Received 19 September 2022; revised 23 November 2022; accepted 3 January 2023. Date of publication 13 January 2023; date of current version 27 January 2023.

Digital Object Identifier 10.1109/OJITS.2023.3236531

Insufficiency-Driven DNN Error Detection in the Context of SOTIF on Traffic Sign Recognition Use Case

LUKAS HACKER [1] AND JÖRG SEEWIG [2]

[1] RD ADAS Software Platform, Mercedes-Benz AG, 71059 Sindelfingen, Germany

[2] Institute of Measurement and Sensor Technology, Technical University of Kaiserslautern, 67663 Kaiserslautern, Germany

CORRESPONDING AUTHOR: L. HACKER (e-mail: lukas.l.hacker@mercedes-benz.com).

ABSTRACT Deep Neural Networks (DNNs) are used in various domains and industry fields with great success due to their ability to learn complex tasks from high-dimensional data. However, the data-driven approach within deep learning results in various DNN-specific insufficiencies (e.g., robustness limitations, overconfidence, lack of interpretability), which makes the usage in safety-critical applications, like automated driving, challenging. An important safety strategy to address these limitations is the detection of DNN errors (e.g., false positives) during runtime. In this work, we present a general error detection approach for DNNs, which combines diverse monitoring methods to address different safety-related DNN insufficiencies simultaneously. To ensure consistency with the automotive safety domain, we take into account established concepts of the automotive safety standard ISO 21448 (SOTIF). We apply our error detection method on the safety-related use case of traffic sign recognition by using self-created 3D driving scenarios. In doing so, we consider different types of DNN errors related to in distribution, out of distribution, and adversarial data. We demonstrate that our approach is able to handle all these error types. Furthermore, we show the performance benefit of our method compared to a baseline DNN and to state of the art DNN monitoring methods.

INDEX TERMS Artificial intelligence, automated driving, deep learning, deep neural networks, safety of the intended functionality, traffic sign recognition.

I. INTRODUCTION

IN HIGHLY automated vehicles, artificial intelligence (AI) and deep learning (DL) are crucial parts for complex tasks, like environmental perception [1]. In deep learning, a data-driven approach is taken, in which deep neural networks (DNNs) learn these tasks from high-dimensional data [2]. After training and testing activities, the DNN black box models are integrated into the overall automated driving (AD) system architecture, as shown in the upper part of Fig. 1, and perform safety-critical tasks, like object detection and image classification. However, ensuring safe behavior of these data-driven algorithms is challenging due to their insufficiencies and the infinite number of scenarios in open-world. For example, distributional shift and adversarial attacks can force the DNN to predict high confidence scores on incorrect outputs [2]. In the context of safety of the intended functionality (SOTIF) and the corresponding automotive safety standard ISO 21448 [3], the open-world problem is addressed by systematically minimizing the area of unknown scenarios with iterative process steps of analysis, functional modifications, verification, and validation. For the remaining area of unknown, potentially unsafe scenarios, appropriate monitoring methods have to be provided during runtime. However, methods for monitoring traditional software, which are recommended in the functional safety standard ISO 26262 [4] (e.g., range checks), are not applicable for DL-based algorithms, like DNNs [2]. For this reason, various DNN runtime monitoring methods have been published in recent years, which we discuss in the related work in Section II-C. However, these methods do not take into account various DNN insufficiencies simultaneously. Rather, they focus on individual error root causes (i.e., triggering

The review of this article was arranged by Associate Editor Johannes Betz.

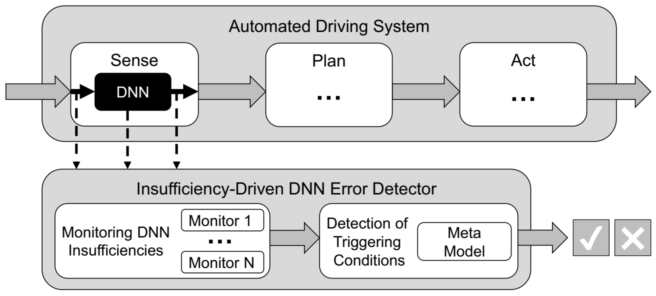

Automated Driving System
Sense
DNN
Plan
...
Act
...
Insufficiency-Driven DNN Error Detector
Monitoring DNN Insufficiencies
Monitor 1
...
Monitor N
Detection of Triggering Conditions
Meta Model

FIGURE 1. Proposed architecture for an insufficiency-driven detection of DNN errors (using the example of DNN-based perception).

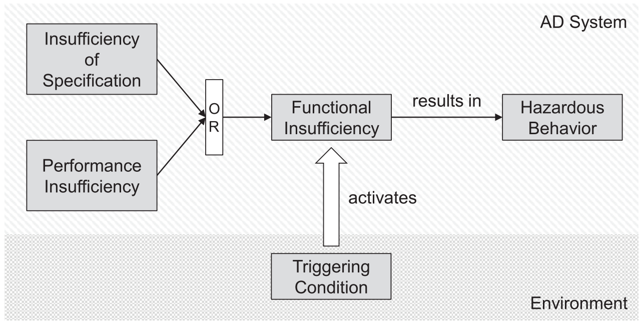


FIGURE 2. SOTIF cause and effect chain in the context of automated driving based on ISO 21448 [3].

conditions), like out of distribution inputs and adversarial attacks. Furthermore, to the best of our knowledge, established SOTIF concepts like the cause and effect chain, which is shown in Fig. 2, have not yet been taken into account in the context of DNN runtime monitoring. To address these gaps, we propose in Section III an insufficiency-driven approach for monitoring DNNs, as shown in the lower part of Fig. 1. In Section IV, we describe the implementation and application of our proposed error detection method on the safety-related automated driving use case of traffic sign recognition (TSR). Afterwards, we present and discuss our results by comparing our error detection approach to a baseline DNN model and to state of the art monitoring methods in Section V. Finally, we summarize our work and draw conclusions in Section VI.

II. RELATED WORK

In this section, we discuss theoretical background and relevant research for modeling, simulating, and testing of our proposed DNN error detection approach. First, we introduce SOTIF and the corresponding safety standard ISO 21448 in Section II-A. Afterwards, we discuss testing in automated driving and deep learning including relevant datasets and toolsets in Section II-B. Finally, we summarize methods for monitoring DNNs during runtime in Section II-C.

A. SAFETY OF THE INTENDED FUNCTIONALITY

According to the automotive safety standard ISO 21448, SOTIF is defined as “the absence of unreasonable risk due to a hazard caused by functional insufficiencies” [3] of the intended functionality. In the context of automated driving, functional insufficiencies can be insufficiencies of specification (e.g., gaps in specification of operational design domain) or performance insufficiencies (e.g., technical limitations of sensors and perception algorithms). Certain conditions of a scenario (i.e., triggering conditions) can activate these functional insufficiencies and might provoke a hazardous behavior of the AD system [3], [5]. Fig. 2 shows the corresponding cause and effect chain and illustrates the connection between functional insufficiency, triggering condition, and hazardous behavior. To ensure the SOTIF, ISO 21448 defines an iterative development process including phases of analysis, design, verification, validation, and monitoring [2], [3]. Thereby it addresses the open-world problem for complex systems (e.g., unknown, unsafe scenarios in automated driving), which are not sufficiently covered within the established functional safety standard ISO 26262 [2], [3]. In doing so, ISO 21448 augments the process of ISO 26262 [2]. Furthermore, ISO 21448 has reasonably foreseeable misuse of the intended functionality in scope and proposes measures, like human machine interface (HMI) improvement and driver monitoring implementation [2], [3]. However, ISO 21448 does not go into detail about insufficiencies in context of deep learning algorithms and related measures to address the safety of DNNs. This is were various research activities come in, like Willers et al. [5], who identified DNN-specific safety concerns and corresponding mitigation measures, or Burton et al. [6], who proposed an approach for the construction of confidence arguments in the context of DNN performance evaluation.

