main aspects of the L^* algorithm will be reiterated here. L^* was originally developed for learning the behavior of deterministic finite automata (DFAs), which are 5-tuples $(Q, \Sigma, \delta, q_0, F)$ [7]:

$$\begin{aligned} &Q \text{ is a finite non-empty set of states.} \\ &\Sigma \text{ is a finite input alphabet.} \\ &\delta : Q \times \Sigma \to Q \text{ is the transition function.} \\ &q_0 \in Q \text{ is the initial state of the DFA.} \\ &F \subset Q \text{ is the set of accepting states of the DFA.} \end{aligned} \tag{1}$$

During execution of L^*, the learner L sends membership and equivalence queries to the teacher T. If the accepted language of the SUL A is $L(A)$ and $Aut(A)$ is the set of all DFAs with the same input alphabet Σ, the two queries are defined in the following manner:

- Membership query $Q_M : \Sigma^* \to \{0, 1\}$ where the learner asks the teacher to test if a given string x is part of the language $L(A)$. If $x \in L(A)$, the teacher's response is 1, 0 otherwise.
- Equivalence query $Q_E : Aut(\Sigma) \to \Sigma^* \cup \{\text{true}\}$ where the learner asks the teacher to test equivalence between the SUL A and the current learned automaton representation $A' \in Aut(\Sigma)$.

With the aim of constructing an internal observation table for storing the results of the queries in systematic fashion, the learner issues membership queries until an initial model A' is obtained. The learner then performs an equivalence query for A'. The teacher subsequently either acknowledges correspondence between the learned and the true model or supplies a counterexample $c \in \Sigma^*$ fulfilling the condition

$$c \in L(A) \wedge c \notin L(A') \text{ or } c \notin L(A) \wedge c \in L(A').$$

Windmüller, Neubauer, Steffen, Howar and Bauer showed in [8] and [9] that the L^* algorithm can be adapted to large-scale software applications with varying degrees of complexity. However, they also noted that this requires a lot of adaptation by the developer within the teacher to properly abstract the behavior of the SUL to the needs of the learner. Furthermore, while AI modules can be interpreted as deterministic software modules, their output for arbitrary, unknown input generally cannot be predicted due to the complexity of implementations such as Artificial Neural Networks (ANN). [10]

B. Passive automata learning

If white-box access to the SUL is not possible, passive automata learning algorithms can be used as an alternative for learning the behavior of a SUL. These algorithms usually obtain a set of traces $S = \{S_+, S_-\}$, where S_+ are positive traces describing the correct behavior of the SUL and S_- are traces that contain known errors that contradict the behavior of the SUL [11]. A trace itself is a list of input symbols and subsequently reached states represented by the corresponding observed output symbols. The Regular Positive Negative Inference algorithm (RPNI) [11] can be used to learn a model of an SUL from such traces. While the approach can correctly learn the behavior of complex software systems given sufficient time [12], it lacks the possibility of mapping the potentially arbitrary response of an AI module for unknown input to a DFA. A common limitation for typical automata learning algorithms is the inefficiency of handling so-called deeply nested states, especially when the SUL is highly complex. On the other hand, although passive automata learning algorithms allow efficient learning of an SUL's behavior from a provided set of traces, testing the conformance of the learned model, e.g., checking the completeness and/or the consistency, could result in relatively long time because the tests are usually performed via repeatedly checking different input combinations [12]. While [12] puts the emphasis on inferring a complete automaton of a black-box system, this paper focuses on monitoring a task learned by an AI module without prior knowledge.

C. Integrity protection through hashes

As mentioned in Section I, [5] provides harmonized technical guidelines for all EU member states regarding the application of securing and protection requirements for software in regulated measuring instruments. However, the Guide currently relies on static methods such as cryptographic hashes for providing evidence of interventions for all types of software modules. If an AI module contains a learning facility for adaptation in the field, any change would result in a new

