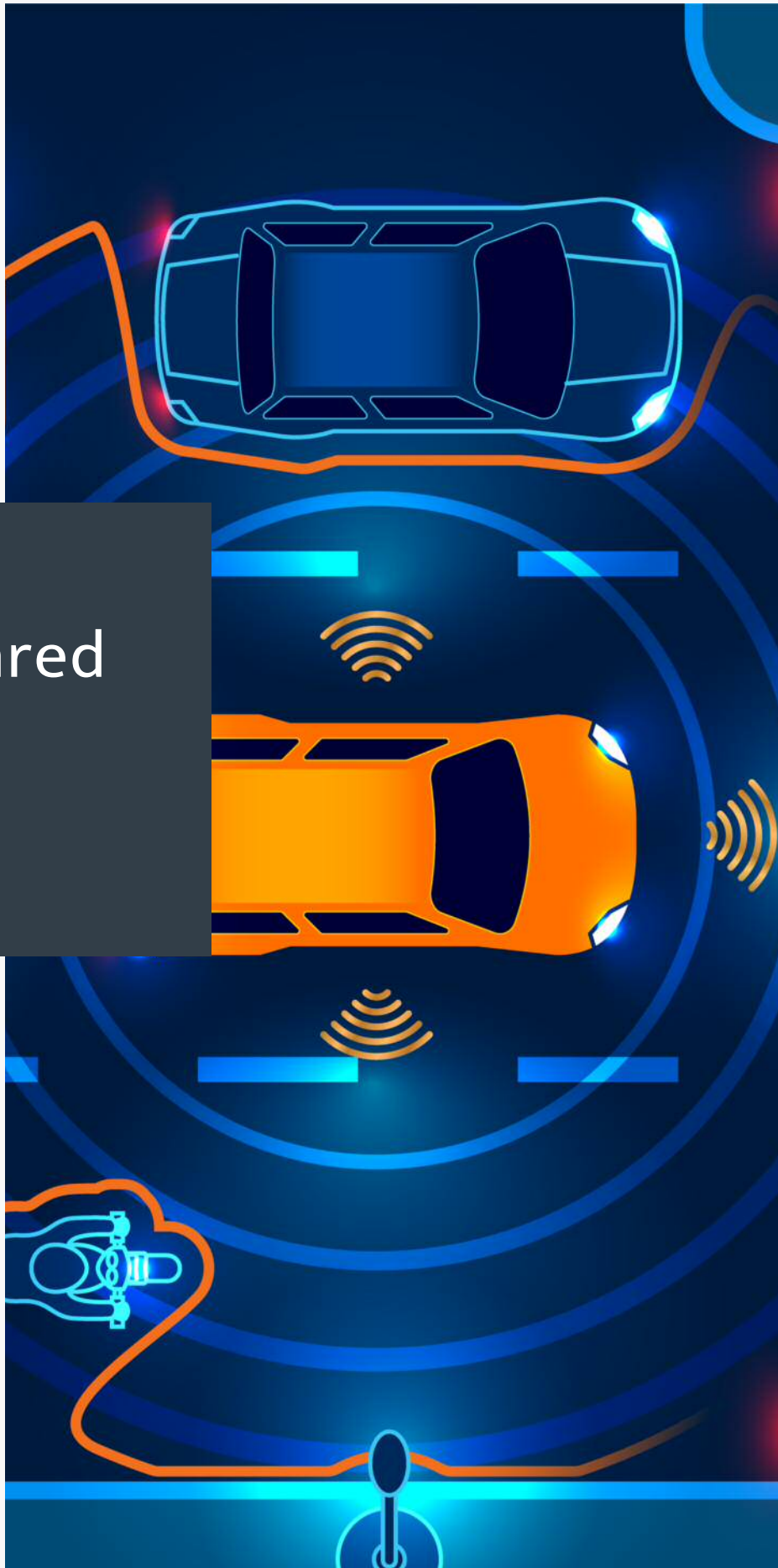
ASSURING AUTONOMY
INTERNATIONAL PROGRAMME

DEMONSTRATOR PROJECT

Final report

Safe-SCAD: Safety of shared control in autonomous driving



FEBRUARY 2022

Safe-SCAD: Safety of Shared Control in Autonomous Driving

Final Technical Report

February 2022

Project Team

Lu Feng, Erfan Pakdamanian, Shili Sheng and Kayla Boggess
University of Virginia, USA

Corina Păsăreanu, John Grese, and Ravi Mangal
Carnegie Mellon University, USA

Radu Calinescu, Calum Imrie, Misael Alpizar Santana, Gricel Vázquez, Naif Alasmari, Emad Alharbi and Mario Gleirscher
University of York, UK

Executive Summary

International standards classify automated driving systems on a six-level scale, from no automation at Level 0 to full automation at Level 5. Despite huge R&D budgets and much hype over the past decade, fully autonomous (Level 5) cars are unlikely to become available to the general public any time soon. In contrast, cars providing Level 2 (i.e., partial) automation can be purchased from manufacturers including Tesla, Nissan and BMW; and the approval of Level 3 (i.e., conditional automation) and 4 (i.e., high automation) cars is being considered by regulators worldwide.

A critical requirement for vehicles operating at autonomy Levels 2 and 3 is that a user resides in the driver's seat and is sufficiently attentive to be able to share the control of the car with the automated driving system. Although Level 4 autonomous vehicles do not rely on human support, they may still issue *timely requests for human intervention* (e.g., when they approach traffic situations they were not designed to handle), performing a minimum-risk manoeuvre (e.g., stopping the car safely) if their user does not respond.

However, drivers find it very challenging to remain attentive when in charge of vehicles with automated driving systems, as shown by accidents involving both Level 2 autonomous cars used by regular drivers, and cars with higher autonomy levels tested by professional safety drivers. To address this challenge, recent regulations—such as the UN regulation on the approval of vehicles with automated lane keeping systems—advocate the use of autonomous systems capable of detecting driver inattentiveness and of issuing automated alerts to mitigate it.

The Assuring Autonomy International Programme demonstrator project 'Safe-SCAD: Safety of Shared Control in Autonomous Driving' has developed a proof-of-concept driver attentiveness management system for this purpose. The Safe-SCAD system comprises a deep neural network (DNN) responsible for predicting the driver control-takeover behaviour, methods for verifying this DNN, and a discrete-event controller that issues optical, acoustic and/or haptic driver alerts based on the predictions of the DNN and the results of its online verification. This report describes the development and integration of the SafeSCAD components, and the testing of the proof-of-concept SafeSCAD solution in a driving simulator at the University of Virginia.

Developed with input from project partner Toyota Info Tech labs, the proof-of-concept SafeSCAD solution shows that intelligent driver-attention management systems have the potential to improve the safety of shared-control automated driving. Additionally, the research carried out to develop this solution has led to two important insights that are applicable to a broad range of autonomous systems. The first insight is that formal analysis techniques for neural networks can be used to quantify the aleatory uncertainty of multiclass deep neural network classifiers within the operational design domain of their autonomous systems. Second, the project showed that the use of a combination of design-time and online verification of neural networks enables the synthesis of conventional discrete-event controllers guaranteed to satisfy key safety, dependability and performance requirements of autonomous systems, and to be Pareto optimal with respect to a set of optimisation objectives.

1 Introduction

Ensuring and assuring the safety of shared control in autonomous driving is very challenging due to the uncertainties associated with measuring the level of situational awareness of safety drivers while not in control of the vehicle, and with the mapping of such measures to control hand-back times and likelihood of success. In this demonstrator project, we extended, adapted and integrated our previous research and the latest advances from human behaviour and cognitive modelling, verification of deep neural networks, and automated controller synthesis to tackle these challenges. We used an advanced semi-autonomous driving simulator to deliver methods for ensuring and assuring the safety of shared control in autonomous driving, and a demonstrator that leverages these methods. The project has made contributions to the AAIP Body of Knowledge, in the areas of shared autonomy/human-machine interaction, verification of machine learning, verification of deciding requirements, and implementation of decision making elements.

Our project, titled "Safe-SCAD: Safety of shared control in autonomous driving," has addressed the following critical barrier about control handover in shared-control autonomous systems: *if (semi-) autonomous cars have to hand (back) control to a safety driver, how can it be ensured and assured that the human has sufficient situational awareness to be able to take over control safely and effectively?* The project has pursued five concrete objectives:

- **Objective 1**: To develop deep-learning based methods for modeling and predicting of driver takeover behavior, using multi-modal sensing data of vehicles and driver biometrics (e.g., eye-tracking, heart rate, galvanic skin response).
- **Objective 2**: To develop methods for the verification of the deep neural networks from Objective 1, in order to guarantee (i.e., to provide assurance evidence for) the bounds of safety drivers' behaviour.
- **Objective 3**: To develop methods and assurance evidence for the stochastic modelling of the safety driver - autonomous car system, and for the synthesis of controllers capable of maintaining suitable levels of driver situational awareness (e.g., by using audio, vibration and light stimuli to improve human alertness).
- **Objective 4**: To develop a proof-of-concept demonstrator system that uses the methods and assurance evidence from Objectives 1-3 and the UVA semi-autonomous driving testbed. Figure 1 shows an overview of the developed demonstrator system.

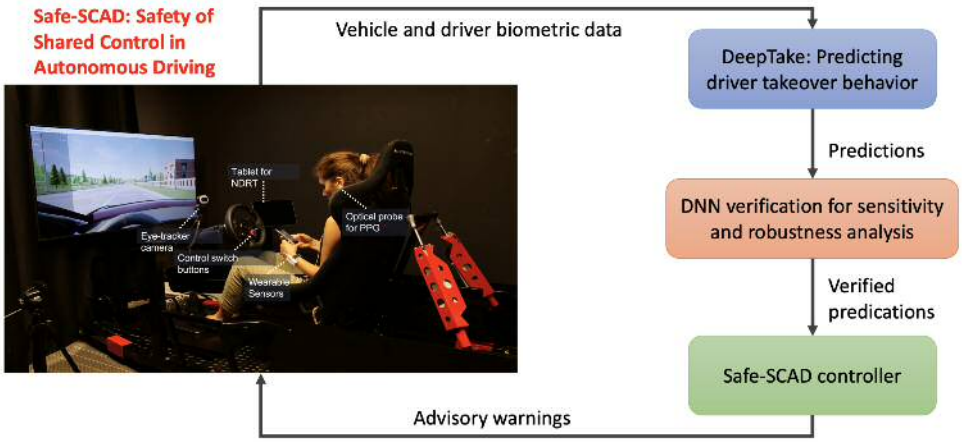


Figure 1: Overview of the Safe-SCAD demonstrator system.

- **Objective 5**: To contribute methods for ensuring and assuring the safety of robotics and autonomous systems (RAS) 'Handover' to the AAIP BoK, and to inform the regulatory community on these advances.

The results achieved by the project for Objectives 1–4 are described in Sections 2–5, with further technical detail provided in Appendixes A–D.

All the project deliverable, including the BoK contributions for Objective 5, are made available at `https://drive.google.com/drive/folders/1RKtESQhvXk-9Qr-CVyiiO2Jrp17aVTBK?usp=sharing`.

2 Prediction of Driver Takeover Behavior using Multimodal Data

2.1 Motivation and Background

Automated vehicles (AVs) promise a future where drivers can engage in non-driving tasks without hands on the steering wheels for a prolonged period. In Level 3 of autonomy (i.e., conditionally automated driving), as defined by the Society of Automotive Engineers (SAE international [12]), the driver does not need to continuously monitor the driving environment. Nevertheless, due to current technology limitations and legal restrictions, AVs may still need to handover the control back to drivers occasionally (e.g., under challenging driving conditions beyond the automated systems' capabilities) [40]. In such cases, AVs would initiate takeover requests (TORs) and alert drivers via auditory, visual, or vibrotactile modalities [44, 54, 46] so that the drivers can resume manual driving in a timely manner. However, there are challenges in making drivers safely take over control. Drivers may need a longer time to shift their attention back to driving in some situations, such as when they have been involved in NDRTs for a prolonged time [56] or when they are stressed or tired [23]. Even if TORs are initiated with enough time for a driver to react, it does not guarantee that the driver will safely take over [41]. Besides, frequent alarms could startle and increase drivers' stress levels leading to detrimental user experience in AVs [47, 33, 35]. These challenges denote the need for AVs to constantly monitor and predict driver behavior and adapt the systems accordingly to ensure a safe takeover.

The vast majority of prior work on driver takeover behavior has focused on the empirical analysis of high-level relationships between the factors influencing takeover time and quality (e.g., [43, 57, 17, 20]). More recently, the prediction of driver takeover behavior using machine learning approaches has been drawing increasing attention. However, only a few studies have focused on the prediction of either takeover time [36, 3] or takeover quality [5, 14, 16, 18]; and their obtained accuracy results (ranging from 61% to 79%) are insufficient for the practical implementation of real-world applications. This is partly due to the fact that takeover prediction involves a wide variety of factors (e.g., drivers' cognitive and physical states, vehicle states, and the contextual environment) that could influence drivers' takeover behavior [55].

2.2 Our Approach

To address the aforementioned challenges, we developed a novel approach, named **DeepTake**, to provide reliable predictions of multiple aspects of takeover behavior, including (1) *takeover intention* – whether the driver would respond to a TOR; (2) *takeover time* – how long it takes for the driver to resume manual driving after a TOR; and (3) *takeover quality* – the quality of driver intervention after resuming manual control.

As illustrated in Figure 2, DeepTake considers multimodal data from various sources, including driver's pre-driving survey response (e.g., gender, baseline of cognitive workload and stress levels), vehicle data (e.g., lane position, steering wheel angle, throttle/brake pedal angles), engagement in NDRTs, and driver biometrics (e.g., eye movement for detecting visual attention, heart rate and galvanic skin responses for the continuous monitoring of workload and stress levels). This data can easily be collected in AVs' driving environment. For instance, all of the driver biometrics utilized in DeepTake can be captured by wearable smartwatches and deployed eye-tracking systems. The multitude of sensing modalities and data sources offer complementary information for the accurate and highly reliable prediction of driver takeover behavior. The collected multimodal data are pre-processed followed by segmentation and feature extraction. The extracted features are then labeled based on the belonging takeover behavior class. In our framework, we define each aspect of takeover behavior as a classification problem (i.e., takeover intention as a binary classes whereas takeover time and quality as three multi-classes). Finally, we built DNN-based predictive models for each aspect of takeover

behavior. DeepTake takeover predictions can potentially enable the vehicle autonomy to adjust the timely initiation of TORs to match drivers' needs and ultimately improve safety.

Below, we describe the detailed steps of the DeepTake approach.

1. **Data Collection**: we collect multimodal data such as driver biometrics, pre-driving surveys, types of engagement in non-driving related tasks (NDRTs), and vehicle data. Collecting multimodal data copes with the main drawback and inability to provide the underlying complicated state of the driver. As driving is a dynamic task and could be impacted by internal and external factors, multiple physiological data streams were used. However, DeepTake can be adjusted to fit the data entry.

2. **Pre-processing**: the collected multimodal data are preprocessed followed by segmentation and feature extraction. Due to sensitiveness of physiological wearables, intensive preprocessing should be applied to remove motion artifacts and extract meaningful information. The extracted features are then labeled based on the belonging to takeover behavior class.

3. **Labelling**: DeepTake tends to cover multiple aspects of takeover behavior to provide more reliable outcomes. We define each aspect of takeover behavior as a multi-class classification problem (i.e., takeover intention as a binary class whereas takeover time and quality as three multi-classes). Thus, we labeled takeover time as the period from the moment the takeover request alarm is triggered to the moment a participant initiates regaining control by pressing the two embedded buttons on the steering. This period defines the takeover time for each participant which categorized as Low, Medium, and High. In addition, we consider a motivating scenario where the driver needs to take over control of the vehicle and swerve away from an obstacle blocking the same lane; meanwhile, the vehicle should not deviate too much from the current lane, risking crashing into nearby traffic. Thus, takeover quality was labeled as the lateral deviation from the current lane. we label the feature vectors into three classes of takeover quality: "low" or staying in a lane when the deviation is smaller than 3.5 meters, "medium" or maneuver the obstacle when the deviation is grater than 7 meters, or "high" or maneuver safely when the deviation is between 3.5 and 7 meters.

4. **Modeling**: DeepTake utilizes a feed-forward deep neural network (DNN) with a mini-batch stochastic gradient descent. The DNN model architecture begins with an input layer to match the input features, and each layer receives the input values from the prior layer and outputs to the next one. Although we used 3classes of takeover time and quality, the output layer of DNN model

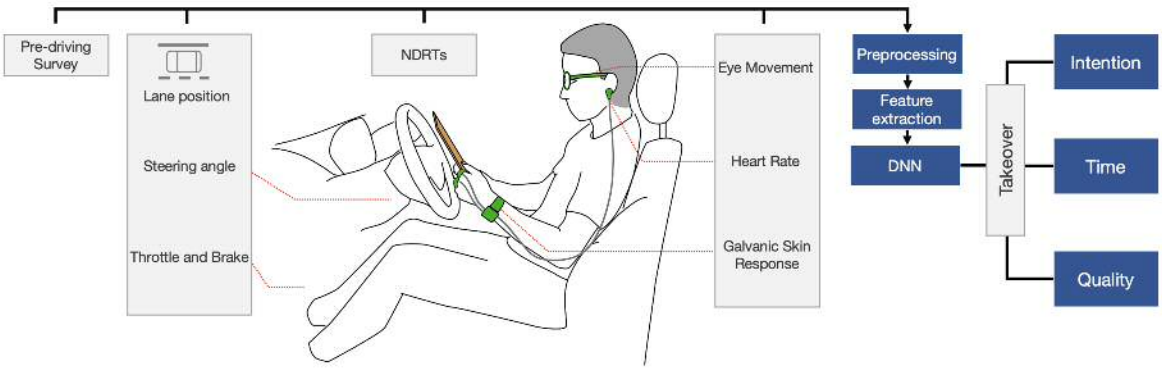


Figure 2: DeepTake uses data from multiple sources (pre-driving survey, vehicle data, non-driving related tasks (NDRTs) information, and driver biometrics) and feeds the preprocessed extracted features into deep neural network models for the prediction of takeover intention, time and quality.

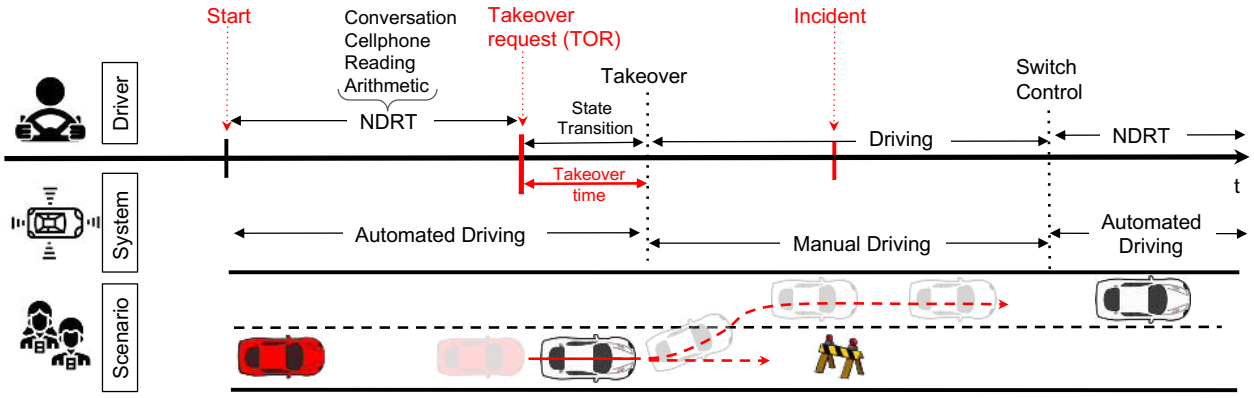


Figure 3: A schematic view of an example of a takeover situation used in our study, consisting of: 1) takeover timeline associated with participants' course of action; 2) system status; and 3) takeover situation. The vehicle was driven in the automated mode to the point after the TOR initiation and transitioning preparation period. The ego vehicle is shown in red and the lead car is white. When the Ego vehicle reaches its limits, the system may initiate (true alarm) or fail (no alarm) to initiate the TOR, and the driver takes the control back from the automated system.

can be customized for the multi-class classification. In fact, we also demonstrated the capability of DeepTake in predicting 5-class takeover time. We evaluate the DNN-model performance against multiple state-of-art models.

2.3 Evaluation

We validate DeepTake framework feasibility using data collected from a driving simulator study. The driving scenarios comprised a 4-lane rural highway, with various trees and houses placed alongside the roadway. We designed five representative situations where the AVs may need to prompt a TOR to the driver, including novel and unfamiliar incidents that appear on the same lane. Figure 3 displays an example of a takeover situation used in our study. The designed unplanned takeovers let participants react more naturally to what they would normally do in AVs, participants' reaction times are in detectable categories. In other words, participants have no previous knowledge of incident appearance, which might happen among other incidents requiring situational awareness and decision-making.

There are a number of potential methods to address reliability of the models. Each of these methods shows a different aspect of the model. We evaluate the performance of DeepTake framework by multiple metrics. Applying different metrics reflect the goodness of the proposed model in different aspects. We first apply 10-fold cross-validation on training data to evaluate the performance of selected features in the prediction of driver takeover intention, time and quality. We then compared the proposed model against 6 other models using Receiver Operating Characteristic (ROC), and weighted F1 scores. We finally use the confusion matrix to further illustrate the summary of DeepTake's performance on the distinction of takeover intention, time, and quality per class. The results show that DeepTake models significantly outperform six machine learning-based models in all predictions of takeover intention, time and quality. Specifically, DeepTake achieves an accuracy of 96% for the binary classification of takeover intention, 93%, and 83% accuracy for multi-class classification of takeover time and quality, respectively. These accuracy results also outperform results reported in the existing work. We refer to Appendix A for details of these results.

2.4 Implications

We believe that our human-centered DeepTake framework makes a step towards enabling a longer interaction with none driving related tasks (NDRTs) for automated driving. DeepTake provides a new approach to help the monitoring systems to constantly observe and predict the driver's mental and physical status by which the automated system can make optimal decisions and improve the safety and user experience in AVs. Specifically, by integrating the DeepTake framework into the monitoring systems of AVs, the automated system infers when the driver has the intention to takeover through multiple sensor streams. Once the system confirms a strong possibility of takeover intention, it can adapt its driving behavior to match the driver's needs for acceptable and safe takeover time and quality. Therefore, a receiver of TOR can be ascertained as having the capability to take over properly, otherwise, the system would have allowed the continued engagement in NDRT or warned about it. Thus, integration of DeepTake into the future design of AVs facilitates the human and system interaction to be more natural, efficient and safe. Since DeepTake should be used in safety-critical applications, we further validated it to ensure that it meets important safety requirements.

DeepTake framework provides a promising new direction for modeling driver takeover behavior to lessen the effect of the general and fixed design of TORs which generally considers homogeneous takeover time for all drivers. This is grounded in the design of higher user acceptance of AVs and dynamic feedback. The information obtained by DeepTake can be conveyed to passengers as well as other vehicles letting their movement decisions have a higher degree of situational awareness.

3 Verification of Deep Neural Networks

We report here on the analysis of a neural network component that was built for predicting *takeover time* in the shared-control autonomous driving system. The network was trained on data collected from a (semi-)autonomous driving simulator. The network is a fully-connected network with three hidden layers and ReLU activation functions. More details about the work can be found in [31].

We report results for the following types of analysis: attribution and trust in neural networks, formal robustness analysis, and confidence analysis via calibration of neural networks. We further performed work on mining properties of neural networks and transfer learning for building personalized models. Due to space constraints, we do not describe them here but details can be found in [31].

3.1 Attribution and Trust in Neural Networks

Neural networks are essentially black-box models, which generate a prediction based on input features and some learned weights. In critical applications, it is imperative to understand how and why the model gives the predictions, by identifying the *important features* that have the highest impact on the model predictions. We therefore designed and evaluated a framework to determine feature importance as viewed by the model.

We examined off-the-shelf state-of-the-art methods such as SHAP [37], LIME [50] and Integrated Gradients (IG) [51]. SHAP and Integrated Gradients are white-box techniques whereas LIME is a black-box method for attribution analysis. Given a set of input samples, we generate an importance vector of size, 1 $\times$ n_features per sample. We randomly selected 3000 samples and created a 3000 $\times$ n_features importance matrix. From the importance matrix, we computed the number of times a feature was regarded as top-k important feature (with the respective method) and created a dictionary where for each feature there are k values and each value signifies the number of times that feature was regarded as kth important feature. For validation, we dropped the features that were found of less importance, re-trained network using the same architecture, and evaluated the resulting accuracies.

3.1.1 Results

In Figure 4, for each feature, the measured importance values are plotted for the three evaluated methods (SHAP, LIME and IG). Each bar has 5 parts, demonstrating the top-5 importance values. It can be observed that *FixationSeq, FixationStart* and *Manualwheel* are given high importance values by the three methods, whereas *ManualBrake* is given a high importance value only by LIME. Some features, such as *FixationDuration, AutoWheel, RightLaneType, RangeW* appear to have low importance values. Figure 5 depicts the model accuracy after dropping low-importance features. It also validates the importance values as measured by the three methods. For example, *ManualWheel* and *FixationSeq* are important features hence dropping those results in lower accuracy. Dropping *FixationDuration, RangeW* and *AutoWheel* results in a model with comparable accuracy, demonstrating that they are indeed of low importance. The results indicate that SHAP and IG have similar performance, with LIME giving some outliers. The experiments indicate that existing attribution techniques can indeed be used to understand the model behaviour and furthermore can be used to optimize the model (by dropping some features that have little influence over model predictions).

3.2 Clustering for Robustness Analysis

Robustness analysis of neural networks aims to formally verify that small perturbations to inputs do not modify the network predictions. One approach for robustness analysis is to use label guided k-means clustering [25] on the training data to find *regions* in the input space where the model prediction does not change, and the model is thus potentially robust to input perturbations. Model robustness in

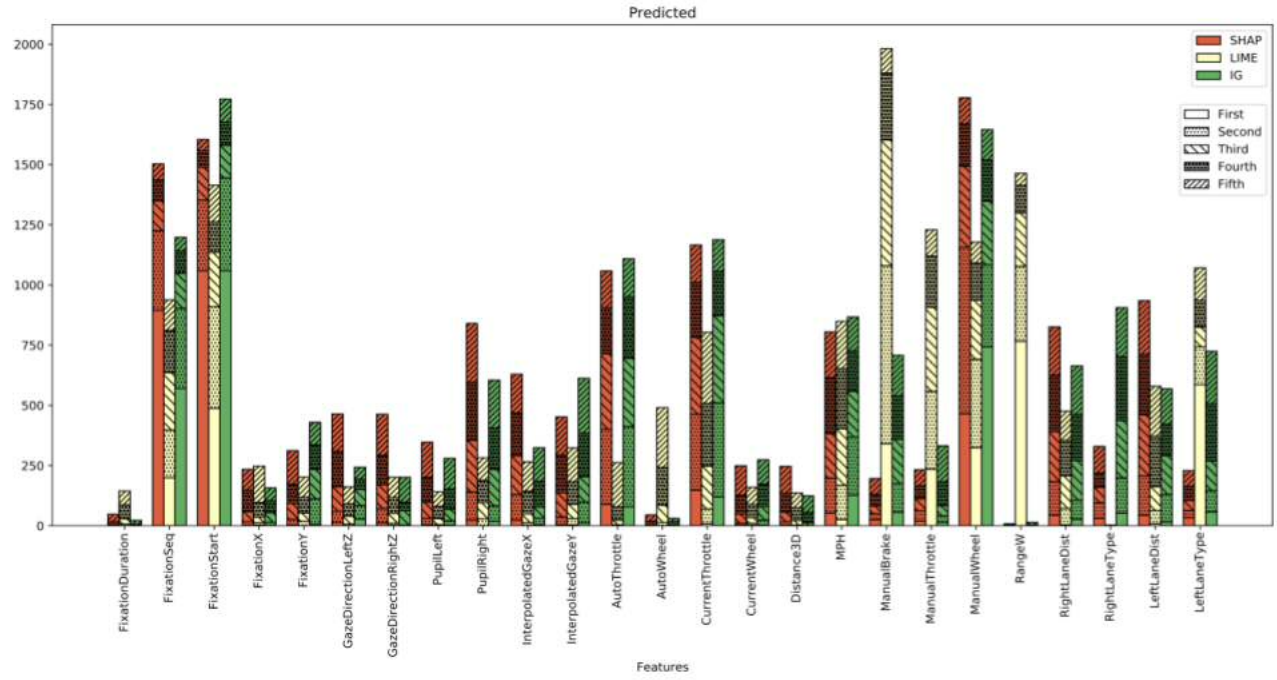


Figure 4: Distribution showing the number of times a feature was regarded top-5 important

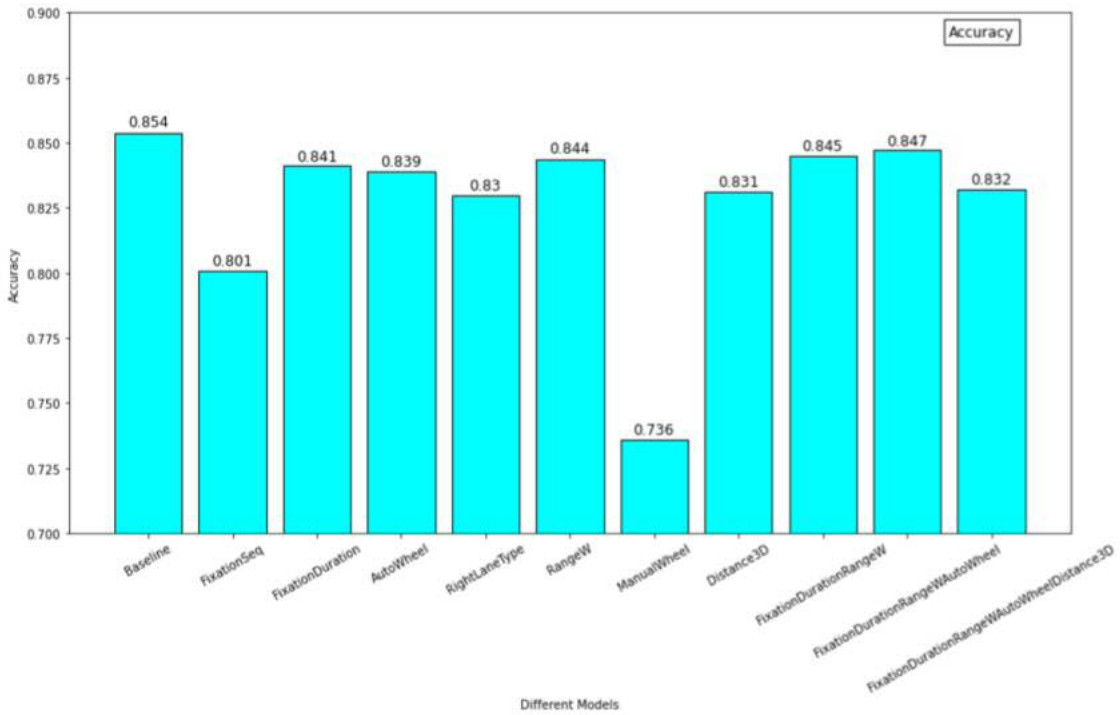


Figure 5: Bar plot depicting the accuracy of models trained with dropped features

these regions is then further validated using formal methods. We report here on an extension of this clustering-based approach for robustness analysis. In particular, we investigated ways to improve the clustering algorithm used to find the robust regions.

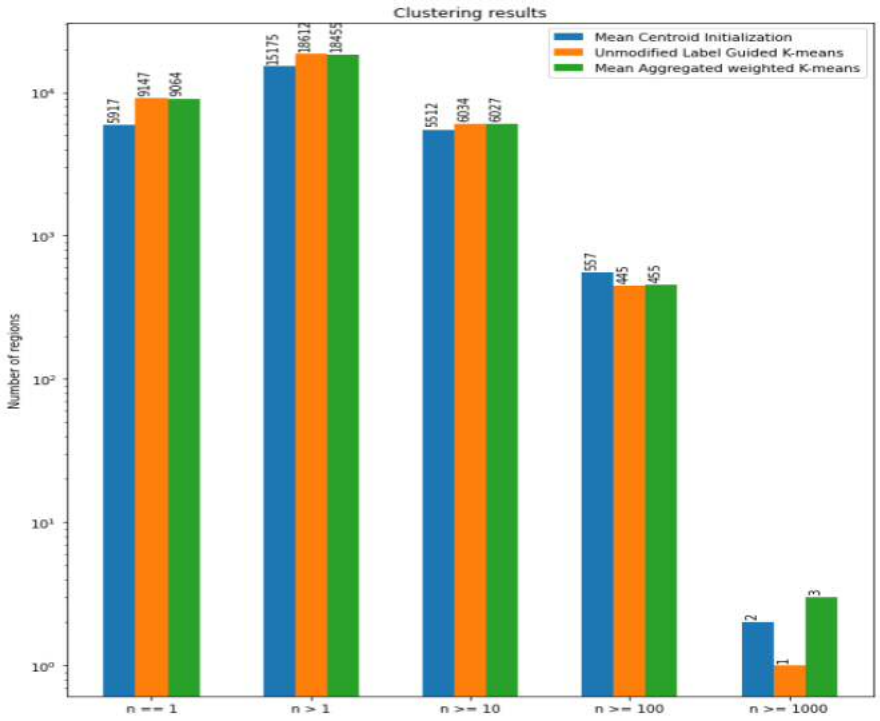


Figure 6: Clustering results

The original label guided k-means clustering algorithm often produces clusters (or regions) that contain a single training data point. Such single point regions are not very useful for robustness analysis since they only cover a small volume of the input space. Moreover, it increases the cost of formal verification since model robustness has to be verified in a large number of small regions as opposed to a small number of large regions (note that the cost of verification scales with the number of regions). The existence of clusters with a single training data point can be attributed to the fact that these points did not fall close to the initialized cluster centroids. To address this issue, methods such as weighted k-means clustering, elbow method, and centroid initializations were implemented and evaluated. For the elbow method, it was found that setting the initial number of clusters to 150 (i.e., k=150) was a good start for the clustering algorithm on our dataset.

3.2.1 Clustering Results

The results in Figure 6 show the number of regions on y-axis with each region containing 'n' number of points, where n is depicted on the x-axis. The 3 bars are the comparison of the clustering results of the 3 methodologies we considered:

- unmodified label guided k-means (initial method)
- mean aggregated weighted k-means
- mean centroid initialized k-means

There are 5 categories of these clusters as represented on the x-axis with 'n' ranging from 1 to 1000. The best case would be to have less number of regions with 'n == 1' and the more regions with 'n >= 1000'. Out of the 3 methods, the mean-centroid initialization has the least amount of regions with n==1 (5917 regions as compared to 9000+ regions for the other two methods.) and higher amount of regions with 'n >= 100'. This indicates that the clustering has fewer singular point clusters. The

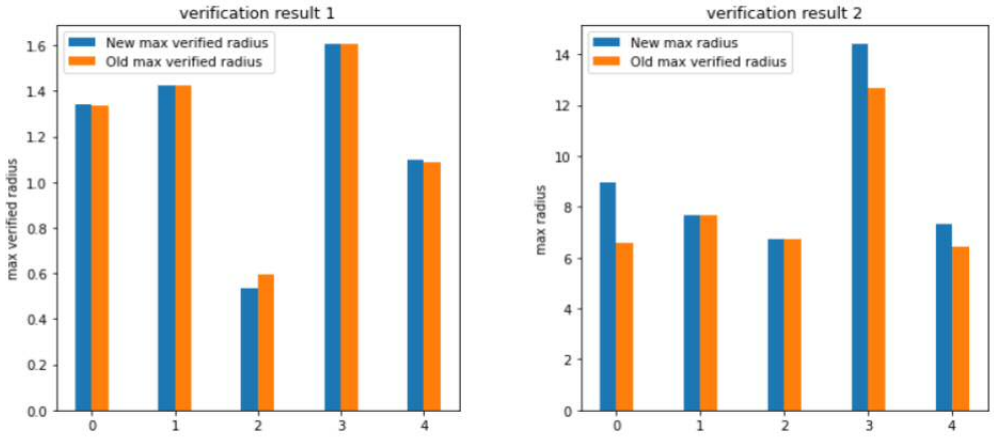


Figure 7: Comparison of max verified radius (left) and max radius (right)

weighted k-means method and the unmodified method produce similar results, so choosing the mean centroid initialization would be more beneficial amongst the 3 methods.

3.2.2 Verification with Marabou

The regions computed with k-means clustering were passed to Marabou [32] for verification of robustness. Essentially, for a region R with centroid C and radius r we use Marabou to check that all the points in the region have the same label (and thus the model is robust – behaves consistently – in that region). The verification results indicated that for the modified k-means clustering, it was possible to verify larger regions. The number of $\epsilon = 0$ regions for the current implementation amounted to 116 whereas the previous implementation had 464 such regions. The value $\epsilon = 0$ indicates a result of either incorrect centroid initialization or the search space being too large for Marabou to verify within the time limit assigned (which was set to be 5 minutes per region).

The plots show the comparison of two parameters, (i) maximum verified radius; (ii) maximum cluster radius. The comparison of the radius (bar plot on the right in figure 7) attribute shows that the modification performed on the k-means clustering resulted in larger regions. When these regions were put through Marabou for verification, the maximum radius verified for each class is marginally better than the unmodified k-means methodology.

When Marabou fails to prove a property, it returns a counterexample. For the properties that we checked, a counterexample has the form of an input that is in a region R and has label different than the other points in the region. We performed experiments to determine if we can increase the robustness of the model by using the counterexamples found from Marabou. The results can be found in [31].

3.3 Calibration and Confidence

For real-world safety-critical systems, neural networks must not only be accurate and robust but they should also indicate when they are likely to be incorrect. The formally verified *safe regions* described above provide one measure of confidence for points that exist within the regions, however they are not capable of providing confidence about points outside those regions. To add a measure of confidence

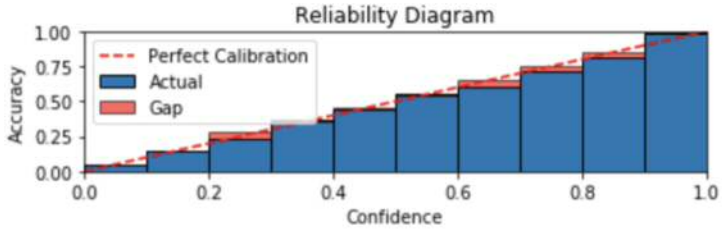


Figure 8: Pre-calibration reliability diagram

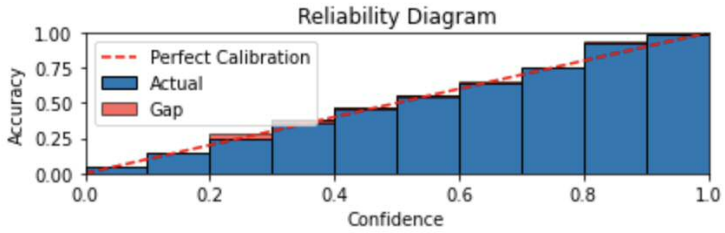


Figure 9: Post-calibration reliability diagram

for inputs which exist outside of the *safe regions*, we also investigated *confidence calibration* [26]. Confidence calibration is concerned with the problem of predicting probability estimates representative of the true correctness likelihood. In particular, we investigated temperature scaling, a single-parameter variant of Platt scaling, as a simple and effective way to post-process a network such that the probabilities associated with each predicted class reflect the correctness likelihood of the prediction. The method does not change the accuracy of the network; it uses the same parameter T (temperature) to soften the network output making the network less confident when making wrong predictions.

3.3.1 Results

Figure 8 depicts a reliability diagram which shows accuracy as a function of confidence measure for the original, un-calibrated model. A well calibrated model would have the bars well-aligned with the diagonal line (identity function), and will have same accuracy and confidence for a given bin. Although, most of the bars seem to be well-aligned, we observe gaps for some confidence intervals. These gaps illustrate low confidence of the model for samples whose prediction fall in that interval.