B. TESTING IN AUTOMATED DRIVING AND DEEP LEARNING

Virtual testing is becoming increasingly important in automated driving due to well-known advantages like repeatability, scaling, safety and costs [7], [8]. Especially with respect to the development of DL algorithms (e.g., DNNs for perception tasks), testing in virtual environments is an important field [9]. Recent survey papers [7], [9] summarize relevant datasets for testing of DL-based AD systems and virtual testing environments for open- and closed-loop simulations, like MATLAB Automated Driving Toolbox. Using the example of traffic sign recognition task, well-known datasets like GTSDB (German Traffic Sign Detection Benchmark) [10] and LISA (Laboratory for Intelligent and Safe Automobiles) [11] are publicly available. Furthermore, Zhu et al. [12] introduced the Tsinghua-Tencent 100K dataset with Chinese traffic signs in bad weather conditions. To test the AD system in relevant scenarios and to derive potential failure cases, Ghodsi et al. [13] and Wang et al. [14] recently published methods for generation of safety-critical scenarios. However, the data-driven approach in deep learning results in major differences compared to traditional software, which requires DL-specific testing methods on software level. Recent survey papers [15], [16] highlight these differences and summarize research of appropriate

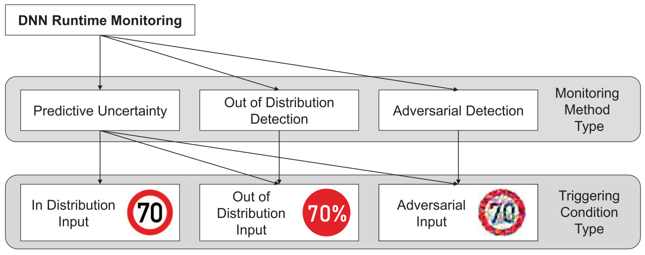


FIGURE 3. Runtime monitoring methods address different types of triggering conditions (using the example of traffic sign recognition).

testing methods to address DL-specific insufficiencies, like robustness limitations and black-box characteristics. For example, Pei et al. [17] introduced the white-box testing framework DeepXplore to measure neuron coverage and to uncover thousands of incorrect DNN corner case behaviors [16]. Furthermore, Tian et al. [18] proposed DeepTest as a systematic testing tool, which supports derivation of erroneous DNN behaviors in the context of automated driving. Additionally, various explainability methods offer the possibility in testing to explain the decisions of a DL algorithm and to analyze and understand its errors [19]. For example, explainability methods like GradCam [20] and Occlusion Sensitivity [21] provide saliency maps to highlight relevant features within the input image.

C. RUNTIME MONITORING METHODS FOR DEEP NEURAL NETWORKS

In addition to DL-specific testing activities, it is of great practical importance to monitor the tested DL algorithm during runtime [22]. To discuss state of the art monitoring methods in deep learning, we define a DNN as a high-dimensional function f_θ, which maps the input data x to output values in form of, e.g., probability scores for different classes. This mapping depends on the DNN's learned parameters θ from training data distribution (i.e., in distribution) [23]. However, the DNN probability scores are often overconfident and do not guarantee error prediction [2], [19]. Therefore, various methods for DNN runtime monitoring have been published in recent years, which can be assigned to three main literature fields [22]:

- Predictive Uncertainty
- Out of Distribution Detection
- Adversarial Detection

The methods of each literature field differ in their error detection approach and address different types of error root causes (i.e., triggering conditions) [19], [22]. Fig. 3 shows the relation between monitoring methods and triggering condition types. For example, out of distribution (OOD) detection methods focus on the detection of input data outside the training data distribution [19]. They deal with a binary classification problem, whether the input is in distribution (ID) or OOD to prevent a DNN error on input data that have never been seen during training time [23]. Adversarial detection methods address intentionally modified input data to fool the DNN (i.e., adversarial attacks [24]), which are often very close to the training data distribution with minimal targeted modifications [22]. Similar to OOD methods, they classify binary, if the current input is an adversarial attack or not. In contrast, predictive uncertainty methods provide additional uncertainty values to reflect the level of confidence for the current DNN prediction [23]. Because predictive uncertainty methods improve uncertainty quantification in general, they cannot be assigned to only one triggering condition type, as shown in Fig. 3. In the following, we describe some well-known representatives of each literature field. More detailed overviews of DNN runtime monitoring methods can be found in recent survey papers [19], [22].

1) PREDICTIVE UNCERTAINTY METHODS

Well-known predictive uncertainty methods are based on Bayes theory, like Monte Carlo (MC) dropout and Bayesian neural networks [25]. These methods estimate an output probability distribution by multiple sampling of non-deterministic forward passes during runtime $f_{\theta,i}(x)$ [23]. For an MC-dropout-based monitoring, dropout layers have to be added to the network architecture, which randomly switch off single neurons and introduce a regularization effect during training time. However, Gal and Ghahramani [25] argued that if dropout is applied during runtime as well, it can be interpreted as an approximation of a Bayesian neural network with Bernoulli distributions as a prior [19]. Mean μ and variance σ^2 of the output distribution are calculated by using the DNN predictions $f_{\theta,i}(x)$ over n forward passes through the dropout layers, as shown in (1). The variance of the predicted class can be seen as a score S_{unc} of the predictive uncertainty.

$$\mu = \frac{1}{n} \cdot \sum_{i=0}^{n-1} f_{\theta,i}(x); \quad \sigma^2 = \frac{1}{n} \cdot \sum_{i=0}^{n-1} \left(f_{\theta,i}(x) - \mu\right)^2 \tag{1}$$

Furthermore, Lakshminarayanan et al. [26] proposed Deep Ensembles, which is another sampling-based approach for estimation of the DNN output probability distribution. In contrast to the MC-dropout method, the sampling procedure is not performed through the dropout layers, but through multiple DNNs, which differ in their underlying training process (i.e., their weights θ) or additionally in their architecture. The output probability distribution over n samples can be calculated the same way as with the MC-dropout method by (1). One drawback of these sampling-based methods are their computational overload due to the fact that multiple forward passes are required to achieve an accurate approximation of the output probability distribution [19].

2) OUT OF DISTRIBUTION DETECTION METHODS

Instead of estimating a probability distribution for the current DNN prediction, OOD detection methods deal with a binary classification problem, whether the current input is inside or outside the training data distribution. For example, Cheng et al. [27] proposed a method for monitoring neuron activation of the hidden DNN layers. The neuron activation patterns in the last hidden layer are stored with Binary Decision Diagrams (BDD) [28] during design phase [23]. These patterns are built up with Boolean variables, which indicate if the monitored neurons are active or not. To detect anomalies, the neuron activation pattern is compared to the stored patterns during runtime by measuring the Hamming distance. Furthermore, Hendrycks et al. [29] proposed Outlier Exposure, where the training process of the DNN is augmented with OOD samples from auxiliary datasets (e.g., 80 Million Tiny Images [30]). The DNN is trained on the original dataset with original labels $D_{ID}(x, y)$ and additionally an OOD dataset $D_{OOD}(x')$. The authors modified the loss function with an additional term, which forces the DNN to output a high entropy score for the added OOD samples. For classification tasks, Lee et al. [31] introduced a modified loss function, which includes the Kullback-Leibler (KL) divergence [32] to force the DNN f_θ to be closer at the uniform distribution $\mathcal{U}$ for OOD inputs x' with a penalty parameter $\lambda > 0$, as shown in (2). Consequently, the OOD score S_{OOD} for the current DNN prediction can be estimated by calculating the entropy of the DNN prediction, according to (3).

$$\min_{\theta} \Big[\mathbb{E}_{D_{ID}(x,y)}\big[-\log f_\theta(x)\big] + \lambda \cdot \mathbb{E}_{D_{OOD}(x')}\big[KL\big(\mathcal{U}(y)\|f_\theta(x')\big)\big]\Big] \tag{2}$$

$$S_{OOD} = H\big(f_\theta(x)\big). \tag{3}$$

3) ADVERSARIAL DETECTION METHODS

To address adversarial perturbations within input data during runtime, Meng and Chen [33] introduced the autoencoder-based method MagNet, which is trained on the original dataset. The input samples are processed through the autoencoder, which compresses and reconstructs the input. Afterwards, adversarial inputs are detected by a high reconstruction loss between original and reconstructed input. Additionally, the reconstructed input is processed through the DNN and shifts within the DNN prediction between the original and reconstructed input are measured for adversarial detection as well. Furthermore, Xu et al. [34] proposed Feature Squeezing, which is based on DNN predictions on squeezed input images. Therefore, a reduced color depth image and a spatial smoothed image are generated from the original input. The DNN predictions on the two squeezed images are compared with the DNN prediction on the original input image. If the difference S_{adv} between the outputs exceeds a threshold, the input is classified as an adversarial attack. To measure the difference between the original prediction $f_\theta(x)$ and the squeezed prediction $f_\theta(x^{squeezed})$, the ℓ_1-norm can be taken into account [34], as shown in (4).

$$S_{adv} = \left\|f_\theta(x) - f_\theta\left(x^{squeezed}\right)\right\|_1 \tag{4}$$

III. MODELING OF AN INSUFFICIENCY-DRIVEN ERROR DETECTOR FOR DEEP NEURAL NETWORKS

In this section, we describe the development of a general model for DNN error detection in the context of safety of the intended functionality, as shown in Fig. 1. Therefore, we take into account the SOTIF cause and effect chain from Fig. 2 and the literature fields of DNN runtime monitoring from Section II-C. First, we create an error root cause model for DNNs with DNN-specific insufficiencies and triggering conditions in Section III-A. Afterwards, we derive general DNN monitor categories and link them to the insufficiencies in Section III-B. Finally, we combine the monitor categories with a meta model, based on stack learning technique, in Section III-C. This enables a general approach for monitoring DNNs, which addresses safety-related DNN insufficiencies and considers various types of DNN errors related to in distribution, out of distribution, and adversarial input data.

A. CREATION OF AN ERROR ROOT CAUSE MODEL FOR DEEP NEURAL NETWORKS

Willers et al. [5] recently defined nine DNN-specific safety concerns leading to insufficiencies, like data distribution shift to real world, distributional shift over time, incomprehensible behavior, unknown behavior in rare critical situations, unreliable confidence information, brittleness, dependence on labeling quality, inadequate separation of test and training data, and insufficient consideration of safety metrics. Cheng et al. [35] identified core properties and corresponding insufficiencies of DNNs, which have to be addressed in safety context, like robustness, interpretability, completeness, and correctness. Dataset limitations as well as limitations regarding robustness, explainability, and uncertainty are also highlighted by Houben et al. [36].