hash over binary code and necessitate a new conformity assessment. To remedy this problem, the International Organization for Legal Metrology (OIML) published a revised version of the OIML Document D31 [13] in 2023. This document is, in theory, applicable to all regulated measuring instruments world-wide and addresses provision of evidence of intervention in a meaningful manner for AI modules. D31 treats AI modules as software modules with a predefined structure that are controlled by a (potentially very large) set of parameters that can be modified by means of a learning facility. Clause 6.2.3.1 of D31:2023, for instance, provides an example of a large ANN that uses version control for identification of the network topology and a cryptographic hash over the network weights in predefined order for tracing changes to the ANN's behavior. To avoid having to re-certify the adapted ANN, D31 recommends providing fingerprints of the network weights and storing the actual configuration of the weights externally. While this method ensures that an AI module within a regulated measuring instrument can continue to be used even after a learning cycle, the size of the externally stored weight configuration will increase linearly over time. It also places the burden of monitoring compliance with legal requirements on inspectors and market surveillance authorities. To check a specific result of the AI module, the authority has to verify if the ANN together with the stored network weight configuration is suitable to produce measurement results within the legally required limits. While the method does not require access to the development environment of an AI module, it does require access to the weights of the ANN and thus only works in a white-box scenario.

D. Admissibility as evidence

It should be noted that the development processes for 'classical' software and AI modules are very similar. For classical software, the developer constructs an initial concept based on known requirements and available data. This concept is implemented and tested to ensure compliance with requirements [14]. Figure 1 provides a visual representation of this workflow for a measuring instrument. During use, changes to the software can be made via updates. In the use case investigated here, any change to the software shall produce evidence of intervention. Consequently, any software update must either result in a broken physical seal or a permanent logbook entry with the same legal consequences.

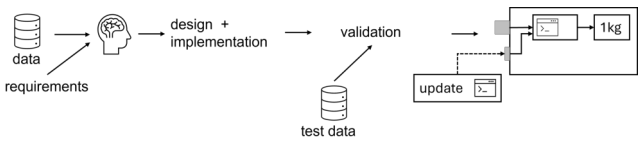


Fig. 1: Classical workflow for software development: Taking into account predefined requirements, the developer uses the available data to create an initial implementation, which is then validated using independent test data. During use, software modules can be modified by means of an update.

TABLE I: Overview of the different monitoring approaches for AI modules and their properties.

approach	black-box support	memory requirements	suitable for AI
active automata learning	no	size of learning table	no
passive automata learning	yes	size of learning table + size of saved traces	no
hash comparison	no	size of AI model per update	yes
remote quality control	yes	size of AI monitor model	yes

Development of an AI module follows a similar pattern [15]. Initially, the developer selects an AI model (such as a decision tree or a deep ANN) taking into account known requirements. The model is then trained to learn a certain behavior based on the available pre-processed training data. Prior to the release of the model, it is validated using a validation data set which is disjunct from the data used for training, see Figure 2. During use of the model, different scenarios for updating it are possible:

1) Updates can follow a pattern similar to 'classical' software products, where the entire trained model is replaced by a new one.
2) A modification of the AI module can be realized by providing it with new training data and initiating another training procedure during use.
3) A learning algorithm as part of the AI module could use observed real-world data together with an externally provided reference for improving its configuration. In this scenario, all individual serial devices in the field will demonstrate different behavior.

These three variants will be revisited in Section IV.

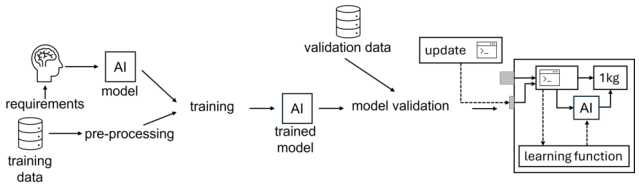


Fig. 2: Workflow for development of AI modules: Based on predefined requirements, the developer selects an initial AI model, which is then trained using pre-processed training data. The trained model is validated using independent test data. During use, the AI module can either be replaced during an update or re-trained using external or internal reference data.

Development of 'classical' software and AI modules both use a two-step approach that first produces an initial implementation which is then validated using an independent test dataset not included during development of the initial implementation. Also, changes to the final implementation can occur during use. While a software update can affect both types of modules, the source for modifications can also be an

internal learning procedure for an AI module. Regardless of the fact if such a learning procedure uses a supervised or an unsupervised training method, the main distinction compared to 'classical' software thus becomes the ability to dynamically change, potentially without an external trigger. It is this property that makes continuous monitoring of AI modules a necessity. Table I provides a summary of the aforementioned different approaches for providing security of evidence for interventions. As can be seen from rows 1, 2, and 3, only the hash comparison between different binary images of an AI module can actually be used for providing evidence of intervention while potentially needing linearly increasing chunks of memory per modification of the AI module. At the same time, the passive automata learning approach also supports black-box scenarios combined with a significantly smaller memory footprint, but is not originally able to monitor the underlying massively complex models behind an AI module. Thus, an extension of the passive automata learning approach to adaptive AI modules will be investigated in the subsequent section to derive an optimized solution with smaller memory usage and black-box applicability.