Figure 9 depicts the model's output after calibration using temperature scaling. We obtain a better calibrated model without affecting the model's accuracy.

In the following section, we describe how to use verification and calibration results to guide the controller synthesis with provable guarantees.

4 Synthesis of Safe-SCAD Controllers

The J3016 standard [45] classifies automated driving systems (ADSs) on a six-level scale, from no automation at Level 0 to full automation at Level 5. Despite huge R&D budgets and much hype over the past decade, fully autonomous (Level 5) cars are unlikely to become available to the general public any time soon. In contrast, cars providing Level 2 (i.e., partial) automation can be purchased from manufacturers including Tesla, Nissan and BMW; and the approval of Level 3 (i.e., conditional automation) and 4 (i.e., high automation) cars is considered by regulators worldwide [10, 6, 21, 29, 52].

A critical requirement for vehicles operating at autonomy Levels 2 and 3 is that a user resides in the driver's seat and is sufficiently attentive to be able to share the control of the car with the ADS. At Level 2, this human in the loop is expected to '*complete the object and event detection and response subtask and* [to] *supervise the driving automation system*', while at Level 3 the user is expected to be '*receptive to ADS-issued requests to intervene* [. . .] *and* [to] *respond appropriately*' [45]. Although Level 4 ADSs do not rely on human support, they may still issue *timely requests for human intervention* (e.g., when they approach roads or traffic situations they were not designed to handle), performing a minimum-risk manoeuvre (e.g., stopping the car safely) if their user does not respond.

In these scenarios, accidents with potentially fatal consequences (for Levels 2 and 3) and frequent emergency stops (for Level 4) can only be avoided if the drivers are sufficiently attentive to be able to take over the control of their vehicles [42]. However, humans find it very difficult to remain attentive when overseeing the operation of automated and autonomous systems [11, 19, 39]. In the automotive domain, this is amply demonstrated by accidents involving both cars with Level 2 ADS used by regular drivers [2, 53] and cars with higher autonomy levels tested by professional safety drivers [22].

SafeSCAD proposes an approach that mitigates this problem by using a sense-understand-decide-act (SUDA) control loop to improve driver attentiveness in shared-control autonomous driving. The sensing component of this control loop uses an array of sensors to collect driver biometrics and vehicle data. The understanding component uses these data and the deep neural network [48] presented earlier in this report to predict the driver response time to a potential ADS intervention request. This results in a classification of the driver as attentive, semi-attentive or inattentive, and guides the planning of driver alerts by a formally verified discrete-event controller. This controller—whose development is described next—ensures that the risk due to driver inattentiveness does not exceed a predetermined level, and achieves Pareto-optimal trade-offs between risk level and driver nuisance.

4.1 Problem definition

Given a shared-control ADS vehicle, we assume that its driver can have one of $n_d \geq 2$ attentiveness levels. The highest level ('attentive') corresponds to the situation in which the driver can respond timely to a transition demand. The other levels correspond to diminished driver attentiveness and therefore to increased risk that can be mitigated through issuing alerts to improve the driver's attentiveness level.

We assume that the ADS can activate one or several of $K \geq 1$ *alerts* (e.g., optical, acoustic and haptic) as needed to improve the driver's attentiveness. As such, the ALKS state at any point in time is characterised by:

1. the driver attentiveness level $k \in \{1, 2, \ldots, K\}$, where $k = 1$ and $k = K$ correspond to the driver being 'attentive' and 'inattentive', respectively;

2. the set of active alerts $a \in \{0, 1\}^{n_a}$, where $a = (a_1, a_2, \ldots, a_{n_a})$ indicates that the i-th alert is inactive when $a_i = 0$, and active when $a_i = 1$.

Using the notation $[K] = \{1, 2, \ldots, K\}$ and $A = \{0, 1\}^{n_a}$ to denote the sets of possible values for the two components of the system state, we further assume that the following measures are defined over the state space $[K] \times A$:

1. $nuisance : A \times [K] \rightarrow \mathbb{R}_{\geq 0}$, where $nuisance(a, k)$ represents the nuisance experienced by the driver when the alerts $a \in A$ are in use and the driver attentiveness level is k, with

$$nuisance((0, 0, \ldots, 0), k) = 0$$

for all $k \in [K]$;

2. $risk : [K] \rightarrow \mathbb{R}_{\geq 0}$, where $risk(k)$ provides a measure of the risk during time periods when the driver attentiveness level is $k \in [K]$.

Finally, we assume that timing data are available about the drivers' transition between the attentiveness levels $[K]$, when different alert combinations are active, and at different vehicle speeds. These data may be available from studies of driver behaviour [36, 38], experiments carried out by autonomous vehicle manufacturers, observations of drivers who are using the deployed ADS, or a combination thereof. Given such data, the *driver attentiveness management problem* is to find a combination of alerts $a \in A$ to use for each *predicted driver attentiveness level* $\hat{k} \in K$ (i.e., attentiveness level predicted by a deep neural network that classifies the driver attentiveness level), such that the system

- does not exceed a predetermined level of risk,
- achieves Pareto optimality between minimising the driver nuisance and minimising the risk

over a period of T hours of driving. For additional details, please see Appendix C.

4.2 SafeSCAD controller synthesis

To solve the problem defined in the previous section, we used a new discrete-event controller synthesis approach called DEEPDECS[1] [8].

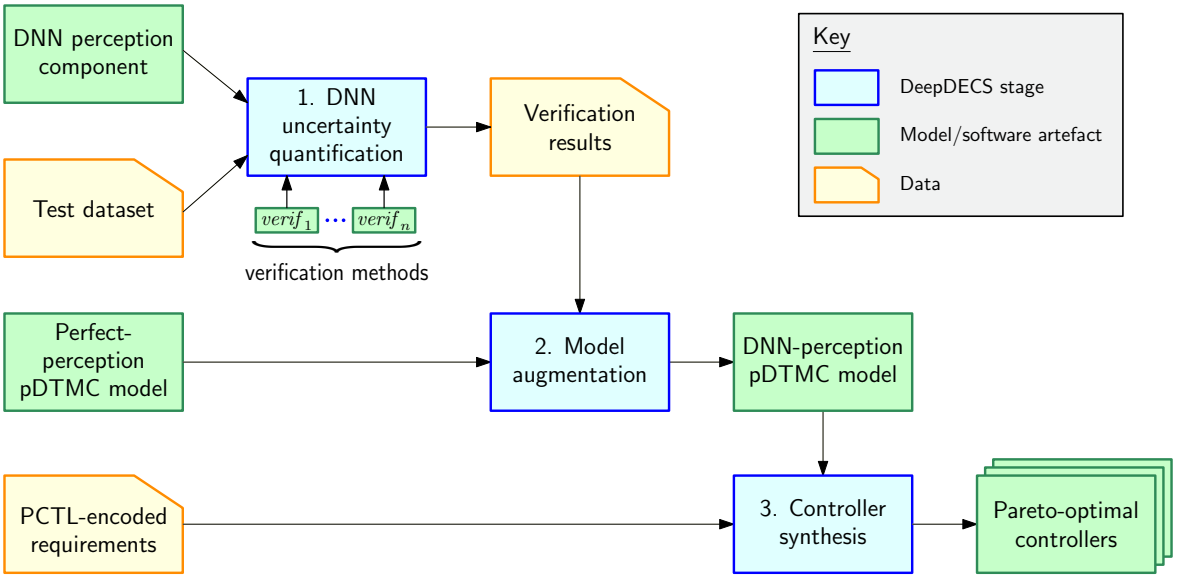
Overview. DEEPDECS uses a *parametric discrete-time Markov chain* (pDTMC) to model the design space of the controller under development. The uncertainty due to the use of a deep-learning perception component within the system to be controlled and, if applicable, the uncertainty inherent to the system and its environment are modelled by the probabilities of transition between the states of this pDTMC. Finally, the controller synthesis problem involves finding combinations of parameter values for which the Markov chain satisfies strict safety, dependability and performance constraints, and is Pareto-optimal with respect to a set of optimisation objectives. These constraints and optimisation objectives are formalised as probabilistic temporal logic formulae.

DEEPDECS derives the pDTMC underpinning its controller synthesis automatically from (i) DNN verification results that quantify the uncertainty due to the deep-learning perception component, and (i) an "ideal" pDTMC that models the behaviour of the controlled system assuming perfect perception (Figure 10a). The set of correct-by-construction, Pareto-optimal DEEPDECS controllers is then synthesised by applying a combination of probabilistic model checking and search techniques to the derived pDTMC. As shown in Figure 10b, each of these controllers operates by reacting to changes in the system, in the DNN outputs and, unique to DEEPDECS, in the results obtained through the online verification of each DNN input and prediction.

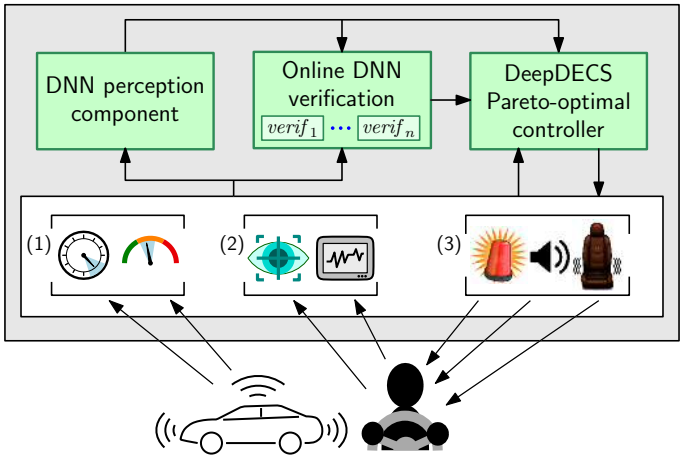
We detail each stage of the DEEPDECS approach and its application to the SafeSCAD problem below.

Stage 1: DNN uncertainty quantification. This section provides a brief introduction to DNN classifier verification, and describes the use of such verification techniques to quantify the aleatory uncertainty of DNN classifiers.

[1] Deep neural network perception Discrete-Event Controller Synthesis



(a) DEEPDECS generates discrete-event controllers aware of the uncertainty induced by the DNN perception component of an autonomous system in three stages. First, in a *DNN uncertainty quantification* stage, n verification methods are used to evaluate the DNN perception component over a test dataset representative for the operational design domain (ODD) of the autonomous system. The verification results provide a quantification of the DNN prediction uncertainty within the system ODD. Next, the *Model augmentation* stage uses these results—and an ideal-system pDTMC model that assumes perfect perception—to assemble a pDTMC system model that takes the DNN-induced uncertainty into account. Finally, the *Controller synthesis* stage uses this pDTMC model to synthesise a set of Pareto-optimal discrete-event controllers guaranteed to satisfy the PCTL-encoded requirements (constraints and optimisation objectives) of the system.



(b) SafeSCAD driver-attentiveness management system. Data from car sensors (1) and driver biometric sensors (2) are supplied to a DNN perception component that classifies the driver state as attentive, semi-attentive or inattentive. The DEEPDECS controller decides when optical, acoustic and/or haptic alerts (3) should be used to increase the driver's attentiveness.

Figure 10: DEEPDECS controller synthesis (a), and deployment (b)

a) Verification of DNN classifiers. A K-class DNN classifier f_θ is a function, composed of linear and non-linear transformations, of the form

$$f_\theta : \mathbb{R}^d \to [K], \tag{1}$$

where $[K]$ denotes the set $\{1, \ldots, K\}$, and θ refers to the *weights* or parameter values that characterize the linear transformations. As the results presented in this article are oblivious to the internal details of DNNs, we will by default omit the subscript θ, and treat f as a black-box function.

DNN classifiers are learnt from data, and are not guaranteed to always classify their input correctly. DNN verification techniques can help assess the quality of a classifier for a given input. A verification technique has the general form

$$verif : (\mathbb{R}^d \to [K]) \times \mathbb{R}^d \to \mathbb{B}, \tag{2}$$

such that, for a classifier $f \in \mathbb{R}^d \to [K]$ and an input $x \in \mathbb{R}^d$, $verif(f, x) = \mathsf{true}$ if the verification technique deems the DNN f likely to classify the input x correctly, and $verif(f, x) = \mathsf{false}$ otherwise. Two examples of simple DNN verification techniques (which we use to evaluate DEEPDECS later in the article) are:

1. *Model confidence threshold*—A K-class DNN classifier is practically implemented as a function of type $\mathbb{R}^d \to [0, 1]^K$, with each input $x \in \mathbb{R}^d$ first mapped to a discrete probability distribution $\delta(x) = (p_1, p_2, \ldots, p_K)$ over the K classes, and the class corresponding to the highest probability is chosen as the classifier prediction. The probability associated with a class can be interpreted as estimating the probability that the class is the *true* label of x. While it has been observed that classifiers may not be well-calibrated, i.e., the estimated correctness probabilities may be far from the true probabilities, a number of methods have been proposed to calibrate DNN classifiers [27]. Assuming that a classifier is well-calibrated using one of these methods, a simple DNN verification technique is to check whether the estimate correctness probability for an input x is greater than a fixed threshold τ for the class with the highest probability:

$$verif_1(f, x) = \begin{cases} \mathsf{true}, \text{ if } \max_{i=1}^{K} p_i \geq \tau \\ \mathsf{false}, \text{ otherwise} \end{cases} \tag{3}$$

2. *Local robustness certification* [9]—A DNN classifier f is ϵ-locally robust at an input x if perturbations within a small distance $\epsilon > 0$ from x (measured using the ℓ_2 metric) do not lead to a change in the classifier prediction. Accordingly, the local robustness verifier is defined by

$$verif_2(f, x) = \begin{cases} \mathsf{true}, \text{ if } \forall x' \in \mathbb{R}^d. \, ||x - x'||_2 \leq \epsilon \\ \qquad\qquad\qquad\qquad \implies f(x) = f(x') \\ \mathsf{false}, \text{ otherwise} \end{cases} \tag{4}$$

   for any input $x \in \mathbb{R}^d$.

b) Quantification of DNN perception uncertainty. The use of DNN perception introduces aleatory uncertainty into the autonomous system since DNNs are not guaranteed to predict accurately on all inputs. In the first DEEPDECS stage, we use a mechanism that relies on DNN verification techniques to empirically quantify the uncertainty of the DNN outcomes.

Let $X \subset \mathbb{R}^d$ be a *representative test dataset* for the DNN classifier (1), i.e., a set of classifier inputs that reflects the inputs that the autonomous system using the DNN will encounter in its ODD. For any test input $x \in X$, let $f^*(x) \in [K]$ be the label (i.e., the true class) of x, which is known since X is a test dataset.

DEEPDECS uses $n \geq 0$ DNN verification techniques $verif_1$, $verif_2$, …, $verif_n$ to identify subsets of X for which the classifier is likely to achieve higher accuracy than for the entire set X.[2] We use the n verification methods to partition the test dataset X into 2^n subsets comprising inputs x with the

[2]Note that DEEPDECS is also applicable in the special case when $n = 0$, i.e., when no verification techniques is used.

same verification results.[3] Formally, for a DNN classifier f and any $v = (v_1, v_2, \ldots, v_n) \in \mathbb{B}^n$, we define the test data subset

$$X_v = \{x \in X \mid verif(f, x) = v\}, \tag{5}$$

where $verif(f, x) = (verif_1(f, x), verif_2(f, x), \ldots, verif_n(f, x))$. We use each of these test data subsets to define a $K \times K$ confusion matrix $\mathcal{C}_v$ such that, for any $k, k' \in [K]$, the element in row k and column k' of this matrix is given by the number of inputs from X_v with true class k that the DNN classifies as belonging to class k'

$$\mathcal{C}_v[k, k'] = \#\left\{x \in X_v \mid f^*(x) = k \wedge f(x) = k'\right\}, \tag{6}$$

where, for any set A, $\#A$ denotes its cardinality.

As the test dataset X is representative of the DNN inputs that the system encounters in operation, we henceforth assume that the probability that a class-k input x satisfies $verif(f, x) = v$ and is (mis)classified by the DNN as belonging to class k' is given by:[4]

$$p_{kk'v} = Pr\left(f(x) = k' \wedge\ verif(f, x) = v \middle|\ f^*(x) = k\right) = \frac{C_v[k, k']}{\sum_{v' \in \mathbb{B}^n} \sum_{k'' \in [K]} C_{v'}[k, k'']}. \tag{7}$$

Stage 2: Model augmentation. This section provides a brief introduction to pDTMCs, defines the discrete-event controller synthesis problem, and presents the DEEPDECS theory underlying the generation of pDTMCs that model the behaviour of, and support the synthesis of controllers for, autonomous systems with deep-learning perception components.

a) Discrete-time Markov chains. DEEPDECS models the design space (i.e., the possible variants) for the controller of an autonomous system as a pDTMC augmented with *rewards*.

Definition 1. *A reward-augmented discrete-time Markov chain (DTMC) over a set of atomic propositions AP is a tuple*

$$\mathcal{M} = (S, s_0, P, L, R), \tag{8}$$

where $S \neq \emptyset$ is a finite set of states; $s_0 \in S$ is the initial state; $P : S \times S \to [0, 1]$ is a transition probability function such that, for any states $s, s' \in S$, $P(s, s')$ gives the probability of transition from state s to state s' and $\sum_{s' \in S} P(s, s') = 1$; $L : S \to 2^{AP}$ is a labelling function that maps every state $s \in S$ to the atomic propositions from AP that hold in that state; and R is a set of reward structures*, i.e., function pairs (ρ, ι) that associate non-negative values with the pDTMC states and transitions:*

$$\rho : S \to \mathbb{R}_{\geq 0}, \quad \iota : S \times S \to \mathbb{R}_{\geq 0}. \tag{9}$$

When (8) includes unknown transition probabilities and/or reward values, the DTMC is termed *parametric*.

Definition 2. *A reward-augmented parametric discrete-time Markov chain is a DTMC* (8) *comprising one or several transition probabilities and/or rewards that are specified as rational functions*[5] *over a set of continuous variables [13].*

DEEPDECS uses *probabilistic computation tree logic* (PCTL) [28, 4] extended with rewards [1] to quantify the safety, dependability and performance properties of an autonomous system whose controller design space is modelled as a pDTMC.

[3] As typical values for n are $n = 1, 2, 3$, there will only be a small number of such subsets.

[4] Formally, this results holds as $\#X \to \infty$.

[5] i.e., functions that can be written as fractions whose numerators and denominators are polynomial functions, e.g., $1 - p$ or $\frac{1-p_1}{p_2}$

Definition 3. State PCTL formulae Φ *and* path PCTL formulae Ψ *over an atomic proposition set* AP*, and* PCTL reward formulae Φ_R *over a reward structure* (9) *are defined by the grammar:*

$$\begin{aligned} \Phi &::= true \mid \alpha \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\sim p}[\Psi] \\ \Psi &::= X\Phi \mid \Phi \ \mathrm{U} \ \Phi \mid \Phi \ \mathrm{U}^{\leq k} \ \Phi \\ \Phi_R &::= \mathcal{R}_{\sim r}[\mathrm{C}^{\leq k}] \mid \mathcal{R}_{\sim r}[\mathrm{F} \ \Phi] \end{aligned} \tag{10}$$

where $\alpha \in AP$ *is an atomic proposition,* $\sim \in \{\geq, >, <, \leq\}$ *is a relational operator,* $p \in [0,1]$ *is a probability bound,* $r \in \mathbb{R}_0^+$ *is a reward bound, and* $k \in \mathbb{N}_{>0}$ *is a timestep bound.*

The PCTL semantics [28, 4, 1] is defined using a satisfaction relation $\models$ over the states of a DTMC. Given a state s of a DTMC $\mathcal{M}$, $s \models \Phi$ means 'Φ holds in state s', and we have: always $s \models true$; $s \models \alpha$ iff $\alpha \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; and $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. The *time-bounded until formula* $\Phi_1 \ \mathrm{U}^{\leq k} \ \Phi_2$ holds for a *path* (i.e., sequence of DTMC states $s_0 s_1 s_2 \ldots$ such that $P(s_i, s_{i+1}) > 0$ for all $i > 0$) iff Φ_1 holds in the first $i < k$ path states and Φ_2 holds in the $(i+1)$-th path state; and the *unbounded until formula* $\Phi_1 \ \mathrm{U} \ \Phi_2$ removes the bound k from the time-bounded until formula. The *next formula* $X\Phi$ holds if Φ is satisfied in the next state. The semantics of the probability $\mathcal{P}$ and reward $\mathcal{R}$ operators are defined as follows: $\mathcal{P}_{\sim p}[\Psi]$ specifies that the probability that paths starting at a chosen state s satisfy a path property Ψ is $\sim p$; $\mathcal{R}_{\sim r}[C^{\leq k}]$ holds if the expected cumulated reward up to time-step k is $\sim r$; and $\mathcal{R}_{\sim r}[F\Phi]$ holds if the expected reward cumulated before reaching a state satisfying Φ is $\sim r$. Replacing $\sim p$ (or $\sim r$) from (10) with '$=?$' specifies that the calculation of the probability (or reward) is required. We use the shorthand notation $pmc(\Phi, \mathcal{M})$ and $pmc(\Phi_R, \mathcal{M})$ for these quantities computed for the initial state s_0 of $\mathcal{M}$.

b) Discrete-event controller synthesis problem. To distinguish between different concerns of the autonomous system to be controlled, DEEPDECS organises each state s of the perfect-perception pDTMC model from Figure 10 into a tuple

$$s = (z, k, t, c), \tag{11}$$

where $z \in Z$ models the (internal) state of the system, $k \in [K]$ models the state of the environment, $c \in C$ models the control parameters of the system, and $t \in [3]$ is a "turn" flag. This flag indicates which elements of (11) can change in each pDTMC state:

$$\begin{aligned} &\forall s = (z,k,t,c), s' = (z',k',t',c') \in S : \\ &\quad ((t = 1 \ \wedge \ P(s,s') > 0) \implies k' = k \ \wedge \ c' = c \ \wedge \ t' < 3) \ \wedge \\ &\quad ((t = 2 \ \wedge \ P(s,s') > 0) \implies z' = z \ \wedge \ c' = c \ \wedge \ t' = 3) \ \wedge \\ &\quad ((t = 3 \ \wedge \ P(s,s') > 0) \implies z' = z \ \wedge \ k' = k \ \wedge \ t' = 1)\,. \end{aligned} \tag{12}$$

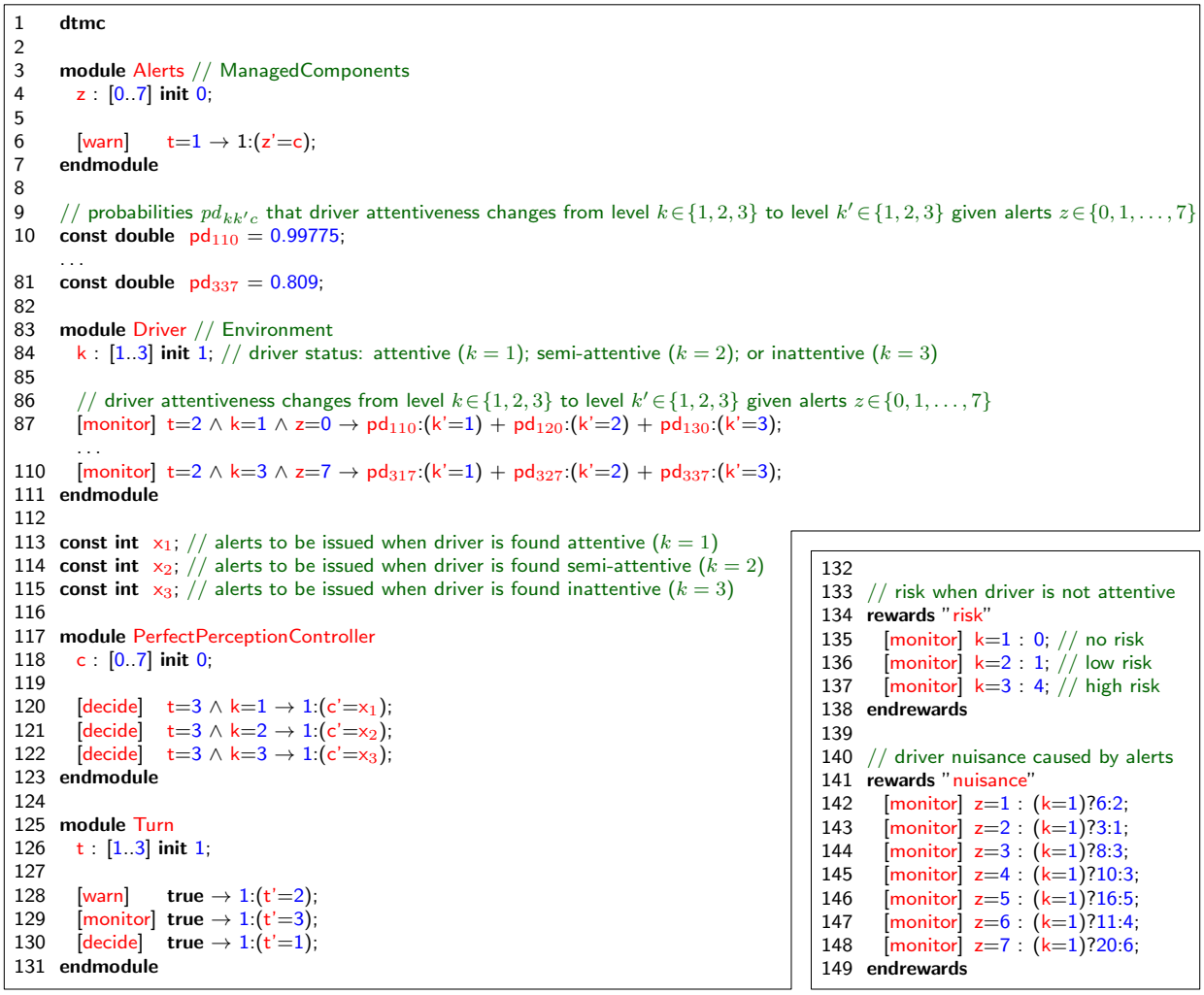
We partition the pDTMC state set into states in which the system can change, states in which the environment can change, and states in which it is the controller's "turn" to act for simplicity, and without loss of generality; the three types of states can be easily collapsed into one.

Finally, we assume that the outgoing transition probabilities from states $(z, k, 3, c) \in S$ are controller parameters that need to be determined and are given by

$$x_{zkcc'} = P((z,k,3,c),(z,k,1,c')) \tag{13}$$

for all $c' \in C$, where $x_{zkcc'} \in \{0,1\}$ for *deterministic controllers* or $x_{zkcc'} \in [0,1]$ for *probabilistic controllers*, and $\sum_{c' \in C} x_{zkcc'} = 1$.

The perfect-perception pDTMC model for the SafeSCAD system (shown in Figure 11) is defined in the high-level modelling language of the PRISM model checker [34]. In this language, the model of a system is specified by the parallel composition of a set of *modules*. The state of a *module* is given



```
1   dtmc
2
3   module Alerts // ManagedComponents
4     z : [0..7] init 0;
5
6     [warn]     t=1 → 1:(z'=c);
7   endmodule
8
9   // probabilities pd_{kk'c} that driver attentiveness changes from level k∈{1,2,3} to level k'∈{1,2,3} given alerts z∈{0,1,...,7}
10  const double  pd_110 = 0.99775;
    ...
81  const double  pd_337 = 0.809;
82
83  module Driver // Environment
84    k : [1..3] init 1; // driver status: attentive (k = 1); semi-attentive (k = 2); or inattentive (k = 3)
85
86    // driver attentiveness changes from level k∈{1,2,3} to level k'∈{1,2,3} given alerts z∈{0,1,...,7}
87    [monitor] t=2 ∧ k=1 ∧ z=0 → pd_110:(k'=1) + pd_120:(k'=2) + pd_130:(k'=3);
      ...
110   [monitor] t=2 ∧ k=3 ∧ z=7 → pd_317:(k'=1) + pd_327:(k'=2) + pd_337:(k'=3);
111 endmodule
112
113 const int  x_1; // alerts to be issued when driver is found attentive (k = 1)
114 const int  x_2; // alerts to be issued when driver is found semi-attentive (k = 2)
115 const int  x_3; // alerts to be issued when driver is found inattentive (k = 3)
116
117 module PerfectPerceptionController
118   c : [0..7] init 0;
119
120   [decide]   t=3 ∧ k=1 → 1:(c'=x_1);
121   [decide]   t=3 ∧ k=2 → 1:(c'=x_2);
122   [decide]   t=3 ∧ k=3 → 1:(c'=x_3);
123 endmodule
124
125 module Turn
126   t : [1..3] init 1;
127
128   [warn]     true → 1:(t'=2);
129   [monitor]  true → 1:(t'=3);
130   [decide]   true → 1:(t'=1);
131 endmodule
```

```
132
133 // risk when driver is not attentive
134 rewards "risk"
135   [monitor] k=1 : 0; // no risk
136   [monitor] k=2 : 1; // low risk
137   [monitor] k=3 : 4; // high risk
138 endrewards
139
140 // driver nuisance caused by alerts
141 rewards "nuisance"
142   [monitor] z=1 : (k=1)?6:2;
143   [monitor] z=2 : (k=1)?3:1;
144   [monitor] z=3 : (k=1)?8:3;
145   [monitor] z=4 : (k=1)?10:3;
146   [monitor] z=5 : (k=1)?16:5;
147   [monitor] z=6 : (k=1)?11:4;
148   [monitor] z=7 : (k=1)?20:6;
149 endrewards
```

Figure 11: Perfect-perception pDTMC model of the SafeSCAD system. The model states are tuples $(z, k, t, c) \in [7] \times [3]^2 \times \{0, 1, \ldots, 7\}$ with the semantics from (11). The Alerts module is responsible for warning the driver by "implementing" the controller-decided alerts c. The Driver module models the driver attentiveness level k, which is monitored every 4s; the probabilities of transition between attentiveness levels depend on the combination of alerts z in place. The control parameters $x_1, x_2, x_3 \in \{0, 1, \ldots, 7\}$ are binary encodings of the alerts to be activated for each of the three driver attentiveness levels, e.g., $x_3 = 5 = 101_{(2)}$ corresponds to a deterministic-controller decision to have the optical alert active, the acoustic alert inactive, and the haptic alert active when the driver is inattentive. The reward structures from lines 134–138 and 141–149 quantify the risk and driver nuisance associated with the different driver attentiveness levels and alert combinations, respectively. The expressions '$(k = 1)?value_1 : value_2$' from lines 142–148 evaluate to the larger $value_1$ if the driver is attentive (i.e., $k = 1$), and $value_2$ otherwise.

by a set of finite-range local variables, and its state transitions are specified by probabilistic guarded commands that change these variables:

$$[action] \quad guard \quad \rightarrow e_1 : update_1 + e_2 : update_2 + \ldots + e_m : update_N; \tag{14}$$

In this command, $guard$ is a boolean expression over the variables of all modules. If $guard$ evaluates to true, the arithmetic expression e_i, $i \in [m]$, gives the probability with which the $update_i$ change of the

module variables occurs. When $action$ is present, all modules comprising commands with this $action$ have to *synchronise*, i.e., to perform one of these commands simultaneously.

With this notation introduced so far, the *controller synthesis problem for the perfect-perception system* is to find the set of Pareto-optimal parameters (13) which ensure that the pDTMC satisfies $n_1 \geq 0$ PCTL-encoded *constraints* of the form in (10),

$$C_i ::= \Phi_i \ \mid \ \Phi_{Ri} \tag{15}$$

and Pareto-optimises $n_2 \geq 1$ PCTL-encoded *objectives* of the form

$$O_j ::= \text{maximise } pmc(\Phi_j, \mathcal{M}) \ \mid \ \text{minimise } pmc(\Phi_j, \mathcal{M}) \ \mid \ \text{maximise } pmc(\Phi_{Rj}), \mathcal{M} \ \mid \ \text{minimise } pmc(\Phi_{Rj}, \mathcal{M}) \tag{16}$$

where $i \in [n_1]$ and $j \in [n_2]$.

For the SafeSCAD system, the controller requirements comprise two constraints that limit the maximum expected risk and driver nuisance accrued over a 45-minute driving trip, and two optimisation objectives requiring that the same two measures are minimised:

$$\begin{aligned} C_1: &\quad \mathcal{R}^{\mathsf{risk}}_{\leq 100}[\mathrm{C}^{\leq 2000}] \\ C_2: &\quad \mathcal{R}^{\mathsf{nuisance}}_{\leq 6\times 10^3}[\mathrm{C}^{\leq 2000}] \\ O_1: &\quad \text{minimise } \mathcal{R}^{\mathsf{risk}}_{=?}[\mathrm{C}^{\leq 2000}] \\ O_2: &\quad \text{minimise } \mathcal{R}^{\mathsf{nuisance}}_{=?}[\mathrm{C}^{\leq 2000}] \end{aligned} \tag{17}$$

where each occurrence of the PCTL reward operator $\mathcal{R}$ is annotated with the name of the reward structure from Figure 11 it refers to (i.e., 'risk' of 'nuisance'). The 2000 time- steps from the PCTL cumulative reward properties correspond to the 45 minutes of the journey: verifying the driver state every 4s requires 667 verifications over 2667s, and each verification is modelled by three pDTMC time-steps, one for the monitoring of the driver state, one for the controller to decide the appropriate alerts for the observed state, and one for the decided alerts to be issued in order to warn the driver.

c) Model augmentation. The controller of an autonomous system with deep-learning perception does not have access to the true value k of the environment state from (11). Instead, DEEPDECS controllers need to operate with an estimate $\hat{k} \in [K]$ of this true value, and with the results $v = (v_1, v_2, \ldots, v_n) \in \mathbb{B}^n$ of $n \geq 0$ verification techniques (2) applied to the DNN and its input that produced the estimate $\hat{k}$. As such, the states $\hat{s}$ of a DEEPDECS *DNN-perception pDTMC model*

$$\hat{\mathcal{M}} = (\hat{S}, \hat{s_0}, \hat{P}, \hat{L}, \hat{R}) \tag{18}$$

are tuples that extend (11) with $\hat{k}$ and v:

$$\hat{s} = (z, k, \hat{k}, v, t, c). \tag{19}$$

To provide a formal definition for the derivation of the DEEPDECS DNN-perception pDTMC from the perfect-perception pDTMC, we use the notation $s(\hat{s}) = (z, k, t, c)$ to refer to the element from $Z \times [K] \times [3] \times C$ that corresponds to a generic element corresponding to $\hat{s} \in Z \times [K]^2 \times \mathbb{B}^n \times [3] \times C$. With this notation, the components of the pDTMC $\hat{\mathcal{M}}$ are obtained from the perfect-perception pDTMC $\mathcal{M} = (S, s_0, P, L, R)$ of the same autonomous system and the probabilities (7) as follows:

$$\hat{S} = \{\hat{s} \in Z \times [K]^2 \times \mathbb{B}^n \times [3] \times C \mid s(\hat{s}) \in S\}; \tag{20}$$

$$\hat{s}_0 = (z_0, k_0, k_0, \mathsf{true}, \ldots, \mathsf{true}, t_0, c_0), \tag{21}$$

where $(z_0, k_0, t_0, c_0) = s_0$; and, for any states $\hat{s} = (z, k, \hat{k}, v, t, c)$, $\hat{s}' = (z', k', \hat{k}', v', t', c') \in \hat{S}$,

$$\hat{P}(\hat{s}, \hat{s}') = \begin{cases} P(s(\hat{s}), s(\hat{s}')), & \text{if } t = 1 \wedge (\hat{k}', v') = (\hat{k}, v) \\ P(s(\hat{s}), s(\hat{s}')) \cdot p_{k'\hat{k}'v'}, & \text{if } t = 2 \\ x_{z\hat{k}vcc'}, & \text{if } t = 3 \ \wedge (z', k', \hat{k}', v', t') \\ & \qquad = (z, k, \hat{k}, v, 1) \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

where $x_{z\hat{k}vcc'}$ are controller parameters associated with state pairs $((z, k, \hat{k}, v, 3, c), (z, k, \hat{k}, v, 3, c')) \in \hat{S}^2$ such that $x_{z\hat{k}vcc'} \in \{0, 1\}$ for deterministic controllers or $x_{z\hat{k}vcc'} \in [0, 1]$ for probabilistic controllers, and $\sum_{c' \in C} \ x_{z\hat{k}vcc'} = 1$. Finally, for any state $\hat{s} \in \hat{S}$,

$$\hat{L}(\hat{s}) = L(s(\hat{s})), \tag{23}$$

and

$$\begin{aligned} \hat{R} = \{(\hat{\rho}, \hat{\iota}) \in (\hat{S} \to \mathbb{R}_{\geq 0}) \times (\hat{S} \times \hat{S} \to \mathbb{R}_{\geq 0}) \mid \\ \exists (\rho, \iota) \in R : \big(\forall \hat{s} \in \hat{S} : \hat{\rho}(\hat{s}) = \rho(s(\hat{s}))\big) \wedge \\ \big(\forall \hat{s}, \hat{s}' \in \hat{S} : \hat{\iota}(\hat{s}, \hat{s}') = \iota(s(\hat{s}), s(\hat{s}'))\big)\} \end{aligned} \tag{24}$$

The following result shows that the DEEPDECS module augmentation produces a valid pDTMC,[6] and Figure 12 depicts this pDTMC for the SafeSCAD system (for the scenario when the verification method $verif_1$ from (3) was used for the DNN uncertainty quantification).

Theorem 1. *The tuple* (18) *with the elements defined by* (20)–(24) *is a valid pDTMC that satisfies the following variant of* (12)*:*

$$\begin{aligned} &\forall \hat{s} = (z, k, \hat{k}, v, t, c), \hat{s}' = (z', k', \hat{k}', v', t', c') \in \hat{S} : \\ &\quad \big((t = 1 \ \wedge \ P(\hat{s}, \hat{s}') > 0) \implies (k', \hat{k}', v', c') = (k, \hat{k}, v, c) \ \wedge \ t' < 3\big) \wedge \\ &\quad \big((t = 2 \ \wedge \ P(\hat{s}, \hat{s}') > 0) \implies (z', c') = (z, c) \ \wedge \ t' = 3\big) \wedge \\ &\quad \big((t = 3 \ \wedge \ P(\hat{s}, \hat{s}') > 0) \implies (z', k', \hat{k}', v') = (z, k, \hat{k}, v) \ \wedge \ t' = 1\big). \end{aligned} \tag{25}$$

Importantly, the next result shows that the controller decisions do not depend on the true state k of the environment.

Theorem 2. *For any* $(z, k_1, \hat{k}, v, 3, c), (z, k_2, \hat{k}, v, 3, c) \in \hat{S}$ *and any control parameters* $c' \in C$*,*

$$\begin{aligned} &\hat{P}((z, k_1, \hat{k}, v, 3, c), (z, k_1, \hat{k}, v, 1, c')) \\ &\qquad = \hat{P}((z, k_2, \hat{k}, v, 3, c), (z, k_2, \hat{k}, v, 1, c')). \end{aligned} \tag{26}$$

Finally, the following theorem and its corollaries prove that for each (probabilistic) discrete-event controller that satisfies constraints (15) and Pareto-optimises objectives (16) for the autonomous system with DNN perception there is an equivalent (probabilistic) discrete-event controller for the autonomous system with perfect perception, but the converse does not hold.

Theorem 3. *Let* $\underline{x}$ *and* $\underline{\hat{x}}$ *be valid instantiations of the perfect-perception controller parameters* $\big\{x_{zkcc'} \in [0, 1] \ \big| \ (\exists k \in [K].(z, k, 3, c) \in S) \ \wedge \ c' \in C\big\}$ *from* (13) *and of the DNN-perception controller parameters* $\big\{x_{z\hat{k}vcc'} \in [0, 1] \ \big| \ (\exists k \in [K].(z, k, \hat{k}, v, 3, c) \in \hat{S}) \wedge c' \in C\big\}$ *from* (22)*, respectively. Also, let* $\mathcal{M}_{\underline{x}}$ *and* $\hat{\mathcal{M}}_{\underline{\hat{x}}}$

[6] The theorem proofs are provided in Appendix D.