In the following, we summarize these results in six general DNN insufficiencies and categorize them by using the SOTIF terms of performance insufficiency and insufficiency of specification, as shown in Table 1. Furthermore, we make a distinction between data- and model-related insufficiencies. Whereas data-related insufficiencies refer to the underlying datasets, which are used in DNN development process for training, validation, and testing, model-related insufficiencies refer to limitations of the trained DNN model. Considering the DNN-specific triggering condition types and the categorized insufficiencies, we create an error root cause model in the context of deep learning. Fig. 4 shows the error root cause model and illustrates the relation between DNN-specific functional insufficiencies, triggering conditions, and DNN errors. In distribution, out of distribution, and adversarial inputs can activate corresponding DNN insufficiencies and lead to a DNN error (i.e., false positive or false negative prediction). If the DNN error is not detected on software level, it can contribute to a hazardous behavior on vehicle level. To prevent DNN error propagation, we propose an error detection approach on software level, which takes into account the DNN insufficiencies to detect all types of DNN-specific triggering conditions.

TABLE 1. Functional insufficiencies of DNNs in the context of SOTIF.

	Insufficiency of Specification	Performance Insufficiency
Data-related Insufficiency	**Lack of Data Completeness:** Shift between dataset and real world distribution; Rare scenarios **Lack of Data Quality:** Labeling quality; Data balance; Data relevance; Data bias	/
Model-related Insufficiency	**Lack of Model Interpretability:** Non-specifiable tasks; Non-explainable DNN black box models	**Lack of Model Correctness:** 100% correctness not possible in open-world **Lack of Model Robustness:** Sensitive to small input perturbations **Lack of Model Uncertainty Representation:** Overconfidence; No reliable probability information

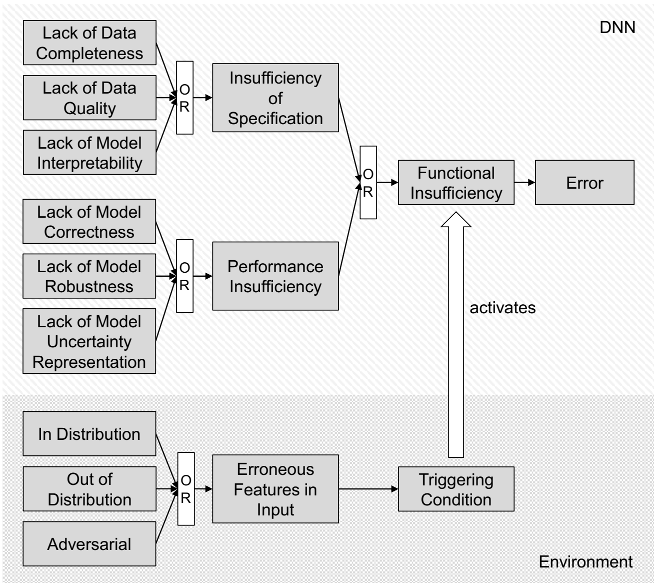

DNN
Lack of Data Completeness
Lack of Data Quality
Lack of Model Interpretability
OR
Insufficiency of Specification
Lack of Model Correctness
Lack of Model Robustness
Lack of Model Uncertainty Representation
OR
Performance Insufficiency
OR
Functional Insufficiency
Error
activates
In Distribution
Out of Distribution
Adversarial
OR
Erroneous Features in Input
Triggering Condition
Environment

FIGURE 4. Error root cause model for DNNs: Triggering conditions in input data activate functional insufficiencies of the DNN and lead to an error.

B. DERIVATION OF GENERAL MONITOR CATEGORIES TO ADDRESS DNN INSUFFICIENCIES DURING RUNTIME

To enable a general error detection approach, which addresses the safety-related DNN insufficiencies in Table 1, appropriate monitoring methods have to be provided for each insufficiency. Therefore, we derive five general monitor categories, as shown in the upper part of Fig. 5. Each monitor category should provide an error score $S_i \in [0, 1]$, which reflects the error probability with respect to the observed insufficiency. In the following, we describe these monitor categories and make suggestions for their implementation.

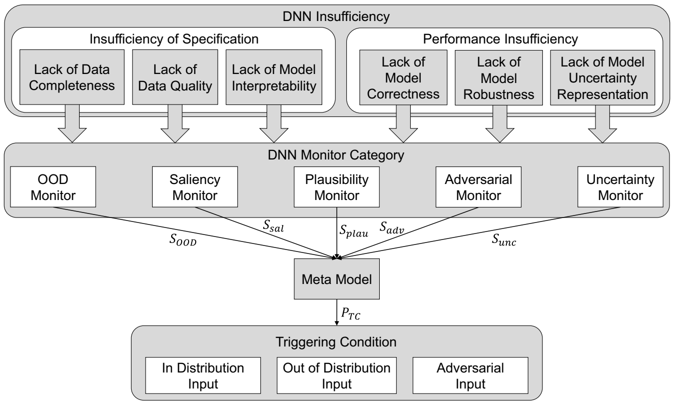


FIGURE 5. DNN monitor categories address safety-related insufficiencies of DNNs for detection of different triggering condition types.

We introduce **OOD-Monitor** category, to address insufficiencies regarding data completeness. For implementation of the OOD monitor, well-known methods from the OOD detection literature field, can be used to calculate an OOD score S_{OOD}. To address insufficiencies regarding DNN interpretability and dataset quality (e.g., data bias), we introduce **Saliency Monitor** category, which includes runtime application of explainability methods for saliency map generation. Similarity metrics [37] can be used to estimate a saliency score S_{sal}, by quantifying how much the DNN is focusing on the right location within input image or the right object artifact. Additional knowledge about the complex task can be used to build up a rule set for cross-checking the DNN prediction, which we cover in **Plausibility Monitor** category. These plausibility checks address the lack of DNN correctness in general. For example, within camera-based computer vision, comparisons over different sensor modalities (e.g., radar, lidar) or non-DL-based software (e.g., conventional computer vision techniques, like *Histogram of Oriented Gradients* [38], [39]) are applicable for estimation of a plausibility score S_{plau} to quantify conformity with expected object attributes like size, color, and shape. With **Adversarial Monitor** category, we cover robustness limitations of the DNN with respect to small adversarial perturbations by using methods from the adversarial detection literature field to calculate an adversarial score S_{adv}. Finally, we cover insufficiencies regarding DNN uncertainty representation with **Uncertainty Monitor** category, which can be implemented with well-known predictive uncertainty methods to estimate an uncertainty score S_{unc}.

C. RUNTIME DETECTION OF TRIGGERING CONDITIONS WITH A COMBINING META MODEL APPROACH

To consider the DNN insufficiencies during runtime, we combine the results of the individual monitor categories from

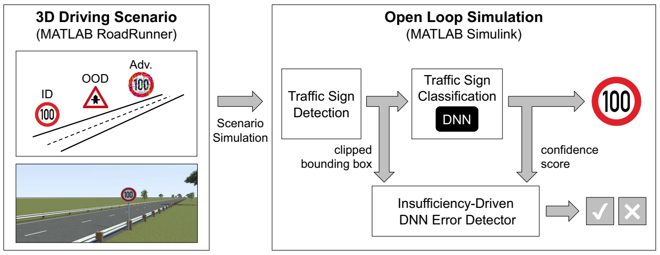


FIGURE 6. **Experimental set-up in MATLAB Simulink and MATLAB RoadRunner environment for training and testing of the insufficiency-driven DNN error detector.**

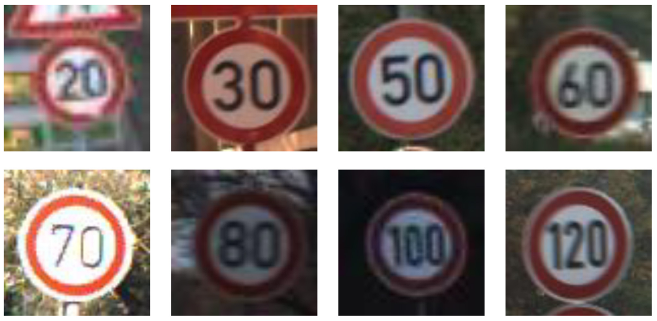


FIGURE 7. **Images of the 8 speed sign classes in GTSRB dataset [43].**

Section III-B, as shown in the lower part of Fig. 5. In doing so, we introduce a meta model $\mathcal{M}$ to estimate the probability of a triggering condition P_{TC} depending on the monitor category outputs S_i, according to (5).

$$P_{TC} = \mathcal{M}(S_{OOD}, S_{sal}, S_{plau}, S_{adv}, S_{unc}) \tag{5}$$

We propose to optimize the meta model with stack learning technique on the monitor category outputs by using a training dataset that contains all types of triggering conditions (i.e., in distribution, out of distribution, and adversarial inputs). This enables the meta model to exploit the different strengths of the individual monitor categories and to cover the problem space of DNN insufficiencies and triggering conditions. Afterwards, the optimized meta model can be used for runtime detection of triggering conditions.