III. Remote quality control approach

As has been demonstrated in [12], passive automata learning algorithms can be used to learn the behavior of the software of complex cyber-physical systems in a quasi black-box scenario given sufficient learning time. To this end, the learning algorithm generates prefixes from the observed positive and negative traces S_+ and S_-. In the case of an SUL containing an AI model, such as a deep ANN, the notion of traces (consisting of input symbols and triggered state changes) has to be replaced by observing pairs of input datasets I and corresponding output datasets O. The mapping between the two will be denoted as $\{I, O\}$. As such, the approach developed here shows some similarity with the learner/teacher approach from the L^* algorithm, see Section II-A: The central aim of the approach will be to approximate an SUL's behavior by the learner. To this end, a teacher instance is added to the SUL, transforming its input I and output O into a data format compatible with the learner. Since a specific model structure needs to be selected prior to training of the learner, it shall be initially assumed that an oracle exists that the learner can use to select a specific model type. The consequences of this restriction will be examined and discussed in Section IV-E. It is assumed that a sufficiently complex learner can properly monitor compliance of a given AI module with predefined requirements, thus providing functional identification of software as defined in [12]. Once an initial version of the trained SUL exists, it is used as a teacher for a subsequent second AI learner model. This second model will be referred to as the AI monitor in the following text. It will be assumed that the SUL is not modified during an initial stabilization period t_S. A graphical representation of the dataflow during the stabilization period is shown in Figure 3. During this stabilization period, the AI monitor will be trained using $\{I, O\}$ observed during t_S. For the purpose of the experiments

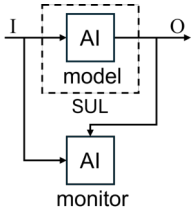


Fig. 3: Stabilization phase of the proposed quality control approach: A second AI module is fed input data I and output data O of the SUL to perform passive learning of the SUL's AI module.

described in Section IV, t_S was selected so that during initial training of the AI monitor/learner, the same amount of input data I was used as for the SUL. It should be noted that this leads to a configuration, where the groundtruth G reference data that corresponds to the input data I of the SUL is not the same as the reference data O used by the AI monitor for initialization and continuous training. After the stabilization period, the trained model of the AI monitor will be used to calculate an individual prediction $p \in P$ for each new input symbol $i \in I$. This will be compared with the corresponding output symbol $o \in O$ of the SUL. i, o and p thus represent individual symbols observed by the AI monitor during normal operation. Over a sliding window of length w matches and mismatches between predictions P and observed output O are monitored. If the resulting prediction accuracy is above a threshold a_{min}, the model of the AI monitor will be updated using i and o. If the threshold is violated, the monitor triggers a compliance warning to all concerned parties. The intended workflow of the method is shown in Figure 4. The restriction regarding the stabilization period is not strictly necessary, but will avoid triggering a large amount of compliance warnings at the beginning of the monitoring process. For the purpose of experimental evaluation, a_{min} was set here to allow a 3% accuracy decrease relative to the initially trained learner. The properties of the approach are shown in row 4 of Table I.

The intention behind the proposed remote quality monitoring approach is to ensure continued compliance of the SUL with requirements while reducing memory consumption and reducing the need for manual interventions. Compared to the hash comparison described in OIML D31 [13] the approach loses some resolution regarding the SUL's behavior since inaccurate predictions of the monitor are tolerated to a certain extent. To check their real-world applicability, both approaches will be described in more detail in Section IV. The section will also perform an in-depth analysis using real-world data.