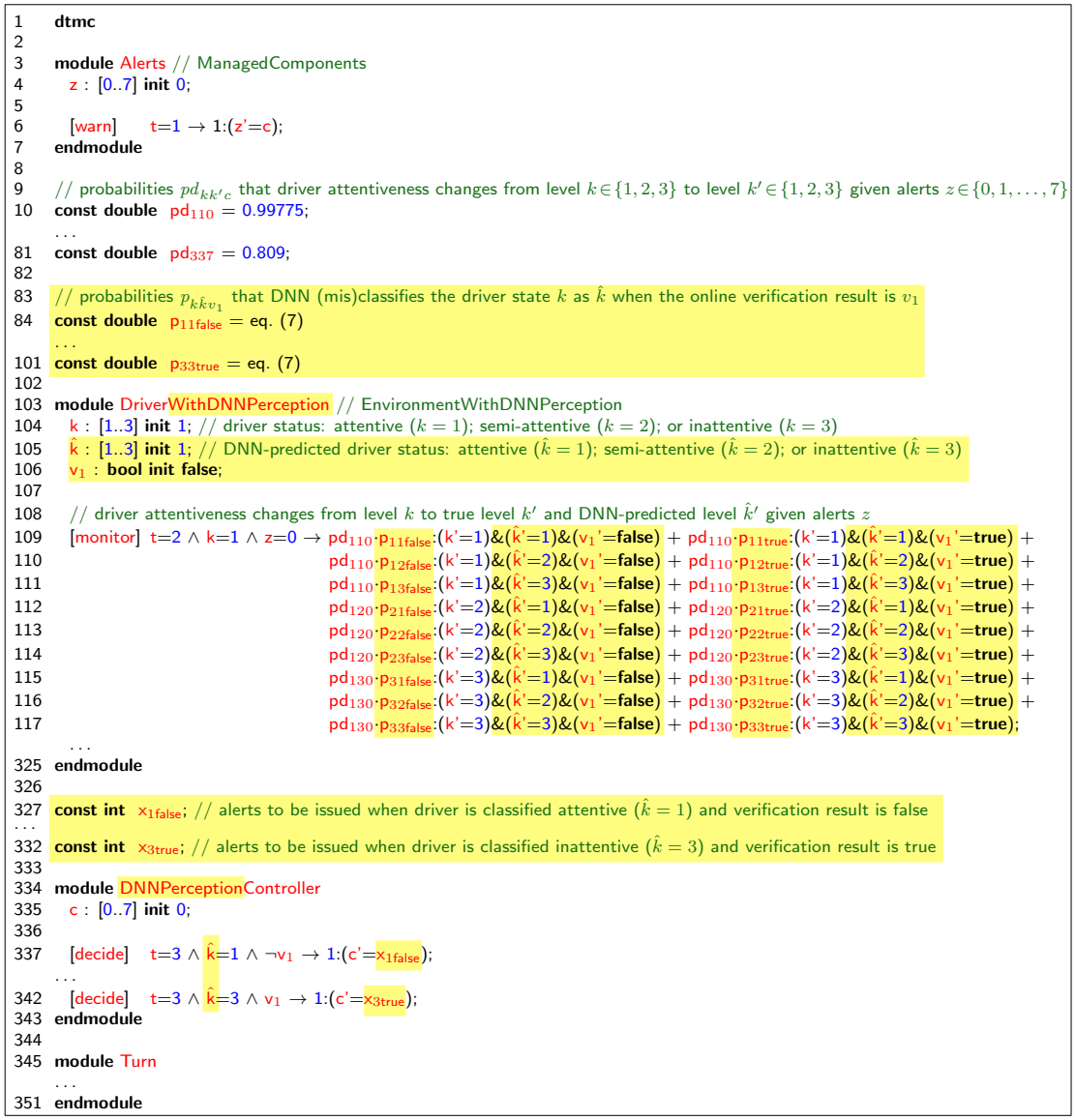


Figure 12: DNN-perception pDTMC model of the SafeSCAD driver-attentiveness management system for the scenario when a single DNN verification method is used to distinguish between "verified" ($v_1 = \text{true}$) and "unverified" ($v_1 = \text{false}$) DNN predictions of the driver's attentiveness level. Lines 103–111 show how all combinations of true (k') and DNN-predicted ($\hat{k}'$) driver attentiveness levels can be reached from the attentive driver state ($k = 1$) when no alerts are used ($c = 0$). The six-parameter deterministic controller decides a combination of alerts c for each pair of DNN-predicted driver attentiveness level $\hat{k}$ and online DNN verification result v_1 (lines 339-344). The Switch module and the two reward structures from the pDTMC in Figure 11 are omitted for brevity.

be the instances of the perfect-perception pDTMC $\mathcal{M}$ and DNN-perception pDTMC $\hat{\mathcal{M}}$ corresponding to the controller parameters $\underline{x}$ and $\underline{\hat{x}}$, respectively. With this notation, we have

$$pmc(\Phi, \hat{\mathcal{M}}_{\underline{\hat{x}}}) = pmc(\Phi, \mathcal{M}_{\underline{x}}), \tag{27}$$

and

$$pmc(\Phi_R, \hat{\mathcal{M}}_{\underline{\hat{x}}}) = pmc(\Phi_R, \mathcal{M}_{\underline{x}}), \tag{28}$$

for any (quantitative) PCTL state formula Φ and reward state formula Φ_R if and only if

$$x_{zkcc'} = \sum_{\hat{k} \in [K]} \sum_{v \in \mathbb{B}^n} p_{k\hat{k}v} x_{z\hat{k}vcc'} \tag{29}$$

for all $(z, k, 3, c) \in S$ and $c' \in C$.

Properties (27) and (28) imply that any constraint (15) is either satisfied or violated by both $\mathcal{M}_{\underline{x}}$ and $\hat{\mathcal{M}}_{\underline{\hat{x}}}$ (since the two DTMCs yield the same value for the system property associated with the constraint). Likewise, $\mathcal{M}_{\underline{x}}$ and $\hat{\mathcal{M}}_{\underline{\hat{x}}}$ are guaranteed to achieve the same value for the system property associated with any optimisation objective (16).

Corollary 1. *For any combination of constraints* (15) *and optimisation objectives* (16) *for which there exists a probabilistic DNN-perception controller that satisfies the constraints, there exists also a probabilistic perfect-perception controller that satisfies the same constraints and yields the same values for the PCTL properties from the optimisation objectives.*

Corollary 2. *There exist an infinite number of combinations of constraints* (15) *and optimisation objectives* (16) *for which there exists a probabilistic perfect-perception controller that satisfies the constraints, and no DNN-perception controller exists that satisfies the constraints and yields the same values for the system properties from the optimisation objectives.*

Corollary 2 shows that the decision-making capabilities of infinitely many perfect-perception controllers cannot be replicated by DNN-perception controllers. While this does not indicate how many of these practically unachievable controllers satisfy constraints (15) and Pareto-optimise objectives (16), the proof of the corollary provides a hint about this by showing that DNN-perception controllers do not exist for large $\alpha_{zcc'}$ values, i.e., for scenarios when the perfect-perception controller decides to use a specific configuration c' with high probability. Intuitively, these scenarios are highly relevant, i.e., many perfect-perception controllers with no equivalent DNN-perception controllers are likely to be Pareto-optimal. For instance, the perfect-perception controller used for the mobile robot application from the next section decides that the robot should mostly or even always wait when a collision with another mobile agent would otherwise occur. This line of reasoning also implies that deterministic perfect-perception controllers are likely to not have equivalent (probabilistic or deterministic) DNN-perception controllers.

Stage 3: Controller synthesis. The *controller synthesis problem for the DNN-perception system* involves finding instantiations $\underline{\hat{x}}$ of the DNN-perception controller parameters for which the pDTMC $\hat{\mathcal{M}}$ from (18) satisfies the constraints (15) and is Pareto optimal with respect to the optimisation objectives (16). Solving the general version of this problem precisely is unfeasible. However, metaheuristics such as multi-objective genetic algorithms for probabilistic model synthesis [7, 24] can be used to generate close approximations of the Pareto-optimal controller set. Alternatively, exhaustive search can be employed to synthesise the actual Pareto-optimal controller set for systems with deterministic controllers and small numbers of controller parameters, or—by discretising the search space—an approximate Pareto-optimal controller set for systems with probabilistic controllers.

For the SafeSCAD system, the DNN verification methods $verif_1$ and $verif_2$ from (3) and (4) were used in all possible combinations (i.e., alone, together, and neither) in the DEEPDECS DNN uncertainty quantification stage. Figure 13 compare the controller Pareto fronts obtained for all these combinations to the Pareto front associated with the perfect-perception model from Figure 11.

The search space (8^{12} controller parameter combinations when both DNN verification methods are used) is very large. As such, an exhaustive search to determine the Pareto-optimal controllers is infeasible. Therefore, the probabilistic model synthesis tool EvoChecker [24], which adopts multi-objective genetic algorithms, was employed to generate close approximations of the Pareto-optimal

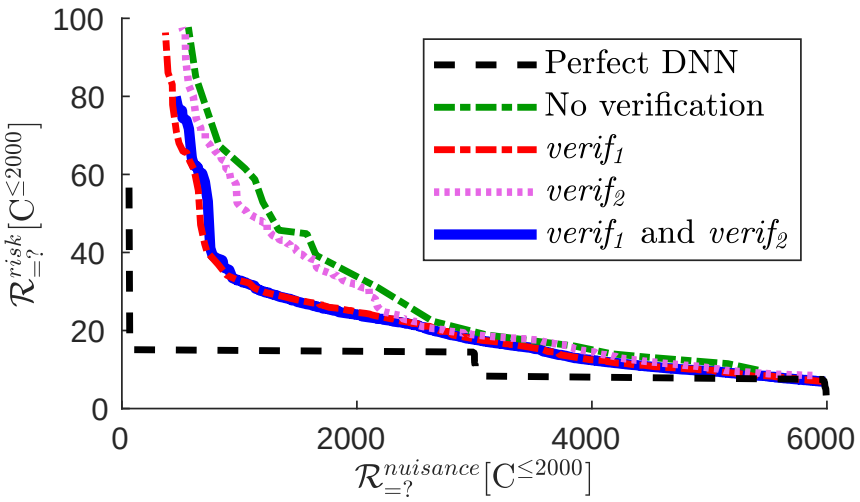


Figure 13: Pareto front associated with the set of Pareto-optimal SafeSCAD controllers

controllers. The Pareto fronts, see Figure 13 show that the inclusion of verification methods achieves Pareto-optimal controllers closer to the perfect-perception Pareto fronts. Furthermore, the knee point of the $verif_1$ and $verif_1$ and $verif_2$ fronts are closest to the knee point of the perfect DNN front. These two fronts share a similar frontier in general.

5 Safe-SCAD Demonstrator Evaluation

We conducted a user study to evaluate the integrated Safe-SCAD demonstrator system. The evaluation is structured around the following research questions:

- **RQ1**: How does Safe-SCAD affect driver takeover reaction time?
- **RQ2**: How does Safe-SCAD affect driver stress and cognitive workload?
- **RQ3**: What are driver perceptions (e.g., safety, disruptiveness, and urgency) about Safe-SCAD?

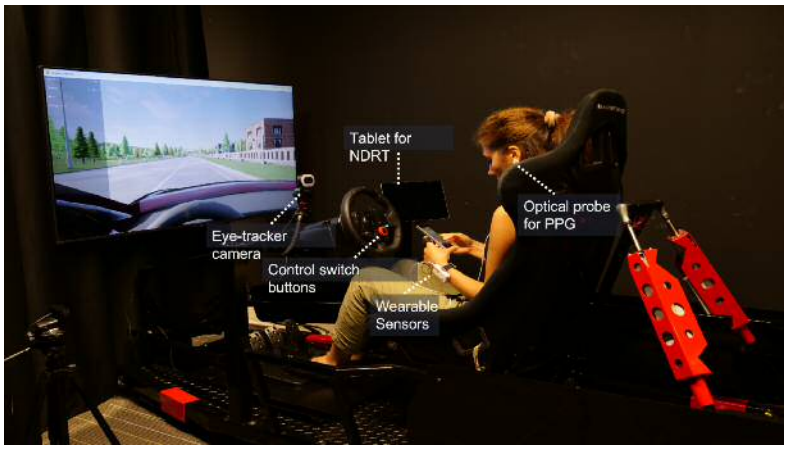


Figure 14: The driving simulator setup for the user study.

5.1 Apparatus and Data Collection

The study was conducted using the SimXperience Stage 5 Full Motion Racing Simulator (Figure 14). The virtual driving environment was simulated using CARLA [15] and projected on a 55-inch display placed about 63 inches away from the driver's seat. A 9.7-inch tablet display was mounted on the right side of the simulator as the infotainment system. Participants could switch between the automated and manual driving modes by press a button on the steering wheel (see Figure 14). In the automated mode, the simulated vehicle was equipped with SAE Level 3 automation and could issue TORs (350 Hz acoustic with 75 ms duration) to ask the driver to resume the control if it was not able to handle a situation by itself. In the manual driving mode, participants could control the vehicle via the steering wheel and pedals.

In this study, we collected drivers' psychophysiological, vehicle-related metrics, workload, and perceived safety. We used a Shimmer3+ wearable device to measure the driver's heart rate (PPG) and galvanic skin response (GSR) signals with a sampling rate of 256 Hz. Heart rate variability (the time elapsed between two successive R-waves) from PPG and maximum and mean phasic components were calculated as the objective metrics reflecting cognitive load variation and stress, respectively. Furthermore, we installed one high resolution camera above the steering wheel to monitor the driver's head and eye movement. Finally, we developed multiple APIs to forward all stream of data to iMotions biometric platform for the real-time aggregation and synchronization.

5.2 Experimental Design

5.2.1 Independent Factors

The study utilized a within-subject design to compare the Safe-SCAD demonstrator system with a baseline system. As illustrated in Figure 1, the Safe-SCAD system automatically triggers advisory

(a) (b)

Figure 15: Advisory warning modalities: (a) visual head-up-display, (b) vibrotactile.

warnings through the following process: (1) DeepTake continuously predicts driver takeover reaction time based on the sensing data, (2) the sensitivity and robustness of DeepTake predictions are verified, and (3) the Safe-SCAD controllers decides if an advisory warning should be triggered based on the verified predictions. Depending on the urgency of the situation (e.g., the predicted reaction time is slow and the verification results confirm the robustness of predictions), multiple warning modalities may be triggered simultaneously, including voice, visual head-up-display, and vibrotactile (see Figure 15). By contrast, the baseline system used the Wizard-of-Oz technique where the Wizard operator (i.e., experimenter) manually issued the warnings after observing the driver immersed in non-driving-related tasks (NDRTs) for certain period.

5.2.2 Dependent Measures

We used the following objective measurements and subjective feedback as dependent variables.

RQ1 questions driver takeover reaction time. We measured the time difference between the TOR initiation and the exact moment of the driver pressing the button on the steering wheel to resume manual control.

RQ2 evaluates driver stress and cognitive workload. We asked participants to complete the Driving Activity Load Index (DALI) [49], which customizes NASA-TLX for the automotive domain.

RQ3 inquires about driver perceptions. We asked participants to rate their perceived safety, disruptiveness, and the urgency of advisory warnings on a 5-point Likert-type scale ranging from 1 (strongly disagree) to 5 (strongly agree), which was adapted from the rating questionnaire used in the prior study by Iqbal et al. [30].

5.3 Procedure

Upon arrival, the participants were briefed about the study. Participants then signed an informed consent form and completed a demographics questionnaire, followed by a 5-minute practice drive to get familiar with the driving simulator and NDRTs. In this study, we considered three common NDRTs: (1) Reading a book out loud; (2) Having a conversation with the passenger; and (3) Watching a movie displayed on the tablet. We fitted the participant with the Shimmer3+ wearable device and calibrated the eye-tracker (which was re-calibrated at the beginning of each trial). Participants were informed that there was no need to actively monitor the driving environments or resume the control of the vehicle unless a TOR was issued. However, they were instructed to resume the vehicle control as

soon as a TOR was issued, then switch back to the automated driving once the incident had passed and continue the engagement with a NDRT. Each participant was asked to perform two experimental drives (Safe-SCAD and baseline), each containing 6 possible takeover events (2 TORs per NDRT). At the beginning of the drive, the participants were asked to activate the automated mode and perform a NDRT based on the experimenter's instructions, followed by two more NDRTs. At the end of each drive, the questionnaire on workload (DALI) and perceived safety and urgency were administered. The entire study took about 90 minutes.

5.4 Result Analysis

We analyzed the data collected from the user study for the proposed research questions. We set the statistical significance level as $\alpha = 0.05$.

5.4.1 Effects on Driver Takeover Reaction Time (RQ1)

Driver reaction to the hazardous situations is a critical component of road safety research. Understanding of driver reaction can help design safe automated vehicles and develop effective collision warning systems. In this study the time from receiving the takeover warning to the moment of relinquishing the vehicle control is considered reaction time. Thus, analysis of reaction time is an essential approach to gauge the effectiveness of the proposed method.

The descriptive analysis on the reaction time (see Fig 16) displays that driver's response throughout the Safe-SCAD trial (Mean=0.695sec, SD=0.34) was slightly better than the baseline (Mean=0.88, SD=0.37). In order to test the RQ1, we conducted t-test on the reaction time obtained in each experimental condition (baseline, and Safe-SCAD method) to see if there is difference among the methods. The results of the t-test ($t(46) = -1.77$, $p = 0.081$) showing no-significant difference between the two groups.

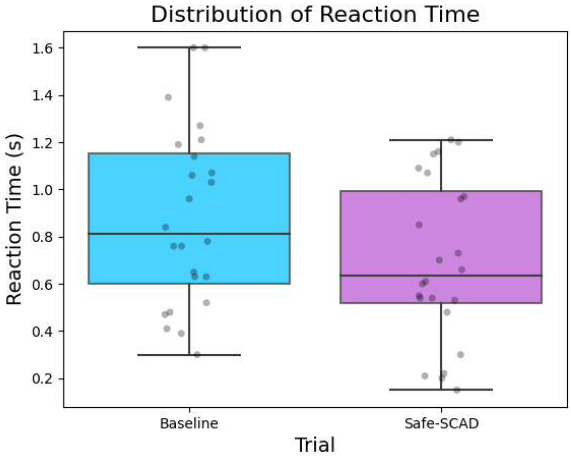


Figure 16: Results of reaction time w.r.t each trial.

5.4.2 Effects on Driver Stress and Cognitive Workload (RQ2)

We also analyzed the participants' subjective rating on DALI, which includes six dimensions of workload as shown in Figure 17. ANOVA analysis found that there were no significant effects on any workload dimensions. However, on average Safe-SCAD required more visual, auditory, tactile, and attention demand from the participants than the baseline. Furthermore, Safe-SCAD shows a much

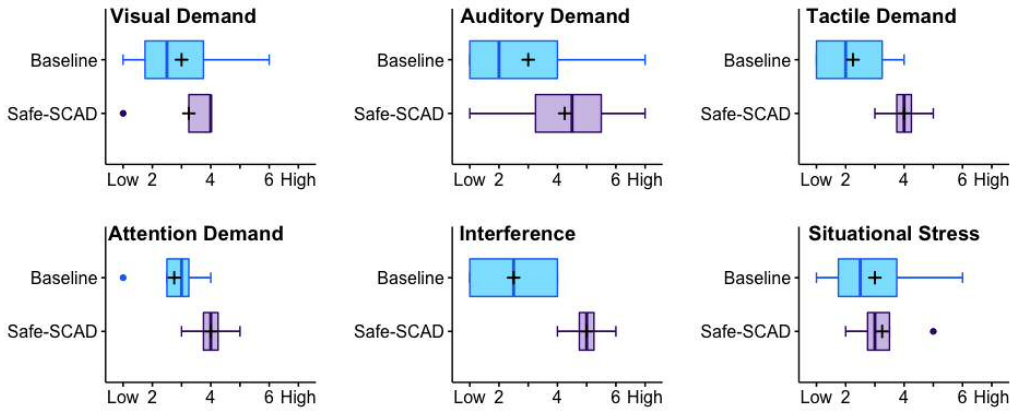


Figure 17: Results on DALI ratings about workload.

higher level of perceived interference (disturbance) than the baseline, but a similar level of situational stress.

5.4.3 Driver Perceptions (RQ3)

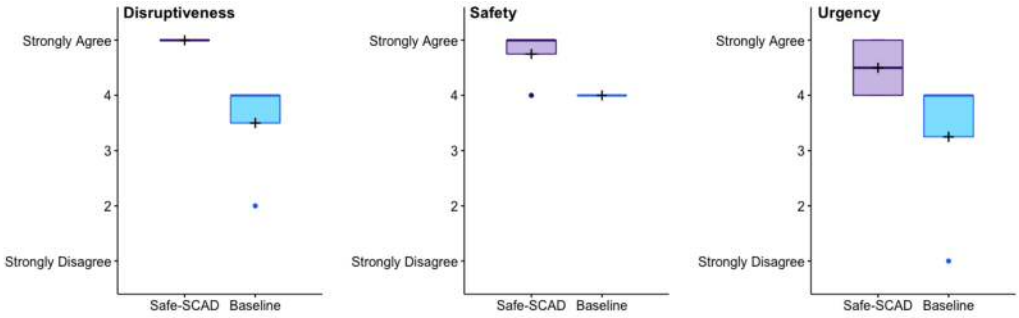


Figure 18: Results on driver perceived safety, disruptiveness, and urgency of advisory warnings.

Figure 18 shows the survey results on drivers' perceived safety, disruptiveness, and urgency of advisory warnings. The data does not show a significant difference between Safe-SCAD and the baseline. However, participants did report that Safe-SCAD was more disruptive of their non-driving tasks and its warnings appeared more urgent. Yet, Safe-SCAD was also perceived as making driving safer.

In the post-session interview, two participants indicated that they preferred Safe-SCAD, as it was more convenient and informative, while the other two indicated their preference for the baseline as Safe-SCAD was more disruptive. Yet, all four participants noted that Safe-SCAD was more useful during driving and three of the four participants noted Safe-SCAD was more effective during takeover requests.

References

[1] Suzana Andova, Holger Hermanns, and Joost-Pieter Katoen. Discrete-time rewards model-checked. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 88–104. Springer, 2003.

[2] Victoria A Banks, Katherine L Plant, and Neville A Stanton. Driver error or designer error: Using the perceptual cycle model to explore the circumstances surrounding the fatal tesla crash on 7th may 2016. *Safety science*, 108:278–285, 2018.

[3] Frauke L Berghöfer, Christian Purucker, Frederik Naujoks, Katharina Wiedemann, and Claus Marberger. Prediction of take-over time demand in conditionally automated driving-results of a real world driving study. In *Proceedings of the Human Factors and Ergonomics Society Europe Chapter 2018 Annual Conference*, pages 69–81, 2018.

[4] Andrea Bianco and Luca De Alfaro. Model checking of probabilistic and nondeterministic systems. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.

[5] Christian Braunagel, Wolfgang Rosenstiel, and Enkelejda Kasneci. Ready for take-over? a new driver assistance system for an automated classification of driver take-over readiness. *IEEE Intelligent Transportation Systems Magazine*, 9(4):10–22, 2017.

[6] California State Assembly. Assembly Bill 2866 Autonomous vehicles, February 2016.

[7] Radu Calinescu, Milan Ceska, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software*, 143:140 – 158, 2018.

[8] Radu Calinescu, Calum Imrie, Ravi Mangal, Corina Pasareanu, Misael Alpizar Santana, and Gricel Vázquez. Discrete-event controller synthesis for autonomous systems with deep-learning perception components, 2022.

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*, pages 39–57. IEEE, 2017.

[10] Centre for Connected and Autonomous Vehicles. Safe use of automated lane keeping system (ALKS). Technical report, UK Department for Transport, August 2020.

[11] Rifai Chai, Ganesh R Naik, Tuan Nghia Nguyen, et al. Driver fatigue classification with independent component by entropy rate bound minimization analysis in an eeg-based system. *IEEE Journal of Biomedical and Health Informatics*, 21(3):715–724, 2016.

[12] SAE On-Road Automated Vehicle Standards Committee et al. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International: Warrendale, PA, USA*, 2018.

[13] Conrado Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *International Colloquium on Theoretical Aspects of Computing*, pages 280–294, 2005.

[14] Nachiket Deo and Mohan M Trivedi. Looking at the driver/rider in autonomous vehicles to predict take-over readiness. *IEEE Transactions on Intelligent Vehicles*, 2019.

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[16] Na Du, Feng Zhou, Elizabeth Pulver, Dawn Tilbury, Lionel P Robert, Anuj K Pradhan, and X Jessie Yang. Predicting takeover performance in conditionally automated driving. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2020.

[17] Na Du, Feng Zhou, Elizabeth M Pulver, Dawn M Tilbury, Lionel P Robert, Anuj K Pradhan, and X Jessie Yang. Examining the effects of emotional valence and arousal on takeover performance in conditionally automated driving. *Transportation research part C: emerging technologies*, 112:78–87, 2020.

[18] Na Du, Feng Zhou, Elizabeth M Pulver, Dawn M Tilbury, Lionel P Robert, Anuj K Pradhan, and X Jessie Yang. Predicting driver takeover performance in conditionally automated driving. *Accident Analysis & Prevention*, 148:105748, 2020.

[19] Jeanne F Duffy, Kirsi-Marja Zitting, and Charles A Czeisler. The case for addressing operator fatigue. *Reviews of Human Factors and Ergonomics*, 10(1):29–78, 2015.

[20] Mahdi Ebnali, Kevin Hulme, Aliakbar Ebnali-Heidari, and Adel Mazloumi. How does training effect users' attitudes and skills needed for highly automated driving? *Transportation research part F: traffic psychology and behaviour*, 66:184–195, 2019.

[21] European Parliament. Regulation (EU) 2019/2144 of the European Parliament and of the Council on type-approval requirements for motor vehicles. *Official Journal of the European Union*, L 325/1, 2019.

[22] Francesca M Favarò, Nazanin Nader, Sky O Eurich, Michelle Tripp, and Naresh Varadaraju. Examining accident reports involving autonomous vehicles in California. *PLoS one*, 12(9):e0184952, 2017.

[23] Anna Feldhütter, Dominik Kroll, and Klaus Bengler. Wake up and take over! the effect of fatigue on the take-over performance in conditionally automated driving. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2080–2085. IEEE, 2018.

[24] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. Synthesis of probabilistic models for quality-of-service software engineering. *Automated Software Engineering*, 25(4):785–831, 2018.

[25] John M. Grese, Corina S. Păsăreanu, and Erfan Pakdamanian. Formal analysis of a neural network predictor in shared-control autonomous driving. *AIAA 2021-1580, Session: Human-Automation Interaction*, 2021.

[26] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. *CoRR*, abs/1706.04599, 2017.

[27] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1321–1330. JMLR.org, 2017.

[28] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[29] Takeyoshi Imai. Legal regulation of autonomous driving technology: Current conditions and issues in Japan. *IATSS Research*, 43(4):263–267, 2019.

[30] Shamsi T Iqbal, Eric Horvitz, Yun-Cheng Ju, and Ella Mathews. Hang on a sec! effects of proactive mediation of phone conversations while driving. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 463–472, 2011.

[31] Vaidehi Joshi, Aiswarya Vinod Kumar, Aman Mohanty, Sai Prahladh Padmanabhan, John Grese, Ravi Mangal, and Corina S. Păsăreanu. Safety of shared control in autonomous driving. *CMU Technical Report*, 2021.

[32] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 443–452, Cham, 2019. Springer International Publishing.

[33] Moritz Körber, Lorenz Prasch, and Klaus Bengler. Why do i have to drive now? post hoc explanations of takeover requests. *Human factors*, 60(3):305–323, 2018.

[34] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of the 23rd Int. Conf. on Computer Aided Verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[35] Jiwon Lee and Ji Hyun Yang. Analysis of driver's eeg given take-over alarm in sae level 3 automated driving in a simulated environment. *International journal of automotive technology*, 21(3):719–728, 2020.

[36] Alexander Lotz and Sarah Weissenberger. Predicting take-over times of truck drivers in conditional autonomous driving. In *International Conference on Applied Human Factors and Ergonomics*, pages 329–338. Springer, 2018.

[37] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[38] Querino Maia, Michael A Grandner, et al. Short and long sleep duration and risk of drowsy driving and the role of subjective sleep insufficiency. *Accident Analysis & Prevention*, 59:618–622, 2013.

[39] Gerald Matthews and Peter A Hancock. *The Handbook of Operator Fatigue*. CRC Press, 2017.

[40] Rod McCall, Fintan McGee, Alexander Mirnig, Alexander Meschtscherjakov, Nicolas Louveton, Thomas Engel, and Manfred Tscheligi. A taxonomy of autonomous vehicle handover situations. *Transportation research part A: policy and practice*, 124:507–522, 2019.

[41] Anthony D McDonald, Hananeh Alambeigi, Johan Engström, Gustav Markkula, Tobias Vogelpohl, Jarrett Dunne, and Norbert Yuma. Toward computational simulations of behavior during automated driving takeovers: a review of the empirical and modeling literatures. *Human factors*, 61(4):642–688, 2019.

[42] Natasha Merat and A Hamish Jamson. How do drivers behave in a highly automated car? In *5th International Driving Symposium on Human Factors in Driver Assessment, Training, and Vehicle Design: Driving Assessment 2009*. University of Iowa, 2009.

[43] Brian Mok, Mishel Johns, David Miller, and Wendy Ju. Tunneled in: Drivers with active secondary tasks need more time to transition from automation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2840–2844, 2017.

[44] Frederik Naujoks, Christoph Mai, and Alexandra Neukum. The effect of urgency of take-over requests during highly automated driving under distraction conditions. *Advances in Human Aspects of Transportation*, 7(Part I):431, 2014.

[45] On-Road Automated Driving (ORAD) committee. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Standard J3016_201806, SAE International, 2018.

[46] Erfan Pakdamanian, Lu Feng, and Inki Kim. The effect of whole-body haptic feedback on driver's perception in negotiating a curve. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 62, pages 19–23. SAGE Publications Sage CA: Los Angeles, CA, 2018.

[47] Erfan Pakdamanian, Nauder Namaky, Shili Sheng, Inki Kim, James Arthur Coan, and Lu Feng. Toward minimum startle after take-over request: A preliminary study of physiological data. In *12th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 27–29, 2020.

[48] Erfan Pakdamanian, Shili Sheng, Sonia Baee, Seongkook Heo, Sarit Kraus, and Lu Feng. DeepTake: Prediction of driver takeover behavior using multimodal data. In *ACM CHI Conference on Human Factors in Computing Systems*, 2021.

[49] Annie Pauzié. A method to assess the driver mental workload: The driving activity load index (dali). *IET Intelligent Transport Systems*, 2(4):315–322, 2008.

[50] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[51] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017.

[52] UNECE World Forum for Harmonization of Vehicle Regulations. ECE/TRANS/WP.29/2020/81: United Nations Regulation on Uniform provisions concerning the approval of vehicles with regard to Automated Lane Keeping Systems, June 2020.

[53] US National Transportation Safety Board. Collision between a sport utility vehicle operating with partial driving automation and a crash attenuator, February 2020.

[54] Jingyan Wan and Changxu Wu. The effects of vibration patterns of take-over request and non-driving tasks on taking-over control of automated vehicles. *International Journal of Human–Computer Interaction*, 34(11):987–998, 2018.

[55] Kathrin Zeeb, Axel Buchner, and Michael Schrauf. What determines the take-over time? an integrated model approach of driver take-over after automated driving. *Accident Analysis & Prevention*, 78:212–221, 2015.

[56] Kathrin Zeeb, Manuela Härtel, Axel Buchner, and Michael Schrauf. Why is steering not the same as braking? the impact of non-driving related tasks on lateral and longitudinal driver interventions during conditionally automated driving. *Transportation research part F: traffic psychology and behaviour*, 50:65–79, 2017.

[57] Bo Zhang, Joost de Winter, Silvia Varotto, Riender Happee, and Marieke Martens. Determinants of take-over time from automated driving: A meta-analysis of 129 studies. *Transportation research part F: traffic psychology and behaviour*, 64:285–307, 2019.

Appendix A DeepTake: Prediction of Driver Takeover Behavior Using Multimodal Data

DeepTake: Prediction of Driver Takeover Behavior using Multimodal Data

Erfan Pakdamanian
School of Engineering
University of Virginia
ep2ca@virginia.edu

Shili Sheng
School of Engineering
University of Virginia
ss7dr@virginia.edu

Sonia Baee
School of Engineering
University of Virginia
sb5ce@virginia.edu

Seongkook Heo
School of Engineering
University of Virginia
seongkook@virginia.edu

Sarit Kraus
Department of Computer Science
Bar-Ilan University
sarit@cs.biu.ac.il

Lu Feng
School of Engineering
University of Virginia
lu.feng@virginia.edu

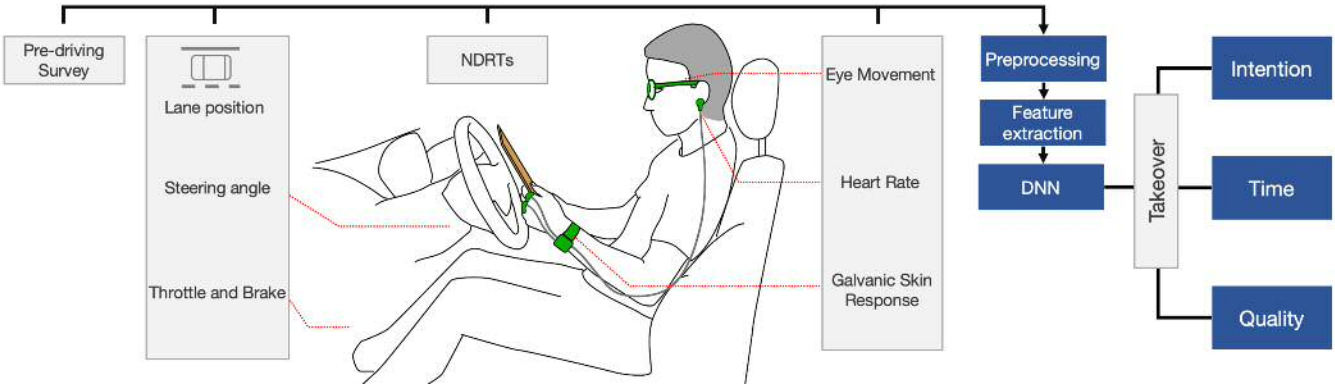


Figure 1: DeepTake uses data from multiple sources (pre-driving survey, vehicle data, non-driving related tasks (NDRTs) information, and driver biometrics) and feeds the preprocessed extracted features into deep neural network models for the prediction of takeover intention, time and quality.

ABSTRACT

Automated vehicles promise a future where drivers can engage in non-driving tasks without hands on the steering wheels for a prolonged period. Nevertheless, automated vehicles may still need to occasionally hand the control back to drivers due to technology limitations and legal requirements. While some systems determine the need for driver takeover using driver context and road condition to initiate a takeover request, studies show that the driver may not react to it. We present DeepTake, a novel deep neural network-based framework that predicts multiple aspects of takeover behavior to ensure that the driver is able to safely take over the control when engaged in non-driving tasks. Using features from vehicle data, driver biometrics, and subjective measurements, DeepTake predicts the driver's intention, time, and quality of takeover. We evaluate DeepTake performance using multiple evaluation metrics. Results show that DeepTake reliably predicts the takeover intention, time, and quality, with an accuracy of 96%, 93%, and 83%, respectively. Results also indicate that DeepTake outperforms previous state-of-the-art methods on predicting driver takeover time and quality. Our findings have implications for the algorithm development of driver monitoring and state detection.

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445563

CCS CONCEPTS

• **Human-centered computing** → *Empirical studies in HCI*.

KEYWORDS

Automated driving; Multimodal data; Takeover behavior; Human-automation interaction; Deep neural networks

ACM Reference Format:
Erfan Pakdamanian, Shili Sheng, Sonia Baee, Seongkook Heo, Sarit Kraus, and Lu Feng. 2021. DeepTake: Prediction of Driver Takeover Behavior using Multimodal Data. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3411764.3445563

1 INTRODUCTION

The rapid development of autonomous driving technologies promises a future where drivers can take their hands off the steering wheels and instead engage in non-driving related tasks (NDRTs) such as reading or using mobile devices. Incorporating cameras, sensors, global positioning systems (GPS), adaptive cruise control, light detection and ranging, and advanced driver assistance systems, automated vehicles (AVs) can navigate automatically. In Level 3 of autonomy (i.e., conditionally automated driving), as defined by the Society of Automotive Engineers (SAE international [9]), the driver does not need to continuously monitor the driving environment. Nevertheless, due to current technology limitations and legal restrictions, AVs may still need to handover the control back to drivers occasionally (e.g., under challenging driving conditions beyond the automated systems' capabilities) [39]. In such cases, AVs would initiate takeover requests (TORs) and alert drivers via auditory, visual, or vibrotactile modalities [44, 47, 61] so that the drivers can resume manual driving in a timely manner. However, there are challenges in making drivers safely take over control. Drivers may need a longer time to shift their attention back to driving in some situations, such as when they have been involved in NDRTs for a prolonged time [68] or when they are stressed or tired [22]. Even if TORs are initiated with enough time for a driver to react, it does not guarantee that the driver will safely take over [40]. Besides, frequent alarms could startle and increase drivers' stress levels leading to detrimental user experience in AVs [33, 34, 48]. These challenges denote the need for AVs to constantly monitor and predict driver behavior and adapt the systems accordingly to ensure a safe takeover.

The vast majority of prior work on driver takeover behavior has focused on the empirical analysis of high-level relationships between the factors influencing takeover time and quality (e.g., [16, 18, 43, 69]). More recently, the prediction of driver takeover behavior using machine learning approaches has been drawing increasing attention. However, only a few studies have focused on the prediction of either takeover time [2, 35] or takeover quality [4, 11, 15, 17]; and their obtained accuracy results (ranging from 61% to 79%) are insufficient for the practical implementation of real-world applications. This is partly due to the fact that takeover prediction involves a wide variety of factors (e.g., drivers' cognitive and physical states, vehicle states, and the contextual environment) that could influence drivers' takeover behavior [66].

In this paper on the other hand, we present a novel approach, named **DeepTake**, to address these challenges by providing reliable predictions of multiple aspects of takeover behavior. **DeepTake** is a unified framework for the prediction of driver takeover behavior in three aspects: (1) *takeover intention* – whether the driver would respond to a TOR; (2) *takeover time* – how long it takes for the driver to resume manual driving after a TOR; and (3) *takeover quality* – the quality of driver intervention after resuming manual control. As illustrated in Figure 1, DeepTake considers multimodal data from various sources, including driver's pre-driving survey response (e.g., gender, baseline of cognitive workload and stress levels), vehicle data (e.g., lane position, steering wheel angle, throttle/brake pedal angles), engagement in NDRTs, and driver biometrics (e.g., eye movement for detecting visual attention, heart rate and galvanic skin responses for the continuous monitoring of workload and stress levels). This data can easily be collected in AVs' driving environment. For instance, all of the driver biometrics utilized in DeepTake can be captured by wearable smartwatches and deployed eye-tracking systems. The multitude of sensing modalities and data sources offer complementary information for the accurate and highly reliable prediction of driver takeover behavior. DeepTake extracts meaningful features from the preprocessed multimodal data, and feeds them into deep neural network (DNN) models with mini-batch stochastic gradient descent. We built and trained different DNN models (which have the same input and hidden layers, but different output layers and weights) for the prediction of takeover behavior: intention, time and quality. We validate DeepTake framework feasibility using data collected from a driving simulator study. Finally, we evaluate the performance of our DNN-based framework with six machine learning-based models on prediction of driver takeover behavior. The results show that DeepTake models significantly outperform six machine learning-based models in all predictions of takeover intention, time and quality. Specifically, DeepTake achieves an accuracy of 96% for the binary classification of takeover intention, 93%, and 83% accuracy for multi-class classification of takeover time and quality, respectively. These accuracy results also outperform results reported in the existing work.