Various machine learning architectures are applicable for implementation of the meta model $\mathcal{M}$, like logistic regression (LR), naive Bayes (NB), k-nearest neighbors (KNN), random forest (RF), gradient-boosted trees (GBT), support vector machines (SVM), and feed forward neural networks (FFNN). We implement these architectures in Section IV on TSR use case and compare them in Section V by considering the trade-off between interpretability and detection performance. For example, a highly interpretable meta model approach, based on logistic regression, can be implemented according to (6) and (7).

$$P_{TC,LR} = \frac{1}{1 + e^{-z}} \tag{6}$$

$$z = \beta_0 + \beta_1 \cdot S_{OOD} + \beta_2 \cdot S_{sal} + \beta_3 \cdot S_{plau} + \beta_4 \cdot S_{adv} + \beta_5 \cdot S_{unc} \tag{7}$$

However, to utilize stack learning for training of the meta model $\mathcal{M}$, the monitor categories have to be sufficiently uncorrelated in their predictions, which has to be verified with appropriate metrics like variance inflation factor (VIF) [40].

IV. EXPERIMENTAL SET-UP AND IMPLEMENTATION ON AUTOMATED DRIVING USE CASE OF TRAFFIC SIGN RECOGNITION

In this section, we implement the proposed error detection method from Section III on the safety-related automated driving use case of traffic sign recognition by using MATLAB Simulink and MATLAB RoadRunner toolset, as shown in Fig. 6. Therefore, we adapt the general monitor categories and the meta model on the TSR use case and implement them in model-based MATLAB Simulink language. The overall model is then embedded in a MATLAB Simulink open loop simulation, which contains a self-developed TSR function consisting of DNN-based traffic sign detection and classification components. We create diverse 3D traffic sign scenarios with MATLAB RoadRunner and feed the simulated camera data into the open loop simulation. Our error detection approach is implemented to supervise the baseline DNN for the traffic sign classification task. In doing so, we treat the DNN as a black box with a non-modifiable architecture and inaccessible inner states to increase flexibility in application. Therefore, we focus on the DNN input (clipped bounding box with traffic sign image from traffic sign detection) and the DNN output (confidence scores for traffic sign classes) as input data for the error detector.

A. TRAFFIC SIGN DETECTION

For the traffic sign detection task, we train a *YOLO v2* [41] object detection algorithm on the German Traffic Sign Detection Benchmark (GTSDB) dataset [10]. The dataset comprises 900 images with a resolution of 1360×1024 pixels in RGB format and contains 1206 German traffic signs with corresponding class and bounding box labels. Training, validation, and test dataset (300 images per dataset) are taken from the full dataset. The feature extraction network of the YOLO v2 algorithm is based on a *MobileNetV2* [42] architecture with input size $700 \times 700 \times 3$ for RGB images. After 100 epochs of training with *Adam* optimization algorithm (learning rate $= 0.001$, mini batch size $= 2$) a mean average precision of 91% (intersection over union $= 0.5$) on the test dataset is reached.

B. TRAFFIC SIGN CLASSIFICATION (BASELINE DNN)

For the classification task, we train a DNN with 5-layer convolutional neural network (CNN) architecture (3 convolutional layers, 2 fully connected layers, softmax output) on the speed signs within the German Traffic Sign Recognition Benchmark (GTSRB) [43]. Therefore, 16.946 images with 8 classes of speed signs are extracted of the full GTSRB dataset (51.840 images), as shown in Fig. 7. The extracted speed sign dataset is then divided into an approximately 50%/25%/25% training, validation, and test split. Afterwards, we resize the images to 32×32 RGB resolution and feed them into the CNN, whereas features within the convolutional layers are extracted by 3×3 *VGG*-like (Visual Geometry Group) [44] kernels. Each convolutional layer is followed by ReLU activation, batch normalization, and max pooling (2×2) layers. After 50 epochs training with Adam optimization algorithm (learning rate = 0.005, mini batch size = 128) an accuracy of 95% is reached on the test dataset. Depending on the maximum softmax output $f_{\theta,max}$ of the baseline DNN, we calculate the triggering condition probability $P_{TC,DNN}$ on input data x according to (8).

$$P_{TC,DNN} = 1 - f_{\theta,max}(x) \tag{8}$$

C. 3D TRAINING AND TEST SCENARIOS

To simulate entire prediction tracks of various triggering conditions, we create diverse 3D scenarios for TSR use case in MATLAB RoadRunner environment. In doing so, we design a straight street with different traffic signs, which are separated in a difference of 100 m to each other. The trajectory of the ego vehicle with a mounted camera in the front grill (height relative to road level: 0.3 m) is defined with a constant speed of 80 km/h by using MATLAB Automated Driving Toolbox. To generate the camera data, we simulate the 3D scenario with the ego vehicle trajectory. Afterwards, we export the simulated camera data with resolution of 1024×900 in RGB format in the MATLAB Simulink open loop simulation. The camera data are then resized to 700×700 RGB resolution and fed into the traffic sign detection algorithm. For training and testing of the meta model, we compile two independent camera simulations based on the described core scenario. To produce the required dissimilarity between training and test data, the core scenario is slightly modified with respect to traffic signs, weather, and background features (e.g., trees).

We model **Out of Distribution Inputs** with traffic sign classes of the GTSRB dataset that are not in the training data distribution of the baseline DNN (i.e., no speed signs). We consider these classes as out of distribution due to the fact, that they have never been seen during training time. To generate **Adversarial Inputs**, we apply Projected Gradient Descent (PGD) [45] method, which is an iterative variant of Fast Gradient Sign Method (FGSM) [24], on in distribution traffic sign data (i.e., speed signs in GTSRB). According to (9), we calculate a FGSM perturbation $\Delta(x, x_{adv})$ by using the gradient of the training loss function $\mathcal{L}$ of the DNN model f_θ and ground truth y. We perform 100 FGSM steps with step size $\alpha = 1$ (random initialization) and limit the maximal ℓ_∞-perturbation to $\epsilon = 0.2$ (51/255) by clipping values outside the perturbation range, according to (10).

$$x_{adv} = x + \alpha \cdot sign\Big(\nabla_x \mathcal{L}\big(f_\theta(x), y\big)\Big) \tag{9}$$

$$x_{adv}^{t+1} = \text{Clip}_\epsilon \Big\{ \text{FGSM}\big(x_{adv}^t\big) \Big\} \tag{10}$$

Algorithm 1 Occlusion Sensitivity

```
Input:
$x$: original input image
$f_\theta$: DNN
$N$: number of samples
Output:
$S_{map}$: saliency map
1: for $n = 1 : N$
2:     $M_{rand} = CreateRandomMask(x)$
3:     $x_{perturbated} = x \odot M_{rand}$
4:     $y_{pred} = f_\theta(x_{perturbated})$
5:     $S_{map} = S_{map} + M_{rand} \cdot y_{pred}$
6: end
7: $S_{map} = \frac{S_{map}}{N}$
8: Return $S_{map}$
```

In contrast to out of distribution and adversarial inputs, erroneous **In Distribution Inputs** do not have to be modeled explicitly. They are already included in the simulation data and are indicated by false DNN predictions on in distribution traffic signs.

D. IMPLEMENTATION OF THE MONITOR CATEGORIES

For implementation of some monitor categories, different runtime monitoring methods from the literature fields in Section II-C have to be chosen. Note that the decision criterion for these methods is based on popularity and citation frequency. A complete benchmark on the use case would go beyond the scope of this work. Rather, the benefit of their combination within the meta model approach, which address different DNN insufficiencies simultaneously during runtime, should be shown.

We treat the baseline DNN as a black box and consider this boundary condition in the following specification of the monitor categories. We implement the **Out of Distribution Monitor** with an Outlier Exposure [29] approach. In doing so, we train a similar DNN architecture on ID data (i.e., speed signs in GTSRB dataset) as well as on OOD data with the help of the modified loss function described in (2) by using a penalty parameter of $\lambda = 0.5$. For OOD data, 50.000 entities are randomly sampled from the 80 Million Tiny Images [30] dataset, which contains 80 million diverse color images in 32×32 resolution. We implement the **Saliency Monitor** with Occlusion Sensitivity [21] according to Algorithm 1 to calculate saliency maps during runtime. We estimate the saliency maps by taking $n = 500$ samples per input image with a saliency map resolution of 8×8 pixels. Next, we interpret the saliency maps by checking if the DNN is currently focusing on relevant features within the detected traffic sign. Therefore, we average saliency maps for each traffic sign class on the GTSRB dataset (only true positive predictions) for runtime comparison to the online saliency map estimation by Euclidean distance, as shown in Fig. 8. For the **Plausibility Monitor**, we define a color- and shape-based plausibility check, which is implemented with a red