IV. Experimental evaluation

For the purpose of this evaluation, SULs will be seen as compliant as long as they perform an originally acquired task correctly. Due to the different scenarios of modifying an AI module, this section is divided as follows: Section IV-A describes the algorithms used for evaluation as well as the

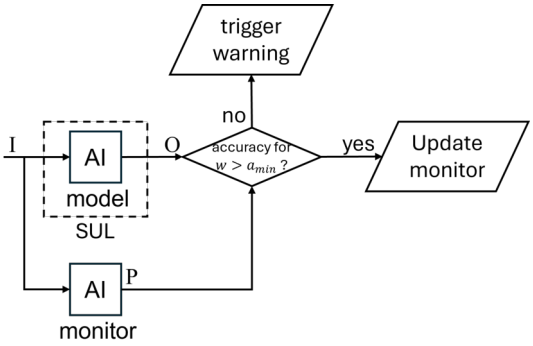
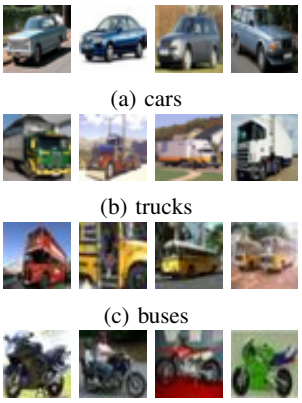


Fig. 4: Monitoring phase of the proposed quality control approach: The second AI module (the AI monitor) continues to monitor the behavior of the SUL and calculates its own prediction accuracy over a sliding window w. If the prediction accuracy drops below a predefined threshold a_{min}, the AI monitor triggers a warning. Otherwise, the newly observed pair of i and o is used to update the AI monitor.

datasets used for experiments. Section IV-B describes how iterative additions of new reference datasets were used to update the SUL and the reaction of the examined methods. Section IV-C extends the use case of providing new global reference datasets to individual reference data for each AI module in use. A replacement of the SUL is addressed in Section IV-D. Section IV-E presents a completely different type of SUL to investigate potential bias in the experiments. Section IV-F discusses the results.

A. Utilized algorithms and datasets

With the aim of testing the applicability of the proposed method of providing evidence of intervention for AI modules, a convolutional neural network (CNN) with six convolutional layers was selected to perform a typical classification task in legal metrology, for which CNNs have already proven their suitability: If speed measurements are performed by law enforcement personell, the used measuring instruments usually incorporate a feature for automatic vehicle classification since different types of vehicles, i.e., cars, trucks, buses, and motorcycles may be subject to different speed limits. As such, this example fits into the EU legislation on measuring instruments and the applicable requirements. It also includes aspects of object detection and recognition, applications for which AI modules have already demonstrated their suitability [2]. This CNN shall serve as the SUL for the remainder of Section IV, except for the use case with CNN3, see Table II. The CNN was implemented using PyTorch and Tensorflow libraries. Training and validation data were obtained from the publicly available CIFAR-10 dataset for images of cars and trucks as well as the CIFAR-100 dataset [16] for images of buses and motorcycles. CIFAR-10 contains 6000 images for 10 different types of objects, whereas the CIFAR-100 dataset contains 100 different classes of objects with 600 images each. The combined dataset used here, thus contained 12000 images of cars and trucks and 1200 images of buses and motorcycles, each image being labeled as belonging to one of the vehicle classes. Exemplary images from each class are shown in Figure 5. For the purpose



(a) cars

(b) trucks

(c) buses

(d) motorcycles

Fig. 5: Exemplary images from the combined CIFAR-10 and CIFAR-100 datasets [16] for the classes ”car/automobile“, ”truck“, ”motorcycle“.

of this experimental evaluation, the input data I are thus the individual training images, whereas the groundtruth G are the corresponding classification labels assigned to those images.

The remote quality control approach described in Section III was similarly implemented using Python Tensorflow and PyTorch libraries. Initially, the internal structure of the AI monitor's model was chosen to be a CNN identical to the one of the SUL. Unless mentioned otherwise, this internal model was used for all subsequent experiments. The AI monitor was used to iteratively learn the behavior of the SUL for all use cases described in Sections IV-B to IV-E using the available input and output data $\{I, O\}$ of the SUL only. Stabilization time t_S and lower accuracy bound a_{min} were configured as described in Section III. The hash comparison algorithm described in [13] was implemented as a reference method in the following manner: The network structure given above was automatically translated to an identifier string that uses a single character to denote the type of the layer, e.g. 'c' for 'convolutional', 'l' for 'linear', followed by the output shape for each layer, where individual dimensions are separated by slash symbols. This results in the string `c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1l128/4` for the described CNN. Similarly, SHA256 hashes were calulated over the exported parameter sets of the CNN for the initial trained network. Both are given in row 1 of Table II.