The main contribution of this work is the development of DeepTake framework that predicts driver takeover intention, time and quality using vehicle data, driver biometrics and subjective measurements[1]. The intersection between ubiquitous computing, sensing and emerging technologies offers promising avenues for DeepTake to integrate modalities into a novel human-centered framework to increase the robustness of drivers' takeover behavior prediction. We envision that DeepTake can be integrated into future AVs, such that the automated systems can make optimal decisions based on the predicted driver takeover behavior. For example, if the predicted takeover time exceeds the duration that the vehicle can detect situations requiring TORs, or the predicted takeover quality is too low to respond to TORs, the automated systems can warn the driver to engage less with the NDRT. In other words, DeepTake facilitates drivers to be distracted as long as they can properly respond and safely maneuver the vehicle. The reliable prediction of driver takeover behavior provided by DeepTake framework would not only improve the safety of AVs, but also improve drivers' user experience and productivity in AVs (e.g., drivers can focus on NDRTs without worrying about missing any TORs and potential tragic circumstances). We believe that our work makes a step towards enabling NDRTs in automated driving, and helps HCI researchers and designers to create user interfaces and systems for AVs that adapt to the drivers' context.

2 RELATED WORK

We discuss prior work on the analysis of takeover time and quality, and position our work in the context of state-of-the-art takeover behavior prediction research.

[1]DeepTake framework configurations, implementation details and code are available at https://github.com/erfpak7/DeepTake

Takeover time. In this paper, we consider the *takeover time* as the period of time from the initiation of TOR to the exact moment of the driver resuming manual control (see Figure 3), following the ISO standard definition in [30]. Note that the same concept has also sometimes been named as takeover reaction time or response time in the literature (e.g., [20, 31, 32, 51]). The empirical literature defines a large variety of takeover time from a mean of 0.87s to brake [63], to an average of 19.8s to response to a countdown TOR [52] and 40s to stabilize the vehicle [42]. This range is derived from influential factors impacting perception, cognitive processing, decision-making and resuming readiness [25, 66]. A meta-analysis of 129 studies by Zhang et al. [69] found that a shorter takeover time is associated with the following factors: a higher urgency of the driving situation, the driver not performing a non-driving related task (NDRT) such as using a handheld device, the driver receiving an auditory or vibrotactile TOR rather than no TOR or a visual-only TOR. Recent studies by Mok et al. [43] and Eriksson et al. [20] both confirmed that drivers occupied by NDRTs have higher responses to TORs. Similarly, [21] found a significant increase in reaction time induced by NDRTs. It is further concluded that the visual distraction causes higher reaction time when it is loaded with cognitive tasks [56]. Studies have also revealed several driving environments, TOR modalities [56, 57], driving expectancy [54], age [60] and gender [62] associated with takeover time. The present study extend previous findings by considering various NDRTs, gender, and objective and subjective measurements of mental workload into the DeepTake framework.

Takeover quality. In addition to takeover time, it is essential to assess the *takeover quality*, which is defined as the quality of driver intervention after resuming manual control [30]. There are a variety of takeover quality measures, depending on different takeover situations (e.g., collision avoidance, lane-keeping), including objective measures (e.g., mean lateral position deviation, steering wheel angle deviation, metrics of distance to other vehicles or objects, minimum time to collision, frequency of emergency braking) and subjective measures (e.g., expert-based assessment, self-reported experience). Prior work has found that takeover quality can be influenced by factors such as drivers' cognitive load [14, 67], emotions and trust [12, 16, 28], and distraction of secondary NDRTs [13, 38]. Takeover time to an obstacle[67] has been used widely studies as an indicator of takeover performance [20]. However, a study by Louw et al. [36] showed that takeover time and quality appear to be independent. This lack of consensus could be due to the fact that studies apply various time budget for takeover control.

Takeover prediction. While existing literature mostly focus on the empirical analysis of drivers' takeover time and quality, there are a few recent efforts on the predication of drivers' takeover behavior using machine learning (ML) approaches. Lotz and Weissenberger [35] applied a linear support vector machine (SVM) method to classify takeover time with four classes, using driver data collected with a remote eye-tracker and body posture camera; the results achieve an accuracy of 61%. Braunagel et al. [4] developed an automated system that can classify the driver's takeover readiness into two levels of low and high (labeled by objective driving parameters related to the takeover quality); their best results reached an overall accuracy of 79% based on a linear SVM classifier, using features including the traffic situation complexity, the driver's gazes on the road and NDRT involvement. Deo and Trivedi [11] proposed a Long Short Term Memory (LSTM) model for continuous estimation of the driver's takeover readiness index (defined by subjective ratings of human observers viewing the feed from in-vehicle vision sensors), using features representing the driver's states (e.g., gaze, hand, pose, foot activity); their best results achieve a mean absolute error (MAE) of 0.449 on a 5 point scale of the takeover readiness index. Du et al. [15, 17] developed random forest models for classifying drivers' takeover quality into two categories of good and bad (given by subjective self-reported ratings), using drivers' physiological data and environment parameters; their best model achieves an accuracy of 70%.

In summary, the existing works only focus on the prediction of either takeover time or takeover quality. By contrast, DeepTake provides a unified framework for the prediction of all three aspects of takeover behavior: intention, time and quality together. Furthermore, DeepTake achieves better accuracy results: 96% for takeover intention (binary classification), 93% for takeover time (three classes), and 83% for takeover quality (three classes).

3 DEEPTAKE: A NEW APPROACH FOR TAKEOVER BEHAVIOR PREDICTION

In this section, we present a novel deep neural network (DNN)-based approach, DeepTake, for the prediction of a driver's takeover behavior (i.e., intention, time, quality). Figure 1 illustrates an overview of DeepTake. First, we collect multimodal data such as driver biometrics, pre-driving survey, types of engagement in non-driving related tasks (NDRTs), and vehicle data. The multitude of sensing modalities and data streams offers various and complementary means to collect data that will help to obtain a more accurate and robust prediction of drivers' takeover behavior. Second, the collected multimodal data are preprocessed followed by segmentation and feature extraction. The extracted features are then labeled based on the belonging takeover behavior class. In our framework, we define each aspect of takeover behavior as a classification problem (i.e., takeover intention as a binary classes whereas takeover time and quality as three multi-classes). Finally, we build DNN-based predictive models for each aspect of takeover behavior. DeepTake takeover predictions can potentially enable the vehicle autonomy to adjust the timely initiation of TORs to match drivers' needs and ultimately improve safety. We describe the details of each step as follows.

3.1 Multimodal Data Sources

3.1.1 Driver Biometrics. The prevalence of wearable devices has made it easy to collect various biometrics for measuring drivers' cognitive and physiological states. Specifically, we consider the following three types of driver biometrics in DeepTake.

Eye movement. Drivers are likely to engage in non-driving tasks when the vehicle is in the automated driving mode [3, 48, 64]. Therefore, it is important to assess the drivers' visual attention and takeover readiness before the initiation of TORs. There is a proven high correlation between a driver's visual attention and eye movement [1, 65, 66]. DeepTake uses eye movement data (e.g., gaze position, fixation duration on areas of interest) measured by eye-tracker devices. We utilize a pair of eye-tracking glasses in our

user study (see Section 4). But the aforementioned eye movement data can be captured with any eye-tracking device.

Heart rate. Studies have found that *heart rate variability* (HRV), fluctuation of heart rate in the time intervals between the nearby beats, is a key factor associated with drivers' workload [49], stress [12], and drowsiness [59]. DeepTake uses features extracted from HRV analysis for monitoring drivers' situational awareness and readiness to respond to TORs. Heart rate can be measured in many different ways, such as checking the pulse or monitoring physiological signals. DeepTake employes photoplethysmographic (PPG) signal, which can be collected continuously via PPG sensors commonly embedded in smartwatches. PPG sensors monitor heart rate by the emission of infrared light into the body and measure the reflection back to estimate the blood flow. Unlike some heart rate monitoring devices that rely on the placement of metal electrodes on the chest, PPG sensors provide accurate heart rate measures without requiring intrusive body contact. Therefore, a PPG signal is preferred for monitoring drivers' heart rate.

Galvanic skin response (GSR). Along with HRV, GSR has been identified as another significant indicator of drivers' stress and workload [12, 23, 41, 53]. A GSR signal measures the skin conduction ability. Drivers' emotional arousal (e.g., stress) can trigger sweating on the hand, which can be detected through distinctive GSR patterns. DeepTake incorporates features extracted from the GSR signal for monitoring drivers' stress levels. GSR sensors are also embedded in many wearable devices, including smartwatches.

3.1. 2Pre-Driving Survey. In addition to the objective measurements of driver biometrics, DeepTake exploits subjective pre-driving survey responses, because drivers' prior experience and background may influence their takeover behavior [69]. However, any subjective rating of factors affecting a driver's cognitive and physical ability as well as driving experience prepare a complete specification of objective metrics, potentially enhancing the distinctive attributes of an algorithm. DeepTake framework exerts demographic information, NASA-Task Load Index (NASA-TLX) [27], and the 10-item Perceived Stress Scale (PSS-10) [7] to measure drivers' perceived workload and psychological stress. In our user study (see Section 4), we asked participants to fill in questionnaires at the beginning of each trial.

3.1.3 Non-Driving Related Tasks (NDRTs). As described in Section 2, prior studies have found that engaging in NDRTs can undermine drivers' takeover performance. Diverse NDRTs require different levels of visual, cognitive and physical demands; thus, the influence varies when drivers are asked to interrupt the secondary task and resume manual control of the vehicle. DeepTake accounts for the impact of different NDRTs on the prediction of drivers' takeover behavior. In our user study, we considered four NDRTs in which drivers are very likely to engage in automated vehicles: (1) *having a conversation with passengers*, (2) *using a cellphone*, (3) *reading*, and (4) *solving problems* such as simple arithmetic questions (more details in Section 4.3). We chose these NDRTs because they are commonly used in driving studies [13, 24], and they follow the framework of difficulty levels in the flow theory [10]. We further designed reading and arithmetic problem solving with two difficulty levels (easy and medium adapted from [46], which reported a strong correlation between the questions and the physiological responses). Nevertheless, DeepTake framework can be easily adjusted to any NDRTs.

3.1.4 Vehicle Data. DeepTake also considers a wide range of data streams captured from the automated vehicles, including lane position, distance to hazards, angles of the steering wheel, throttle and brake pedal angles, and the vehicle velocity. Such vehicle data can help to determine the driving condition, the urgency of a takeover situation, and the impact of drivers' takeover behavior.

3.2 Data Preparation

3. .12 Feature E txaction and Multimodal Data Fusion. The goal of DeepTake is to provide a procedure to reliably predict drivers' takeover behavior (i.e., intention, time and quality) before a TOR initiation. Hence, the taken procedure for data preparation depends on the driving setting, collected data and the context. Herein, we incorporate data of drivers' objective and subjective measurements, as well as vehicle dynamic data. We initially apply data preprocessing techniques including outliers elimination, missing value imputation using mean substitutions, and smoothing to reduce artifacts presented in raw data. It is worth mentioning that we exclude any data stream providing insights about the unknown future (e.g., type of alarm) or containing more than 50% missing value. The preprocessed time series data are then segmented into 10-second fixed time windows *prior to the occurrences of TORs*. In other words, if TOR happened at time t, we only used data captured in the fixed time window of [t-10s, t] and did not include any data later than t. We started with time window values of 2s and 18s, suggested in the literature [4, 17, 69], and experimentally settled on 10s, as real-world applications require a shorter time window with better prediction. We then aggregated the values of all multimodal data over this time interval, resulting in 256 (max sampling rate)$\times 10sec = 2560$ observations *per takeover* event. However, depending on specific applications and contextual requirements, the selected time window length could vary. Subsequently, the segmented windows from modalities are processed to extract meaningful features describing the attributes impacting takeover behavior.

For the eye movement, we acquire interpolated features extracted from raw data through iMotion software [29]. The extracted eye movement attributes include gaze position, pupil diameters of each eye, time to first fixation, and fixation duration/sequence on the detected area of interest (i.e., cellphone, tablet and monitor).

To compute the heart rate features, we first apply a min-max normalization on the raw PPG signal, and then filter the normalized PPG signal by applying a 2^{nd} order Butterworth high pass filter with a cut-off of 0.5Hz followed by a 1^{st} order Butterworth low pass filter with a cut-off frequency of 6Hz. We use an open-source toolkit HeartPy [58] to filter the PPG signals and extract the following features from heart rate variability (HRV) analysis: the standard deviation of normal beats (SDNN), root mean square of successive differences between normal heartbeats (RMSSD), and the proportion of pairs of successive beats that differ by more than 50ms (pNN50). These metrics are to correlate with driver's cognitive workload and stress [50].

Furthermore, we obtain two common and important GSR features: the number and amplitude of peaks [37, 46]. A peak occurs when there is a quick burst of raised conductance level. The peak

Table 1: List of extracted features used in DeepTake

Data Source	Feature	Type	Values
Eye movement	Gaze position	float	(1920×1080)
	Pupil size	float	(0-7)
	Time to first fixation	int	(1-90)
	Fixation duration	float	(100-1500ms)
	Fixation sequence	int	(1-2500)
Heart rate (PPG signal)	SDNN	float	(45-75ms)
	RMSSD	float	(25-43ms)
	pNN50	float	(18-28%)
GSR signal	Number of peaks	int	(1-6)
	Amplitude of peaks	float	(0.01- 1.58µs)
Pre-driving survey	Gender	binary	(M-W)
	NASA-TLX	categorical	(1-21)
	PSS-10	categorical	(0-4)
Secondary tasks	NDRTs	categorical	(C,U,R,S)[1]
Vehicle data	Right lane distance	float	(0.73-2.4m)
	Left lane distance	float	(1.02-2.8m)
	Distance to hazard	float	(98-131m)
	Steering wheel angle	float	(-180-114°)
	Throttle pedal angle	float	(15-21°)
	Brake pedal angle	float	(0-17°)
	Velocity	float	(0-55mph)

1: *C*; Conversation, *U*; Using cellphone, *R*; Reading articles on tablet, and *S*: Solving arithmetic questions

amplitude measures how far above the baseline the peak occurred. Thus, peaks are valuable indicator of stress and mental workload.

While the variety of a driver's subjective and objective measurements along with vehicle dynamic data provide complementary information to draw better insights into drivers' takeover behavior, we need to finally fuse these multimodal data into a joint representation as input to the DNN model. Beforehand, however, we employ the Z-score normalization for most of the features except extracted PPG features to accentuate key data and binding relationships within the same range. To normalize the features associated with PPG, we use the min-max normalization, as explained above. For any remaining features still containing missing values, their missing values are imputed by using their means. Table 1 summarizes the list of data sources and extracted features used in DeepTake. Finally, the generated features from each modality concatenated to create a rich vector representing driver takeover attributes. The joint representations of all feature vectors with the provision of their associated labels are eventually fed into DNN models for training. Below, the labeling procedure of these feature vectors is explained.

3. . 2 2Data Labeling. The target labels greatly depend on the context in which the labels are presented. Herein, we define the ground truth labeling for an attribute set denoting the feature vector. Each label indicates the classification outcome of takeover intention, time, and quality that is more representative of our user study and the three takeover behavior aspects.

Takeover intention. DeepTake classifies a driver's takeover intention into the binary outcomes, indicating whether or not the driver would resume manual control of the vehicle. In our user study, if a participant initiated the takeover action by pressing the two buttons mounted on the steering wheel (see Figure 2) upon receiving a TOR, we label the feature vector as "TK", showing the takeover intention; if no takeover action was initiated between the moment of TOR initiation and the incident (e.g., obstacle avoidance), we use a "NTK" label displaying the absence of intention.

Takeover time. Recall from Section 2 that takeover time is defined as the time period between a TOR and the exact moment of a driver resuming manual control. Prior works have considered the starting time of manual control as the first contact with the steering wheel/pedals [66] or the takeover buttons [32]. In our user study, we timed the takeover moment once a participant pressed the two takeover buttons on the steering wheel simultaneously (see Figure 2). We categorize takeover time into three classes, using threshold values consistent with the pre-defined i^{th} percentile of takeover time in prior driving studies [8]. Let T denote the takeover time, thus the labels are defined as "low" when $T < 2.6s$, "medium" when $2.6s \leq T \leq 6.1s$, or "high" when $T > 6.1s$.

Takeover quality. As we alluded to earlier in Section 2, there are a wide range of metrics [30] for measuring takeover quality, depending on the needs of various takeover scenarios. In our user study (see Section 4), we consider a motivating scenario where the driver needs to take over control of the vehicle and swerve away from an obstacle blocking the same lane; meanwhile, the vehicle should not deviate too much from the current lane, risking crashing into nearby traffic. Therefore, we measure the takeover quality using the lateral deviation from the current lane, denoted by P. In our study, we design a 4-lane rural highway with a lane width of $3.5m$. Therefore, we label the feature vectors into three classes of takeover quality: "low" or staying in a lane when $P < 3.5m$, "medium" or maneuver the obstacle but too much deviations when $7m < P \leq 10m$, or "high" or maneuver safely and one lane deviates when $3.5 \leq P \leq 7m$.

3.3 DNN Models for Takeover Behavior Prediction

DeepTake utilizes a feed-forward deep neural network (DNN) with a mini-batch stochastic gradient descent. The DNN model architecture begins with an input layer to match the input features, and each layer receives the input values from the prior layer and outputs to the next one. There are three hidden layers with 23, 14, and 8 ReLu units, respectively. The output layer can be customized for the multi-class classification of takeover intention, takeover time and takeover quality. For example, for the classification of takeover quality, the output layer consists of three Softmax units representing three classes (low-, medium-, and high-) of takeover quality. DeepTake framework uses Softmax cross-entropy loss with an Adam optimizer with a learning rate of 0.001 to update the parameters and train the DNN models over 400 epochs. In each iteration, DeepTake randomly samples a batch of data in order to compute the gradients with a batch size of 30. Once the gradients are computed, the initiated parameters get updated. The early stopping method set to 400 epochs prevents overfitting. In addition, DeepTakes randomly divides the given labeled data into 70% for training (necessary for learning the weights for each node), 15% for validation (required to stop learning and overtraining), and 15% for testing (the final phase for evaluating the proposed model's

robustness to work on unseen data). Finally, in order to address imbalanced data issues where the number of observations per class is not equally distributed, DeepTake utilizes Synthetic Minority Oversampling Technique (SMOTE) [6] which uses the nearest neighbor's algorithm to generate new and synthetic data.

In summary, our DeepTake framework employs different DNN models to predict takeover intention, takeover time and takeover quality. All of the DNN models in DeepTake have the same number of inputs and hidden layers, yet different output layers and associated weights.

4 USER STUDY

To test the feasibility of our proposed DeepTake framework, we conducted a user study with 20 participants featuring takeover behavior using a driving simulator[2]. The following section describes the experimental setup and design of our user study as follows.

4.1 Participants

In this study, 20 subjects (11 female, 9 male) aged 18-30 (mean= 23.5, SD= 3.1) were recruited. All participants were hired through the university and were required to have normal or corrected-to-normal vision, to not be susceptible to simulator sickness, and to have at least one year of driving experience to be eligible for participation in this study. Before the experiment, participants were questioned as to their age and driving experience. None of them had prior experience of interaction with AVs. They were reminded of their right to abort their trial at any point with no question asked. Three participants' data were later excluded from the analysis, due to biometric data loss and a large amount of missing values. Participants received $20 to compensate for the time they spent in this study.

4.2 Apparatus

Figure 2 shows our low fidelity driving simulator setup, which consists of a Logitech G29 steering wheel, accelerator, brake pedal and paddle shifters. The simulator records driver control actions and vehicle states with a sampling frequency of 20Hz and sent the captured data through a custom API using iMotions software [29]. The simulated driving environments along with the tasks were created using PreScan Simulation Platform. The driving environment was displayed on a 30-inch monitor. The distance between the center of the Logitech G29 steering wheel and the monitor was set at 91cm. A set of stereo speakers was used to generate the driving environment sounds along with the auditory alarm of TORs (more details in Section 4.3). An Apple iPad Air (10.5-inch) was positioned to the right side of the driver and steering wheel to mimic the infotainment system and displayed an article for NDRT.

We used Tobii Pro-Glasses 2 with the sample rate of 60Hz to collect the eye movement data, and a Shimmer3+ wearable device with a sampling rate of 256Hz to measure PPG and GSR signals. To maintain consistency across all participants, we positioned the Shimmer3+ to the left of all subjects. This consistency helps reduce the motion artifact where the subjects needed to frequently interact with the tablet on the right-hand side. Although we designed our scenarios in a way to minimize the inevitable motion artifacts, we performed necessary signal processing on the PPG and GSR signals to remove potentially corrupted data, as discussed in Section 3.1.

[2]This study complies with the American Psychological Association Code of Ethics and was approved by the Institutional Review Board at University of Virginia.

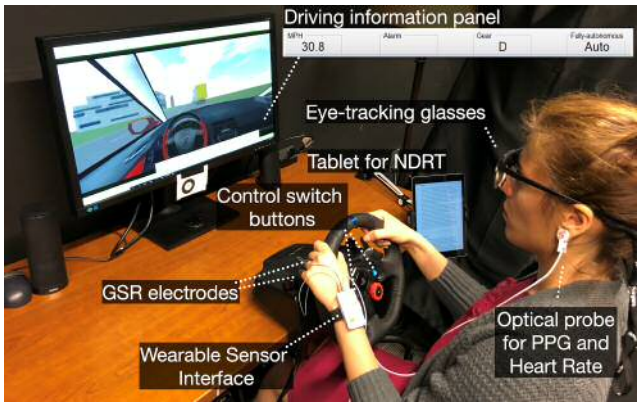


Figure 2: User study setup. This custom driving simulator consists of a 30-inch monitor, a Logitech G29 steering wheel, and 10.5-inch Apple iPad Air on which the non-driving tasks are displayed. For switching between the automated and manual control of the vehicle, the participant needs to press the two blue buttons on the steering wheel simultaneously. The participant wears a pair of eye-tracking glasses, and a wearable device with GSR and PPG sensors for the biometrics acquisition.

Table 2: Non-driving related tasks (NDRTs) used in our study

Task Type	Definition
Conversation with passenger	Interacting with the experimenter who sits close to the participants
Using cellphone	Interacting with their cellphones for texting and browsing
Reading articles	Reading three types of articles (i.e.easy,mid,hard) on the tablet
Solving questions	Answering 2-level arithmetic questions (i.e. easy and medium)

4.3 Experimental design

A within-subjects design with independent variables of stress and cognitive load manipulated by NDRTs and the TOR types was conducted with three trials in a controlled environment as shown in Figure 2. We designed driving scenarios in which the simulated vehicle has enough functionality similar to AVs, such that the full attention of the driver was not required at all times.

Non-Driving Related Tasks. We used four common NDRTs with various difficulty levels and cognitive demand as shown in Table 2. Participants used the tablet to read the designated articles and answer the arithmetic questions. Additionally, they were asked to use their own hand-held phones, needed for the browsing tasks. Each participant performed all NDRTs with the frequency of four times in each trial (except for solving the arithmetic questions which occurred three times;15×3 in total). The conditions and the three driving scenarios were counterbalanced among all participants to reduce order and learning effects. To have natural behavior to the greatest extent possible, participants were allowed to depart from NDRTs to resume control of the vehicle at any given time. During manual driving, participants controlled all aspects of the vehicle, including lateral and longitudinal velocity control.

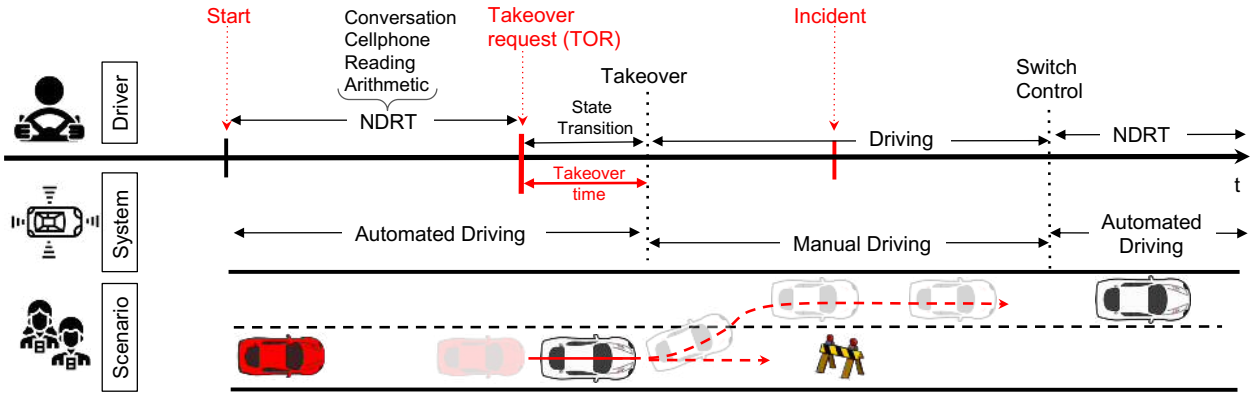


Figure 3: A schematic view of an example of a takeover situation used in our study, consisting of: 1) takeover timeline associated with participants' course of action; 2) system status; and 3) takeover situation. The vehicle was driven in the automated mode to the point after the TOR initiation and transitioning preparation period. The ego vehicle is shown in red and the lead car is white. When the Ego vehicle reaches its limits, the system may initiate (true alarm) or fail (no alarm) to initiate the TOR, and the driver takes the control back from the automated system.

Driving Scenarios. The driving scenarios comprised a 4-lane rural highway, with various trees and houses placed alongside the roadway. We designed five representative situations where the AVs may need to prompt a TOR to the driver, including novel and unfamiliar incidents that appear on the same lane. Figure 3 shows an example of a takeover situation used in our study. The designed unplanned takeovers let participants react more naturally to what they would normally do in AVs [39] or as introduced by Kim and Yang [32], participants' reaction times are in detectable categories. In other words, participants have no previous knowledge of incident appearance, which might happen among other incidents requiring situational awareness and decision-making.

Takeover Requests. In order to incorporate DeepTake in the design of adaptive in-vehicle alert systems in a way that not only monitors driver capability of takeover, but also to enhance takeover performance of automated driving, various types of TOR were required. An auditory alarm was used to inform participants about an upcoming hazard that required takeover from the automated system. The warning was a single auditory tone (350Hz, duration: 75ms) presented at the time of hazard detection (≈140m or ≈13sec before the incidents, depending the speed of the vehicle). In a precarious world, AVs should be expected to fail to always provide correct TORs. Herein, the scenarios were constructed conservatively to include flawed TORs by which subjects would not over-trust the system's ability. In other words, the scenario demands that the participant be partially attentive and frequently perceive the environment. In order to cover the scenarios that one might encounter while driving an AV, we designed multiple critical types of TORs, including an explicit alarm (true alarm), silent failure (no alarm), and nuisance alarm (false alarm). True alarm indicates the situation in which the system correctly detects the hazard and triggers a TOR, no alarm represents the system's failure to identify the existing hazard, and false alarm presents misclassification of a non-hazardous situation as an on-road danger requiring takeover. We randomized the 15 TOR occurrences in each trial (45 in total for each participant) with 6, 3, 6 repetitions for true alarm, no alarm, false alarm, respectively. In addition, we also designed an information panel where the participants could see the status of the vehicle along with the cause of TOR (see Figure 2).

4.4 Procedure

Upon arrival in the lab, participants were asked to sign a consent form and fill out a short demographic and driving history questionnaires. Subsequently, they were briefed on how the automated system functions, how to enable the system by simultaneously pressing two blue buttons on the steering wheel, and what they would experience during NDRTs. They were further instructed that if the system detected a situation beyond its own capabilities to handle, it would ask (true alarm) or fail to ask (no alarm) to take over control. Afterward, participants completed a short training drive along a highway for a minimum of 5 minutes to get familiar with the driving and assure a common level of familiarity with the setup, NDRTs, and auditory signals pitch.

Once the subjects felt comfortable with the driving tasks and NDRTs, they proceeded to the main driving scenario. Prior to beginning the main experiment, we calibrated the eye-tracking glasses (repeated at the beginning of each trial) and set participants up with the Shimmer3+ wearable device. Then, participants were required to complete the baseline NASA-TLX questionnaire followed by the PSS-10 questionnaire. The participants were also instructed to follow the lead car, stay on the current route, and follow traffic rules as they normally do. The participants were cautioned that they were responsible for the safety of the vehicle regardless of its mode (manual or automated). Therefore, they were required to be attentive and to safely resume control of the vehicle in case of failures and TORs. Since the scenarios were designed to have three types of TORs, they needed to adhere to the given instruction whenever they felt the necessity. The given instruction enabled the drivers to respond meticulously whenever it was required and to

reinforce the idea that they were in charge of the safe operation of the vehicle. Due to the system's limitations, participants were told to maintain the speed within the acceptable range (< 47mph). The experiment was conducted utilizing scenarios consisting of sunny weather conditions without considering the ambient traffic. The order of NDRT engagement was balanced for participants (see Figure 3).

The remainder of the experiment consisted of three trials, each containing 15 TORs, followed by a 5-minute break between trials. At the end of each trial, participants were requested to fill out the NASA-TLX. After completion of the last trial, participants filled out the last NASA-TLX followed by a debrief and a $20 compensation. The experiment took about one hour for each participant.

5 PERFORMANCE EVALUATION

We evaluate the performance of DeepTake framework using the multimodal data collected from our user study. We describe the baseline methods, metrics, results, and analysis as follows.

5.1 Baseline Methods

Overall, we obtained about 2 million observations to train, test, and validate DeepTake with; 2560 observations per TOR × 15 TORs per trial × 3 trials × 17 subjects. We evaluate the performance of DeepTake DNN-based models with six other ML-based predictive models, including Logistic Regression, Gradient Boosting, Random Forest, Bayesian Network, Adaptive Boosting (Adaboost), and Regularized Greedy Forest (RGF). Our process of choosing the ML models is an exploratory task with trials and tests of multiple off-the-shelf algorithms and choosing those that perform the best. To evaluate the prediction performance of DeepTake framework with other ML models, we were obligated to utilize some feature importance techniques. The reasons to apply feature importance techniques for an ML algorithm are: to train the predictive model faster, reduce the complexity and increase the interpretability and accuracy of the model. In order to do so, after splitting the labeled data into training, testing, and validation sets (see Section 3.3), we employ the following feature importance methods on each training set: Absolute Shrinkage and Selection Operator (LASSO), and random forest. LASSO helps us with not only selecting a stable subset of features that are nearly independent and relevant to the drivers' takeover behavior, but also with dimensionality reduction. The random forest method, on the other hand, ranks all of the features based on their importance levels with the drivers' takeover behavior. The overlapped features chosen by the two methods were used to train the ML-based classification models of takeover behavior.

5.2 Metrics

We apply 10-fold cross-validation on training data to evaluate the performance of selected features in the prediction of driver takeover intention, time and quality. Cross-validation provides an overall performance of the classification and presents how a classifier algorithm may perform once the distribution of training data gets changed in each iteration. In cross-validation, we utilize the training fold to tune model hyper-parameters (e.g., regularization strength, learning rate, and the number of estimators), which maximizes prediction performance. Therefore, we train predictive models with the best hyper-parameters. Cross-validation randomly partitions the training data into n subsets without considering the distribution of data from a subject in each set. A possible scenario is that data from one subject could be unevenly distributed in some subsets, causing overestimation of the prediction performance of a model. To avoid this situation, we check the subjects' identifiers in both the training and testing sets to ensure that they belong to just one group. We achieve this by forcing the subject to be in one group. To determine the *accuracy* of the binary classification of takeover intention performed by predictive models, accuracy was defined as $Acc = \frac{TP+TN}{TP+TN+FP+FN}$ (TP, TN, FP, and FN represent True Positive, True Negative, False Positive, and False Negative, respectively). For the multi-class classification of takeover time and quality, we used the average accuracy per class. We also used the metric of *weighted F1 scores* given by

$$WF_1 = \sum_{n=1}^{l} \frac{2 \times Pr_i \times Rc_i}{Pr_i + Rc_i} \times W_i, \tag{1}$$

where $Pr_i = \frac{\sum_{i=1}^{l} \frac{TP_i}{TP_i+FP_i}}{l}$ is the precision, $Rc_i = \frac{\sum_{i=1}^{l} \frac{TP_i}{TP_i+FN_i}}{l}$ is the recall, and W_i is the weight of the i^{th} class depending on the number of positive examples in that class. It is worth mentioning that to deal with our imbalanced data, where the number of observations per class is not equally distributed, DeepTake framework along with ML-based predictive models use SMOTE to have a well-balanced distribution within class (see Section 3.3).

Given multiple classifiers, we use the *Receiver Operating Characteristic* (ROC) curve to compare the performance of DeepTake alongside other ML-based models. The ROC curve is a widely-accepted method that mainly shows the trade-off between TP and FP rates. A steep slope at the beginning of the curve shows a higher true positive (correct) classification of the algorithm, whereas increasing the FP rate causes the curve to flatten. The ROC curve provides an effective way to summarize the overall performance of classification algorithms by its only metric, AUC. The AUC values provided in Figure 4 can be interpreted as the probability of correctly classifying the driver takeover behavior into the candidate category compared to a random selection (black line in Figure 4).

In addition, we use the *confusion matrix* to further illustrate the summary of DeepTake's performance on the distinction of takeover intention, time, and quality per class.

5.3 Results and Analysis

Multiple classification algorithms were employed to compare the performance of DeepTake on obtaining a reliable discriminator of driving takeover behavior, including intention, time, and quality. As the prediction of driver takeover time and quality are contingent upon the driver's intention to take over from the autonomous systems after receiving TOR, the classification algorithms were initially carried out on this first stage of driver takeover prediction, followed by takeover time and quality.

Takeover intention. Analysis of the binary classification of drivers' takeover intention is shown in Table 3. The results show that DeepTake outperforms other ML-based models. However, among the ML-based algorithms, RGF attains the highest accuracy and weighted F1 score (92% and 89%) followed by AdaBoost (88% and

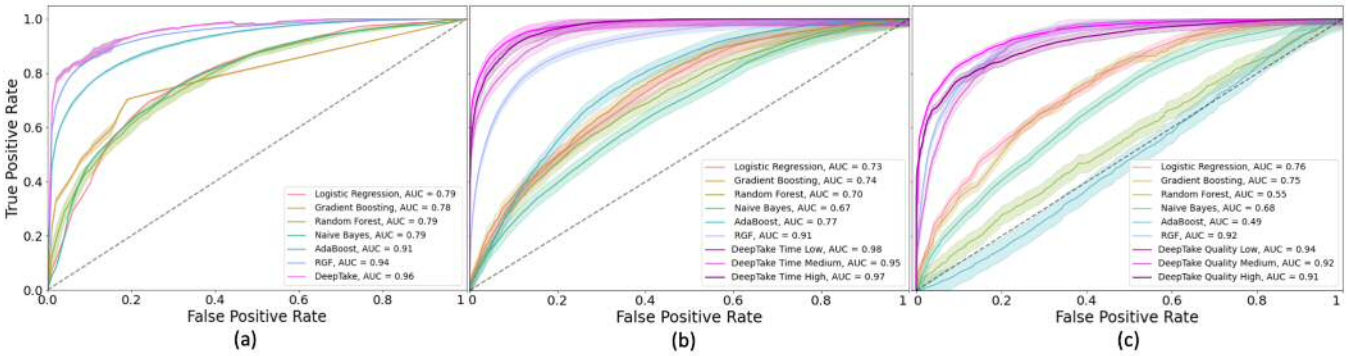


Figure 4: The ROC curve comparison of our DeepTake and six ML classification algorithms for classification of takeover behavior: (a) takeover intention, (b) takeover time, and (c) takeover quality. The ROC curve shows the average performance of each classifier and the shadowed areas represent the 95% confidence interval. The macro AUC associated with each classifier is shown where AUC value of 0.5 refers to a chance.[Best viewed in color]

88%) and Logistic Regression (77% and 88%). Moreover, ROC was applied in order to better evaluate each of the classifiers. Figure 4.a shows ROC curves and AUC values for all six ML models along with DeepTake to infer the binary classification of takeover intention. Although DeepTake shows outperformance on correctly classifying a driver's intention (AUC=0.96) using the multimodal features, RGF shows promising performance with an AUC of 0.94. Similar to the accuracy level, AdaBoost had a slightly lower performance with an AUC= 0.91. Furthermore, we obtained the confusion matrix for takeover intention (Figure 6.a) showing that the percentage of misclassifications is insignificant. Table 3, together with the results obtained from the AUC in Figure 4.a and the confusion matrix in Figure 6.a, ensure that our multimodal features with the right DNN classifier surpass the takeover intention prediction.

Table 3: Classification performance comparison.

Target value	Classifier	Accuracy	W-F1[1] score
Takeover Intention	Logistic Regression	0.77	0.81
	Gradient Boosting	0.76	0.75
	RF[2]	0.75	0.72
	Naive Bayes	0.71	0.66
	Ada Boost	0.88	0.87
	RGF[3]	0.92	0.89
	DeepTake	**0.96**	**0.93**
Takeover Time	Logistic Regression	0.47	0.45
	Gradient Boosting	0.47	0.46
	RF	0.44	0.45
	Naive Bayes	0.36	0.38
	Ada Boost	0.64	0.58
	RGF	0.73	0.71
	DeepTake	**0.93**	**0.87**
Takeover Quality	Logistic Regression	0.65	0.63
	Gradient Boosting	0.60	0.59
	RF	0.53	0.52
	Naive Bayes	0.41	0.39
	Ada Boost	0.42	0.39
	RGF	0.82	0.77
	DeepTake	**0.83**	**0.78**

1: *Weighted F1-score*; 2:*Random Forest*; 3:*Regularized Greedy Forests*

Takeover time. DeepTake's promising performance in takeover intention estimation leads us to a more challenging multi-class prediction of driver takeover time. As some of the ML-based models attained reasonably high accuracy in the binary classification of takeover, their performances, along with our DeepTake DNN based

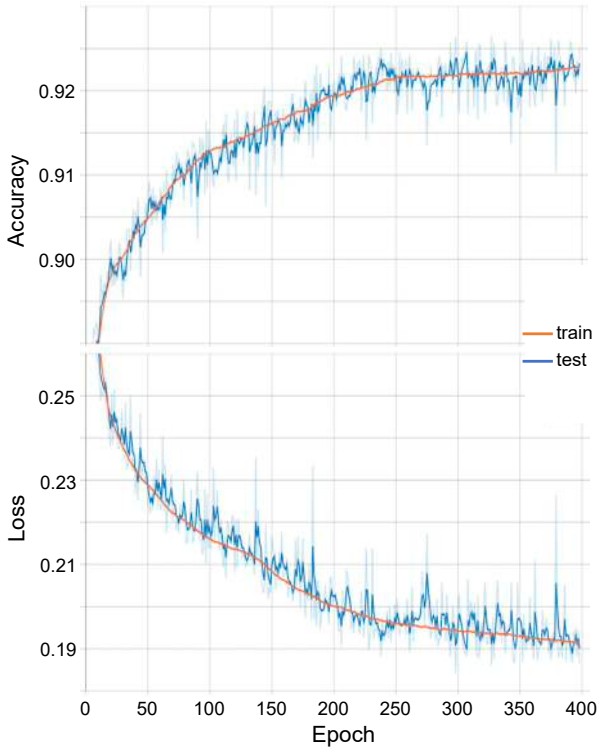


Figure 5: The top graph shows the prediction accuracy of training and test sets for 400 epochs, whereas the bottom graph indicates the loss for DeepTake on prediction of three classes of low-, mid-, and high- takeover time.