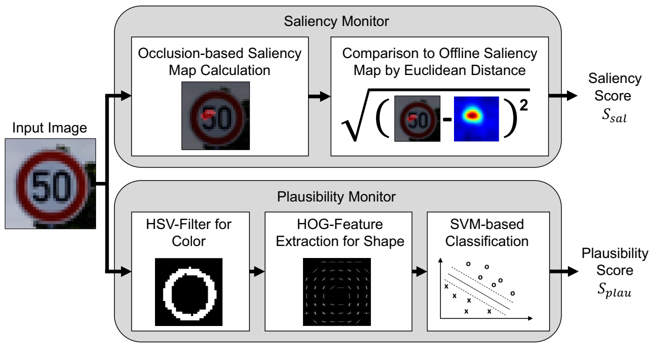

Saliency Monitor
Occlusion-based Saliency Map Calculation
Comparison to Offline Saliency Map by Euclidean Distance
Saliency Score S_{sal}
Input Image
Plausibility Monitor
HSV-Filter for Color
HOG-Feature Extraction for Shape
SVM-based Classification
Plausibility Score S_{plau}

FIGURE 8. Saliency monitor calculates saliency maps for current DNN prediction and interprets the results, whereas plausibility monitor performs color- and shape-based plausibility checks.

color detection in hue-saturation-value (HSV) color space and a circular shape detection with Histogram of Oriented Gradients (HOG) method. We fine-tune the HSV interval for red color detection on GTSRB dataset. The HOG features are classified with a support vector machine [39], as shown in Fig. 8. Training of the HOG model and the SVM is performed on the GTSRB dataset, whereas training images are labeled with circular shape vs. no circular shape. The **Adversarial Monitor** is implemented with Feature Squeezing [34], whereas reducing color depth of the input image from original 8-bit (per RGB channel) to 5-bit and performing spatial smoothing with a median filter (2×2 sliding window). To estimate the difference between DNN prediction on original and squeezed images, we apply the ℓ_1-norm on the DNN predictions, as shown in (4). We implement the **Uncertainty Monitor** with a Deep Ensembles approach [26]. In doing so, ten identical DNN classification architectures (Section IV-B) are trained on the GTSRB dataset with different initial weights. During runtime, the uncertainty is estimated by calculating the variance of the ten DNN predictions, according to (1).

E. IMPLEMENTATION OF THE META MODELS

Finally, we implement different meta model architectures for triggering condition detection. We optimize the meta models on training data (from training scenario in Section IV-C) to classify the results of the implemented monitor categories. For optimization of the LR meta model, we use a maximum likelihood estimation with an iteratively reweighted least squares algorithm. The optimized LR meta model is then used for detection of triggering conditions on test data (from test scenario in Section IV-C) by estimating triggering condition probability P_{TC}. In addition to logistic regression, we optimize further classification methods, like naive Bayes (Gaussian kernel), k-nearest neighbors (50 neighbors), random forest (100 trees), gradient-boosted trees with XGBoost algorithm [46] (100 trees, max. tree depth: 9, learning rate: 0.15), support vector machine (linear kernel), and feed forward neural network (2 layers with 15 neuron each) on training data.

TABLE 2. Confusion matrix for detection of triggering conditions.

	Prediction: Triggering Condition	Prediction: Safe Input
Ground Truth: Triggering Condition	TP (True Positive)	FN (False Negative)
Ground Truth: Safe Input	FP (False Positive)	TN (True Negative)

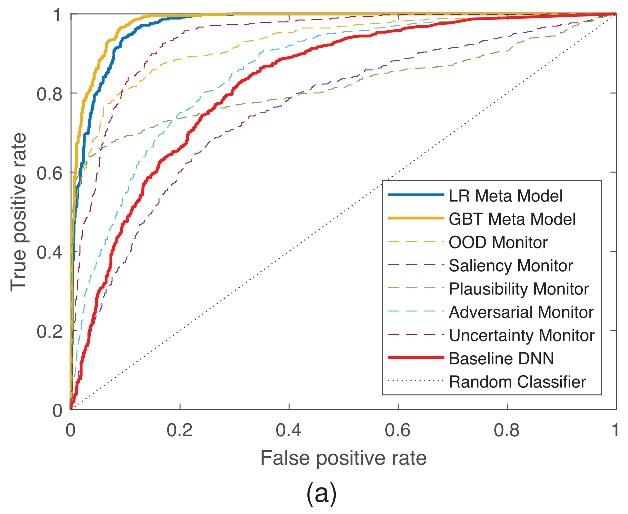
V. RESULTS AND DISCUSSION

In this section, we present and discuss our results related to the error detection performance of baseline DNN, monitor categories, and meta models. Table 2 shows our adapted confusion matrix for the task of triggering condition detection. The diagonal entries (TP, TN) indicate that a prediction of the DNN error detector (i.e., triggering condition vs. safe input) fits to the ground truth, whereas entries on the side diagonal indicate misclassification (FP, FN). We label our training and test data from Section IV-C according to the baseline DNN prediction: False DNN predictions are labeled as triggering conditions, whereas correct DNN predictions are labeled as safe inputs.

First, we analyze the baseline DNN and the individual monitor categories with respect to their detection performance on different triggering condition types in Section V-A. Finally, we compare the different meta model architectures in Section V-B under consideration of the trade-off between interpretability and detection performance.

A. RESULTS AND DISCUSSION FOR MONITOR CATEGORIES

We analyze the performance of the baseline DNN and the monitor categories on the test data from test scenario. The test dataset consists of 72 traffic sign tracks (number of test images $N = 1758$) with equally distributed in distribution, out of distribution, and adversarial inputs ($\sim$ 24 traffic sign images per track). Fig. 9(a) shows a receiver operating characteristic (ROC) diagram, which indicates the detection performance of the baseline DNN (bold red line) and the individual monitor categories (dashed lines). The illustrated ROC curves arise by applying true positive rate over false positive rate with variable detection threshold $t \in [0, 1]$ over the whole test dataset. ROC curves at the top left demonstrate strong detection performance, whereas random classification is characterized by a straight line with a slope of 1. It is shown that every monitor category indicates triggering conditions due to the fact that all ROC curves lie above the random classifier baseline (dotted black line). The baseline DNN's detection performance lies in the lower range of the implemented monitor categories. Highest ROC curves are achieved by out of distribution and uncertainty monitor. Table 3 shows a more detailed view on the detection performance. We analyze each monitor category with respect to the different triggering condition types by taking AUC (area under ROC curve) and TPR$_i$ (true positive rate at i percent false positive rate) as performance metrics, according



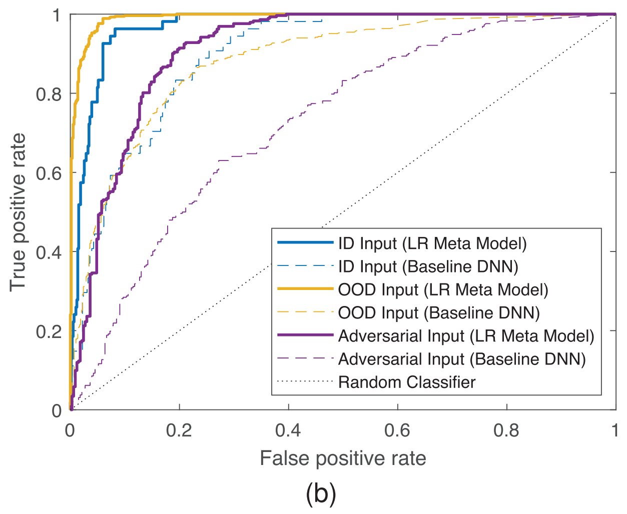


FIGURE 9. Triggering condition detection performance on test dataset, shown by ROC diagram. (a) Overall performance of monitor categories, LR & GBT meta model, and baseline DNN. (b) Performance of LR meta model and baseline DNN on different triggering condition types (ID, OOD, Adversarial).

TABLE 3. Performance evaluation of baseline DNN and monitor categories on test dataset.

	Overall	ID Inputs	OOD Inputs	Adv. Inputs
Baseline DNN	AUC: 0.826 TPR_5: 29.4 TPR_{10}: 47.6 TPR_{20}: 66.0	AUC: 0.896 TPR_5: 44.4 TPR_{10}: 64.8 TPR_{20}: 83.3	AUC: 0.881 TPR_5: 46.8 TPR_{10}: 62.8 TPR_{20}: 82.3	AUC: 0.725 TPR_5: 11.0 TPR_{10}: 28.4 TPR_{20}: 49.0
OOD	AUC: 0.928 TPR_5: 69.3 TPR_{10}: 80.1 TPR_{20}: 88.6	AUC: 0.910 TPR_5: 61.1 TPR_{10}: 70.4 TPR_{20}: 81.5	AUC: 0.985 TPR_5: 92.7 TPR_{10}: 95.5 TPR_{20}: 97.6	AUC: 0.762 TPR_5: 20.9 TPR_{10}: 40.4 TPR_{20}: 60.6
Sal.	AUC: 0.759 TPR_5: 24.0 TPR_{10}: 37.3 TPR_{20}: 60.1	AUC: 0.833 TPR_5: 18.5 TPR_{10}: 48.2 TPR_{20}: 70.4	AUC: 0.822 TPR_5: 31.2 TPR_{10}: 52.8 TPR_{20}: 70.7	AUC: 0.638 TPR_5: 10.6 TPR_{10}: 19.9 TPR_{20}: 41.1
Plau.	AUC: 0.819 TPR_5: 65.4 TPR_{10}: 69.1 TPR_{20}: 73.6	AUC: 0.875 TPR_5: 57.4 TPR_{10}: 74.1 TPR_{20}: 85.2	AUC: 0.968 TPR_5: 92.2 TPR_{10}: 94.2 TPR_{20}: 95.8	AUC: 0.493 TPR_5: 16.4 TPR_{10}: 19.9 TPR_{20}: 26.0
Adv.	AUC: 0.857 TPR_5: 38.1 TPR_{10}: 54.3 TPR_{20}: 75.0	AUC: 0.912 TPR_5: 57.4 TPR_{10}: 68.5 TPR_{20}: 83.3	AUC: 0.891 TPR_5: 48.6 TPR_{10}: 67.4 TPR_{20}: 83.6	AUC: 0.801 TPR_5: 38.4 TPR_{10}: 46.9 TPR_{20}: 65.1
Unc.	AUC: 0.939 TPR_5: 59.9 TPR_{10}: 82.3 TPR_{20}: 94.7	AUC: 0.935 TPR_5: 61.1 TPR_{10}: 83.3 TPR_{20}: 92.6	AUC: 0.965 TPR_5: 80.9 TPR_{10}: 92.9 TPR_{20}: 96.9	AUC: 0.872 TPR_5: 27.7 TPR_{10}: 55.5 TPR_{20}: 76.4