B. Iterative provision of new external reference data

For the initial model, 500 training images + 100 validation images from each of the four classes were used for its first training. To determine how the two evaluated algorithms react to a modified more precise SUL, CNN1 was updated after its initial training. The update was performed iteratively by adding 1000 images to the training dataset with each new

TABLE II: Identifiers produced by the hash comparison described in [13] for the various SULs used for experimental evaluation.

no.	AI model	classes	topology ID	parameter hash digest
1	**original CNN1**	car, truck, bus, motorcycle	c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1c128/64/64/1/l128/4	53899e22277658092a576cb65f29e443 64107c39080c687cbc29e9afe392d69f
2	**CNN1 update 1**	car, truck, bus, motorcycle	c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1c128/64/64/1/l128/4	9aaba668b4cb6f99d608e54b7fa051de 1cf3465994dc2722888ec2db9df1855d
3	**CNN1 update 2**	car, truck, bus, motorcycle	c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1c128/64/64/1/l128/4	b8e431a95e7f1b335f1779a61854fd8f dee9de46e416aa56864de3c045175422
4	**CNN1 update 3**	car, truck, bus, motorcycle	c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1c128/64/64/1/l128/4	09143768afae4bff1b814de3777a7185 72445e807728b75a1e056362d059eb4f
5	**CNN1 update 4**	car, truck, bus, motorcycle	c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1c128/64/64/1/l128/4	4c8f57d109b4e4deabf8f53b1be4f730 126fc5f505e43fa67572eb1e4cc7eed7
7	**CNN2**	car, truck, bus, motorcycle	c128/512/64/1c128/512/64/1c128/256/64/1 c128/256/64/1c128/128/64/1c128/128/64/1/l128/4	c5e303dcb9d30729cc5e65ef44054283 1650ff6cc9490132ff4ce90744347ca3
8	**CNN3**	horse, bird, car, truck	c128/256/64/1c128/256/64/1c128/128/64/1 c128/128/64/1c128/64/64/1c128/64/64/1/l128/4	3cd2c07c569770c7f5bdbae50007bc7f 08f00102f6f0678e56832d7e44790dce
9	**ResNet50**	car, truck, bus, motorcycle	c128/256/1/1c128/256/1/1c128/256/1/1 c128/512/1/1c128/512/1/1c128/512/1/1 c128/512/1/1c128/512/1/1c128/1024/1/1...	98318830b1eccd5a51422c5c5cf11c4f 7428e0f664dc0cda43e8a76732980ac1

dataset pair $\{I_\Delta, G_\Delta\}$ being used to retrain the SUL. The resulting classification accuracy for each incremental modification is given in Figure 6. At the same time, the AI monitor was fed in-between classification output of the SUL for smaller chunks of added images consisting of 250 images each, to continually observe the SULs behavior within a sliding window. Figure 7 illustrates this continuous monitoring for an excerpt of Figure 6 between the original CNN1 and its first update. In the excerpt, the AI-monitor's accuracy changes slightly with each new batch of images, as these are unknown to both SUL and monitor. Nevertheless, a_{min} is not violated within this excerpt. It should be noted that the accuracy of the SUL is measured between its classification output O and the groundtruth G. For the AI monitor however, accuracy is measured between its own prediction P and the SUL's classification output O. Thus, the AI monitor's accuracy can, theoretically, be higher than that of the SUL. This would indicate that the AI monitor has learned the SUL's behavior correctly, even when the SUL itself performs false classifications. As can be seen from Figure 6, the AI monitor continually achieves an accuracy above the threshold a_{min}. Consequently, the AI monitor's model is updated to include new classification output from the SUL for the observed chunks of images.

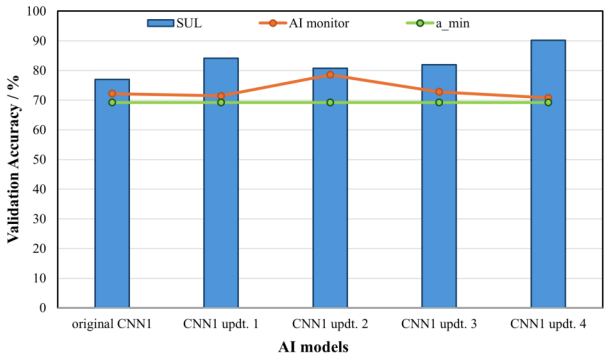


Fig. 6: Timeseries of the AI monitor's classification accuracy for iterative updates. The AI monitor's accuracy is measured relative to the classification output O of the SUL. The SUL's accuracy is measured relative to the groundtruth G.