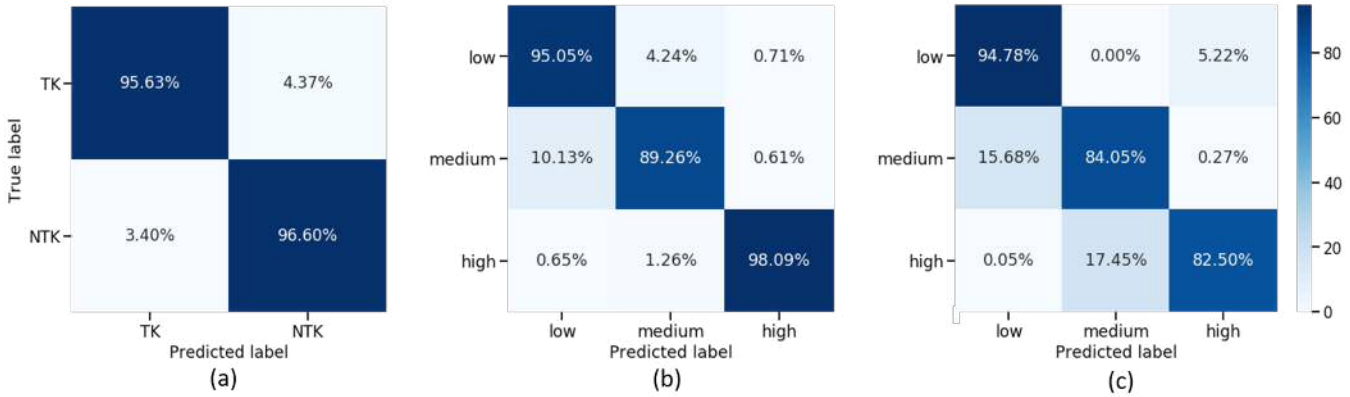


Figure 6: Confusion matrix for the prediction of takeover behavior.The results are averaged over 10 fold cross validation splits. (a) Binary class takeover intention takeover(TK) vs. Not Takeover(NTK), (b) 3-Class classification results of takeover time, (c) 3-class classification of takeover quality.

in classifying multi-class classification of takeover time could assess the robustness of the DeepTake.

Figure4.b shows a comparison amongst the models explored in this paper along with DeepTake for prediction of takeover time. It displays that DeepTake produces the best overall result with an AUC value of 0.96 ± 0.02 for each takeover low-, mid-, and high- time. We next consider the accuracy comparison of our DeepTake model with other classifier algorithms, reported in Table 3. It is evident that DeepTake outperforms all of the classic algorithms. In the three-class classification of takeover time (low, mid, high), DeepTake achieves a weighted-F1 score of 0.87, thereby achieving the best performance on this task by a substantially better accuracy result of 92.8%. Among the classifiers, RGF and AdaBoost still performed better (73.4% and 64.1%). As shown in Figure 5, DeepTake gained a high accuracy for both the training and testing sets. However, the model did not significantly improve and stayed at around 92% accuracy after the epoch 250.

To capture a better view of the performance of DeepTake on the prediction of each class of takeover time, we also computed the confusion matrix. Figure 6 displays the performance of DeepTake DNN model as the best classifier of three-class takeover time. As the diagonal values represent the percentage of elements for which the predicted label is equal to the true label, it can be seen that the misclassification in medium takeover time is the highest. Also, marginal misclassifications are found in the 2%-5% of the high and low takeover time classes, respectively. Overall, all three evaluation metrics of AUC, accuracy, and confusion matrix indicate that DeepTake robustness and promising performances in correctly classifying the three-class takeover time.

Takeover quality. The test accuracy results of the 3-class classification of all classifiers are presented in Table 3. DeepTake achieves the highest accuracy with an average takeover quality of 83.4%. While the value of RGF was close to DeepTake, the rest of the algorithms were not reliable enough to discriminate each class of takeover. However, we should note that RGF training time is very slow and it takes about two times longer than DeepTake to perform prediction.

In addition, Figure 4.c presents the multi-class classification of takeover quality. Analysis of the discriminatory properties of DeepTake achieve the highest AUC of 0.92 ± 0.01 scores among the other models for each individual class. RGF model yields an impressive average macro AUC of 0.91. Such a model indicates a high-performance achievement with informative features.

We further investigated DeepTake robustness in correctly classifying each class of takeover quality and the results achieved by the method are shown in Figure 6.c. For the 3-class quality estimation, DeepTake achieved an average accuracy of 87.2%.

6 DISCUSSION

6.1 Summary of major findings

In the current design of takeover requests, AVs do not account for human cognitive and physical variability, as well as their possibly frequent state changes. In addition, most previous studies emphasize the high-level relationships between certain factors and their impacts on takeover time or quality. However, a safe takeover behavior consists of a driver's willingness and readiness together. The focus of this paper is to utilize multimodal data into a robust framework to reliably predict the three main aspects of drivers' takeover behavior: takeover intention, time and quality. To the best of our knowledge, the DeepTake framework is the first method for the estimation of all three components of safe takeover behavior together within the context of AVs and it has also achieved the highest accuracy compared to previous studies predicting each aspect individually. To ensure the reliability of DeepTake's performance, we applied multiple evaluation metrics and compared the results with six well-known classifiers. Despite the promising accuracy of some of the classifiers, namely the RGF classifier, the accuracy of DeepTake surpassed in its prediction of takeover behavior. In general, our model performed better in classifying driver takeover intention, time and quality with an average accuracy of 96%, 93%, and 83%, respectively.

In order to further assess the robustness of DeepTake, we increase the number of classes to the more challenging five-class classification of takeover time where the classes defined as "lowest"

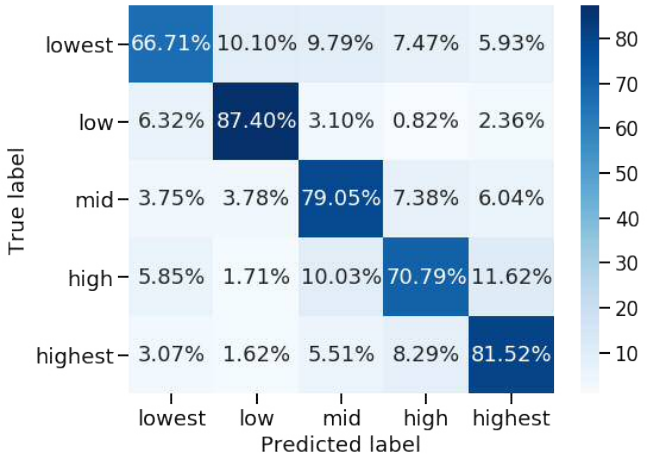


Figure 7: Confusion matrix for the prediction of five classes of driver takeover time.

when $T < 1.5s$, "low" when $1.5s \leq T < 2.6s$, "medium" when $2.6s \leq T < 4.7s$, "high" when $4.7s \leq T \leq 6.1s$, or "highest" when $T > 6.1s$. Figure 7 represents the performance of DeepTake on classifying the five-class takeover time. Although DeepTake was not as distinctive in five-class classification as in the three-class, it still achieved promising results. Lowest, high, and medium takeover times are the top three pairs that were the most frequently misclassified by the DNN model. The reason might be that the selected features do not have the required distinctive characteristics to perfectly divide the low and medium takeover time. In each class, it could still distinguish between five other classes with an average accuracy of 77%. With a future larger amount of data collection satisfying each class need, DeepTake could further improve its distinctive aspect of each feature for more precise classification.

6.2 Descriptive analysis of takeover time and quality

Although DeepTake takes advantage of a DNN-based model integrated into its framework, understanding the reasons behind its predictions is still a black-box and a challenging problem which will be tackled in our future works. However, to comprehend the effects of multimodal variables on takeover time and quality, a repeated measure Generalized Linear Mixed (GLM) model with a significance level of $\alpha = 0.05$ to assess the correlation of suboptimal features was used to predict takeover time and quality. The analysis of the results shows the significant main effect of NDRTs on takeover time and quality ($F_{3,28} = 13.58$, $p < 0.001$) followed by fixation sequence ($F_{1,28} = 35.87$, $p < 0.001$) and vehicle velocity ($F_{1,28} = 13.06$, $p < 0.001$). Post-hoc tests using Bonferroni demonstrated a higher impact of interaction with the tablet and reading articles ($p < 0.001$) as opposed to a conversation with passengers. This result could be based on the amount of time spent and the level of cognitive load on the takeover task. This finding is aligned with the previous results of [20, 21]. Additionally, there was no significant effect of brake and throttle pedal angle on the takeover time($F_{1,28} = 3.05$, $p = 0.085$) and quality ($F_{1,28} = 1.27$ $p = 0.256$). This could be because our scenarios did not take place on crowded roads and participants were not forced to adopt a specific behavior after the TOR. Therefore, they could maneuver the vehicle without significant adjustment to either pedal.

On the other hand, takeover quality tied into drivers' lane keeping control and was impacted by the alarm type and the category of takeover time shown in Figure 8. Although we did not consider the type of alarm and category of takeover time for prediction of takeover behavior as they could simply manipulate DeepTake outcomes by providing insights about the future, it is worth additional investigation of their impacts on the takeover quality. Since participants' takeover times and the speed of the vehicle on the manual driving were different, Figure 8 shows the average time of TOR. The top graph in Figure 8 depicts the average lateral position of the vehicle with respect to no alarm and true alarm. These two types of the alarm were considered due to the necessity of taking over. Under the impact of the true alarm, the vehicle deviates less than when there is no alarm, yet not significantly ($F_{2,28} = 7.07$, $p = 0.78$). Moreover, the drivers performed more abrupt steering wheel maneuvers to change lanes on true alarm. Similarly, the bottom graph in Figure 8 shows the lateral position with respect to different takeover times (low, mid, and high). It can be seen that the longer the takeover time is, the farther the vehicle deviates from the departure lane. Differences in takeover time were also analyzed to investigate the takeover quality. The main effect of the type of takeover time was not significant ($F_{2,19} = 0.44$). Although prior research has revealed various timing efforts to fully stabilize the vehicle [42], our observations are comparable to [45] and [5].

6.3 Implications on the design of future interactive systems

We believe that our human-centered framework makes a step towards enabling a longer interaction with NDRTs for automated driving. DeepTake helps the system to constantly monitor and predict the driver's mental and physical status by which the automated system can make optimal decisions and improve the safety and user experience in AVs. Specifically, by integrating the DeepTake framework into the monitoring systems of AVs, the automated system infers when the driver has the intention to takeover through multiple sensor streams. Once the system confirms a strong possibility of takeover intention, it can adapt its driving behavior to match the driver's needs for acceptable and safe takeover time and quality. Therefore, a receiver of TOR can be ascertained as having the capability to take over properly, otherwise, the system would have allowed the continued engagement in NDRT or warned about it. Thus, integration of DeepTake into the future design of AVs facilitates the human and system interaction to be more natural, efficient and safe. Since DeepTake should be used in safety-critical applications, we further validated it to ensure that it meets important safety requirements [26]. We analyzed DeepTake sensitivity and robustness with several techniques using the Marabou verification tool. The sensitivity analysis provides insight into the importance of input features, in addition to providing formal guarantees with respect to the regions in the input space where the DeepTake behaves as expected.

DeepTake framework provides a promising new direction for modeling driver takeover behavior to lessen the effect of the general

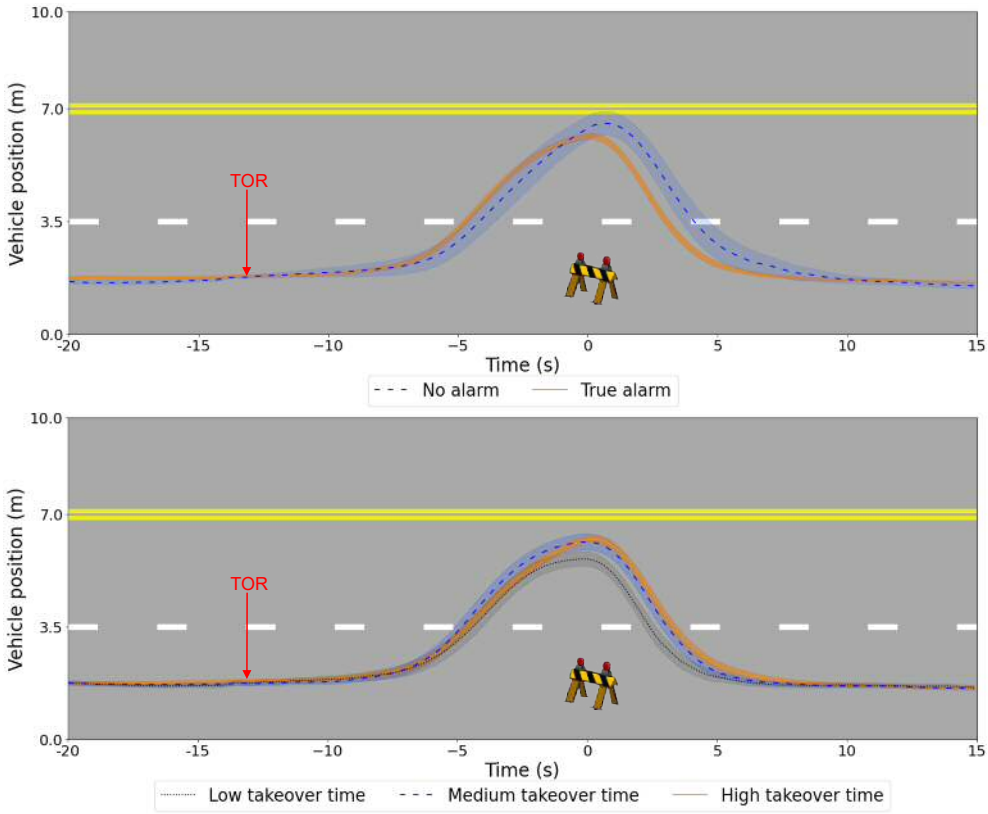


Figure 8: Average trajectories when drivers took over control from automated system after receiving TORs. Top graph shows the lateral position of the vehicle with respect to no alarm (silent failure) and true alarm (explicit alarm). Bottom graph shows the lateral position of the vehicle for three categories of takeover time (low, mid, and high). The light shaded area representing standard deviation at each time point.

and fixed design of TORs which generally considers homogeneous takeover time for all drivers. This is grounded in the design of higher user acceptance of AVs and dynamic feedback [19, 55]. The information obtained by DeepTake can be conveyed to passengers as well as other vehicles letting their movement decisions have a higher degree of situational awareness. We envision that DeepTake would help HCI researchers and designers to create user interfaces and systems for AVs that adapt to the drivers' state.

6.4 Limitations and future work

The following limitations should be taken into consideration for future research and development of DeepTake.

First, it is acknowledged that the DeepTake dataset is vulnerable to the low fidelity driving simulator used for data collection. It is possible that the takeover behavior of subjects were influenced by the simplicity of driving setup and activities. To apply DeepTake on the road, we will need more emphasis on various user's activities and safety, and exclude subjective surveys causing biases. Second, while we increased the number of classes, future development of DeepTake should predict takeover time numerically. For this purpose, a larger dataset will be needed which accounts for a high variation of individual takeover time and probabilistic nature of DNNs by which the DeepTake framework can still learn and reliably predicts takeover time.

Third, although we tried to avoid overfitting, it is possible that DeepTake emphasized more on few features that frequently appeared in TORs, and the performance may not be the same if more scenarios are being tested. Thus, DeepTake decision boundaries need to be experimented with different adversarial training techniques. Forth, DeepTake lacks using real-world data which often significantly different and could potentially impact the results of DeepTake framework. Testing the framework on real-world data helps users to gain confidence in DeepTake's performance. DeepTake was developed and assessed offline using a driving simulator in a controlled environment. Future work should explore the deployment of DeepTake online and in the wild for real-world applications in future AVs. We plan to integrate the DeepTake and its verification results [26] into the safety controller, which will be then evaluated using the on-road vehicle. In our future work we also plan to try to reduce the number of features in the model by using the results from the sensitivity analysis along with feature importance analysis techniques (i.e. LIME and SHAP) to discover features that may be able to be dropped from the model.

7 CONCLUSION

In this work, we present DeepTake, a novel method that predicts driver takeover intention, time and quality using data obtained from the vehicle, wearable sensors, and a self-administered survey taken before driving. By using DNN-based models, DeepTake enables prediction of driver takeover intention, time and quality, all of which are crucial in ensuring the safe takeover of an automated vehicle. Our evaluation showed that DeepTake outperforms the best accuracy results of prior work on takeover prediction with an accuracy of 96 %, 93 %, and 83% for the multi-class classification of takeover intention, time and quality, respectively. As prior studies demonstrated, alarming drivers when the system detects a situation requiring takeover does not guarantee safe driver takeover behavior [33, 34, 40]. We believe that accurate takeover prediction afforded by DeepTake would allow drivers to work on non-driving related tasks while ensuring that they safely take over the control when needed. DeepTake opens up new perspectives for HCI researchers and designers to create user interfaces and systems for AVs that adapt to the drivers' context.

8 ACKNOWLEDGMENT

We would like to thank Prof. Corina Păsăreanu from Carnegie Mellon University and Prof. Radu Calinescu from University of York for their valuable inputs, and John Grese for his help in evaluating DeepTake with a high number of epochs. This work was supported in part by National Science Foundation CCF-1942836 grant, Assuring Autonomy International Programme, and Toyota InfoTech Labs.

REFERENCES

[1] Areen Alsaid, John D Lee, and Morgan Price. 2019. Moving into the loop: An investigation of drivers' steering behavior in highly automated vehicles. *Human factors* (2019), 0018720819850283.

[2] Frauke L Berghöfer, Christian Purucker, Frederik Naujoks, Katharina Wiedemann, and Claus Marberger. 2018. Prediction of take-over time demand in conditionally automated driving-results of a real world driving study. In *Proceedings of the Human Factors and Ergonomics Society Europe Chapter 2018 Annual Conference*. 69–81.

[3] Shadan Sadeghian Borojeni, Lars Weber, Wilko Heuten, and Susanne Boll. 2018. From reading to driving: priming mobile users for take-over situations in highly automated driving. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*. 1–12.

[4] Christian Braunagel, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2017. Ready for take-over? A new driver assistance system for an automated classification of driver take-over readiness. *IEEE Intelligent Transportation Systems Magazine* 9, 4 (2017), 10–22.

[5] Mercedes Bueno, Ebru Dogan, F Hadj Selem, Eric Monacelli, Serge Boverie, and Anne Guillaume. 2016. How different mental workload levels affect the take-over control after automated driving. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2040–2045.

[6] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[7] Sheldon Cohen, Tom Kamarck, and Robin Mermelstein. 1983. A global measure of perceived stress. *Journal of health and social behavior* (1983), 385–396.

[8] G Coley, A Wesley, N Reed, and I Parry. 2009. Driver reaction times to familiar, but unexpected events. *TRL Published Project Report* (2009).

[9] SAE On-Road Automated Vehicle Standards Committee et al. 2018. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International: Warrendale, PA, USA* (2018).

[10] Mihaly Csikszentmihalyi and Mihaly Csikzentmihaly. 1990. *Flow: The psychology of optimal experience*. Vol. 1990. Harper & Row New York.

[11] Nachiket Deo and Mohan M Trivedi. 2019. Looking at the driver/rider in autonomous vehicles to predict take-over readiness. *IEEE Transactions on Intelligent Vehicles* (2019).

[12] Nicole Dillen, Marko Ilievski, Edith Law, Lennart E Nacke, Krzysztof Czarnecki, and Oliver Schneider. 2020. Keep Calm and Ride Along: Passenger Comfort and Anxiety as Physiological Responses to Autonomous Driving Styles. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[13] Ebru Dogan, Vincent Honnêt, Stéphan Masfrand, and Anne Guillaume. 2019. Effects of non-driving-related tasks on takeover performance in different takeover situations in conditionally automated driving. *Transportation research part F: traffic psychology and behaviour* 62 (2019), 494–504.

[14] Na Du, Jinyong Kim, Feng Zhou, Elizabeth Pulver, Dawn Tilbury, Lionel Robert, Anuj Pradhan, X Jessie Yang, et al. 2020. Evaluating Effects of Cognitive Load, Takeover Request Lead Time, and Traffic Density on Drivers' Takeover Performance in Conditionally Automated Driving. *AutomotiveUI* (2020).

[15] Na Du, Feng Zhou, Elizabeth Pulver, Dawn Tilbury, Lionel P Robert, Anuj K Pradhan, and X Jessie Yang. 2020. Predicting Takeover Performance in Conditionally Automated Driving. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–8.

[16] Na Du, Feng Zhou, Elizabeth M Pulver, Dawn M Tilbury, Lionel P Robert, Anuj K Pradhan, and X Jessie Yang. 2020. Examining the effects of emotional valence and arousal on takeover performance in conditionally automated driving. *Transportation research part C: emerging technologies* 112 (2020), 78–87.

[17] Na Du, Feng Zhou, Elizabeth M Pulver, Dawn M Tilbury, Lionel P Robert, Anuj K Pradhan, and X Jessie Yang. 2020. Predicting driver takeover performance in conditionally automated driving. *Accident Analysis & Prevention* 148 (2020), 105748.

[18] Mahdi Ebnali, Kevin Hulme, Aliakbar Ebnali-Heidari, and Adel Mazloumi. 2019. How does training effect users' attitudes and skills needed for highly automated driving? *Transportation research part F: traffic psychology and behaviour* 66 (2019), 184–195.

[19] Fredrick Ekman, Mikael Johansson, and Jana Sochor. 2017. Creating appropriate trust in automated vehicle systems: A framework for HMI design. *IEEE Transactions on Human-Machine Systems* 48, 1 (2017), 95–101.

[20] Alexander Eriksson and Neville A Stanton. 2017. Takeover time in highly automated vehicles: noncritical transitions to and from manual control. *Human factors* 59, 4 (2017), 689–705.

[21] Anna Feldhütter, Christian Gold, Sonja Schneider, and Klaus Bengler. 2017. How the duration of automated driving influences take-over performance and gaze behavior. In *Advances in ergonomic design of systems, products and processes*. Springer, 309–318.

[22] Anna Feldhütter, Dominik Kroll, and Klaus Bengler. 2018. Wake up and take over! The effect of fatigue on the take-over performance in conditionally automated driving. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2080–2085.

[23] Hannah J Foy and Peter Chapman. 2018. Mental workload is reflected in driver behaviour, physiology, eye movements and prefrontal cortex activation. *Applied ergonomics* 73 (2018), 90–99.

[24] Michael A Gerber, Ronald Schroeter, Li Xiaomeng, and Mohammed Elhenawy. 2020. Self-Interruptions of Non-Driving Related Tasks in Automated Vehicles: Mobile vs Head-Up Display. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–9.

[25] Christian Gold, Moritz Körber, David Lechner, and Klaus Bengler. 2016. Taking over control from highly automated vehicles in complex traffic situations: the role of traffic density. *Human factors* 58, 4 (2016), 642–652.

[26] John M Grese, Corina Pasareanu, and Erfan Pakdamanian. 2021. Formal Analysis of a Neural Network Predictor inShared-Control Autonomous Driving. In *AIAA Scitech 2021 Forum*. 1580.

[27] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.

[28] Sebastian Hergeth, Lutz Lorenz, and Josef F Krems. 2017. Prior familiarization with takeover requests affects drivers' takeover performance and automation trust. *Human factors* 59, 3 (2017), 457–470.

[29] iMotions. 2015. *Affectiva iMotions Biometric Research Platform*. https://imotions.com/

[30] ISO 21959:2020 2020. *Road vehicles — Human performance and state in the context of automated driving*. Standard. International Organization for Standardization.

[31] Mishel Johns, Brian Mok, David Sirkin, Nikhil Gowda, Catherine Smith, Walter Talamonti, and Wendy Ju. 2016. Exploring shared control in automated driving. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 91–98.

[32] Hyung Jun Kim and Ji Hyun Yang. 2017. Takeover requests in simulated partially autonomous vehicles considering human factors. *IEEE Transactions on Human-Machine Systems* 47, 5 (2017), 735–740.

[33] Moritz Körber, Lorenz Prasch, and Klaus Bengler. 2018. Why do I have to drive now? Post hoc explanations of takeover requests. *Human factors* 60, 3 (2018), 305–323.

[34] Jiwon Lee and Ji Hyun Yang. 2020. Analysis of driver's EEG given take-over alarm in SAE level 3 automated driving in a simulated environment. *International journal of automotive technology* 21, 3 (2020), 719–728.

[35] Alexander Lotz and Sarah Weissenberger. 2018. Predicting take-over times of truck drivers in conditional autonomous driving. In *International Conference on Applied Human Factors and Ergonomics*. Springer, 329–338.

[36] Tyron Louw, Gustav Markkula, Erwin Boer, Ruth Madigan, Oliver Carsten, and Natasha Merat. 2017. Coming back into the loop: Drivers' perceptual-motor performance in critical events after automated driving. *Accident Analysis & Prevention* 108 (2017), 9–18.

[37] Udara E Manawadu, Hiroaki Hayashi, Takaaki Ema, Takahiro Kawano, Mitsuhiro Kamezaki, and Shigeki Sugano. 2018. Tactical-Level Input with Multimodal Feedback for Unscheduled Takeover Situations in Human-Centered Automated Vehicles. In *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 634–639.

[38] Nikolas Martelaro, Jaime Teevan, and Shamsi T Iqbal. 2019. An Exploration of Speech-Based Productivity Support in the Car. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[39] Rod McCall, Fintan McGee, Alexander Mirnig, Alexander Meschtscherjakov, Nicolas Louveton, Thomas Engel, and Manfred Tscheligi. 2019. A taxonomy of autonomous vehicle handover situations. *Transportation research part A: policy and practice* 124 (2019), 507–522.

[40] Anthony D McDonald, Hananeh Alambeigi, Johan Engström, Gustav Markkula, Tobias Vogelpohl, Jarrett Dunne, and Norbert Yuma. 2019. Toward computational simulations of behavior during automated driving takeovers: a review of the empirical and modeling literatures. *Human factors* 61, 4 (2019), 642–688.

[41] Bruce Mehler, Bryan Reimer, and Joseph F Coughlin. 2012. Sensitivity of physiological measures for detecting systematic variations in cognitive demand from a working memory task: an on-road study across three age groups. *Human factors* 54, 3 (2012), 396–412.

[42] Natasha Merat, A Hamish Jamson, Frank CH Lai, Michael Daly, and Oliver MJ Carsten. 2014. Transition to manual: Driver behaviour when resuming control from a highly automated vehicle. *Transportation research part F: traffic psychology and behaviour* 27 (2014), 274–282.

[43] Brian Mok, Mishel Johns, David Miller, and Wendy Ju. 2017. Tunneled in: Drivers with active secondary tasks need more time to transition from automation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2840–2844.

[44] Frederik Naujoks, Christoph Mai, and Alexandra Neukum. 2014. The effect of urgency of take-over requests during highly automated driving under distraction conditions. *Advances in Human Aspects of Transportation* 7, Part I (2014), 431.

[45] Frederik Naujoks, Christian Purucker, Katharina Wiedemann, and Claus Marberger. 2019. Noncritical State Transitions During Conditionally Automated Driving on German Freeways: Effects of Non–Driving Related Tasks on Takeover Time and Takeover Quality. *Human factors* 61, 4 (2019), 596–613.

[46] Nargess Nourbakhsh, Yang Wang, Fang Chen, and Rafael A Calvo. 2012. Using galvanic skin response for cognitive load measurement in arithmetic and reading tasks. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*. 420–423.

[47] Erfan Pakdamanian, Lu Feng, and Inki Kim. 2018. The effect of whole-body haptic feedback on driver's perception in negotiating a curve. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 62. SAGE Publications Sage CA: Los Angeles, CA, 19–23.

[48] Erfan Pakdamanian, Nauder Namaky, Shili Sheng, Inki Kim, James Arthur Coan, and Lu Feng. 2020. Toward Minimum Startle After Take-Over Request: A Preliminary Study of Physiological Data. In *12th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. 27–29.

[49] Julie Paxion, Edith Galy, and Catherine Berthelon. 2014. Mental workload and driving. *Frontiers in psychology* 5 (2014), 1344.

[50] Margherita Peruzzini, Mara Tonietti, and Cristina Iani. 2019. Transdisciplinary design approach based on driver's workload monitoring. *Journal of Industrial Information Integration* 15 (2019), 91–102.

[51] Sebastiaan Petermeijer, Fabian Doubek, and Joost de Winter. 2017. Driver response times to auditory, visual, and tactile take-over requests: A simulator study with 101 participants. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 1505–1510.

[52] Ioannis Politis, Patrick Langdon, Damilola Adebayo, Mike Bradley, P John Clarkson, Lee Skrypchuk, Alexander Mouzakitis, Alexander Eriksson, James WH Brown, Kirsten Revell, et al. 2018. An evaluation of inclusive dialogue-based interfaces for the takeover of control in autonomous cars. In *23rd International Conference on Intelligent User Interfaces*. 601–606.

[53] Jonas Radlmayr, Christian Gold, Lutz Lorenz, Mehdi Farid, and Klaus Bengler. 2014. How traffic situations and non-driving related tasks affect the take-over quality in highly automated driving. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 58. Sage Publications Sage CA: Los Angeles, CA, 2063–2067.

[54] Daniele Ruscio, Maria Rita Ciceri, and Federica Biassoni. 2015. How does a collision warning system shape driver's brake response time? The influence of expectancy and automation complacency on real-life emergency braking. *Accident Analysis & Prevention* 77 (2015), 72–81.

[55] Bobbie D Seppelt and John D Lee. 2019. Keeping the driver in the loop: Dynamic feedback to support appropriate use of imperfect vehicle control automation. *International Journal of Human-Computer Studies* 125 (2019), 66–80.

[56] Qiuyang Tang, Gang Guo, Zijian Zhang, Bingbing Zhang, and Yingzhang Wu. 2020. Olfactory Facilitation of Takeover Performance in Highly Automated Driving. *Human Factors* (2020), 0018720819893137.

[57] Remo MA van der Heiden, Shamsi T Iqbal, and Christian P Janssen. 2017. Priming drivers before handover in semi-autonomous cars. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 392–404.

[58] Paul van Gent, Haneen Farah, Nicole van Nes, and Bart van Arem. 2019. HeartPy: A novel heart rate algorithm for the analysis of noisy signals. *Transportation research part F: traffic psychology and behaviour* 66 (2019), 368–378.

[59] José Vicente, Pablo Laguna, Ariadna Bartra, and Raquel Bailón. 2011. Detection of driver's drowsiness by means of HRV analysis. In *2011 Computing in Cardiology*. IEEE, 89–92.

[60] Marcel Walch, Kristin Mühl, Johannes Kraus, Tanja Stoll, Martin Baumann, and Michael Weber. 2017. From car-driver-handovers to cooperative interfaces: Visions for driver–vehicle interaction in automated driving. In *Automotive user interfaces*. Springer, 273–294.

[61] Jingyan Wan and Changxu Wu. 2018. The effects of vibration patterns of take-over request and non-driving tasks on taking-over control of automated vehicles. *International Journal of Human–Computer Interaction* 34, 11 (2018), 987–998.

[62] Lora Warshawsky-Livne and David Shinar. 2002. Effects of uncertainty, transmission type, driver age and gender on brake reaction and movement time. *Journal of safety research* 33, 1 (2002), 117–128.

[63] JCF De Winter, Neville A Stanton, Josh S Price, and Harvey Mistry. 2016. The effects of driving with different levels of unreliable automation on self-reported workload and secondary task performance. *International journal of vehicle design* 70, 4 (2016), 297–324.

[64] Philipp Wintersberger, Andreas Riener, Clemens Schartmüller, Anna-Katharina Frison, and Klemens Weigl. 2018. Let me finish before I take over: Towards attention aware device integration in highly automated vehicles. In *Proceedings of the 10th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. 53–65.

[65] Yanbin Wu, Ken Kihara, Yuji Takeda, Toshihisa Sato, Motoyuki Akamatsu, and Satoshi Kitazaki. 2019. Assessing the Mental States of Fallback-Ready Drivers in Automated Driving by Electrooculography. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 4018–4023.

[66] Kathrin Zeeb, Axel Buchner, and Michael Schrauf. 2015. What determines the take-over time? An integrated model approach of driver take-over after automated driving. *Accident Analysis & Prevention* 78 (2015), 212–221.

[67] Kathrin Zeeb, Axel Buchner, and Michael Schrauf. 2016. Is take-over time all that matters? The impact of visual-cognitive load on driver take-over quality after conditionally automated driving. *Accident Analysis & Prevention* 92 (2016), 230–239.

[68] Kathrin Zeeb, Manuela Härtel, Axel Buchner, and Michael Schrauf. 2017. Why is steering not the same as braking? The impact of non-driving related tasks on lateral and longitudinal driver interventions during conditionally automated driving. *Transportation research part F: traffic psychology and behaviour* 50 (2017), 65–79.

[69] Bo Zhang, Joost de Winter, Silvia Varotto, Riender Happee, and Marieke Martens. 2019. Determinants of take-over time from automated driving: A meta-analysis of 129 studies. *Transportation research part F: traffic psychology and behaviour* 64 (2019), 285–307.

Appendix B Formal Analysis of a Neural Network Predictor in Shared-Control Autonomous Driving

Formal Analysis of a Neural Network Predictor in Shared-Control Autonomous Driving

John Grese*
Carnegie Mellon University, CyLab

Corina Păsăreanu†
Carnegie Mellon University, CyLab and NASA Ames Research Center

Erfan Pakdamanian‡
University of Virginia, LinkLab

Autonomous driving systems may encounter scenarios where it is necessary to transfer control to the human driver, for instance when encountering unpredictable dangerous road conditions. To be able to do so safely, the autonomous system needs an estimate of how long it will take for the human driver to take control of the vehicle. Deep neural networks can be used for making such predictions, however proving that neural networks meet critical safety requirements presents a challenge. We present a formally verified neural network which predicts "Takeover-time" in a shared-control autonomous driving system. The network is trained on data collected from a (semi-)autonomous driving simulator. We use *Marabou* (a formal verification tool), to analyze the network's sensitivity, local robustness, contextual robustness, and to find adversarial inputs which produce unsafe outputs.

I. Objectives and Impacts

The work reported here is done within a project called *Safe-SCAD — Safety of shared control in autonomous driving** whose objective is to answer *how humans and machines can safely share control of an autonomous car*. Ensuring that drivers retain sufficient situational awareness to *be able to take control of the vehicle in an emergency* [1] is a challenging problem. This is due to the uncertainties associated with measuring the level of situational awareness of drivers while not in control of the vehicle, and with the mapping of such measures to takeover times. As emphasized in the US Department of Transportation strategic documents, there is an urgent need for solutions to *critical research questions regarding driver transitions between automated and manual driving modes* [2].

The Safe-SCAD project aims to extend, adapt and integrate the recent research and the latest advances from human behavior and cognitive modeling, verification of deep neural networks, and automated controller synthesis to tackle these challenges.

The project will make significant and generalizable *impacts* in the areas of:

- shared autonomy
- training and verification of machine learning
- monitoring of autonomous systems by human operators

The project's team has designed and conducted a human subject study on a driving simulator located at University of Virginia. The simulator is a PC-based system consisting primarily of three 30-inch monitors, a steering wheel, accelerator and brake pedals, and eye tracking glasses. In this study, 20 subjects (11 female, 9 male) aged 18-30 (mean= 23.5, SD= 3.1) were recruited. All participants were hired at University of Virginia and were required to have normal or corrected-to-normal vision, to not be susceptible to simulator sickness, and to have at least one year of driving experience to be eligible for participation in this study. Before the experiment, participants were questioned as to their age and driving experience. None of them had prior experience of interaction with autonomous vehicles. Three participants' data were later excluded from the analysis, due to biometric data loss and a large amount of missing values. Participants received $20 to compensate for the time they spent in this study.

*Carnegie Mellon University
†NASA
‡University of Virginia

*This is a joint project between University of Virginia, University of York and Carnegie Mellon University, funded by the Assuring Autonomy International Program from University of York, UK

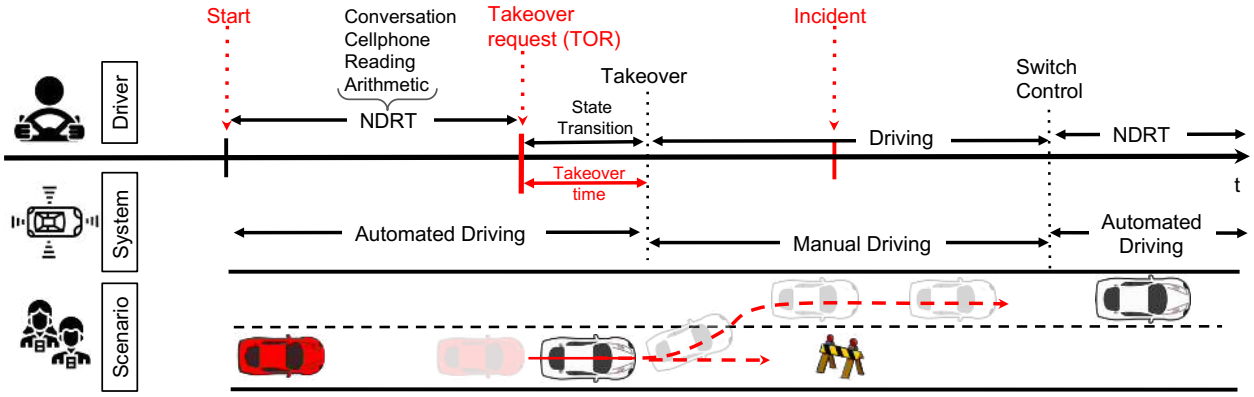


Fig. 1 A schematic view of an example of a takeover situation used in our study, consisting of: 1) takeover timeline associated with participants' course of action; 2) system status; and 3) takeover situation. The vehicle was driven in the automated mode to the point after the TOR initiation and transitioning preparation period. The ego-vehicle is shown in red and the lead car is white. When the ego-vehicle reaches its limits, the system may initiate (true alarm) or fail (no alarm) to initiate the TOR, and the driver takes the control back from the automated system.

The preliminary data from this study was used to train a neural network that achieves 86% accuracy of driver takeover time prediction, with the takeover time organized into several categories, e.g. fast, med-fast, medium, med-slow, and slow. In this paper we report on the formal verification of the neural network using the Marabou [3] verification tool to analyze its robustness and sensitivity to input perturbations.

II. Takeover Time Network

A. User Study

The subjects were briefed about the semi-autonomous systems, the driving tasks and non-driving-tasks (NDRTs), they proceeded to the main driving scenario. The participants were instructed to follow the lead car, stay on the current route, and follow traffic rules as they normally do. *Figure 1* illustrates the scenario that the participants went through in this study. The participants were cautioned that they were responsible for the safety of the vehicle regardless of its mode (manual or automated). As the vehicle approaches the obstacle, it alerts the driver with an alarm requesting that the driver take control of the vehicle. "Takeover time" refers to the amount of time between when the "takeover request" alarm (*TOR*) is triggered and when the driver has taken control (*Takeover*). It is calculated by subtracting the timestamps of *TOR* and *Takeover*. Therefore, they were required to be attentive and to safely resume control of the vehicle in case of failures and takeover requests (TORs). The given instruction enabled the drivers to respond meticulously whenever it was required and to reinforce the idea that they were in charge of the safe operation of the vehicle. Due to the system's limitations, participants were told to maintain the speed within the acceptable range (< 47mph). The experiment was conducted utilizing scenarios consisting of sunny weather conditions without considering the ambient traffic. In addition, the order of NDRT engagement was balanced for participants (see Figure 1). The experiment consisted of three trials, each containing 15 TORs, followed by a 5-minute break between trials.

B. Data Preparation

The goal of this project is to provide a procedure to not only reliably predict drivers' takeover time before a TOR initiation (reported in [4]), but also formally verify safety properties of the model. Hence, the taken procedure for data preparation depends on the driving setting, collected data and the context. Herein, we incorporate data of drivers' physiological measurements, as well as vehicle dynamic data. We initially apply data preprocessing techniques including outlier elimination, missing value imputation using mean substitutions, and smoothing to reduce artifacts presented in raw data. It is worth mentioning that we exclude any data stream providing insights about the unknown future (e.g.,

label	lbound (ms)	ubound (ms)
fast	0	1612
med-fast	1612	2802
med	2802	5180
med-slow	5180	6370
slow	6370	+inf

Table 1 Takeover time categories

type of alarm) or containing more than 50% missing value. The preprocessed time series data are then segmented into 10-second fixed time windows *prior to the occurrences of TORs* with the offset sliding window of 1, experimentally [4]. For instance, if TOR happened at T, we only used data captured in the fixed time window of (T-10s, T) and did not include any data later than T. However, depending on specific applications and contextual requirements, the selected time window length could vary. Subsequently, the segmented windows from modalities are processed to extract meaningful features describing the attributes impacting takeover behavior.