to (11) and (12).

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}; \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{11}$$

$$\text{AUC} = \int_0^1 \text{ROC}(x)\, dx \tag{12}$$

As expected, each monitor category has different strengths and weaknesses with respect to in distribution, out of distribution, and adversarial inputs. Whereas the Deep Ensembles approach of uncertainty monitor performs best on in distribution data, the Outlier Exposure approach of OOD monitor has the highest AUC value and TP rates on input data that lie outside the training data distribution. However, it can be seen that the Deep Ensemble approach improve uncertainty quantification in general due to its high AUC values and TP rates for in distribution, out of distribution, and adversarial data (highest overall AUC). In contrast, OOD monitor achieves good generalization behavior for in distribution and out of distribution inputs, whereas adversarial detection performance is proportionally low. This can be explained by the training process of Outlier Exposure, which only covers OOD inputs instead of both, OOD and adversarial input data. As expected, adversarial monitor, based on Feature Squeezing, achieves good performance on adversarial attacks, especially at low false positive rates (highest TPR_5). Nevertheless, uncertainty monitor has the highest AUC value (also highest TPR_{10} and TPR_{20}) for intentionally modified inputs. Plausibility monitor achieves a high AUC value and high TP rates on out of distribution data. Traffic signs that lie outside the training data distribution can be detected with low false positive rates due to color and shape anomalies. However, these plausibility checks are not suitable to detect adversarial attacks, which is shown by the low TP rates and the AUC value close to the random classifier baseline (AUC ≈ 0.5). The intentional modifications within the adversarial attacks are too small for a simple color-based detection approach. Furthermore, the plausibility checks achieve good performance above the random classifier baseline for in distribution data, despite correct shape and color features of in distribution traffic signs. This can be explained by poorly clipped bounding boxes (i.e., shape anomalies due to insufficiencies of the object detection algorithm), which lead to errors of the baseline DNN. These anomalies are detected by the shape-based plausibility

TABLE 4. Logistic regression analysis on training dataset.

Predictor Variable S_i	Coef β_i	SE Coef $\beta_{SE,i}$	z-Value	p-Value	VIF
Constant	-3.73	0.18	-21.03	0.000	/
OOD (S_{OOD})	3.92	0.37	10.52	0.000	6.15
Sal. (S_{sal})	1.05	0.42	2.48	0.013	1.33
Plau. (S_{plau})	1.18	0.30	3.94	0.000	5.41
Adv. (S_{adv})	3.45	0.24	14.17	0.000	1.30
Unc. (S_{unc})	5.04	0.31	16.22	0.000	2.60

TABLE 5. Performance evaluation of meta models on test dataset.

	Overall	ID Inputs	OOD Inputs	Adv. Inputs
LR	AUC: 0.972 TPR_5: 80.4 TPR_{10}: 93.9 TPR_{20}: 99.1	AUC: 0.970 TPR_5: 81.5 TPR_{10}: 96.3 TPR_{20}: 100	AUC: 0.992 TPR_5: 96.9 TPR_{10}: 99.6 TPR_{20}: 100	AUC: 0.912 TPR_5: 43.2 TPR_{10}: 65.4 TPR_{20}: 91.4
NB	AUC: 0.971 TPR_5: 81.0 TPR_{10}: 91.3 TPR_{20}: 98.3	AUC: 0.963 TPR_5: 83.3 TPR_{10}: 96.3 TPR_{20}: 96.3	AUC: 0.987 TPR_5: 96.7 TPR_{10}: 99.6 TPR_{20}: 99.6	AUC: 0.903 TPR_5: 51.4 TPR_{10}: 62.3 TPR_{20}: 84.6
KNN	AUC: 0.976 TPR_5: 81.9 TPR_{10}: 94.9 TPR_{20}: 99.5	AUC: 0.968 TPR_5: 77.8 TPR_{10}: 98.2 TPR_{20}: 100	AUC: 0.994 TPR_5: 96.0 TPR_{10}: 99.6 TPR_{20}: 100	AUC: 0.925 TPR_5: 55.5 TPR_{10}: 71.2 TPR_{20}: 91.8
RF	AUC: 0.981 TPR_5: 88.6 TPR_{10}: 97.7 TPR_{20}: 99.8	AUC: 0.965 TPR_5: 79.6 TPR_{10}: 94.4 TPR_{20}: 98.2	AUC: 0.993 TPR_5: 96.7 TPR_{10}: 99.5 TPR_{20}: 99.8	AUC: 0.945 TPR_5: 60.3 TPR_{10}: 83.9 TPR_{20}: 97.6
GBT	AUC: 0.982 TPR_5: 86.3 TPR_{10}: 96.9 TPR_{20}: 99.6	AUC: 0.975 TPR_5: 83.3 TPR_{10}: 96.3 TPR_{20}: 100	AUC: 0.995 TPR_5: 96.7 TPR_{10}: 99.6 TPR_{20}: 100	AUC: 0.941 TPR_5: 56.9 TPR_{10}: 83.9 TPR_{20}: 97.6
SVM	AUC: 0.972 TPR_5: 80.6 TPR_{10}: 94.2 TPR_{20}: 98.9	AUC: 0.970 TPR_5: 77.8 TPR_{10}: 96.3 TPR_{20}: 98.2	AUC: 0.991 TPR_5: 96.2 TPR_{10}: 99.6 TPR_{20}: 99.8	AUC: 0.912 TPR_5: 46.2 TPR_{10}: 67.1 TPR_{20}: 91.4
FFNN	AUC: 0.979 TPR_5: 82.9 TPR_{10}: 96.3 TPR_{20}: 99.9	AUC: 0.974 TPR_5: 88.9 TPR_{10}: 96.3 TPR_{20}: 100	AUC: 0.994 TPR_5: 97.6 TPR_{10}: 99.3 TPR_{20}: 100	AUC: 0.932 TPR_5: 53.4 TPR_{10}: 73.3 TPR_{20}: 94.9

checks accordingly. The implemented saliency checks, based on Occlusion Sensitivity, provide indications for all types of triggering conditions (AUC > 0.5). However, the AUC values and TP rates of saliency monitor are in the lower range compared to the other monitor categories. This can be explained by the similarity of the saliency maps for different traffic sign classes, which makes metric-based differentiation difficult.