Fig. 7: Timeseries of the AI monitor's classification accuracy for chunks of new images between updates of the SUL (CNN1 to CNN1 update 1). The AI monitor's accuracy is measured relative to the classification output O of the SUL. The SUL's accuracy is measured relative to the groundtruth G.

C. Iterative provision of individual reference data

From the point of view of both the D31 method [13] and the remote quality control method, no distinction can be made between AI modules being updated with new common external reference data and provision of individual reference data per AI module. Thus, all results from Section IV-B also apply for this use case. The main distinction lies in the required memory capacity needed for storing the configuration of the CNNs for later manual inspection: If each AI module can change independently, such traceability data also must be provided for each module, thus increasing memory requirements linearly with the number of AI modules used in the field.

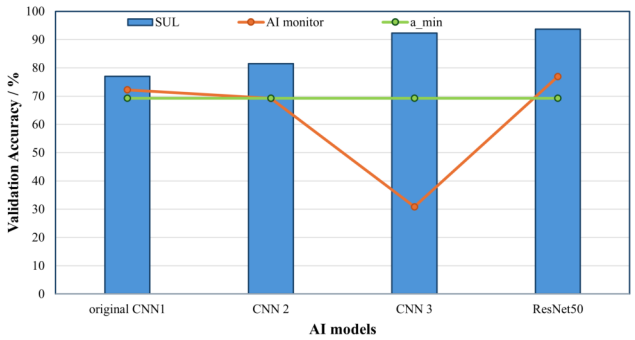


Fig. 8: Timeseries of the AI monitor's accuracy of classification results for the replaced SUL. The AI monitor's accuracy is measured relative to the classification output O of the SUL. The SUL's accuracy is measured against the groundtruth G.

D. Replacement of the CNN

To test the provision of evidence of an intervention by the two algorithms for a modified SUL, the CNN1 was replaced by a different CNN2 with six convolutional layers and one linear layer, where each layer has twice the number of neurons. CNN2 was trained using the aforementioned combined vehicle training dataset from CIFAR-10 and CIFAR-100. Consequently, the topology identification string changed to `c128/512/64/1c128/512/64/1c128/256/64/1 c128/256/64/1c128/128/64/1c128/128/64/1 l128/4`, and all weights within the CNN of the SUL were abruptly changed, too. The new identifications provided by the D31 method are shown in row 7 of Table II. The classification accuracy of the AI monitor for CNN1 and CNN2 is shown in Figure 8. Even though CNN2 uses a different network topology, the AI monitor still achieves an accuracy similar to the one for the original SUL CNN1.

In order to compare observations made for the classification task performed by CNN1 and CNN2 with a common reference, a third CNN3 was trained to detect birds, horses and cars from the CIFAR-10 dataset, i.e., with different groundtruth data. The identifications obtained by the D31 method for this new SUL CNN3 are shown in row 8 of Table II. Similarly, the reaction of the AI monitor was tested by providing it input data and the SUL's output classification for CNN3 for the image classification task. The resulting change in classification accuracy is shown in Figure 8. As anticipated, the accuracy drops to below 40%, indicating that there is a mismatch between behavior of SUL and AI monitor. How modifications of t_S can influence the detection rate, will be discussed in Section IV-F. At the same time, the output of the D31 method does indicate a modification, but fails to illustrate the impact of the modifications compliance with requirements.

E. Comparison with a reference classifier

To avoid bias because of the known network structure of CNN1 during experiments, a generic KERAS ImageClassifier with the preset "resnet_50_imagenet" (afterwards referred to as ResNet50) was used as an additional reference. It consists of a total of 49 convolutional layers and one output layer. The resulting data are shown in row 9 of Table II, where the topology ID produced by the D31 method had to be truncated since it is too long to be repeated here. In practice, this corresponds to a scenario where the internal topology of the SUL is unknown and the oracle introduced in Section III is no longer needed. However, the learner still needs to know the general task performed by the SUL, i.e., the classification of images into predefined classes. Thus, the oracle from Section III is reduced to providing a general task description which can easily be done by a human expert with knowledge of the data types for I and O. The corresponding prediction accuracy of the AI monitor after stabilization was 76.95%. As can be seen from Figure 8, the AI monitor successfully learned the behavior of ResNet50 despite its unknown internal structure.