For the eye movement, we acquire interpolated features extracted from raw data through iMotion software. The extracted eye movement attributes include gaze position, pupil diameters of each eye, time to first fixation, and fixation duration/sequence on the detected area of interest.

Finally, the generated features from each modality concatenated to create a rich vector representing driver takeover attributes. The joint representations of all feature vectors with the provision of their associated labels are eventually fed into neural network for training. The final processed dataset consists of 25 input features including: *FixationDuration, FixationSeq, FixationStart, FixationX, FixationY, GazeDirectionLeftZ, GazeDirectionRightZ, PupilLeft, PupilRight, InterpolatedGazeX, InterpolatedGazeY, AutoThrottle, AutoWheel, CurrentThrottle, CurrentWheel, Distance3D, MPH, ManualBrake, ManualThrottle, ManualWheel, RightLaneDist, RightLaneType, LeftLaneDist, LeftLaneType*. These features were labeled with 5 output targets of *fast, med-fast, med, med-slow, slow*.

C. Neural Network Architecture

These features are labeled into the five classes shown in Table 1 and one-hot encoded. Then, the minority classes are upsampled to ensure that all of the classes are represented equally. The target (y) columns are separated from the training data, and the input (x) values are scaled using standard scaling, which standardizes features by removing the mean and scaling to the unit variance. The dataset is then divided, using 80% for training, 10% for testing, and 10% for validation.

The neural network is a fully-connected feed-forward classifier with three hidden layers as shown in figure 2. The input layer has 25 nodes. The three hidden layers have 21, 18, and 11 nodes, and use ReLU as the activation function. The output layer has 5 nodes and uses Softmax. The input layer's 25 inputs (x_0 through x_{24}) map to the feature names listed in theData Preparation section. Inputs x_0 through x_{10} are provided by the simulator's eye tracking system, and describe the driver's fixation, gaze, and pupil diameter. Inputs x_{11} through x_{19} provide information related to both autonomous and manual steering, braking, throttle, and vehicle speed. Inputs x_{20} through x_{24} provide information around the vehicle such as lane type, position, and distance. The output layer's 5 nodes (y_0 through y_4) represent the takeover time class labels listed in table 1.

We utilize Softmax cross-entropy loss with an Adam optimizer and a learning rate of 0.001 to update the parameters and train the network. In each iteration, we randomly sample a batch of data in order to compute the gradients with a batch size of 16. Once the gradients are computed, the initiated parameters are updated. The early stopping method set to 50 epochs prevents overfitting. The resulting network has an accuracy of ~86%.

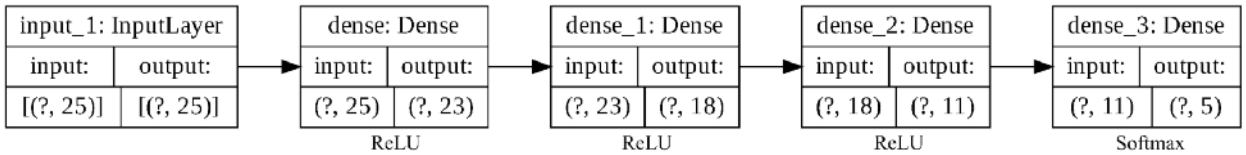


Fig. 2 Neural Network

III. Verification with Marabou

Deep neural networks (DNNs) are increasingly used in safety-critical applications such as autonomous transport, raising serious safety and security concerns. To address these concerns, DNNs need to be validated to ensure that they meet important safety requirements. However, validation of DNNs is challenging due to the nature of data-driven learning and lack of meaningful specifications. Evaluating sensitivity to input perturbations and robustness against adversarial attacks is also challenging due to the huge input space and unclear boundaries.

In this paper we report our investigation of techniques that provide assurance guarantees for neural networks. The problem is difficult as it is known that neural networks are unstable with respect to so called *adversarial perturbations* [5, 6], which are (minimal) changes that cause the network to misclassify an input. They can be devised without access to the training set [7] and are transferable [8] in the sense that an example misclassified by one network is also misclassified by a network with a different architecture, even if it is trained on different data. Existing testing and approximation techniques [6, 8–11] that can be used for the analysis of neural networks are inherently limited as they can not provide guarantees.

To address this limitation, recent work has employed formal verification methods based on Satisfiability Modulo Theory (SMT) that provide sound assurances that no adversarial examples exist within a given neighborhood of an input. Marabou [3] is an SMT-based neural network verification framework which can be used to provide formal guarantees about a neural network. Marabou works by accepting queries about the network's properties and transforming these queries into SMT constraint satisfaction problems. It is capable of accommodating networks with different activation functions and topologies, and performs high-level reasoning on the network to curtail the search space and improve performance. Common Marabou queries are expressed in terms of upper and lower bounds on the network's inputs, and an expected output. After solving a query, Marabou either returns "UNSAT", or "SAT" with a counterexample if one was found.

Using Marabou, we performed local robustness checks, targeted robustness checks, a sensitivity analysis, and data-driven verification on the takeover-time network. All verification was performed on an Amazon Web Services t2.x-large EC2 instance with 16GB of RAM and a 64bit Intel(R) Xeon(R) E5-2686 v4 CPU @ 2.30GHz with 4 cores.

A. Local Robustness

Local robustness can be defined as the minimum perturbation $\pm\delta$ applied uniformly to all features ($x_0 ... x_{24}$) from an single input x, which causes a change in the network's prediction. We find the minimum $\pm\delta$ using Marabou by evaluating a series of input queries with the lower bounds set to $x - \delta$, and the upper bounds set to $x + \delta$ over a range of values for δ until finding the minimum $\pm\delta$ that causes the predicted label to change. Specifically, we perform something similar to a binary search over the range of possible values for δ. For each value of δ, we perform a query for every label other than the expected label for x to ensure that no other labels exist between $\|x - \delta\|$ and $\|x + \delta\|$. Using the value of δ we find from local robustness for a given input x, we are given the guarantee described by equation 1, which states that for any input $x\prime$ such that the distance from x is smaller than δ, the network will make the same prediction.

$$\forall x\prime \; s.t. \; \|x\prime - x\| < \delta \Rightarrow f(x\prime) = f(x). \tag{1}$$

We checked the network's local robustness on a set of ~2500 inputs from the dataset. Table 2 shows the minimum and average values of $\pm\delta$ grouped by label. With respect to the inputs we tested, the minimum observed δ was 0.00011, which belonged to the *med* class. The highest observed δ was 0.65801, which belonged to the *med-slow* class. The class with the lowest mean δ was *med-slow*, and the class with the highest mean δ was *fast*. We can see from figure 3 that the δ for the majority of inputs fell within the range 0.001 to 0.05. The runtime per input of the local robustness checks took between ~1.2 seconds and ~61 minutes, with an average of ~4.9 minutes.

label	mean δ	min δ	max δ
fast	0.028733	0.00030	0.26471
med-fast	0.026576	0.00015	0.16012
med	0.027524	0.00011	0.13579
med-slow	0.023497	0.00040	0.65801
slow	0.026293	0.00033	0.19719
overall	0.026513	0.000110	0.658010

Table 2 Local robustness results

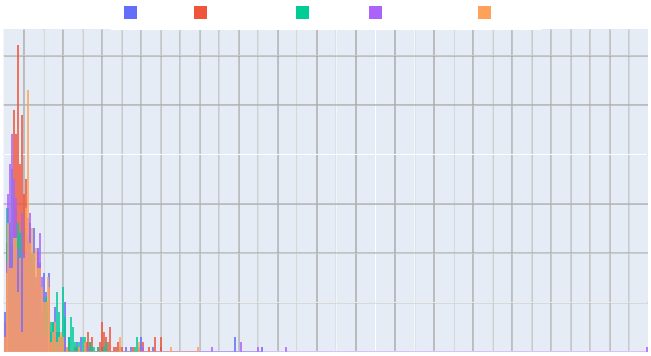


Fig. 3 Local robustness distribution

B. Targeted Robustness

We define targeted robustness as the minimum perturbation $\pm\delta$ applied uniformly to all features $(x_0...x_{24})$ of an input x which produces a *specific* change in the network's predicted label. Focusing on specific changes in the network's prediction provides more specific robustness guarantees than local robustness, and can also be helpful in finding targeted adversarial inputs. The process for targeted robustness checks is similar to local robustness, however instead of proving that only one label exists between $x+\delta$ and $x-\delta$, we prove that the targeted label *does not* exist. The guarantee provided by targeted robustness checks for a given x is described in equation 2, which states that for any input $x\prime$ such that the distance from x is smaller than δ, the network will not predict the target label.

$$\forall x\prime \; s.t. \; \|x\prime - x\| < \delta \Rightarrow f(x\prime) \neq target. \tag{2}$$

Because safety is our primary goal, we targeted the most unsafe change in classification — *slow* inputs which change to *fast* with a perturbation of δ. The results of the targeted robustness checks on a set of 500 *slow* inputs can be seen in Table 3. For the *slow – fast* target, the minimum δ was 0.0013, the mean δ was 0.073864, and the maximum δ was 0.749950. We can see from the distribution in figure 4 that the δ for the majority of inputs fell within the range 0.008 - 0.06.

The targeted robustness checks for the *slow-fast* target took longer to complete on average than the local robustness checks, which is a little counter-intuitive because fewer queries are performed in the targeted robustness. The reason for this is that the *slow-fast* target requires Marabou to work harder to find a counterexample. The runtime per input for our targeted robustness checks took between ~0.7 seconds and ~78 minutes, with an average of ~10.7 minutes.

target	mean δ	min δ	max δ
slow – fast	0.073864	0.00130	0.749950

Table 3 Targeted robustness results

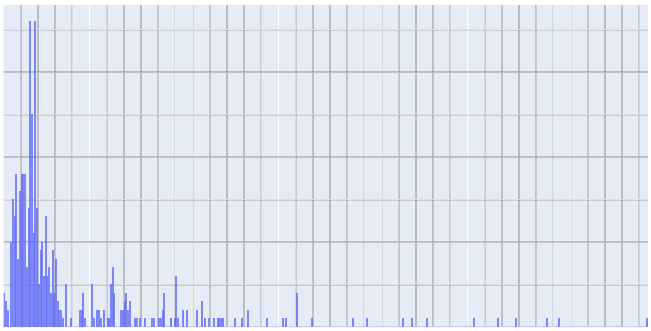


Fig. 4 Targeted robustness distribution

C. Sensitivity Analysis

To learn more about the model's sensitivity to perturbations of individual features, we performed a sensitivity analysis using Marabou. Sensitivity can be defined as the minimum perturbation δ applied to an individual feature x_i of an input x which causes a change in the network's prediction. Perturbing each feature individually also provides some visibility into feature importance. In addition, using a formal verification tool such as Marabou for this type of analysis also gives strong formal guarantees. We took two different approaches to analyzing the network's sensitivity — *symmetric* and *asymmetric*. The symmetric approach considers a single value of δ for the negative and positive perturbations to each feature ($x_0 ... x_{24}$), which extends the space around x_i symmetrically. The asymmetric approach considers individual values of δ for the negative and positive perturbations ($l\delta$ and $u\delta$) to each feature. These two different approaches to sensitivity analysis yield slightly different pictures of the network's sensitivity. In the following two sections, we discuss the symmetric and asymmetric sensitivity analyses in more detail.

1. Symmetric Sensitivity Analysis

The symmetric variant of the sensitivity analysis searches the input space around each feature x_i of an input x to find the minimum value $\pm\delta$ which causes a change in prediction. The process is similar to local robustness, however we only perturb a single feature at a time. Using Marabou, we evaluate different values of $\pm\delta$ by generating input queries for each one with the lower bound of x_i set to $x_i - \delta$ and the upper bound set to $x_i + \delta$ until discovering the minimum δ that causes a change in the predicted label. For this type of sensitivity analysis, Marabou provides the guarantee that any perturbation smaller than $\pm\delta$ applied to a single feature x_i will not change the network's prediction. Equation 3 describes this guarantee provided by Marabou for a feature x_i of an input x.

$$\forall x_i\prime \text{ s.t. } \|x_i\prime - x_i\| < \delta,\ f([x_0 ... x_i\prime ... x_n]) = f([x_0 ... x_i ... x_n]) \tag{3}$$

Using this method, we analyzed the sensitivity of all 25 features on ~2500 inputs from the dataset. Figure 5 shows the results from the symmetric sensitivity analysis. The results show that the model is most sensitive to changes of *ManualWheel* (x_{19}) and least sensitive to *FixationX* (x_3). These results make sense in the context of predicting takeover time because changes in manual wheel indicate that the driver's hands are on the wheel, and the way that the simulation was designed did not necessarily require the human to move their fixation to the left or right. Analyzing a single feature x_i took a minimum of ~0.6 seconds, a maximum of ~4.5 minutes, with an average of ~36.4 seconds, which equates to an average of ~15.7 minutes per input x.

2. Asymmetric Sensitivity Analysis

The asymmetric variant of our sensitivity analysis operates similarly to the symmetric variant, but adjusts the lower and upper bounds of x_i independently. We refer to the perturbation to the lower bound as $l\delta$ and the perturbation to the upper bound as $u\delta$. For each feature x_i of input x, we use Marabou to evaluate separate queries for $l\delta$ and $u\delta$ over a range of values until discovering the minimum $l\delta$ and $u\delta$ required to cause a change in the predicted label. The queries on the lower bound perturb feature x_i by $x_i - l\delta$, and the queries on the upper bound perturb the feature x_i by $x_i + u\delta$.

For the lower bound of the asymmetric sensitivity analysis, Marabou provides the guarantee that any perturbation smaller than $l\delta$ *subtracted* from feature x_i of an input x, the network will predict the same label. For the upper bound, it is guaranteed that the network will predict the same label when any perturbation smaller than $u\delta$ is *added* to feature x_i. The guarantee for the lower bound's perturbation $l\delta$ is described by equation 4, and the upper bound's perturbation $u\delta$ is described by equation 5.

$$\forall x_i\prime \text{ s.t. } \|x_i\prime - x_i\| < l\delta,\ f([x_0 ... x_i\prime ... x_n]) = f([x_0 ... x_i ... x_n]) \tag{4}$$

$$\forall x_i\prime \text{ s.t. } \|x_i\prime - x_i\| < u\delta,\ f([x_0 ... x_i\prime ... x_n]) = f([x_0 ... x_i ... x_n]) \tag{5}$$

We used the asymmetric sensitivity analysis to evaluate ~2500k inputs from the dataset. Figure 6 shows the model's mean sensitivity by feature. Again, we can see that the model is sensitive to changes in the *ManualWheel* feature. Another interesting observation is that on average, the model is significantly more sensitive to negative perturbations of *CurrentWheel*, indicating that the model is more sensitive when the vehicle maneuvers to the left. This is likely due to the fact that the simulation always had the obstacle on the right side of the road, so the vehicle always maneuvered to the left during the takeover.

The asymmetric approach takes approximately twice as long as the symmetric approach to compute due to the fact that the $l\delta$ and $u\delta$ must be discovered in isolation. However, even though it takes more time, it yields more information with respect to how sensitive the model is to negative vs positive perturbations, and thus has the possibility of discovering bias in the model.

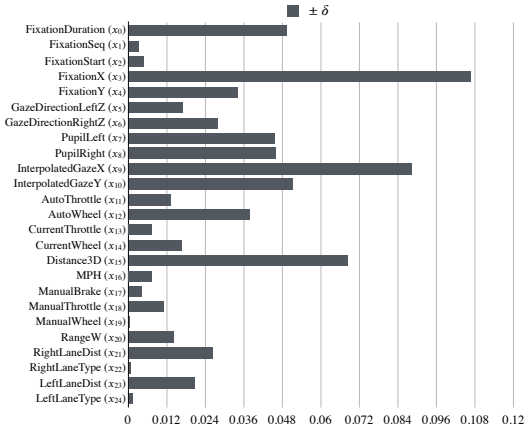


Fig. 5 Symmetric sensitivity results

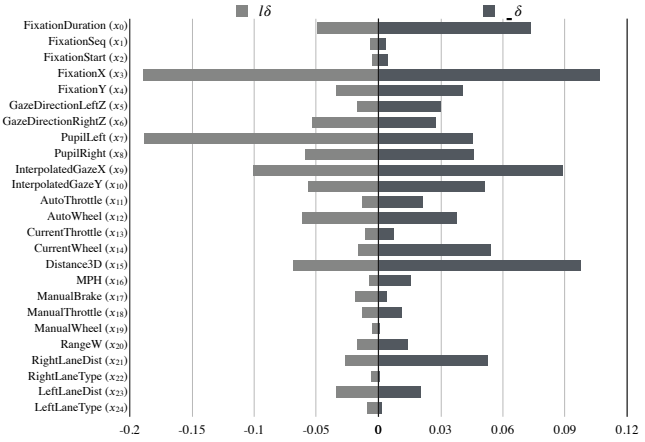


Fig. 6 Asymmetric sensitivity results

D. Clustering for Data-driven Verification

The data-driven verification technique tests the robustness of targeted regions from the input space that contain a dense population of points of a single label. This approach, which is adopted from the *Deep Safe*[12] technique, allows us to target the most relevant (densely populated) regions of the input space for verification. We start by running a modified K-Means clustering algorithm on the dataset to produce regions which contain points of a single label. Each region consists of a centroid, a radius, and a label. Then, we verify these regions with Marabou, using the centroids as inputs. The result of this verification technique is a set of targeted, "safe regions" which have been proven to contain points of a single label. Another benefit of this approach is that by targeting relevant regions of the input space, we can verify larger regions using fewer inputs, thus covering more of the input space with fewer queries. Furthermore, the results from this data-driven approach can also be useful to provide a measure of confidence about the network's predictions. The following sections describe the clustering algorithm, verification technique, results, and the proposed run-time usage of the verification results.

1. Label-guided K-Means Clustering

To identify regions of points of a the same label, we use a modified K-Means clustering algorithm called *label-guided K-Means* [12]. The regular K-Means algorithm is an unsupervised approach which does not provide any guarantees that the clusters will contain points of the same label, which means that the clusters may not be useful for verification. Label-guided K-Means clustering solves this problem by using the inputs' labels to help guide the K-Means algorithm to produce regions containing points of the same label. This is accomplished by applying K-Means with n set to the number of unique labels, checking the number of labels in each cluster, and then repeating the process iteratively to divide the K-Means clusters into regions which contain points of a single label. The output of the algorithm is a set of regions, each one consisting of a centroid, a radius, and a label. The $L2$ (euclidean) distance metric is used to measure distance between the points and compute the radius. Fig 7 shows Label-Guided K-means applied to a simplified example. The first step in the figure shows the initial K-Means clustering, the second shows the final iteration of the algorithm, and the third shows the resulting regions' centroids and radii. We also compute the density of each region as $r \div n$ where r is the region's radius, and n is the number of points in the region. Code listing 1 shows a python implementation of label-guided K-Means.

When applied to the takeover time dataset, the algorithm produced a 6138 regions with 10 or more points, however 484 of those regions had centroids which were incorrectly predicted by the network, so they were discarded, leaving us with a total of 5654 regions containing 10 or more inputs which were used for verification. These 5654 regions effectively cover ~88% of the points from the dataset.

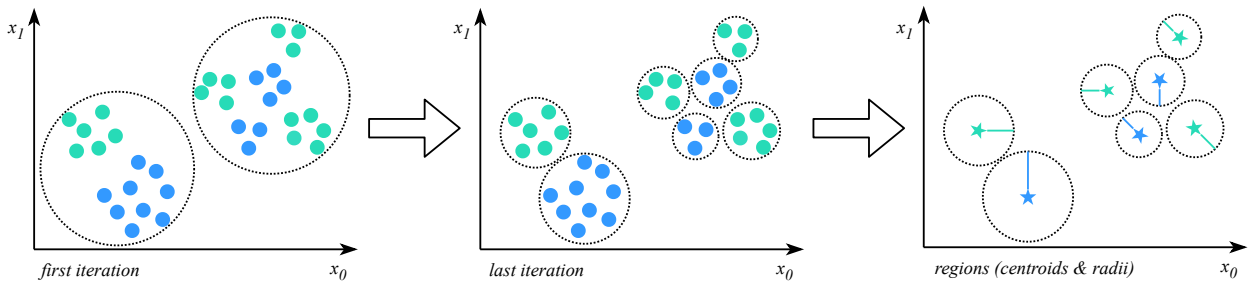


Fig. 7 Label-guided K-means

```python
import numpy as np
from sklearn.cluster import KMeans
from scipy.spatial import distance

# X:np.ndarray of inputs, Y:np.ndarray of labels
def label_guided_kmeans(X, Y):
    regions = [] # list of completed regions
    remaining = [(X, Y)] # stack of remaining inputs/labels
    while len(remaining) > 0:
        X, Y = remaining.pop(0) # pop inputs/labels to cluster from stack
        Yuniq = np.unique(Y, axis=0) # unique labels in Y
        n = Yuniq.shape[0] # number of unique labels in Y
        # initial centroids for KMeans (mean of inputs from each label)
        init = np.array([X[np.where(Y==y)[0]].mean(axis=0) for y in Yuniq])
        model = KMeans(n_clusters=n, init=init).fit(X) # run kmeans
        Yhat = model.predict(X)
        for c in np.unique(Yhat, axis=0):
            idxs = np.where(Yhat == c)[0] # indexes of inputs in cluster
            Xc, Yc = X[idxs], Y[idxs] # get inputs/labels in the cluster
            if np.unique(Yc, axis=0).shape[0] == 1:
                # cluster contained a single label; save as a region
                centroid = model.cluster_centers_[c]
                radius = max([distance.euclidean(x, centroid) for x in X])
                region = dict(centroid=centroid, radius=radius, label=Yc[0],
                    n=Xc.shape[0], density=radius/Xc.shape[0])
                regions.append(region)
            else:
                # cluster contained multiple labels; repeat KMeans on cluster
                remaining.append((Xc, Yc))
    return regions
```

Listing 1 Label-guided K-Means Python code

2. Region Verification

The regions discovered by the modified K-Means algorithm were verified using a technique similar to local robustness. For each region, we performed a robustness test with Marabou, using the region's centroids as the input. We set the queries' upper bounds to $centroid + \delta$, and the lower bounds to $centroid - \delta$, and iterated over a range of values for δ until discovering the minimum $\pm\delta$ which caused a change in the predicted label for the region. Using the δ we discovered for a given region, we computed the verified radius of the region as $\|(centroid + \delta) - centroid\|_{L2}$. The guarantee provided the verification of a given region with a radius r and *label* can be described by equation 6, which states that for all x

within the verified radius r of a safe region R, the predicted label will be *label*.

$$[h] \forall x \; s.t. \; \|x - centroid\|_{L2} \leq r \Rightarrow f(x) = label. \tag{6}$$

The results from the data-driven verification results can be seen in Table 4. In the results, we compare the verified radius with the original radius. We can see from the results that for the *med-fast* regions, the average verified radius was larger than original radius. For all other labels, the average verified radius was roughly half of the original. These results show that we were able to verify a larger portion of the *med-fast* regions than the regions from other labels. Even though we were not able to verify the entire region for the other labels, the data-driven approach still allowed us to cover a large portion of the dataset while testing only 5654 individual points. The data-driven verification tests per region took a minimum of ~0.558 seconds and a maximum of ~149.55 minutes, with an average of ~3.38 minutes per region. In general, the regions with higher densities took longer to verify than the regions with lower densities.

category	**verified radius**			**original radius**		
	mean	**min**	**max**	**mean**	**min**	**max**
fast	0.81743	0.00250	8.88250	1.67943	0.11222	8.68825
med_fast	2.07260	0.01250	15.51750	1.98323	0.17113	10.43312
med	0.74406	0.00250	3.62750	1.46866	0.12483	6.87752
med_slow	0.58589	0.00250	6.04750	1.36602	0.04696	13.84459
slow	0.82720	0.00250	4.49500	1.34232	0.11944	5.79814
all	0.84691	0.00250	15.51750	1.50298	0.04696	13.84459

Table 4 Verified regions comparison

3. Run-time Usage

One of the goals of the Safe-SCAD project involves designing a safety-controller which was designed by the University of York. Among other things, the proposed safety-controller has the job of ensuring that the takeover action is handled safely. When the takeover alarm is triggered, the safety-controller will consume the neural network's reaction time prediction to help make a decision on the safest action to take. However, neural networks are probabilistic in nature and may make incorrect predictions under certain conditions. So, to provide a measure of confidence about the neural network's prediction, we have devised a way to use the results from the data-driven verification within the safety controller. To accomplish this, the verified "safe regions" (each consisting of a centroid, a radius, and a label) are provided to the safety-controller in a searchable format, and then searched at runtime to discover whether or not a given input resides within one of the regions and has a matching label. Code listing 2 shows a simplified python implementation of a search function that the safety-controller could call to determine whether or not a given input resides within one of the safe regions. If the input resides within a safe region with a matching label, the safety-controller would have a strong degree of confidence that the prediction is correct. Otherwise, the safety-controller knows that it has a lower degree of confidence about the prediction because it is either from part of the input space that was not formally verified, or exists in a verified region but has a label mismatch.

```
from scipy.spatial import distance

# regions:list of safe regions, x:the input, y:predicted label for x
def exists_in_safe_region(regions, x, y):
    for r in regions:
        label_match = r['label'] == y
        in_region = distance.euclidean(r['centroid'], x) <= r['radius']
        if label_match and in_region:
            return True
    return False
```

Listing 2 Searching safe regions

IV. Conclusion and Next Steps

We presented the formal verification of a neural network that predicts takeover-time in a shared-control semi-autonomous driving system. We analyzed its sensitivity and robustness with several techniques using the Marabou verification tool. The sensitivity analysis provides insight into the importance of input features, in addition to providing formal guarantees with respect to the regions in the input space where the network behaves as expected. The asymmetric sensitivity analysis has the added benefit of providing the opportunity to discover biases with respect to negative and positive perturbations to individual features. We also evaluated the network's robustness using local robustness, targeted robustness, and a data-driven verification approach. Compared to targeted and local robustness, the data-driven approach has the benefit of verifying the robustness of larger regions of the input space while testing fewer inputs. We have also shown an example of how these results can be leveraged and included in a "safety controller" to provide confidence about the network's predictions.

For next steps, we plan to integrate the results of the neural network and verification results into the safety controller, which will be then evaluated using the simulator. We also plan to try to improve the network's robustness by experimenting with different adversarial training techniques. One idea is to use the counterexamples discovered by Marabou during re-training, and another is to use an adversarial training framework such as "Clever Hans". Another idea for future work is to experiment with additional clustering algorithms to try to consolidate some of the regions. We also plan to try to reduce the number of features in the model by using the results from our sensitivity analysis along with other feature importance analysis techniques to discover features that may be able to be dropped from the model.

References

[1] KPMG International, "Autonomous Vehicles Readiness Index," , 2018. URL https://home.kpmg.com/content/dam/kpmg/xx/pdf/2018/01/avri.pdf.

[2] US Department of Transportation – Intelligent Transportation System Joint Program Office, "Automation Research at USDOT," , 2018. URL https://www.its.dot.gov/automated_vehicle/avr_plan.htm.

[3] Katz, G., and et al., "The Marabou Framework for Verification and Analysis of Deep Neural Networks," *CAV*, 2019, pp. 443–452.

[4] Pakdamanian, E., Sheng, S., Baee, S., Heo, S., Kraus, S., and Feng, L., "DeepTake: Prediction of Driver Takeover Behavior using Multimodal Data," *arXiv preprint arXiv:2012.15441*, 2020.

[5] Roli, F., Biggio, B., and Fumera, G., "Pattern Recognition Systems under Attack," *CIARP (1)*, Lecture Notes in Computer Science, Vol. 8258, Springer, 2013, pp. 1–8.

[6] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R., "Intriguing Properties of Neural Networks," , 2013. Technical Report. http://arxiv.org/abs/1312.6199.

[7] Papernot, N., McDaniel, P. D., Goodfellow, I. J., Jha, S., Celik, Z. B., and Swami, A., "Practical Black-Box Attacks against Machine Learning," *AsiaCCS*, ACM, 2017, pp. 506–519.

[8] Goodfellow, I. J., Shlens, J., and Szegedy, C., "Explaining and Harnessing Adversarial Examples," , 2014. Technical Report. http://arxiv.org/abs/1412.6572.

[9] Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B., "Detecting Adversarial Samples from Artifacts," , 2017. Technical Report. http://arxiv.org/abs/1703.00410.

[10] Carlini, N., and Wagner, D., "Towards evaluating the robustness of neural networks," *Proc. 38th IEEE Symposium on Security and Privacy*, 2017.

[11] Chang, K., Parvez, M. R., Chakraborty, S., and Ray, B., "Building Language Models for Text with Named Entities," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 2018, pp. 2373–2383. URL https://aclanthology.info/papers/P18-1221/p18-1221.

[12] Gopinath, D., Katz, G., Pasareanu, C. S., and Barrett, C. W., "DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks," *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, Lecture Notes in Computer Science, Vol. 11138, edited by S. K. Lahiri and C. Wang, Springer, 2018, pp. 3–19. https://doi.org/10.1007/978-3-030-01090-4_1, URL https://doi.org/10.1007/978-3-030-01090-4_1.

Appendix C Maintaining driver attentiveness in shared-control autonomous driving

Maintaining driver attentiveness in shared-control autonomous driving

Radu Calinescu, Naif Alasmari and Mario Gleirscher
Department of Computer Science, *University of York*, York, U.K.
{radu.calinescu,nnma500,mario.gleirscher}@york.ac.uk

***Abstract*—We present a work-in-progress approach to improving driver attentiveness in cars provided with automated driving systems. The approach is based on a control loop that *monitors* the driver's biometrics (eye movement, heart rate, etc.) and the state of the car; *analyses* the driver's attentiveness level using a deep neural network; *plans* driver alerts and changes in the speed of the car using a formally verified controller; and *executes* this plan using actuators ranging from acoustic and visual to haptic devices. The paper presents (i) the self-adaptive system formed by this monitor-analyse-plan-execute (MAPE) control loop, the car and the monitored driver, and (ii) the use of probabilistic model checking to synthesise the controller for the planning step of the MAPE loop.**

***Index Terms*—autonomous driving, shared control, MAPE control loop, controller synthesis, probabilistic model checking**

I. Introduction

The J3016 standard [1] classifies automated driving systems (ADSs) on a six-level scale, from no automation at Level 0 to full automation at Level 5. Despite huge R&D budgets and much hype over the past decade, fully autonomous (Level 5) cars are unlikely to become available to the general public any time soon. In contrast, cars providing Level 2 (i.e., partial) automation can be purchased from manufacturers including Tesla, Nissan and BMW; and the approval of Level 3 (i.e., conditional automation) and 4 (i.e., high automation) cars is considered by regulators worldwide [2], [3], [4], [5], [6].

A critical requirement for vehicles operating at autonomy Levels 2 and 3 is that a user resides in the driver's seat and is sufficiently attentive to be able to share the control of the car with the ADS. At Level 2, this human in the loop is expected to '*complete the object and event detection and response subtask and* [to] *supervise the driving automation system*', while at Level 3 the user is expected to be '*receptive to ADS-issued requests to intervene* [. . .] *and* [to] *respond appropriately*' [1]. Although Level 4 ADSs do not rely on human support, they may still issue *timely requests for human intervention* (e.g., when they approach roads or traffic situations they were not designed to handle), performing a minimum-risk manoeuvre (e.g., stopping the car safely) if their user does not respond.

In all these scenarios, accidents with potentially fatal consequences (for Levels 2 and 3) and frequent emergency stops (for Level 4) can only be avoided if the drivers are sufficiently attentive to be able to take over the control of their vehicles [7]. However, humans find it extremely difficult to remain attentive when overseeing the operation of automated and autonomous systems [8], [9], [10]. In the automotive domain, this is amply demonstrated by accidents involving both cars with Level 2 ADS used by regular drivers [11], [12] and cars with higher autonomy levels tested by professional safety drivers [13].

Our paper proposes an approach that mitigates this problem by using a monitor-analyse-plan-execute (MAPE) control loop to improve driver attentiveness in shared-control autonomous driving. The monitoring component of this MAPE loop uses an array of sensors to collect driver biometrics and vehicle data. The analysis component uses these data and a deep neural network [14] developed by a complementary research strand of our Safe-SCAD project[1] to predict the driver response time and response quality to a potential ADS intervention request. These predictions are processed based on a simple driver attentiveness model that factors in the speed of the car, and the resulting classification of the driver as attentive, semi-attentive or inattentive guides the planning of driver alerts and car speed changes by a formally verified discrete-event controller. This controller achieves Pareto-optimal trade-offs between risk level, driver nuisance, and progress with the journey.

The first version of our Safe-SCAD approach is aimed at Level 3/4 ADSs, with a particular focus on the *Automated Lane Keeping System* (ALKS) for which the United Nations World Forum for Harmonization of Vehicle Regulations adopted a new UN regulation [6] in June 2020, and that the UK Department for Transport plans to implement on UK motorways [2]. For these advanced ADSs, the Safe-SCAD improvement in driver attentiveness can lead to fewer minimum-risk manoeuvres and better progress with the car journey, and with minimal risk of accidents. Future enhancements (summarised later in the paper) will mitigate additional uncertainties associated with the challenging problem tackled by our approach, extending its applicability to Level 2 ADSs, and to autonomous car testing by professional test drivers.

The key contributions of the paper are the presentation of the driver attentiveness management problem in ALKS (as a motivating example, in Sect. II), the Safe-SCAD approach to improving driver attentiveness in shared-control autonomous driving (Sect. III), and the probabilistic model checking method for synthesising the Safe-SCAD planning component (Sect. IV). The paper also discusses related work (Sect. V) and summarises our plans for future work (Sect. VI).

This research was funded by the Lloyds Register Foundation under the Assuring Autonomy International Programme grant Safe-SCAD.

[1] Safety of Shared Control in Autonomous Driving, https://cutt.ly/Safe-SCAD

II. Driver attentiveness management problem

A. Background

We consider an ADS with the characteristics stipulated in the United Nations' ALKS regulation [6]. The ALKS can be activated by a driver (who must be available in the driving seat, with the seatbelt fastened) when all its components are fully operational, and the vehicle is on roads and in environment (e.g., weather) conditions within its operational design domain (ODD). When activated, the system keeps the vehicle inside its lane, controlling the vehicle speed (within the range 0 to 60 km/h) to adapt to the surrounding road traffic. Additionally, the ALKS can detect the risk of collision (e.g., due to a stationary vehicle) and can stop to avoid the collision, e.g., by performing an emergency manoeuvre.

In certain situations, all of which it must recognise, the ALKS issues a *transition demand*, i.e., a request for the driver to take over the control of the vehicle. The regulation allows these situations to differ across manufacturers. However, they must include the situations in which the ALKS activation conditions are not met (e.g., the vehicle approaches a road outside its ODD), and those in which the driver is unavailable (i.e., not in the driving seat or *inattentive*) and not responding to ALKS alerts aimed at restoring the driver's availability. Transition demands are issued timely, allowing (i) an *attentive driver* to resume the manual driving safely, or (ii) the ALKS to perform a minimum-risk manoeuvre (MRM), e.g., to bring the vehicle to a standstill, if the driver is inattentive. The ALKS may reduce the vehicle speed to ensure safety, e.g., by allowing the driver additional time for the control takeover.

It follows from the summary so far that the ALKS must be capable of assessing the driver's availability, including their position in the car (in the driving seat, wearing the seatbelt) and their attentiveness. For the latter, the regulation proposes the use of driver biometrics such as *'eye blinking, eye closure, conscious head or body movement'* [6], but allows manufacturers to select their own methods for assessing driver attentiveness. Likewise, the regulation and UK's ALKS plan [2] recommend the use of optical, acoustic and haptic warning signals (i) to announce transition demands to the driver, and (ii) to improve driver attentiveness, but are not prescriptive about how these alerts should be used. In the next section, we use these recommendations to define the *driver attentiveness management problem* for ALKS-like ADsS.

B. Problem definition

Given an ALKS, we assume that its driver can have one of $n \geq 2$ attentiveness levels. The highest level ('attentive') corresponds to the situation in which the driver can respond timely to a transition demand, even at the maximum speed permitted for the vehicle. The lowest level ('inattentive') corresponds to the situation where the ALKS needs to execute an MRM unless the driver improves their level of attentiveness within a mandated time period $\tau > 0$.[2] If present, any intermediate levels (e.g., 'semi-attentive' for $n = 3$) correspond to diminished driver attentiveness that does not require an MRM. However, they provide an opportunity for issuing alerts to improve the driver's attentiveness level before it drops further, and drastic action is required: MRMs involve stopping the vehicle in a motorway lane [2], [6], and should only be used as a last resort.

We assume that the ADS has two mechanisms it can use when the driver is not 'attentive'. First, it can activate one or several of $m \geq 1$ *alerts* (e.g., optical, acoustic and haptic) as needed to improve the driver's attentiveness. Second, it can reduce the car speed to one of $q \geq 1$ speed levels, where we allow $q = 1$ for the case when this feature is not available. As such, the ALKS state at any point in time is characterised by:

1) the driver attentiveness level $l \in \{0, 1, \ldots, n-1\}$, where $l = 0$ and $l = n - 1$ correspond to the driver being 'attentive' and 'inattentive', respectively;
2) the set of active alerts $a \in \{0, 1\}^m$, where $a = (a_1, a_2, \ldots, a_m)$ indicates that the i-th alert is inactive when $a_i = 0$, and active when $a_i = 1$;
3) the vehicle speed level $v \in \{0, 1, \ldots, q - 1\}$.

Using the notation $L = \{0, 1, \ldots, n - 1\}$, $A = \{0, 1\}^m$ and $V = \{0, 1, \ldots, q - 1\}$ to denote the range for the three components of the ALKS state, we further assume that the following measures are defined over the state space $L \times A \times V$:

1) $nuisance : A \to \mathbb{R}_{\geq 0}$, where $nuisance(a)$ represents the nuisance experienced by the driver when the alerts $a \in A$ are in use, with $nuisance(0, 0, \ldots, 0) = 0$;
2) $progress : V \to \mathbb{R}_{\geq 0}$, where $progress(v)$ reflects the progress with the journey made when the vehicle travels at speed $v \in V$ (e.g., the distance travelled in one hour);
3) $risk : L \times V \to \mathbb{R}_{\geq 0}$, where $risk(l, v)$ provides a measure of the risk associated with travelling at speed $v \in V$ when the driver attentiveness level is $l \in L$,

and that $risk_{\mathsf{MRM}} > 0$ denotes the risk associated with performing an MRM.

Finally, we assume that timing data are available about the drivers' transition between the attentiveness levels L, when different alert combinations are active, and at different vehicle speeds. These data may be available from studies of driver behaviour [15], [16], experiments carried out by ALKS manufacturers, observations of drivers who are using the deployed ADS, or a combination thereof. Given such data, the *driver attentiveness management problem* is to find a combination of alerts $a(s) \in A$ to use in each ALKS state $s \in L \times A \times V$, such that the ALKS achieves Pareto optimality between minimising the driver nuisance, maximising the progress with the journey, and minimising the risk over a period of T hours of driving.

III. The Safe-SCAD approach

Our Safe-SCAD approach addresses the driver attentiveness management problem from Section II-B by using a MAPE control loop with the components shown in Figure 1. These components and the four stages of the MAPE loop are described in the following sections. We provide only a brief summary for the components used in the first two MAPE

[2] The UK consultation document proposes $\tau = 15$s [2, p. 17].