B. RESULTS AND DISCUSSION FOR META MODELS

We optimize the meta models (LR, NB, KNN, RF, GBT, SVM, FFNN) on training data from training scenario. The training dataset consists of 144 traffic sign tracks (number of training images $N = 3643$) with equally distributed in distribution, out of distribution, and adversarial inputs (~ 25 traffic sign images per track). Since all variance inflation factors (VIF) on training data are lower than 10, we assume that the monitor categories are sufficiently uncorrelated in their predictions to utilize stack learning for meta model training [40]. Table 4 shows the parameters of the LR meta model, which we optimized on the training dataset. All predictor variables are significant ($p_i < 0.05$). The highest coefficients β_i and z-values are given for uncertainty score S_{unc}, OOD score S_{OOD}, and adversarial score S_{adv}. Furthermore, the highest VIFs are related to OOD score and plausibility score S_{plau}, which can be explained by the fact that both monitor categories mainly address out of distribution errors. In Table 5, all meta model architectures are summarized with respect to their detection performance on in distribution, out of distribution, and adversarial test data by using AUC and TPR metrics. It can be seen that the AUC values and true positive rates of all meta models are significantly higher than those of the baseline DNN and the individual monitor categories from Table 3. The GBT meta model, which we optimized with XGBoost algorithm, achieves the highest AUC value on the overall test dataset. However, Fig. 9(a) shows that the ROC curve of the interpretable LR meta model (bold blue line) is close to the ROC curve of the more complex GBT meta model (bold yellow line), which can be explained by the linear relationship of monitor scores S_i and logit transformation of triggering condition probability P_{TC}. The AUC value of GBT meta model is approximately 1% higher than the AUC value of the LR meta model on the overall test dataset. However, both meta model ROC curves lie significantly above the baseline DNN ROC curve (bold red line), as shown in Fig. 9(a). Thus, the meta model approach, based on stack learning, covers for every detection threshold $t \in [0, 1]$ more triggering conditions than the baseline DNN. Table 5 shows that the performance of the other meta model architectures are also within the range of LR and GBT meta model with respect to their TPR and AUC values. For example, RF meta model achieves similar performance as the GBT meta model on in distribution and out of distribution data and exceeds GBT performance on adversarial input data at low false positive rates (highest TPR_5). FFNN meta model achieves highest TPR_{20} rates on the overall test dataset. Its lower AUC value can be explained by a flatter rising ROC curve compared to RF and GBT meta model (lower TPR_5 and TPR_{10}). Whereas KNN meta model performs better than the LR meta model (overall AUC value 0.4% higher), the SVM meta model with linear kernel achieves as expected approximately the same performance as logistic regression for all triggering condition types. The NB meta model has the lowest AUC values of the tested meta model architectures for all triggering condition types, which can be attributed to a stagnation of the ROC curve for in distribution and out of distribution inputs at higher false positive rates ($\text{TPR}_{10} = \text{TPR}_{20}$). Fig. 9(b) shows the ROC curves of LR meta model and baseline DNN with respect

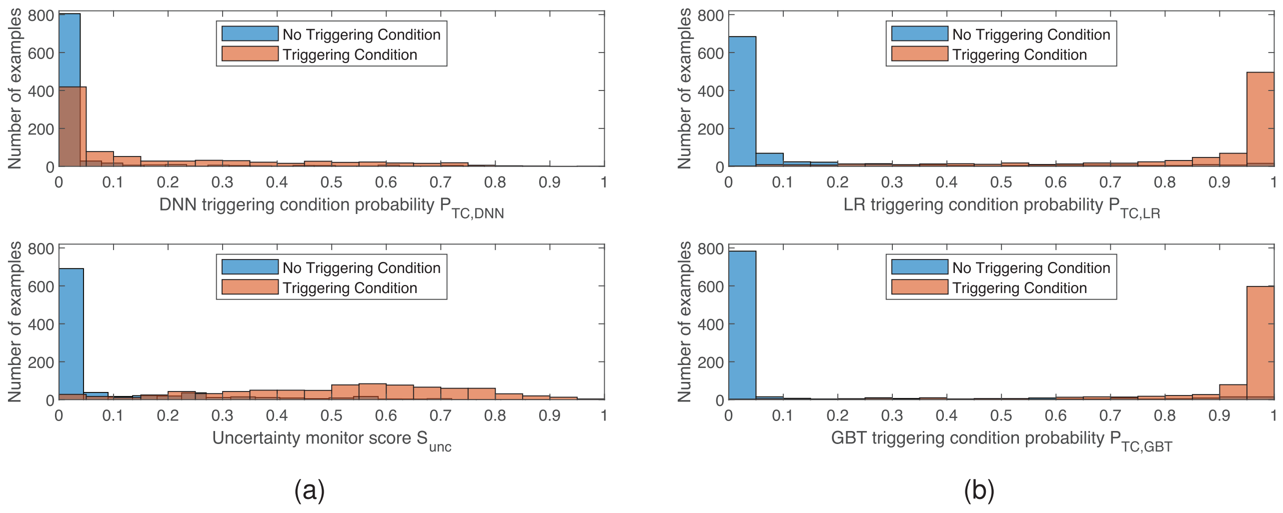


FIGURE 10. Separation ability between safe inputs and triggering conditions on overall test dataset, shown by histogram. (a) Baseline DNN and uncertainty monitor (Deep Ensembles). (b) LR and GBT meta model.

to in distribution, out of distribution and adversarial data. The error detection rates of our approach are for all triggering condition types above the baseline DNN performance. It can be seen that adversarial errors are the most difficult to detect for both, LR meta model and baseline DNN. However, the AUC value of the LR meta model is 20.5% higher on adversarial test data compared to the baseline DNN. In and out of distribution errors are easier to detect than adversarial attacks: At a fixed FP rate of 20%, all DNN errors on in and out of distribution inputs are covered by the LR meta model. In contrast to the baseline DNN, the LR meta model detects out of distribution errors proportionately better than in distribution errors ($AUC_{OOD} > AUC_{ID}$). This can be ascribed to the high AUC values of the OOD detection methods on out of distribution data, which we implemented with Outlier Exposure and color- as well as shape-based plausibility checks. Furthermore, the relation between triggering condition types in training data determines detection performance for in distribution, out of distribution, and adversarial errors, which has to be considered in training data collection. Most DNN errors on training data are related to out of distribution inputs, thus the meta model is optimized more in direction of out of distribution detection performance. Fig. 10 shows the separation ability between safe inputs and triggering conditions of baseline DNN, uncertainty monitor category, and meta models (LR and GBT) on the overall test dataset. In the context of deep learning, triggering conditions are usually hard to distinguish from safe input data. The baseline DNN fails at the prediction of reliable probability scores for triggering conditions, as shown in the upper part of Fig. 10(a). Many triggering conditions ($\approx 50\%$) are wrongly assigned with a low error score ($P_{TC,DNN} < 0.1$) by the baseline DNN. The uncertainty monitor based on Deep Ensembles (monitor category, which achieves highest overall AUC), in the lower part of Figure 10(a), achieves better separation between safe inputs and triggering conditions than the baseline DNN. However, the Deep Ensembles method predicts most triggering conditions ($\approx 82\%$) within a wide probability range between 0.2 and 0.8. In contrast, Fig. 10(b) shows that the meta models are able to clearly distinguish between safe inputs and triggering conditions. Both meta models predict high error scores ($P_{TC,LR} > 0.9$, $P_{TC,GBT} > 0.9$) for most triggering conditions (LR: $\approx 59\%$, GBT: $\approx 71\%$) and predict low error scores ($P_{TC,LR} < 0.1$, $P_{TC,GBT} < 0.1$) for most safe inputs (LR: $\approx 75\%$, GBT: $\approx 85\%$). This can be explained by the fact that the insufficiency-related monitor categories are not highly correlated in their predictions ($VIF_i < 10$). Thus, stack learning combines their different strengths and increases performance in the context of triggering condition detection. However, the increase in detection performance compared to the baseline DNN and to state of the art monitoring methods, like Deep Ensembles, is associated with an increase in runtime computing effort due to the usage of multiple monitoring methods in parallel. Taking into account the low complexity and the comparable performance of the LR meta model approach (AUC_{LR} is 1% lower than AUC_{GBT}), logistic regression represents a good trade-off between performance and interpretability for our implemented safety-related use case. Due to the fact that the meta model performance depends on the use case and the implemented monitor categories, an application-specific assessment is required in the context of method adaption.

VI. CONCLUSION

Reliable detection of environmental error root causes (i.e., triggering conditions) and related DNN errors is a crucial part of automated driving systems. In this work, we proposed a general error detection approach for DNNs, which addresses various safety-related DNN insufficiencies simultaneously during runtime. We developed our approach, taking into account concepts of the automotive safety standard ISO 21448 (SOTIF). Therefore, we created an error root cause model with DNN-specific insufficiencies and triggering conditions. Afterwards, we derived general monitor categories to address the identified DNN insufficiencies during runtime. In doing so, we considered the main literature fields of DNN runtime monitoring. Finally, we introduced a meta model, based on stack learning, that combines the results of the individual monitor categories for an insufficiency-driven error detection. We applied our error detection approach on traffic sign recognition use case in MATLAB Simulink simulation by using self-created 3D scenarios with MATLAB RoadRunner. Our simulation covered all types of triggering conditions, like in distribution inputs, out of distribution inputs, and adversarial attacks. In performance evaluation, we showed that each monitor category has different strengths and weaknesses with respect to the different triggering condition types. Furthermore, we optimized various meta model architectures like logistic regression, naive Bayes, k-nearest neighbors, random forest, support vector machine, gradient boosted trees (XGBoost), and feed forward neural network to predict an error probability based on the monitor category outputs. We showed that the meta models are able to clearly distinguish between safe input data and all types of triggering conditions in contrast to a baseline DNN and to state of the art DNN monitoring methods. Furthermore, the interpretable meta model architecture, based on logistic regression, achieves similar error detection performance as more complex models like gradient boosted trees and feed forward neural network due to linear relationship between the monitor category predictions and the logit transformation of triggering condition probability. However, the performance of the meta models is influenced by the use case and the implemented monitor categories, which requires an application-specific assessment of meta model architectures. Furthermore, care must be taken that the implemented monitor categories are not highly correlated in their predictions to utilize stack learning. Additionally, data balance in training data is important to achieve optimal detection performance with respect to different triggering condition types. Finally, the available computing capacity on target hardware has to be taken into account due to the usage of multiple monitoring methods in parallel. In the future work, we plan to reduce the needed computing effort by minimizing the amount of sample-based approaches. Furthermore, an extension of the 3D driving scenarios with further OOD data (e.g., U.S. and Chinese traffic signs) and adversarial data (e.g., sticker-based adversarial attacks) is planned. We also plan field studies on automated driving target hardware for traffic sign recognition use case to evaluate the runtime performance of our approach on live data.