F. Analysis

From Table II, it should be clear that due to the avalanche effect in cryptographic hashes [17], even slight modifications of the network weights result in a completely different hash digest. Differentiating between incremental improvements and the complete replacement of the SUL from the hash digest alone thus becomes impossible. Even when combined with the topology ID, the hash digest only provides general information on whether or not any parameter of the ANN was changed. The topology identification on the other hand does allow a human expert to evaluate if the network is still able to perform a certain classification task after a modification. As described in Section II-C, OIML Document D31 solves this issue of differentiating between two identical networks trained for different tasks by imposing storage requirements on the parameter set used for each instance of the ANN. In the particular setup, the CNNs 1 and 3 have 1,153,114 parameters due to the number of connections between successive layers and the ouput layers respectively. CNN2 similarly uses 4,591,642 parameters. For later manual inspection of a specific instance of the CNNs, CNN1 and CNN3 thus require 92.83 MB storage capacity, whereas CNN2 requires 370.92 MB. The remote monitoring approach proposed here, however, does not detect minor gradual changes of an AI module, see Figure 6. In fact, a complete replacement of the SUL with another CNN performing an identical task does not result in any compliance warnings. Only the abrupt modification of the SUL, in case it is replaced with a CNN that portrays a different behavior for similar input data, is automatically detected, see Figure 8. However, if a replacement is done gradually, without violating a_{min} within t_S, the AI monitor would similarly fail to trigger any warnings. The detection of compliance violations is influenced significantly by the allowed stabilization time t_S for training an AI monitor and by the allowed lower accuracy bound a_{min}. While both can be fine-tuned for a specific task, an inadequate selection of one or the other can result in either too many or too few compliance. While the default values chosen in Section IV-A may have worked for the examined use cases, they are not guaranteed

to work as well in other settings. In practice, t_S could be set to a default value, such as one day and then decreased iteratively, if too many changes occur. Similarly, $a_{\min}$ could be initialized with a value of 90% and then be reduced to fit the particular application. The applicability of the method developed here does not only depend on the parameter settings, but also on the size of available observed data. Minimum requirements for these data for different use cases and AI models still remain to be formally specified. A full oracle, which can tell the topology of a SUL, is not always needed, see Section IV-E. In fact, one could envision a scenario where, in Legal Metrology in particular, the usage of certain types of ANNs is prescribed for specific tasks, thus eliminating the need for an oracle altogether. At the moment, it appears unfeasible to use a general-purpose AI model and attempt to learn an SUL's behavior using an extremely large observation dataset $\{I, O\}$. It does, however appear plausible that the setup and AI monitor used here, can also be applied to different kinds of image or more general signal classification tasks. Nevertheless, there is the potential bias in the findings shown here since image classification is particularly suitable for the remote monitoring approach proposed here. Therefore, further work will include extending the method developed here to audio classification and other common AI tasks.

V. Summary

As stated in Section IV the D31 method described in [13] appears to be realizable for providing evidence of intervention for AI modules in practice if a white-box scenario is given. However, the method has very high memory consumption if traceability of individual changes is required. Nevertheless, this paper can be seen as a first proof of concept of the method from [13]. In addition, the AI monitor introduced here was able to correctly identify compliance violations as required, for example, for regulated measuring instruments in the EU. [18] As demonstrated in Section IV-F, the AI monitor can be seen as another form of the functional identification of a software module introduced in [12]. It would allow inspectors or market surveillance authorities to remotely monitor the compliance of AI modules in quasi black-box settings in regulated industry sectors, such as legal metrology. Moreover, the method could equally be applied by users of AI services in other industry sectors to determine if a service quality is reduced without prior warning. Of course, the monitoring approach requires resources similar to those for operating the actual SUL. The obvious advantage of the approach, however, is that the monitoring does not have to be conducted permanently. The monitoring can also be carried out at a later point when resources are available as long as observed data can be buffered for an intermediate timespan. Further work will address the tradeoff between resources and algorithmic complexity as well as the impact of t_S and $a_{\min}$ on monitoring accuracy. It will also include application of the remote monitoring method to other use cases and investigations into minimal observable data requirements for different use cases. Such investigations would also provide some insight into the applicability of general-purpose AI modules as AI monitors for a larger range of tasks. Additionally, benchmark tests comparing actual memory usage between the remote monitoring system and traditional techniques will be performed.