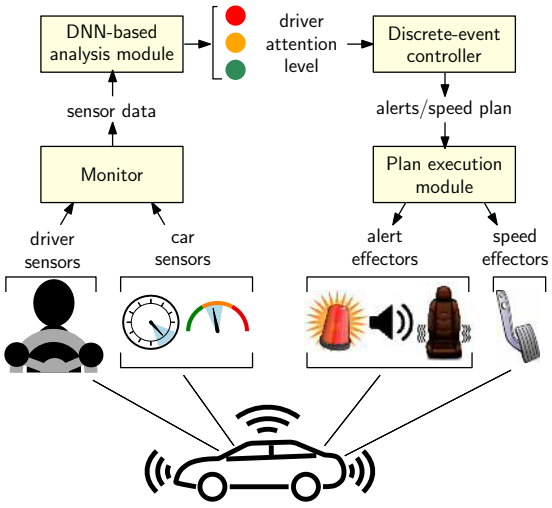


Fig. 1. Safe-SCAD driver attentiveness management approach for ALKS with $n = 3$ attentiveness levels (attentive, semi-attentive, and inattentive)

stages (monitoring and analysis), as their technical details are available in [14], [17], and we focus instead on their integration into a MAPE loop and on the planning MAPE stage, which represent the two key theoretical contributions of this paper.

A. Monitoring

In this MAPE stage, data are collected from a combination of driver-biometrics sensors and vehicle sensors. In our project, driver biometrics are obtained [14] using: (i) eye-tracking glasses to monitor eye movement data (e.g., gaze position and fixation time); (ii) smartwatch photoplethysmographic sensors to monitor heart rate; and (iii) smartwatch galvanic skin response sensors to monitor hand sweating. A broad range of vehicle data streams are already collected and used by ADSs, and can easily be exploited within our MAPE loop. These range from vehicle velocity and steering wheel angle to lane position and throttle/brake pedal angles.

B. Analysis

This MAPE stage (Figure 2) uses the DeepTake predictor of driver takeover behaviour [14] developed by another Safe-SCAD research strand. DeepTake is a deep neural network (DNN) that uses the driver-biometrics and vehicle data from the monitoring stage to predict the driver's control takeover:

1) *intention*, i.e., whether the driver would react to an ADS control-transition demand or not;
2) *time* elapsed from the transition demand until the driver assumes manual control of the vehicle, as defined by the ISO 21959 standard [18];
3) *quality* of the driver's manoeuvring of the vehicle after manual control is resumed.

DeepTake was shown [14] to predict these driver takeover metrics with an accuracy of 96%, 93% and 83%, respectively. We emphasise that these accuracy levels are sufficient for the Safe-SCAD driver attentiveness management because our solution is intended for use with ALKS that can ensure safety at all times, e.g., by performing an MRM if necessary.

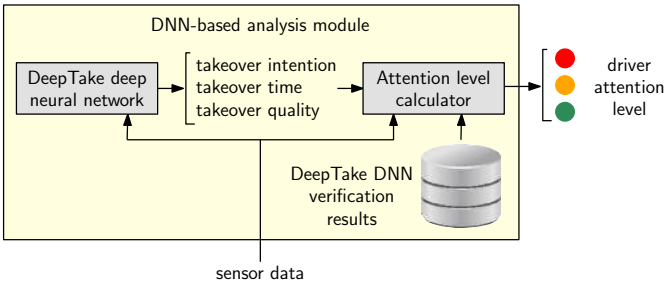


Fig. 2. Safe-SCAD analysis MAPE stage

Additionally, our DNN-based analysis module operates conservatively by also exploiting the results from the design-time *robustness verification of DeepTake* [17]. Figure 3 shows how we intend to use these verification results in the post-processing of the DeepTake predictions, such that the computed driver attention level is lowered when the sensor data belongs to regions of the DeepTake input space that were not identified as robust by the DNN verification. This part of our Safe-SCAD approach is under development.

C. Planning

In this MAPE stage, a discrete-event controller plans the set of active alerts $a \in A$ and the speed level $v \in V$ that the vehicle should employ. This controller is activated by the occurrence of two types of events:

- changes in the attentiveness level of the driver;
- the expiry of a timer that is used to activate the controller periodically at all times when the driver attentiveness level is not 'attentive'.

The timer enables the controller to periodically "try" new or additional alerts and/or speed adjustments when the driver attentiveness level was not improved by the execution of the plan devised by the previous controller activation. The synthesis of the Safe-SCAD controller is detailed in Section IV.

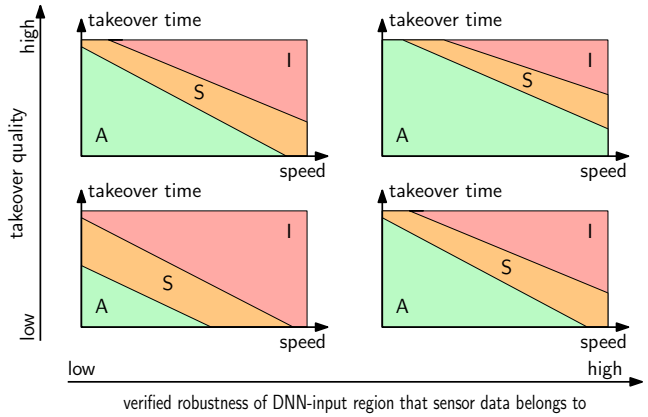


Fig. 3. The driver attention level (A=attentive, S=semi-attentive, I=inattentive) depends on the predicted driver takeover time and quality, on the speed of the vehicle, and on the verified robustness of the DeepTake input region that the sensor data belongs to. The diagram applies to a positive takeover intention (i.e., driver responsive to a control-transition demand); when DeepTake predicts a negative intention, the driver state is deemed inattentive.

D. Execution

In this MAPE stage, the alert and speed effectors of the vehicle are used to implement the alerts and speed plan provided by the Safe-SCAD controller.

IV. Safe-SCAD controller synthesis

To synthesise the Safe-SCAD controller used in the planning MAPE stage, we model the relevant behaviour of the self-adaptive system from Figure 1 as a parametric continuous-time Markov chain (CTMC). The parameters of this CTMC are chosen such that the (non-parametric) CTMC induced by each combination of parameter values corresponds to a different feasible controller, and the CTMCs obtained by considering all valid combinations of parameter values define the set of feasible Safe-SCAD controllers, i.e., the *controller design space*. Given this design space, we synthesise formally verified controllers by using (i) probabilistic model checking to determine the *nuisance*, *progress* and *risk* associated with any specific controller; and (ii) multi-objective genetic algorithms to find controllers that achieve Pareto-optimal trade-offs between these three measures. We detail the steps of our controller synthesis process in Sections IV-B and IV-C, after a brief introduction to continuous-time Markov chains and their probabilistic model checking in Section IV-A.

A. Continuous-time Markov chains

A CTMC is a finite state-transition model $M = (S, s_0, \mathbf{R})$, where S is a finite set of states, $s_0 \in S$ is the initial state, and $\mathbf{R} : S \times S \to [0, \infty)$ is a transition rate matrix such that for any state $s_i \in S$, the probability that the CTMC will transition from state s_i to another state within $t > 0$ time units is $1 - e^{-t \cdot \sum_{s_k \in S} \mathbf{R}(s_i, s_k)}$, and the probability that the new state is $s_j \in S$ is given by $\mathbf{R}(s_i, s_j) / \sum_{s_k \in S} \mathbf{R}(s_i, s_k)$.

To enable the analysis of a broader range of CTMC properties, the states and transitions of a CTMC are often annotated with *rewards*. A *reward structure* over a CTMC with state set S is a pair of functions $r^X = (r_1, r_2)$, where $r_1 : S \to \mathbb{R}_{\geq 0}$ is a *state reward function* that defines the rate $r_1(s)$ at which the reward is obtained while the Markov chain is in state s; and $r_2 : S \times S \to \mathbb{R}_{\geq 0}$ is a *transition reward function* that defines the reward obtained each time a transition occurs.

Probabilistic model checkers such as PRISM [19] and Storm [20] use efficient symbolic model checking techniques to analyse a wide range of CTMC properties expressed in *continuous stochastic logic* (CSL) augmented with rewards [21]. These properties include bounded and unbounded probabilistic reachability, and several types of reward properties. For our Safe-SCAD controller synthesis, we are interested in *cumulative reward properties*. These properties are expressed using the CSL formula $R^X_{=?}[C^{\leq T}]$, which denotes the expected value of the reward X accrued within the time interval $[0, T]$.

B. Safe-SCAD controller design space

We model the driver attentiveness management problem using a family of CTMCs to define the *design space* (i.e., the possible variants) of the Safe-SCAD controller. Each CTMC in the family has the state set $S = L \times A \times V \times \{c, \overline{c}\}$, where L, A and V are defined in Section II-B, and c is a Boolean state variable that indicates whether the discrete-event controller is active or not. As shown in Figure 4, which depicts the controller design space for a specific instance of the problem, the model has three types of state transitions:

1) *Transitions corresponding to changes in driver attentiveness.* These transitions occur from states $(l, a, v, \overline{c})$, in which the controller is inactive, to states (l', a, v, c) with a different driver attentiveness level (i.e., $l' \neq l$), no changes in the alerts a and speed v, and the controller activated.
2) *Transitions corresponding to fixed controller actions.* There are two classes of such actions. In the first, the CTMC transitions from states $('\mathsf{A}', a, v, c)$, in which the driver is attentive and the controller activated, to the initial state $('\mathsf{A}', (0,0), 0, \overline{c})$; this happens whenever the controller is activated by a change in driver attentiveness level, finds the driver fully attentive, and therefore switches off any activated alerts, and selects the nominal driving speed. In the second, the controller is activated by a timer whenever the driver has not become fully attentive after a period of time since a previous controller action. In this situation, the

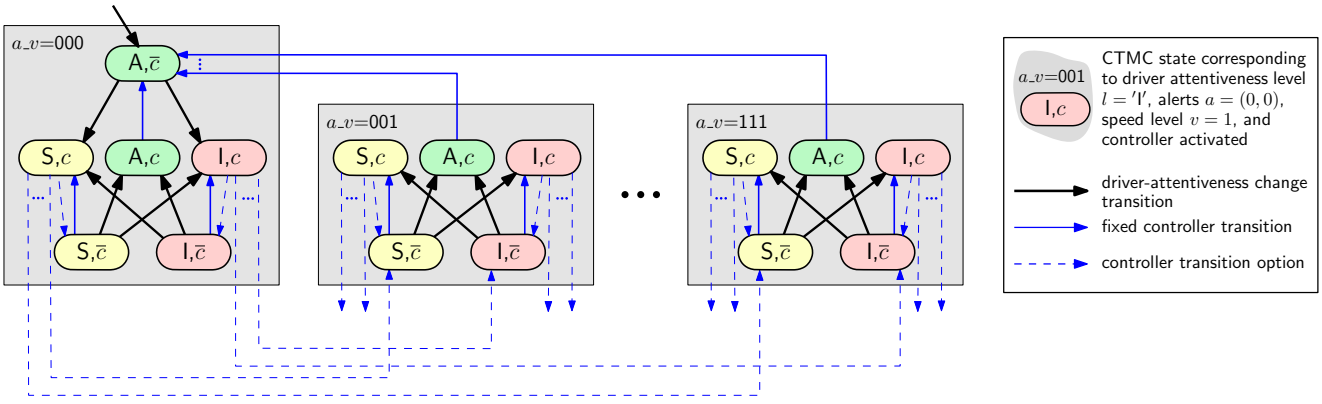


Fig. 4. Safe-SCAD controller design space for the driver attentiveness management problem with three driver attentiveness levels ($L = \{'\mathsf{A}', '\mathsf{S}', '\mathsf{I}'\}$, where A=attentive, S=semi-attentive and I=inattentive), two alerts ($A = \{0,1\}^2$) and two speed levels ($V = \{0,1\}$). In the initial state (indicated by an incoming arrow) the driver is attentive, the controller is inactive, the alerts $a = (0,0)$ are inactive, and the car drives at nominal speed level $v = 0$; for brevity, this combination of alert activations and speed level is denoted $a\_v = 000$ in the diagram.

CTMC transitions from each state $(l, a, v, \overline{c})$ with $l \neq \text{'A'}$ to the counterpart state (l, a, v, c) in which the activated controller has the opportunity to switch on new alerts and/or to select a new speed level.

3) *Transitions corresponding to controller options.* When the controller is activated (by a change in driver attentiveness level or by the timer) and finds the driver to not be fully attentive, it has a choice of using any available combination of alerts and any speed level in order to make the driver attentive and to reduce risk. Thus, from each state $(l, a, v, c) \in S$ with $l \neq \text{'A'}$, the CTMC can transition to any state $(l, a', v', c) \in S$. These controller options are indicated by dashed transitions in Figure 4. In the general case from Section II-B, there are n driver attentiveness levels, m alerts and q speed levels, giving the controller $2^m q$ combinations of alerts and speed level options to choose from in each of the $(n-1)2^m q$ CTMC states in which $l \neq \text{'A'}$. We encode the controller option for each of the $n-1$ driver attentiveness levels $l \in L \setminus \{\text{'A'}\}$ and each of the $2^m q$ combinations of alerts and speed level $a\_v \in A \times V$ using a design-space parameter

$$option_{l,a\_v} \in \{0, 1, \ldots, 2^m q - 1\}. \quad (1)$$

We have $(n-1)2^m q$ such parameters, and each assignment of values to these parameters defines a candidate deterministic controller[3] solution for the driver attentiveness management problem. There are $(2^m q)^{(n-1)2^m q}$ candidate solutions in total, and thus $(2^2 2)^{(3-1)2^2 2} = 8^{16} \approx 10^{14}$ candidate solutions for the problem instance encoded by the parametric CTMC from Figure 4.

To complete the definition of the controller design space, we also need to specify its CTMC transition rates. The last two types of transitions described above correspond to controller actions. Therefore, their rates must reflect the mean operation time that the controller requires: (i) to plan the new alerts and speed level when it is activated, and (ii) to be activated by its timer. For instance, a controller operation time of 500ms gives a rate of 2s^{-1}. These rates are easy to determine, e.g., by worst-case execution time analysis of the controller code. In contrast, the rates for the first type of transition are much more difficult to determine because they encode the mean time taken by the driver to transition between attentiveness levels, for each combination of active alerts and speed levels. These rates must be estimated, e.g., by using data sources such as:

1) the numerous available studies and surveys of driver attentiveness, e.g. [22], [15], [16], [23];
2) additional data from controlled experiments with drivers of ALKS vehicles;
3) driver data collected during the actual driving of ALKS vehicles, e.g., by using a black-box solution similar to that already employed by many insurers of new drivers [24], [25], either across a fleet of vehicles or for a specific driver.

[3]A deterministic controller is a controller which, for any state $s \in S$, performs the same action each time when state s is reached.

We note that using the last data source enables both (i) the definition of personalised controller design spaces for each driver, and (ii) the continual updating of these design spaces to support the runtime synthesis of new Safe-SCAD controllers when the transition rates for a driver change significantly [26].

C. Synthesis of Pareto-optimal Safe-SCAD controllers

The synthesis of Safe-SCAD controllers solves the driver attentiveness management problem. For this purpose, we define the reward structures $r^{nuisance}$, $r^{progress}$ and r^{risk} over the CTMCs from our controller design space. The definitions of these reward structures are directly based on the definitions of the three measures with the same names from Section II-B. For instance, the state and transition reward functions for the first reward structure are given by $r^{nuisance}.r_1(l, a, v, c?) = nuisance(a)$ for any CTMC state $(l, a, v, c?) \in S$, and $r^{nuisance}.r_2(s_1, s_2) = 0$ for any CTMC transition $(s_1, s_2) \in S \times S$, respectively. Given these reward structures, the driver attentiveness management problem for a journey of duration $T > 0$ can be formalised as:

> *Find the set of controller options* (1) *whose associated CTMCs from the controller design space achieve Pareto-optimal trade-offs between minimising* $R^{nuisance}_{=?}[C^{\leq T}]$, *maximising* $R^{progress}_{=?}[C^{\leq T}]$ *and minimising* $R^{risk}_{=?}[C^{\leq T}]$,

where the three CSL cumulative reward properties represent the overall nuisance, overall progress with the journey, and overall risk accrued over a journey of duration T, respectively.

We solve this problem using the search-based software engineering tool EvoChecker [27], [28], which:

1) obtains the precise values of the three reward properties for any given CTMC from the controller design space using a probabilistic model checker (the tool can be configured to use PRISM [19] or Storm [20]);
2) synthesises a close approximation of the Pareto-optimal set of controller options (1) by using multi-objective genetic algorithm (MOGA) optimisation (the tool can be configured to work with any of the NGSA-II [29], SPEA2 [30] or MOCell [31] MOGAs).

To this end, we supply EvoChecker with: (i) our controller design space from Section IV-B, encoded in the high-level PRISM modelling language [19] extended with EvoChecker constructs that we use to specify the possible values for the parameters (1); and (ii) the three CSL reward properties specifying the optimisation objectives from our problem. Figure 5 shows how the controller design space from Figure 4 is expressed in this encoding. Due to space constraints, only a fragment of the encoding is shown, but we made the entire encoding (and the other artifacts from this section) available for inspection at https://cutt.ly/SafeSCAD-SEAMS21. Given the controller design space and the optimisation objectives, EvoChecker synthesises a close approximation of the Pareto-optimal set of Safe-SCAD controllers, and the Pareto front associated with this set. Figure 6 shows the Pareto front obtained for the instance of the driver attentiveness management

```
// Controller options specifying the next a_v ∈ {000(2), 001(2), ..., 111(2)} value
evolve int option_S,0 [0..7]; // when driver is semi-attentive and a_v = 000(2)
...
evolve int option_S,7 [0..7]; // when driver is semi-attentive and a_v = 111(2)
evolve int option_I,0 [0..7]; // when driver is inattentive and a_v = 000(2)
...
evolve int option_I,7 [0..7]; // when driver is inattentive and a_v = 111(2)

module Controller
  c : [0..1] init 0;    // 0 = controller inactive, 1 = controller active
  a_v : [0..7] init 0; // current alerts a and speed level v

  // activate controller
  [driver_change] c=0 → (c' = 1);          // when driver state changes
  [ ] c = 0 ∧ l ≠ 0 → timerRate : (c' = 1); // periodically if driver not attentive

  // switch off alerts and use nominal speed if the driver is attentive (l = 0)
  [ ] c = 1 ∧ l = 0 → controllerRate : (a_v' = 0)&(c' = 0);

  // controller actions for driver attentiveness level l = 1 (semi-attentive)
  [ ] c = 1 ∧ l = 1 ∧ a_v = 0 → controllerRate : (a_v' = option_S,0)&(c' = 0);
  ...
  [ ] c = 1 ∧ l = 1 ∧ a_v = 7 → controllerRate : (a_v' = option_S,7)&(c' = 0);

  // controller actions for driver attentiveness level l = 2 (inattentive)
  [ ] c = 1 ∧ l = 2 ∧ a_v = 0 → controllerRate : (a_v' = option_I,0)&(c' = 0);
  ...
  [ ] c = 1 ∧ l = 2 ∧ a_v = 7 → controllerRate : (a_v' = option_I,7)&(c' = 0);
endmodule
```

Fig. 5. Fragment of EvoChecker-encoded controller design space for $m = 2$ independent alerts and $q = 2$ speed levels

with the design space from Figure 4 and a driving time of $T = 4$ hours. Each element of this Pareto front corresponds to a controller variant whose nuisance, risk and progress values from Figure 6 were obtained by EvoChecker through formal verification using the Storm model checker.

V. Related work

Human-machine interaction in driving automation includes the identification and handling of control transitions between the driver and the vehicle [32], [18] and, in support of that, managing the driver's attentiveness and involvement necessary for such transitions, taking into account the current traffic situation with its potential adversity.

Recent research deals with managing such control transitions [18], [33] or continuous shared control [32], and includes inventions about control transitions that require but do not manage driver attentiveness [33], [34], [35]. Similarly, an earlier invention [36] focuses on classifying the driver state and on technologies for situation-adapted collision avoidance by combining braking and driver warnings. A range of experiments [37], [38], [39] investigate parameters of the driver state and behaviour (e.g. response times, drowsiness, influence of traffic density and driver workload), however, with little data about the time it takes drivers to become inattentive.

Overall, none of the works we found discusses how attentiveness monitoring and control software can be automatically designed and adapted for optimal safety and performance, and how such software can be confidently assured to fulfil regulatory requirements [40]. In contrast, our Safe-SCAD approach focuses on the design and analysis of a MAPE control loop supported by such software, assuming the availability of specific sensor technology for estimating the driver state [37]. Moreover, we provide a generic method for defining the design space for this control software, and for its automated synthesis with probabilistic guarantees. Finally, our exhaustive formal

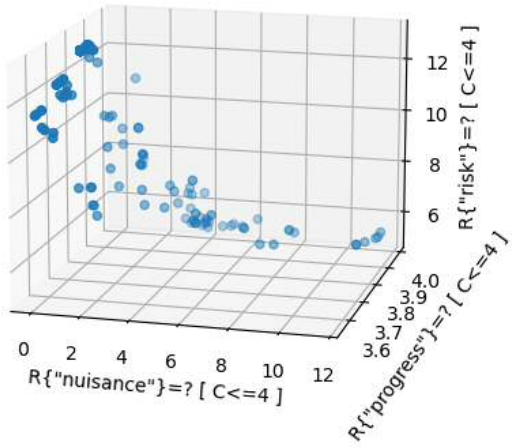


Fig. 6. Pareto front associated with the set of Pareto-optimal Safe-SCAD controllers for the controller design space instance from Figure 4, as synthesised in 98.58s by EvoChecker configured to use Storm [20] and NSGA-II [29] (population size 7000 × 1000 iterations) and running on a 3.6GHz Intel Core i3 Mac OSX 10.14.6 Mac mini computer with 16 GB of memory

verification approach based on probabilistic model checking is also an improvement over the purely testing-based approach that the safety of the intended functionality (SOTIF) standard recommends for the verification of automated driving systems [40, §10.3, Table 5-7].

VI. Conclusion

We introduced a MAPE control loop for improving driver attentiveness in ADS-enhanced cars, and we described a novel method for the rigorous synthesis of the controller used in the planning stage of this MAPE loop. In the next stage of our project, we will complete the development of the DNN-based analysis module from Figure 2 by integrating our project's DeepTake predictor of driver takeover behaviour [14] and its verification results [17] into the end-to-end solution presented in this paper. The complete solution will then be evaluated experimentally, in a study carried out using our driving simulator from [14].

Additionally, we will assess whether the good EvoChecker scalability reported in [28] extends to our controller synthesis problem with larger numbers of alerts m and speed levels q (cf. Section II-B). Finally, we plan to explore: (i) the use of personalised and adaptive controllers (as described in Section IV-B); (ii) the use of the CTMC-refinement technique from [41], [42] to improve the accuracy of the Safe-SCAD controller design space; (iii) the use of the robust synthesis techniques from [43], [44] to generate controllers tolerant to variations in driver behaviour; and (iv) the potential advantages of using probabilistic Safe-SCAD controllers, i.e., controllers for which the options (1) are discrete probability distributions over the set of actions available to the controller.

Acknowledgements

This project has received funding from the Assuring Autonomy International Programme project 'Safety of shared control in autonomous driving' and the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'.

References

[1] On-Road Automated Driving (ORAD) committee, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE International, Standard J3016_201806, 2018. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/preview/

[2] Centre for Connected and Autonomous Vehicles, "Safe use of automated lane keeping system (ALKS)," UK Department for Transport, Tech. Rep., August 2020. [Online]. Available: https://cutt.ly/UK-DfT-ALKS

[3] California State Assembly, "Assembly Bill 2866 Autonomous vehicles," February 2016. [Online]. Available: http://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201520160AB2866

[4] European Parliament, "Regulation (EU) 2019/2144 of the European Parliament and of the Council on type-approval requirements for motor vehicles," *Official Journal of the European Union*, vol. L 325/1, 2019.

[5] T. Imai, "Legal regulation of autonomous driving technology: Current conditions and issues in Japan," *IATSS Research*, vol. 43, no. 4, pp. 263–267, 2019.

[6] UNECE World Forum for Harmonization of Vehicle Regulations, "ECE/TRANS/WP.29/2020/81: United Nations Regulation on Uniform provisions concerning the approval of vehicles with regard to Automated Lane Keeping Systems," June 2020. [Online]. Available: https://cutt.ly/ALKS-regulation

[7] N. Merat and A. H. Jamson, "How do drivers behave in a highly automated car?" in *5th International Driving Symposium on Human Factors in Driver Assessment, Training, and Vehicle Design: Driving Assessment 2009*. University of Iowa, 2009.

[8] R. Chai, G. R. Naik, T. N. Nguyen *et al.*, "Driver fatigue classification with independent component by entropy rate bound minimization analysis in an eeg-based system," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 3, pp. 715–724, 2016.

[9] J. F. Duffy, K.-M. Zitting, and C. A. Czeisler, "The case for addressing operator fatigue," *Reviews of Human Factors and Ergonomics*, vol. 10, no. 1, pp. 29–78, 2015.

[10] G. Matthews and P. A. Hancock, *The Handbook of Operator Fatigue*. CRC Press, 2017.

[11] V. A. Banks, K. L. Plant, and N. A. Stanton, "Driver error or designer error: Using the perceptual cycle model to explore the circumstances surrounding the fatal Tesla crash on 7th May 2016," *Safety Science*, vol. 108, pp. 278–285, 2018.

[12] US National Transportation Safety Board, "Collision between a sport utility vehicle operating with partial driving automation and a crash attenuator," February 2020. [Online]. Available: https://www.ntsb.gov/news/events/Documents/2020-HWY18FH011-BMG-abstract.pdf

[13] F. M. Favarò, N. Nader, S. O. Eurich, M. Tripp, and N. Varadaraju, "Examining accident reports involving autonomous vehicles in California," *PLoS one*, vol. 12, no. 9, p. e0184952, 2017.

[14] E. Pakdamanian, S. Sheng, S. Baee, S. Heo, S. Kraus, and L. Feng, "DeepTake: Prediction of driver takeover behavior using multimodal data," in *ACM CHI Conference on Human Factors in Computing Systems*, 2021. [Online]. Available: https://arxiv.org/abs/2012.15441

[15] A. Lotz and S. Weissenberger, "Predicting take-over times of truck drivers in conditional autonomous driving," in *International Conference on Applied Human Factors and Ergonomics*, 2018, pp. 329–338.

[16] Q. Maia, M. A. Grandner *et al.*, "Short and long sleep duration and risk of drowsy driving and the role of subjective sleep insufficiency," *Accident Analysis & Prevention*, vol. 59, pp. 618–622, 2013.

[17] J. M. Grese, C. Pasareanu, and E. Pakdamanian, "Formal analysis of a neural network predictor in shared-control autonomous driving," in *AIAA Scitech 2021 Forum*, 2021. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1580

[18] ISO/TR 21959, "Road vehicles – human performance and state in the context of automated driving," ISO, Standard, 2020. [Online]. Available: https://www.iso.org/standard/78088.html

[19] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV'11*, 2011, pp. 585–591.

[20] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A Storm is coming: A modern probabilistic model checker," in *CAV'17*, 2017, pp. 592–600.

[21] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation*. Springer, 2007, pp. 220–270.

[22] M. Körber, L. Prasch, and K. Bengler, "Why do I have to drive now? Post hoc explanations of takeover requests," *Human Factors*, vol. 60, no. 3, pp. 305–323, 2018.

[23] W. Vanlaar, H. Simpson, D. Mayhew, and R. Robertson, "Fatigued and drowsy driving: A survey of attitudes, opinions and behaviors," *Journal of safety research*, vol. 39, no. 3, pp. 303–309, 2008.

[24] A. Kassem, R. Jabr, G. Salamouni, and Z. K. Maalouf, "Vehicle black box system," in *2nd IEEE Systems Conference*, 2008, pp. 1–6.

[25] M. A. Kumar, M. V. Suman, Y. Misra, and M. G. Pratyusha, "Intelligent vehicle black box using IoT," *Int. J. Eng. Technol*, vol. 7, no. 2, pp. 215–218, 2018.

[26] X. Zhao, R. Calinescu, S. Gerasimou, V. Robu, and D. Flynn, "Interval change-point detection for runtime probabilistic model checking," in *35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 163–174.

[27] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 319–330.

[28] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, no. 4, pp. 785–831, 2018.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[30] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[31] A. J. Nebro, J. J. Durillo, F. Luna *et al.*, "MOCell: A cellular genetic algorithm for multiobjective optimization," *International Journal of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, 2009.

[32] M. Walch, K. Mühl, J. Kraus, T. Stoll, M. Baumann, and M. Weber, "From car-driver-handovers to cooperative interfaces: Visions for driver-vehicle interaction in automated driving," in *Automotive User Interfaces*. Springer, 2017, pp. 273–294.

[33] R. Latotzki and F. Goseberg, "Handover procedure for driver of controlled vehicle," Germany Patent US 2020/0 103 898 A1, 2020. [Online]. Available: https://patents.google.com/patent/US20200103898A1/en

[34] P. L. G. Martinez and J. Yu, "Collision mitigation systems and methods using driver attentiveness," Worldwide Patent US9 047 780B2, 2013. [Online]. Available: https://patents.google.com/patent/US9047780B2/en

[35] B. Hoye, "Determining driver engagement with autonomous vehicle," U.S. Patent US10 209 708B2, 2016. [Online]. Available: https://patents.google.com/patent/US10209708B2/en

[36] M. Kopf and N. Farid, "Systems and methods for evaluating driver attentiveness for collision avoidance," Germany Patent US7 592 920B2, 2004. [Online]. Available: https://patents.google.com/patent/US7592920B2/en

[37] C. Gold, D. Damböck, K. Bengler, and L. Lorenz, "Partially automated driving as a fall-back level of high automation," in *6. Tagung Fahrerassistenzsysteme: Der Weg zum automatischen Fahren*, vol. 28, 2013. [Online]. Available: https://mediatum.ub.tum.de/doc/1187198/

[38] F. Biondi, D. L. Strayer, R. Rossi *et al.*, "Advanced driver assistance systems: Using multimodal redundant warnings to enhance road safety," *Applied Ergonomics*, vol. 58, pp. 238–244, 2017.

[39] T. E. Trimble, R. Bishop *et al.*, "Human factors evaluation of level 2 and level 3 automated driving concepts: Past research, state of automation technology, and emerging system concepts," National Highway Traffic Safety Administration, Tech. Rep., 2015. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/812043_hf-evaluationlevel2andlevel3automateddrivingconceptsv2.pdf

[40] ISO/PAS 21448, "Road vehicles – safety of the intended functionality (SOTIF)," International Standards Organisation, Standard, 2019. [Online]. Available: https://www.iso.org/standard/70939.html

[41] C. Paterson and R. Calinescu, "Accurate analysis of quality properties of software with observation-based markov chain refinement," in *IEEE International Conference on Software Architecture*, 2017, pp. 121–130.

[42] ——, "Observation-enhanced QoS analysis of component-based systems," *IEEE Trans. Softw. Eng.*, vol. 46, no. 5, pp. 526–548, 2018.

[43] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Efficient synthesis of robust models for stochastic systems," *Journal of Systems and Software*, vol. 143, pp. 140–158, 2018.

[44] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Designing robust software systems through parametric Markov chain synthesis," in *IEEE International Conference on Software Architecture*, 2017, pp. 131–140.

Appendix D Discrete-Event Controller Synthesis for Autonomous Systems with Deep-Learning Perception Components

Discrete-Event Controller Synthesis for Autonomous Systems with Deep-Learning Perception Components

Radu Calinescu*, Calum Imrie*, Ravi Mangal†, Corina Păsăreanu†,
Misael Alpizar Santana*, and Gricel Vázquez*

We present DEEPDECS, a new method for the synthesis of correct-by-construction discrete-event controllers for autonomous systems that use deep neural network (DNN) classifiers for the perception step of their decision-making processes. Despite major advances in deep learning in recent years, providing safety guarantees for these systems remains very challenging. Our controller synthesis method addresses this challenge by integrating DNN verification with the synthesis of verified Markov models. The synthesised models correspond to discrete-event controllers guaranteed to satisfy the safety, dependability and performance requirements of the autonomous system, and to be Pareto optimal with respect to a set of optimisation criteria. We use the method in simulation to synthesise controllers for mobile-robot collision avoidance, and for maintaining driver attentiveness in shared-control autonomous driving.

Autonomous systems perceive the environment they operate in, and adapt their behaviour in response to changes in it. In application domains as diverse as medicine,[1,2] finance[3] and transportation,[4,5] this perception is often performed using deep neural network (DNN) classifiers. The integration of deep-learning perception components into the control loop of autonomous systems used in such critical applications poses major challenges for their assurance.[6] In particular, the long-established methods for formal software verification[7] cannot be used to provide safety and performance guarantees for systems comprising both traditional software and deep-learning components. Newer verification methods developed specifically for DNNs focus on verifying robustness to changes in individual[8,9] or clusters[10] of DNN inputs. Therefore, they are equally unable to provide system-level guarantees for the controllers of autonomous systems with DNNs perception components.

Our paper presents DEEPDECS,[a] a controller synthesis method that addresses this significant limitation. DEEPDECS employs a suite of DNN verification methods to quantify the *aleatory uncertainty* that the use of DNN perception components introduces in the control loop of the autonomous system under development. Discrete-event controllers guaranteed to satisfy the requirements of the autonomous system are then synthesised from a stochastic model that takes this uncertainty into account *and* leverages the high accuracy that DNNs can achieve for their verified inputs.

We start by introducing our DEEPDECS controller synthesis method, and its unique approach to exploiting both DNN and traditional-software verification techniques. Next, we describe the use of DEEPDECS to devise controllers for mobile-robot collision avoidance, and for using a combination of optical, audio and haptic alerts to improve driver attentiveness in vehicles provided with Level 3 automated driving systems.[11] Finally, we discuss DEEPDECS in the context of related work, and we propose directions for future research.

*Department of Computer Science, University of York, York, UK.
†Carnegie Mellon University, Silicon Valley, USA
[a]Deep neural network perception Discrete-Event Controller Synthesis

1 DEEPDECS controller synthesis

Overview. DEEPDECS uses a *parametric discrete-time Markov chain* (pDTMC) to model the design space of the controller under development. The uncertainty due to the use of a deep-learning perception component within the system to be controlled and, if applicable, the uncertainty inherent to the system and its environment are modelled by the probabilities of transition between the states of this pDTMC. Finally, the controller synthesis problem involves finding combinations of parameter values for which the Markov chain satisfies strict safety, dependability and performance constraints, and is Pareto-optimal with respect to a set of optimisation objectives. These constraints and optimisation objectives are formalised as probabilistic temporal logic formulae.

DEEPDECS derives the pDTMC underpinning its controller synthesis automatically from (i) DNN verification results that quantify the uncertainty due to the deep-learning perception component, and (i) an "ideal" pDTMC that models the behaviour of the controlled system assuming perfect perception (Figure 1a). The set of correct-by-construction, Pareto-optimal DEEPDECS controllers is then synthesised by applying a combination of probabilistic model checking and search techniques to the derived pDTMC. As shown in Figure 1b, each of these controllers operates by reacting to changes in the system, in the DNN outputs and, unique to DEEPDECS, in the results obtained through the online verification of each DNN input and prediction.

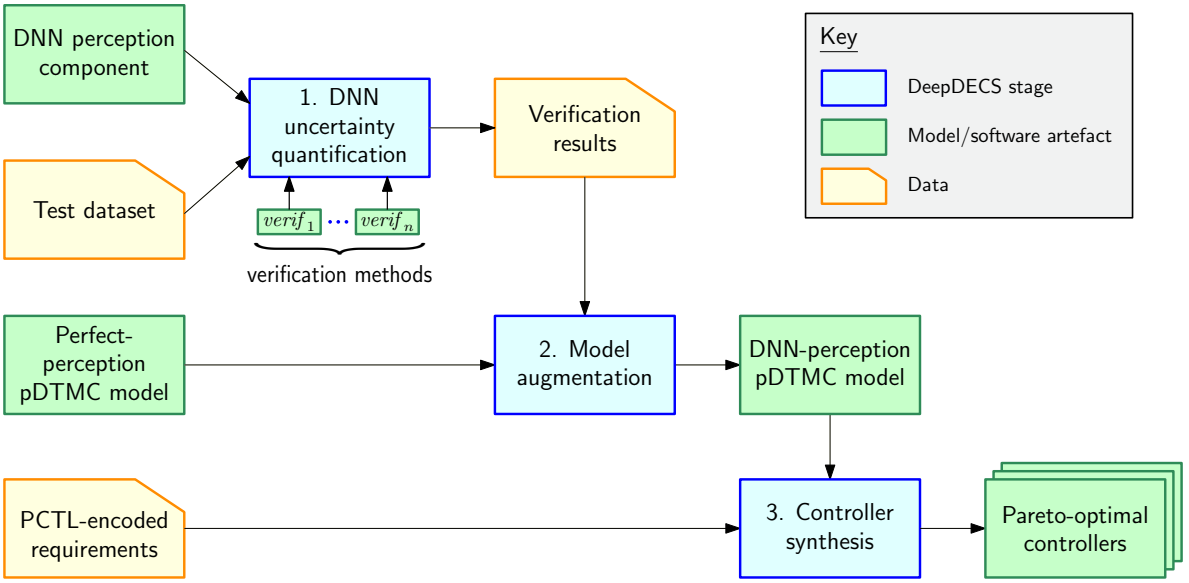
We detail each stage of the DEEPDECS approach below.

Stage 1: DNN uncertainty quantification. This section provides a brief introduction to DNN classifier verification, and describes the use of such verification techniques to quantify the aleatory uncertainty of DNN classifiers.

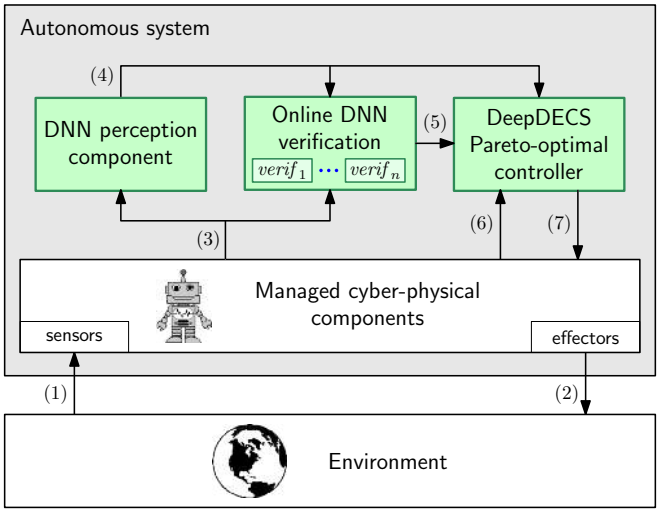
a) Verification of DNN classifiers. A K-class DNN classifier f_θ is a function, composed of linear and non-linear transformations, of the form

$$f_\theta : \mathbb{R}^d \to [K], \tag{1}$$

where $[K]$ denotes the set $\{1, \ldots, K\}$, and θ refers to the



(a) DEEPDECS generates discrete-event controllers aware of the uncertainty induced by the DNN perception component of an autonomous system in three stages. First, in a *DNN uncertainty quantification* stage, n verification methods are used to evaluate the DNN perception component over a test dataset representative for the operational design domain (ODD) of the autonomous system. The verification results provide a quantification of the DNN prediction uncertainty within the system ODD. Next, the *Model augmentation* stage uses these results—and an ideal-system pDTMC model that assumes perfect perception—to assemble a pDTMC system model that takes the DNN-induced uncertainty into account. Finally, the *Controller synthesis* stage uses this pDTMC model to synthesise a set of Pareto-optimal discrete-event controllers guaranteed to satisfy the PCTL-encoded requirements (constraints and optimisation objectives) of the system.



(b) The cyber-physical components of an autonomous system managed by a DEEPDECS controller monitor their environment through sensors (1) and perform actions that affect it through effectors (2). A DNN perception component uses a combination (3) of preprocessed sensor data and data about these managed components to classify the state of the environment (4). The n verification methods used for the DEEPDECS controller synthesis are also applied to the classification (4) and the DNN input (3) that produced it. Using the online DNN verification results (5) alongside the classification (4) and additional state information (6) obtained directly from the managed cyber-physical components, the DEEPDECS controller updates (7) the controllable parameters of these components in line with the system requirements.