ACKNOWLEDGMENT

Thanks to Heiko Möckel, Nico Kem, Markus Baumeister, and Marek Kowalczyk for inspiring discussions during the research activities.

REFERENCES

[1] *Road Vehicles—Safety and Cybersecurity for Automated Driving Systems—Design, Verification and Validation*, Standard ISO/TR 4804, 2020.

[2] S. Mohseni, M. Pitale, V. Singh, and Z. Wang, “Practical solutions for machine learning safety in autonomous vehicles,” Dec. 2019, *arXiv:1912.09630*.

[3] *Road Vehicles—Safety of the Intended Functionality*, Standard ISO 21448, 2022.

[4] *Road Vehicles—Functional Safety*, Standard ISO 26262, 2018.

[5] O. Willers, S. Sudholt, S. Raafatnia, and S. Abrecht, “Safety concerns and mitigation approaches regarding the use of deep learning in safety-critical perception tasks,” Jan. 2020. *arXiv:2001.08001*.

[6] S. Burton, L. Gauerhof, B. B. Sethy, I. Habli, and R. Hawkins, “Confidence arguments for evidence of performance in machine learning for highly automated driving functions,” in *Computer Safety, Reliability, and Security*, vol. 11699, 2019, pp. 365–377.

[7] P. Ji, L. Ruan, Y. Xue, L. Xiao, and Q. Dong, “Perspective, survey and trends: Public driving datasets and toolsets for autonomous driving virtual test,” Jun. 2021, *arXiv:2104.00273*.

[8] R. Dona and B. Ciuffo, “Virtual testing of automated driving systems. A survey on validation methods,” *IEEE Access*, vol. 10, pp. 24349–24367, 2022.

[9] Y. Kang, H. Yin, and C. Berger, “Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments,” *IEEE Trans. Intell. Veh.*, vol. 4, no. 2, pp. 171–185, Jun. 2019.

[10] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, “Detection of traffic signs in real-world images: The German traffic sign detection benchmark,” in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Dallas, TX, USA, Aug. 2013, pp. 1–8.

[11] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, “Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey,” *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1484–1497, Dec. 2012.

[12] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu, “Traffic-sign detection and classification in the wild,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 2110–2118.

[13] Z. Ghodsi et al., “Generating and characterizing scenarios for safety testing of autonomous vehicles,” Mar. 2021, *arXiv:2103.07403*.

[14] J. Wang et al., “AdvSim: Generating safety-critical scenarios for self-driving vehicles,” Jan. 2022, *arXiv:2101.06549*.

[15] J. Sekhon and C. Fleming, “Towards improved testing for deep learning,” in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. New Ideas Emerg. Results (ICSE-NIER)*. Montreal, QC, Canada, May 2019, pp. 85–88.

[16] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine learning testing: Survey, landscapes and horizons,” *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, pp. 1–36, Jan. 2022.

[17] K. Pei, Y. Cao, J. Yang, and S. Jana, “DeepXplore: Automated whitebox testing of deep learning systems,” in *Proc. 26th Symp. Oper. Syst. Principles*, Oct. 2017, pp. 1–18.

[18] Y. Tian, K. Pei, S. Jana, and B. Ray, “DeepTest: Automated testing of deep-neural-network-driven autonomous cars,” Mar. 2018, *arXiv:1708.08559*.

[19] M. Henne, A. Schwaiger, K. Roscher, and G. Weiss, “Benchmarking uncertainty estimation methods for deep learning with safety-related metrics,” in *Proc. SafeAI AAAI*, New York, NY, USA, 2020, pp. 83–90.

[20] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 618–626.

[21] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” Nov. 2013, *arXiv:1311.2901*.

[22] J. Lust and A. P. Condurache, “A survey on assessing the generalization envelope of deep neural networks: Predictive uncertainty, out-of-distribution and adversarial samples,” Sep. 2021, *arXiv:2008.09381*.

[23] S. Luan, Z. Gu, L. B. Freidovich, L. Jiang, and Q. Zhao, “Out-of-distribution detection for deep neural networks with isolation forest and local outlier factor,” *IEEE Access*, vol. 9, pp. 132980–132989, 2021.

[24] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” Mar. 2015, *arXiv:1412.6572*.

[25] Y. Gal and Z. Ghahramani, “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning,” Oct. 2016, *arXiv:1506.02142*.

[26] B. Lakshminarayanan, A. Pritzel, and C. Blundell, “Simple and scalable predictive uncertainty estimation using deep ensembles,” Nov. 2017, *arXiv:1612.01474*.

[27] C.-H. Cheng, G. Nührenberg, and H. Yasuoka, “Runtime monitoring neuron activation patterns,” Sep. 2018, *arXiv:1809.06573*.

[28] R. E. Bryant, “Symbolic Boolean manipulation with ordered binary-decision diagrams,” *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, Sep. 1992.

[29] D. Hendrycks, M. Mazeika, and T. Dietterich, “Deep anomaly detection with outlier exposure,” Jan. 2019, *arXiv:1812.04606*.

[30] A. Torralba, R. Fergus, and W. T. Freeman, “80 million tiny images: A large data set for nonparametric object and scene recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.

[31] K. Lee, H. Lee, K. Lee, and J. Shin, “Training confidence-calibrated classifiers for detecting out-of-distribution samples,” Feb. 2018, *arXiv:1711.09325*.

[32] S. Kullback and R. A. Leibler, “On information and sufficiency,” *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, Mar. 1951.

[33] D. Meng and H. Chen, “MagNet: A two-pronged defense against adversarial examples,” Sep. 2017, *arXiv:1705.09064*.

[34] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” in *Proc. Netw. Distrib. Syst. Security Symp.*, San Diego, CA, USA, 2018, pp. 1–16.

[35] C.-H. Cheng, G. Nührenberg, C.-H. Huang, H. Ruess, and H. Yasuoka, “Towards dependability metrics for neural networks,” Jun. 2018, *arXiv:1806.02338*.

[36] S. Houben et al., “Inspect, understand, overcome: A survey of practical methods for AI safety,” Apr. 2021, *arXiv:2104.14235*.

[37] V. Chen, M.-K. Yoon, and Z. Shao, “Novelty detection via network saliency in visual-based deep learning,” in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. Workshops (DSN-W)*, Portland, OR, USA, Jun. 2019, pp. 52–57.

[38] R. McConnell, “Method of and apparatus for pattern recognition,” U.S. Patent 4 567 610, 1986.

[39] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, San Diego, CA, USA, 2005, pp. 886–893.

[40] D. W. Marquardt, “Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation,” *Technometrics*, vol. 12, no. 3, pp. 591–612, Aug. 1970.

[41] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” Dec. 2016, *arXiv:1612.08242*.

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” Mar. 2019, *arXiv:1801.04381*.

[43] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural Netw.*, vol. 32, pp. 323–332, Aug. 2012.

[44] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Apr. 2015, *arXiv:1409.1556*.

[45] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” Sep. 2019, *arXiv:1706.06083*.

[46] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, Aug. 2016, pp. 785–794.

LUKAS HACKER received the B.Sc. and M.Sc. degrees from the Technical University of Kaiserslautern in 2017 and 2020, respectively. During the bachelor and master thesis with the Research Departments of Opel Automobile GmbH, Rüsselsheim and Mercedes Benz AG, Sindelfingen, he specialized in control theory and artificial intelligence. Since 2020, he takes part in a Ph.D. (Dr.-Ing.) Program with Mercedes-Benz AG, Sindelfingen, in cooperation with the Institute of Measurement and Sensor Technology, Technical University of Kaiserslautern. His research focuses on SOTIF and safe AI in the context of automated driving. For his research activities, he received the Ferchau Young Talents Award in 2017. Furthermore, he is a member of the DIN Mirror Committee “Artificial Intelligence for Automotive” for ISO/TC 22/SC 32/WG 14 “Safety and Artificial Intelligence” and works on standardization of safe AI.

JÖRG SEEWIG received the electrical engineering degree from the University of Hanover and the Ph.D. degree (Dr.-Ing.) in 2000. From 1995 to 1999, he was a Scientific Assistant in the field of Production Measurement Technology with the Institute for Measurement and Control Technology, University of Hanover. In 2000, he co-founded an engineering office with focus on software development for production measurement technology. From 2003 to 2008, he also took over the management of the area of production measurement and testing technology with the Institute for Measurement and Control Technology, University of Hanover. Since 2008, he has been a Full Professor and the Chairman of the Institute of Measurement and Sensor Technology with the Technical University of Kaiserslautern within the Department of Mechanical and Process Engineering. Furthermore, he is the German delegate in the ISO/TC 213, WG 15 “Filtration” and WG 16 “Surface Texture” and the Chairman of the Mirror Committee in DIN. He is the author of various international and national standards in the field “Geometric Product Specification”. Beyond that, he has been a Partner in Opti-Cal GmbH, since 2018.