References

[1] K. McElheran, J. F. Li, E. Brynjolfsson, Z. Kroff, E. Dinlersoz, L. Foster, and N. Zolas, "Ai adoption in america: Who, what, and where," *Journal of Economics & Management Strategy*, vol. 33, no. 2, pp. 375–415, 2024.

[2] Y. Wei, N. Song, L. Ke, M.-C. Chang, and S. Lyu, "Street object detection / tracking for ai city traffic analysis," in *2017 IEEE SmartWorld*, 2017. doi: 10.1109/UIC-ATC.2017.8397669 pp. 1–5.

[3] G. Rani, J. Singh, and A. Khanna, "Comparative analysis of generative ai models," in *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*, 2023. doi: 10.1109/ICAICCIT60255.2023.10465941 pp. 760–765.

[4] EC, "Regulation (eu) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence," European Union, Council of the European Union; European Parliament, Directive, February 2024.

[5] "WELMEC 7.2 Software Guide," European cooperation in legal metrology, WELMEC Secretariat, Braunschweig, Standard, Mar. 2022.

[6] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987. doi: 10.1016/0890-5401(87)90052-6

[7] M. Sipser, *Introduction to the theory of computation*, 2nd ed. Boston, Massachusetts: Thomson, 2006. ISBN 0-534-95097-3

[8] S. Windmüller, J. Neubauer, B. Steffen, F. Howar, and O. Bauer, "Active continuous quality control," in *Proceedings of the International Symposium on Component-Based Software Engineering*. ACM, Jun. 2013. doi: 10.1145/2465449.2465469 pp. 111–120.

[9] J. Neubauer, S. Windmüller, and B. Steffen, "Risk-based testing via active continuous quality control," *International Journal on Software Tools for Technology Transfer*, vol. 16, pp. 569–591, 2014. doi: 10.1007/s10009-014-0321-6

[10] C. Oliva and L. F. Lago-Fernández, "On the interpretation of recurrent neural networks as finite state machines," in *Proceedings of ICANN 2019: Theoretical Neural Computation: 28th International Conference on Artificial Neural Networks,*, vol. I 28. Munich, Germany: Springer International Publishing, September 2019, pp. 312–323.

[11] B. Aichernig, E. Muskardin, and A. Pferscher, "Active vs. Passive: A Comparison of Automata Learning Paradigms for Network Protocols," in *Formal Methods for Autonomous Systems and Automated and verifiable Software system development*, ser. Electronic Proceedings in Theoretical Computer Science, EPTCS, vol. 371. National ICT Australia Ltd, September 2022. doi: 10.4204/EPTCS.371.1 pp. 1–19.

[12] L. C. X. Ho, M. Esche, M. Nischwitz, and S. Glesner, "Black-box conformity tests on regulated measuring instruments: A machine learning approach," in *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference*. Chemnitz, Germany: IEEE, 05 2025, to be published.

[13] "OIML D 31: General requirements for software controlled measuring instruments," International Organisation of Legel Metrology, Paris, France, Tech. Rep., 2023.

[14] M. Gorrod, *The software development lifecycle*. London: Palgrave Macmillan UK, 2004, pp. 93–120. ISBN 978-0-230-51029-6. [Online]. Available: https://doi.org/10.1057/9780230510296_4

[15] S. Mohseni, N. Zarei, and E. D. Ragan, "A multidisciplinary survey and framework for design and evaluation of explainable ai systems," *ACM Trans. Interact. Intell. Syst.*, vol. 11, no. 3–4, Sep. 2021. doi: 10.1145/3387166. [Online]. Available: https://doi.org/10.1145/3387166

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[17] D. Upadhyay, N. Gaikwad, M. Zaman, and S. Sampalli, "Investigating the avalanche effect of various cryptographically secure hash functions and hash-based applications," *IEEE Access*, vol. 10, pp. 112 472–112 486, 2022. doi: 10.1109/ACCESS.2022.3215778

[18] EC, "Directive 2014/32/EU of the European Parliament and of the Council of 26 February 2014 on the harmonisation of the laws of the Member States relating to the making available on the market of measuring instruments," European Union, Council of the European Union; European Parliament, Directive, February 2014.