Figure 1: DEEPDECS controller synthesis (a), and deployment (b)

weights or parameter values that characterize the linear transformations. As the results presented in this article are oblivious to the internal details of DNNs, we will by default omit the subscript θ, and treat f as a black-box function.

DNN classifiers are learnt from data, and are not guaranteed to always classify their input correctly. DNN verification techniques can help assess the quality of a classifier for a given input. A verification technique has the general form

$$verif : (\mathbb{R}^d \to [K]) \times \mathbb{R}^d \to \mathbb{B}, \tag{2}$$

such that, for a classifier $f \in \mathbb{R}^d \to [K]$ and an input $x \in \mathbb{R}^d$, $verif(f,x) = \text{true}$ if the verification technique deems the DNN f likely to classify the input x correctly, and $verif(f,x) = \text{false}$ otherwise. Two examples of simple DNN verification techniques

(which we use to evaluate DEEPDECS later in the article) are:

1. *Model confidence threshold*—A K-class DNN classifier is practically implemented as a function of type $\mathbb{R}^d \to [0,1]^K$, with each input $x \in \mathbb{R}^d$ first mapped to a discrete probability distribution $\delta(x) = (p_1, p_2, ..., p_K)$ over the K classes, and the class corresponding to the highest probability is chosen as the classifier prediction. The probability associated with a class can be interpreted as estimating the probability that the class is the *true* label of x. While it has been observed that classifiers may not be well-calibrated, i.e., the estimated correctness probabilities may be far from the true probabilities, a number of methods have been proposed to calibrate DNN classifiers.[12] Assuming that a classifier is well-calibrated using one of these methods, a simple DNN verification technique is to check whether the estimate correctness probability for an input x is greater than a fixed threshold τ for the class with the highest probability:

$$verif_1(f,x) = \begin{cases} \text{true}, & \text{if } \max_{i=1}^{K} p_i \geq \tau \\ \text{false}, & \text{otherwise} \end{cases} \tag{3}$$

2. *Local robustness certification*[13]—A DNN classifier f is ϵ-locally robust at an input x if perturbations within a small distance $\epsilon > 0$ from x (measured using the ℓ_2 metric) do not lead to a change in the classifier prediction. Accordingly, the local robustness verifier is defined by

$$verif_2(f,x) = \begin{cases} \text{true}, & \text{if } \forall x' \in \mathbb{R}^d.\ ||x - x'||_2 \leq \epsilon \\ & \qquad \implies f(x) = f(x') \\ \text{false}, & \text{otherwise} \end{cases} \tag{4}$$

for any input $x \in \mathbb{R}^d$.

b) Quantification of DNN perception uncertainty. The use of DNN perception introduces aleatory uncertainty into the autonomous system since DNNs are not guaranteed to predict accurately on all inputs. In the first DEEPDECS stage, we use a mechanism that relies on DNN verification techniques to empirically quantify the uncertainty of the DNN outcomes.

Let $X \subset \mathbb{R}^d$ be a *representative test dataset* for the DNN classifier (1), i.e., a set of classifier inputs that reflects the inputs that the autonomous system using the DNN will encounter in its ODD. For any test input $x \in X$, let $f^*(x) \in [K]$ be the label (i.e., the true class) of x, which is known since X is a test dataset.

DEEPDECS uses $n \geq 0$ DNN verification techniques $verif_1$, $verif_2$, ..., $verif_n$ to identify subsets of X for which the classifier is likely to achieve higher accuracy than for the entire set X.[b] We use the n verification methods to partition the test dataset X into 2^n subsets comprising inputs x with the same verification results.[c] Formally, for a DNN classifier f and any $v = (v_1, v_2, ..., v_n) \in \mathbb{B}^n$, we define the test data subset

$$X_v = \{x \in X \mid verif(f,x) = v\}, \tag{5}$$

where $verif(f,x) = (verif_1(f,x), verif_2(f,x), ..., verif_n(f,x))$. We use each of these test data subsets to define a $K \times K$ confusion matrix $\mathcal{C}_v$ such that, for any $k, k' \in [K]$, the element in row k and column k' of this matrix is given by the number of inputs from X_v with true class k that the DNN classifies as belonging to class k'

$$\mathcal{C}_v[k,k'] = \#\{x \in X_v \mid f^*(x) = k \wedge f(x) = k'\}, \tag{6}$$

where, for any set A, $\#A$ denotes its cardinality.

As the test dataset X is representative of the DNN inputs that the system encounters in operation, we henceforth assume that the probability that a class-k input x satisfies $verif(f,x) = v$ and is (mis)classified by the DNN as belonging to class k' is given by:[d]

$$p_{kk'v} = Pr(f(x) = k' \wedge verif(f,x) = v \mid f^*(x) = k) = \frac{C_v[k,k']}{\sum_{v' \in \mathbb{B}^n} \sum_{k'' \in [K]} C_{v'}[k,k'']}. \tag{7}$$

Stage 2: Model augmentation. This section provides a brief introduction to pDTMCs, defines the discrete-event controller synthesis problem, and presents the DEEPDECS theory underlying the generation of pDTMCs that model the behaviour of, and support the synthesis of controllers for, autonomous systems with deep-learning perception components.

a) Discrete-time Markov chains. DEEPDECS models the design space (i.e., the possible variants) for the controller of an autonomous system as a pDTMC augmented with *rewards*.

Definition 1. *A reward-augmented discrete-time Markov chain (DTMC) over a set of atomic propositions AP is a tuple*

$$\mathcal{M} = (S, s_0, P, L, R), \tag{8}$$

where $S \neq \emptyset$ is a finite set of states; $s_0 \in S$ is the initial state; $P : S \times S \to [0,1]$ is a transition probability function such that, for any states $s, s' \in S$, $P(s,s')$ gives the probability of transition from state s to state s' and $\sum_{s' \in S} P(s,s') = 1$; $L : S \to 2^{AP}$ is a labelling function that maps every state $s \in S$ to the atomic propositions from AP that hold in that state; and R is a set of reward structures*, i.e., function pairs (ρ, ι) that associate non-negative values with the pDTMC states and transitions:*

$$\rho : S \to \mathbb{R}_{\geq 0},\ \iota : S \times S \to \mathbb{R}_{\geq 0}. \tag{9}$$

When (8) includes unknown transition probabilities and/or reward values, the DTMC is termed *parametric*.

Definition 2. *A reward-augmented parametric discrete-time Markov chain is a DTMC* (8) *comprising one or several transition probabilities and/or rewards that are specified as rational functions*[e] *over a set of continuous variables.*[14]

DEEPDECS uses *probabilistic computation tree logic* (PCTL)[15,16] extended with rewards[17] to quantify the safety, dependability and performance properties of an autonomous system whose controller design space is modelled as a pDTMC.

[b] Note that DEEPDECS is also applicable in the special case when $n = 0$, i.e., when no verification techniques is used.

[c] As typical values for n are $n = 1,2,3$, there will only be a small number of such subsets.

[d] Formally, this results holds as $\#X \to \infty$.

[e] i.e., functions that can be written as fractions whose numerators and denominators are polynomial functions, e.g., $1 - p$ or $\frac{1-p_1}{p_2}$

Definition 3. State PCTL formulae Φ *and* path PCTL formulae Ψ *over an atomic proposition set* AP*, and* PCTL reward formulae Φ_R *over a reward structure* (9) *are defined by the grammar:*

$$\begin{aligned}\Phi &::= true \,|\, \alpha \,|\, \Phi \wedge \Phi \,|\, \neg\Phi \,|\, \mathcal{P}_{\sim p}[\Psi] \\ \Psi &::= X\Phi \,|\, \Phi \,\mathrm{U}\, \Phi \,|\, \Phi \,\mathrm{U}^{\leq k}\, \Phi \\ \Phi_R &::= \mathcal{R}_{\sim r}[\mathrm{C}^{\leq k}] \,|\, \mathcal{R}_{\sim r}[\mathrm{F}\,\Phi]\end{aligned} \tag{10}$$

where $\alpha \in AP$ *is an atomic proposition,* $\sim \in \{\geq, >, <, \leq\}$ *is a relational operator,* $p \in [0,1]$ *is a probability bound,* $r \in \mathbb{R}_0^+$ *is a reward bound, and* $k \in \mathbb{N}_{>0}$ *is a timestep bound.*

The PCTL semantics[15–17] is defined using a satisfaction relation $\models$ over the states of a DTMC. Given a state s of a DTMC $\mathcal{M}$, $s \models \Phi$ means 'Φ holds in state s', and we have: always $s \models true$; $s \models \alpha$ iff $\alpha \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; and $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. The *time-bounded until formula* $\Phi_1 \mathrm{U}^{\leq k} \Phi_2$ holds for a *path* (i.e., sequence of DTMC states $s_0 s_1 s_2 ...$ such that $P(s_i, s_{i+1}) > 0$ for all $i > 0$) iff Φ_1 holds in the first $i < k$ path states and Φ_2 holds in the $(i+1)$-th path state; and the *unbounded until formula* $\Phi_1 \mathrm{U} \Phi_2$ removes the bound k from the time-bounded until formula. The *next formula* $X\Phi$ holds if Φ is satisfied in the next state. The semantics of the probability $\mathcal{P}$ and reward $\mathcal{R}$ operators are defined as follows: $\mathcal{P}_{\sim p}[\Psi]$ specifies that the probability that paths starting at a chosen state s satisfy a path property Ψ is $\sim p$; $\mathcal{R}_{\sim r}[C^{\leq k}]$ holds if the expected cumulated reward up to time-step k is $\sim r$; and $\mathcal{R}_{\sim r}[F\Phi]$ holds if the expected reward cumulated before reaching a state satisfying Φ is $\sim r$. Replacing $\sim p$ (or $\sim r$) from (10) with '$=?$' specifies that the calculation of the probability (or reward) is required. We use the shorthand notation $pmc(\Phi, \mathcal{M})$ and $pmc(\Phi_R, \mathcal{M})$ for these quantities computed for the initial state s_0 of $\mathcal{M}$.

b) Discrete-event controller synthesis problem. To distinguish between different concerns of the autonomous system to be controlled, DEEPDECS organises each state s of the perfect-perception pDTMC model from Figure 1 into a tuple

$$s = (z, k, t, c), \tag{11}$$

where $z \in Z$ models the (internal) state of the system, $k \in [K]$ models the state of the environment, $c \in C$ models the control parameters of the system, and $t \in [3]$ is a "turn" flag. This flag indicates which elements of (11) can change in each pDTMC state:

$$\begin{aligned}&\forall s = (z,k,t,c), s' = (z',k',t',c') \in S: \\ &\quad ((t = 1 \wedge P(s,s') > 0) \implies k' = k \wedge c' = c \wedge t' < 3) \wedge \\ &\quad ((t = 2 \wedge P(s,s') > 0) \implies z' = z \wedge c' = c \wedge t' = 3) \wedge \\ &\quad ((t = 3 \wedge P(s,s') > 0) \implies z' = z \wedge k' = k \wedge t' = 1).\end{aligned} \tag{12}$$

We partition the pDTMC state set into states in which the system can change, states in which the environment can change, and states in which it is the controller's "turn" to act for simplicity, and without loss of generality; the three types of states can be easily collapsed into one.

Finally, we assume that the outgoing transition probabilities from states $(z,k,3,c) \in S$ are controller parameters that need to be determined and are given by

$$x_{zkcc'} = P((z,k,3,c),(z,k,1,c')) \tag{13}$$

for all $c' \in C$, where $x_{zkcc'} \in \{0,1\}$ for *deterministic controllers* or $x_{zkcc'} \in [0,1]$ for *probabilistic controllers*, and $\sum_{c' \in C} x_{zkcc'} = 1$.

Figure 2a shows the general format of a DEEPDECS perfect-perception pDTMC model, defined in the high-level modelling language of the PRISM model checker.[18] In this language, the model of a system is specified by the parallel composition of a set of *modules*. The state of a *module* is given by a set of finite-range local variables, and its state transitions are specified by probabilistic guarded commands that change these variables:

$$[action]\ guard \rightarrow e_1 : update_1 + e_2 : update_2 + ... + e_m : update_N; \tag{14}$$

In this command, $guard$ is a boolean expression over the variables of all modules. If $guard$ evaluates to true, the arithmetic expression e_i, $i \in [m]$, gives the probability with which the $update_i$ change of the module variables occurs. When $action$ is present, all modules comprising commands with this $action$ have to *synchronise*, i.e., to perform one of these commands simultaneously.

With this notation introduced so far, the *controller synthesis problem for the perfect-perception system* is to find the set of Pareto-optimal parameters (13) which ensure that the pDTMC satisfies $n_1 \geq 0$ PCTL-encoded *constraints* of the form in (10),

$$C_i ::= \Phi_i \mid \Phi_{Ri} \tag{15}$$

and Pareto-optimises $n_2 \geq 1$ PCTL-encoded *objectives* of the form

$$\begin{aligned}O_j ::= &\text{maximise } pmc(\Phi_j, \mathcal{M}) \mid \text{minimise } pmc(\Phi_j, \mathcal{M}) \\ &\text{maximise } pmc(\Phi_{Rj}), \mathcal{M} \mid \text{minimise } pmc(\Phi_{Rj}, \mathcal{M})\end{aligned} \tag{16}$$

where $i \in [n_1]$ and $j \in [n_2]$.

c) Model augmentation. The controller of an autonomous system with deep-learning perception does not have access to the true value k of the environment state from (11). Instead, DEEPDECS controllers need to operate with an estimate $\hat{k} \in [K]$ of this true value, and with the results $v = (v_1, v_2, ..., v_n) \in \mathbb{B}^n$ of $n \geq 0$ verification techniques (2) applied to the DNN and its input that produced the estimate $\hat{k}$. As such, the states $\hat{s}$ of a DEEPDECS *DNN-perception pDTMC model*

$$\hat{\mathcal{M}} = (\hat{S}, \hat{s_0}, \hat{P}, \hat{L}, \hat{R}) \tag{17}$$

are tuples that extend (11) with $\hat{k}$ and v:

$$\hat{s} = (z, k, \hat{k}, v, t, c). \tag{18}$$

The derivation of the DEEPDECS DNN-perception pDTMC from the perfect-perception pDTMC is shown in Figure 2b. To provide a formal definition of this derivation, we use the notation $s(\hat{s}) = (z,k,t,c)$ to refer to the element from $Z \times [K] \times [3] \times C$ that corresponds to a generic element corresponding to $\hat{s} \in Z \times [K]^2 \times \mathbb{B}^n \times [3] \times C$. With this notation, the components of the pDTMC $\hat{\mathcal{M}}$ are obtained from the perfect-perception pDTMC $\mathcal{M} = (S, s_0, P, L, R)$ of the same autonomous system and the probabilities (7) as follows:

$$\hat{S} = \{\hat{s} \in Z \times [K]^2 \times \mathbb{B}^n \times [3] \times C \mid s(\hat{s}) \in S\}; \tag{19}$$

dtmc

module ManagedComponents
z : Z **init** z_0;

[action$_1$] t=1 $\wedge$ $guard_1^{\mathsf{Z}}$(z,c) $\rightarrow$ e_{11}^{Z}:(z'=z_{11})+...+$e_{1N_1}^{\mathsf{Z}}$:(z'=z_{1N_1});
[action$_2$] t=1 $\wedge$ $guard_2^{\mathsf{Z}}$(z,c) $\rightarrow$ e_{21}^{Z}:(z'=z_{21})+...+$e_{2N_2}^{\mathsf{Z}}$:(z'=z_{2N_2});
...
endmodule

module Environment
k : [K] **init** k_0;

[monitor] t=2 $\wedge$ $guard_1^{\mathsf{K}}$(z,k) $\rightarrow$ e_{11}^{K}:(k'=1)+...+e_{1K}^{K}:(k'=K);
[monitor] t=2 $\wedge$ $guard_2^{\mathsf{K}}$(z,k) $\rightarrow$ e_{21}^{K}:(k'=1)+...+e_{2K}^{K}:(k'=K);
...
endmodule

module PerfectPerceptionController
c : C **init** c_0;

[decide] t=3 $\wedge$ $guard_1^{\mathsf{C}}$(z,k,c) $\rightarrow$ $\sum_{c' \in C}\left(x_{\mathrm{zkc}c'}:(\mathrm{c}'=c')\right)$;
[decide] t=3 $\wedge$ $guard_2^{\mathsf{C}}$(z,k,c) $\rightarrow$ $\sum_{c' \in C}\left(x_{\mathrm{zkc}c'}:(\mathrm{c}'=c')\right)$;
...
endmodule

module Turn
t : [1..3] **init** 1;

[action$_\alpha$] **true** $\rightarrow$ 1:(t'=2);
[action$_\beta$] **true** $\rightarrow$ 1:(t'=2);
...
[monitor] **true** $\rightarrow$ 1:(t'=3);
[decide] **true** $\rightarrow$ 1:(t'=1);
endmodule

(a) A DEEPDECS perfect-perception pDTMC comprises four modules. The module ManagedComponents models the controlled components of the system by specifying how their state z changes as a result of set of actions $Act=\{\text{action}_1,\text{action}_2,...\}$ performed when the value of the turn flag is $t=1$; the guards of this module depend only on the state z of the components and the control parameters c. The module Environment models how the evolving state of environment k changes when observed through monitoring when the value of the turn flag is $t=2$; the guards of this module depend only on the state z of the system components and on the current state k of the environment. The controller decisions are defined by the PerfectPerceptionController module with parameters (13). Finally, the module Turn sets the turn flag to $t=2$ when, after specific actions $\text{action}_\alpha,\text{action}_\beta,... \in Act$, it is the monitor's turn to observe the environment, sets the turn flag to $t=3$ after each such monitor action to trigger a controller decision, and restores the turn flag to $t=1$ immediately after that to enable the controlled system components to react to the new control parameters of the system.

dtmc

module ManagedComponents
z : Z **init** z_0;

[action$_1$] t=1 $\wedge$ $guard_1^{\mathsf{Z}}$(z,c) $\rightarrow$ e_{11}^{Z}:(z'=z_{11})+...+$e_{1N_1}^{\mathsf{Z}}$:(z'=z_{1N_1});
[action$_2$] t=1 $\wedge$ $guard_2^{\mathsf{Z}}$(z,c) $\rightarrow$ e_{21}^{Z}:(z'=z_{21})+...+$e_{2N_2}^{\mathsf{Z}}$:(z'=z_{2N_2});
...
endmodule

module EnvironmentWithDNNPerception
k : [K] **init** k_0;
$\hat{\mathrm{k}}$: [K] **init** k_0;
v : $\mathbb{B}^n$ **init** (**true**,**true**,...,**true**);

[monitor] t=2 $\wedge$ $guard_1^{\mathsf{K}}$(z,k) $\rightarrow$ $\sum_{\hat{k}' \in [K]} \sum_{v' \in \mathbb{B}^n} \left(e_{11}^{\mathsf{K}} p_{1\hat{k}'v'}:(\mathrm{k}'=1 \ \&\ \hat{\mathrm{k}}'=\hat{k}' \ \&\ \mathrm{v}=v')\right)$
$+\ldots+\sum_{\hat{k}' \in [K]} \sum_{v' \in \mathbb{B}^n} \left(e_{1K}^{\mathsf{K}} p_{K\hat{k}'v'}:(\mathrm{k}'=K \ \&\ \hat{\mathrm{k}}'=\hat{k}' \ \&\ \mathrm{v}=v')\right)$;
[monitor] t=2 $\wedge$ $guard_2^{\mathsf{K}}$(z,k) $\rightarrow$ $\sum_{\hat{k}' \in [K]} \sum_{v' \in \mathbb{B}^n} \left(e_{21}^{\mathsf{K}} p_{1\hat{k}'v'}:(\mathrm{k}'=1 \ \&\ \hat{\mathrm{k}}'=\hat{k}' \ \&\ \mathrm{v}=v')\right)$
$+\ldots+\sum_{\hat{k}' \in [K]} \sum_{v' \in \mathbb{B}^n} \left(e_{2K}^{\mathsf{K}} p_{K\hat{k}'v'}:(\mathrm{k}'=K \ \&\ \hat{\mathrm{k}}'=\hat{k}' \ \&\ \mathrm{v}=v')\right)$;
...
endmodule

module DNNPerceptionController
c : C **init** c_0;

[decide] t=3 $\wedge$ $guard_1^{\mathsf{C}}$(z,$\hat{\mathrm{k}}$,c) $\wedge$ v=(**false**,**false**,...,**false**) $\rightarrow$ $\sum_{c' \in C}\left(x_{\mathrm{z}\hat{\mathrm{k}}\mathrm{v}\mathrm{c}c'}:(\mathrm{c}'=c')\right)$;
...
[decide] t=3 $\wedge$ $guard_1^{\mathsf{C}}$(z,$\hat{\mathrm{k}}$,c) $\wedge$ v=(**true**,**true**,...,**true**) $\rightarrow$ $\sum_{c' \in C}\left(x_{\mathrm{z}\hat{\mathrm{k}}\mathrm{v}\mathrm{c}c'}:(\mathrm{c}'=c')\right)$;
[decide] t=3 $\wedge$ $guard_2^{\mathsf{C}}$(z,$\hat{\mathrm{k}}$,c) $\wedge$ v=(**false**,**false**,...,**false**) $\rightarrow$ $\sum_{c' \in C}\left(x_{\mathrm{z}\hat{\mathrm{k}}\mathrm{v}\mathrm{c}c'}:(\mathrm{c}'=c')\right)$;
...
[decide] t=3 $\wedge$ $guard_2^{\mathsf{C}}$(z,$\hat{\mathrm{k}}$,c) $\wedge$ v=(**true**,**true**,...,**true**) $\rightarrow$ $\sum_{c' \in C}\left(x_{\mathrm{z}\hat{\mathrm{k}}\mathrm{v}\mathrm{c}c'}:(\mathrm{c}'=c')\right)$;
...
endmodule

module Turn
t : [1..3] **init** 1;

[action$_\alpha$] **true** $\rightarrow$ 1:(t'=2);
[action$_\beta$] **true** $\rightarrow$ 1:(t'=2);
...
[monitor] **true** $\rightarrow$ 1:(t'=3);
[decide] **true** $\rightarrow$ 1:(t'=1);
endmodule

(b) DEEPDECS DNN-perception pDTMC model obtained by performing the highlighted modification in the perfect-perception pDTMC from Figure 2a. As a consequence of using a DNN to perceive the true environment state k, the DNNPerceptionController does not have access to its value; instead, it needs to rely on its classification $\hat{k}$ and on the verification result v, respectively. The EnvironmentWithDNNPerception module continues to track the ground truth k (necessary to establish the true safety, energy consumption, and other key properties of the system). Additionally, this module uses the DNN uncertainty quantification probabilities (7) to model the evolution of the DNN output $\hat{k}$ and the online DNN verification result v associated with this output. The modules ManagedComponents and Turn are unchanged.

Figure 2: Perfect-perception and DNN-perception DEEPDECS pDTMC models

$$\hat{s}_0=(z_0,k_0,k_0,\text{true},...,\text{true},t_0,c_0), \tag{20}$$

where $(z_0,k_0,t_0,c_0)=s_0$; and, for any states $\hat{s}=(z,k,\hat{k},v,t,c)$, $\hat{s}'=(z',k',\hat{k}',v',t',c') \in \hat{S}$,

$$\hat{P}(\hat{s},\hat{s}')=\begin{cases} P(s(\hat{s}),s(\hat{s}')), & \text{if } t=1 \wedge (\hat{k}',v')=(\hat{k},v) \\ P(s(\hat{s}),s(\hat{s}'))\cdot p_{k'\hat{k}'v'}, & \text{if } t=2 \\ x_{z\hat{k}vcc'}, & \text{if } t=3 \wedge (z',k',\hat{k}',v',t') \\ & \quad =(z,k,\hat{k},v,1) \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

where $x_{z\hat{k}vcc'}$ are controller parameters associated with state pairs $((z,k,\hat{k},v,3,c),(z,k,\hat{k},v,3,c')) \in \hat{S}^2$ such that $x_{z\hat{k}vcc'} \in \{0,1\}$ for deterministic controllers or $x_{z\hat{k}vcc'} \in [0,1]$ for probabilistic controllers, and $\sum_{c' \in C} x_{z\hat{k}vcc'}=1$. Finally, for any state $\hat{s} \in \hat{S}$,

$$\hat{L}(\hat{s})=L(s(\hat{s})), \tag{22}$$

and

$$\hat{R}=\{(\hat{\rho},\hat{\iota}) \in (\hat{S} \rightarrow \mathbb{R}_{\geq 0}) \times (\hat{S} \times \hat{S} \rightarrow \mathbb{R}_{\geq 0}) \mid \exists (\rho,\iota) \in R: \left(\forall \hat{s} \in \hat{S}: \hat{\rho}(\hat{s})=\rho(s(\hat{s}))\right) \wedge \left(\forall \hat{s},\hat{s}' \in \hat{S}: \hat{\iota}(\hat{s},\hat{s}')=\iota(s(\hat{s}),s(\hat{s}'))\right)\} \tag{23}$$

The following result shows that the DEEPDECS module augmentation produces a valid pDTMC.

Theorem 1. *The tuple* (17) *with the elements defined by* (19)–(23) *is a valid pDTMC that satisfies the following variant of* (12)*:*

$$\begin{aligned}&\forall \hat{s}=(z,k,\hat{k},v,t,c),\hat{s}'=(z',k',\hat{k}',v',t',c')\in\hat{S}\colon\\&\quad\big((t=1\wedge P(\hat{s},\hat{s}')>0)\Longrightarrow(k',\hat{k}',v',c')=(k,\hat{k},v,c)\wedge t'<3\big)\wedge\\&\quad\big((t=2\wedge P(\hat{s},\hat{s}')>0)\Longrightarrow(z',c')=(z,c)\wedge t'=3\big)\wedge\\&\quad\big((t=3\wedge P(\hat{s},\hat{s}')>0)\Longrightarrow(z',k',\hat{k}',v')=(z,k,\hat{k},v)\wedge t'=1\big).\end{aligned}\tag{24}$$

Proof. To demonstrate that (17) is a valid pDTMC, we need to show that, for any state $\hat{s}=(z,k,\hat{k},v,t,c)\in\hat{S}$, $\sum_{\hat{s}'\in\hat{S}}\hat{P}(\hat{s},\hat{s}')=1$. We prove this and property (24) for each value of $t\in[3]$.

For $t=1$, (21) implies that $\hat{P}(\hat{s},\hat{s}')>0$ only for states $\hat{s}'=(z',k',\hat{k},v,t',c')\in\hat{S}$, so

$$\begin{aligned}\sum_{\hat{s}'\in\hat{S}}\hat{P}(\hat{s},\hat{s}')&=\sum_{(z',k',\hat{k},v,t',c')\in\hat{S}}\hat{P}(\hat{s},\hat{s}')\\&=\sum_{(z',k',\hat{k},v,t',c')\in\hat{S}}P((z,k,1,c),(z',k',t',c'))\\&=\sum_{(z',k',t',c')\in S}P((z,k,1,c),(z',k',t',c'))=1,\end{aligned}$$

as the last sum adds up all outgoing transition probabilities of state (z',k',t',c') from the perfect-perception pDTMC $\mathcal{M}$. Consider now any $\hat{s}'=(z',k',\hat{k}',v',t',c')\in\hat{S}$ such that $\hat{P}(\hat{s},\hat{s}')>0$. We already noted that this requires $\hat{k}'=\hat{k}\wedge v'=v$. Additionally, since $\hat{P}(\hat{s},\hat{s}')=P((z,k,1,c),(z',k',t',c'))$, (12) implies that $k'=k\wedge c'=c\wedge t'<3$, as required by (24).

For $t=2$, we have

$$\begin{aligned}\sum_{\hat{s}'\in\hat{S}}\hat{P}(\hat{s},\hat{s}')&=\sum_{(z',k',\hat{k}',v',t',c')\in\hat{S}}\Big(\hat{P}(\hat{s},\hat{s}')p_{k'\hat{k}'v'}\Big)\\&=\sum_{(z',k',t',c')\in S}\Big(P((z,k,2,c),(z',k',t',c'))\cdot\sum_{(\hat{k}',v')'\in[K]\times\mathbb{B}^n}p_{k'\hat{k}'v'}\Big)\\&=\sum_{(z',k',t',c')\in S}(P((z,k,2,c),(z',k',t',c'))\cdot1)=1.\end{aligned}$$

Consider again a generic $\hat{s}'=(z',k',\hat{k}',v',t',c')\in\hat{S}$ such that $\hat{P}(\hat{s},\hat{s}')>0$. Since $\hat{P}(\hat{s},\hat{s}')=P((z,k,2,c),(z',k',t',c'))\cdot p_{k'\hat{k}'v'}$, (12) implies that $(z',c')=(z,c)\wedge t'=3$.

Finally, for $t=3$, we have $\sum_{\hat{s}'\in\hat{S}}\hat{P}(\hat{s},\hat{s}')=\sum_{c'\in C}x_{z\hat{k}vcc'}=1$ and the property (24) is explicitly stated in (21). □

Importantly, the next result shows that the controller decisions do not depend on the true state k of the environment.

Theorem 2. *For any* $(z,k_1,\hat{k},v,3,c),(z,k_2,\hat{k},v,3,c)\in\hat{S}$ *and any control parameters* $c'\in C$*,*

$$\begin{aligned}&\hat{P}((z,k_1,\hat{k},v,3,c),(z,k_1,\hat{k},v,1,c'))\\&\qquad=\hat{P}((z,k_2,\hat{k},v,3,c),(z,k_2,\hat{k},v,1,c')).\end{aligned}\tag{25}$$

Proof. According to definition (21), both transition probabilities from (25) are equal to $x_{z\hat{k}vcc'}$. □

Finally, the following theorem and its corollaries prove that for each (probabilistic) discrete-event controller that satisfies constraints (15) and Pareto-optimises objectives (16) for the autonomous system with DNN perception there is an equivalent (probabilistic) discrete-event controller for the autonomous system with perfect perception, but the converse does not hold.

Theorem 3. *Let* $\underline{x}$ *and* $\underline{\hat{x}}$ *be valid instantiations of the perfect-perception controller parameters* $\big\{x_{zkcc'}\in[0,1]\;\big|\;(\exists k\in[K].(z,k,3,c)\in S)\;\wedge\;c'\in C\big\}$ *from* (13) *and of the DNN-perception controller parameters* $\big\{x_{z\hat{k}vcc'}\in[0,1]\,\big|\,(\exists k\in[K].(z,k,\hat{k},v,3,c)\in\hat{S})\wedge c'\in C\big\}$ *from* (21)*, respectively. Also, let* $\mathcal{M}_{\underline{x}}$ *and* $\hat{\mathcal{M}}_{\underline{\hat{x}}}$ *be the instances of the perfect-perception pDTMC* $\mathcal{M}$ *and DNN-perception pDTMC* $\hat{\mathcal{M}}$ *corresponding to the controller parameters* $\underline{x}$ *and* $\underline{\hat{x}}$*, respectively. With this notation, we have*

$$pmc(\Phi,\hat{\mathcal{M}}_{\underline{\hat{x}}})=pmc(\Phi,\mathcal{M}_{\underline{x}}),\tag{26}$$

and

$$pmc(\Phi_R,\hat{\mathcal{M}}_{\underline{\hat{x}}})=pmc(\Phi_R,\mathcal{M}_{\underline{x}}),\tag{27}$$

for any (quantitative) PCTL state formula Φ *and reward state formula* Φ_R *if and only if*

$$x_{zkcc'}=\sum_{\hat{k}\in[K]}\sum_{v\in\mathbb{B}^n}p_{k\hat{k}v}x_{z\hat{k}vcc'}\tag{28}$$

for all $(z,k,3,c)\in S$ *and* $c'\in C$*.*

Proof. Let $Paths^{\mathcal{M}_{\underline{x}}}(s_0)$ and $Paths^{\hat{\mathcal{M}}_{\underline{\hat{x}}}}(\hat{s}_0)$ be the set of all $\mathcal{M}_{\underline{x}}$ paths starting at s_0 and the set of all $\hat{\mathcal{M}}_{\underline{\hat{x}}}$ paths starting at $\hat{s}_0$, respectively. Equalities (26) and (27) hold iff, for any path $\pi=s_0s_1s_2...\in Paths^{\mathcal{M}_{\underline{x}}}(s_0)$, set of associated paths $\hat{\Pi}=\big\{\hat{s}_0\hat{s}_1\hat{s}_2...\in Paths^{\hat{\mathcal{M}}_{\underline{\hat{x}}}}(\hat{s}_0)\,|\,\forall i\geq0\,.\,s(\hat{s}_i)=s_i\big\}$, and $i\geq0$, the following property holds:

$$P(s_i,s_{i+1})=\sum_{\hat{s}_0\hat{s}_1\hat{s}_2...\in\hat{\Pi}}\hat{P}(\hat{s}_i,\hat{s}_{i+1}).\tag{29}$$

This is required because, according to (22) and (23), the $(i+1)$-th state of π and of any path $\hat{\pi}\in\hat{\Pi}$ are labelled with the same atomic propositions and assigned the same state rewards, respectively; and, according to (23), the transition rewards for the transition between their i-th state and $(i+1)$-th state are also identical. Thus, if this equality holds, the path π and path set $\hat{\Pi}$ are indistinguishable in the evaluation of PCTL state and state reward formulae; and, if the equality does not hold, a labelling function L and a PCTL state formula Φ (or state reward formula Φ_R) can be handcrafted to provide a counterexample for (26) (or for (27)).

Given the definition of $\hat{P}$ from (21), property (29) holds trivially for any state $s_i=(z,k,t,c)\in S$ with $t=1$, and also holds for states s_i with $t=2$ because

$$\begin{aligned}\sum_{\hat{s}_0\hat{s}_1\hat{s}_2...\in\hat{\Pi}}\hat{P}(\hat{s}_i,\hat{s}_{i+1})&=\sum_{\hat{s}_0\hat{s}_1\hat{s}_2...\in\hat{\Pi}}(P(s_i,s_{i+1})\cdot p_{k\hat{k}_{i+1}v_{i+1}})\\&=P(s_i,s_{i+1})\cdot\sum_{\hat{s}_0\hat{s}_1\hat{s}_2...\in\hat{\Pi}}p_{k\hat{k}_{i+1}v_{i+1}}=P(s_i,s_{i+1}),\end{aligned}$$

where $\hat{k}_{i+1}$ and v_{i+1} represent the DNN prediction and verification result for each state $\hat{s}_{i+1}$ from the sum, respectively. Finally, for $t = 3$, property (29) holds if and only if the perfect-perception and DNN-perception controllers select each next controller configuration $c' \in C$ with the same probability for s_i and for all the states $\hat{s}_i$ from $\hat{\Pi}$ taken together, i.e., if and only if (28) holds, which completes the proof. □

Properties (26) and (27) imply that any constraint (15) is either satisfied or violated by both $\mathcal{M}_{\underline{x}}$ and $\hat{\mathcal{M}}_{\underline{\hat{x}}}$ (since the two DTMCs yield the same value for the system property associated with the constraint). Likewise, $\mathcal{M}_{\underline{x}}$ and $\hat{\mathcal{M}}_{\underline{\hat{x}}}$ are guaranteed to achieve the same value for the system property associated with any optimisation objective (16).

Corollary 1. *For any combination of constraints* (15) *and optimisation objectives* (16) *for which there exists a probabilistic DNN-perception controller that satisfies the constraints, there exists also a probabilistic perfect-perception controller that satisfies the same constraints and yields the same values for the PCTL properties from the optimisation objectives.*

Proof. We prove this result by showing that the application of (28) to any valid instantiation of the DNN-perception controller parameters $x_{z\hat{k}vcc'}$ produces a valid instantiation of the perfect-perception controller parameters $x_{zkcc'}$. First, since $x_{z\hat{k}vcc'} \in [0,1]$ for any valid $(z,\hat{k},v,c,c')$ tuple, we have

$$0 = \sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^n} (p_{k\hat{k}v} \cdot 0) \leq \sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^n} p_{k\hat{k}v} x_{z\hat{k}vcc'} \leq \sum_{v\in\mathbb{B}^n} (p_{k\hat{k}v} \cdot 1) = 1,$$

so $x_{zkcc'} \in [0,1]$ for any valid tuple (z,k,c,c'). Additionally, for any valid combination of z, k and c, we have

$$\sum_{c'\in C} x_{zkcc'} = \sum_{c'\in C} \sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^n} p_{k\hat{k}v} x_{z\hat{k}vcc'}$$

$$= \sum_{c'\in C} \left(x_{z\hat{k}vcc'} \cdot \left(\sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^n} p_{k\hat{k}v} \right) \right) = \sum_{c'\in C} \left(x_{z\hat{k}vcc'} \cdot 1 \right) = 1,$$

which completes the proof. □

Corollary 2. *There exist an infinite number of combinations of constraints* (15) *and optimisation objectives* (16) *for which there exists a probabilistic perfect-perception controller that satisfies the constraints, and no DNN-perception controller exists that satisfies the constraints and yields the same values for the system properties from the optimisation objectives.*

Proof. We prove this result by showing that, for an infinite number of instantiations $\underline{x}$ of the perfect-perception controller parameters $x_{zkcc'}$, no valid instantiation $\underline{\hat{x}}$ of the DNN-perception controller parameters $x_{z\hat{k}vcc'}$ satisfies (28). Let $(k_0,\hat{k}_0,v_0) \in [K]^2 \times \mathbb{B}^n$ such that $p_{k_0\hat{k}_0v_0} \in (0,1)$. Such combinations of true class k, DNN-predicted class $\hat{k}$ and verification results v exist, as otherwise all $p_{k\hat{k}v} \in \{0,1\}$, which would require the DNN to be perfectly accurate or to only err by always swapping class labels in the same way, and this is not possible. Consider a state $\hat{s} = (z,k,\hat{k}_0,v_0,3,c) \in \hat{S}$, any $c' \in C$, and any of the infinite number of valid instantiations $\underline{x}$ of the perfect-controller parameters such that $x_{zk_0cc'} = \alpha_{zcc'} p_{k_0\hat{k}_0v_0} + \sum_{(\hat{k},v)\in[K]\times\mathbb{B}^n\setminus\{(\hat{k}_0,v_0)\}} p_{k_0\hat{k}_0v_0}$ with $\alpha_{zcc'} \in [0,1]$. We use (28) to calculate $x_{z\hat{k}_0v_0cc'}$:

$$\begin{aligned} x_{z\hat{k}_0v_0cc'} &= \frac{x_{zk_0cc'} - \sum_{(\hat{k},v)\in[K]\times\mathbb{B}^n\setminus\{(\hat{k}_0,v_0)\}} p_{k\hat{k}v} x_{z\hat{k}vcc'}}{p_{k_0\hat{k}_0v_0}} \\ &\geq \frac{x_{zk_0cc'} - \sum_{(\hat{k},v)\in[K]\times\mathbb{B}^n\setminus\{(\hat{k}_0,v_0)\}} p_{k\hat{k}v}}{p_{k_0\hat{k}_0v_0}} \\ &= \frac{\alpha_{zcc'} p_{k_0\hat{k}_0v_0}}{p_{k_0\hat{k}_0v_0}} = \alpha_{zcc'}. \end{aligned}$$

Accordingly, the outgoing transition probabilities from the considered state $\hat{s}$ add up to

$$\sum_{c'\in C} x_{z\hat{k}_0v_0cc'} \geq \sum_{c'\in C} \alpha_{zcc'}. \tag{30}$$

As the right-hand side of this inequality can take any value in the interval $[0,\#C]$, and the only valid value for $\sum_{c'\in C} x_{z\hat{k}_0v_0cc'}$ is 1, we identified an infinite number of instantiations $\underline{x}$ for which a valid instantiation $\underline{\hat{x}}$ cannot be built. □

Corollary 2 shows that the decision-making capabilities of infinitely many perfect-perception controllers cannot be replicated by DNN-perception controllers. While this does not indicate how many of these practically unachievable controllers satisfy constraints (15) and Pareto-optimise objectives (16), the proof of the corollary provides a hint about this by showing in (30) that DNN-perception controllers do not exist for large $\alpha_{zcc'}$ values, i.e., for scenarios when the perfect-perception controller decides to use a specific configuration c' with high probability. Intuitively, these scenarios are highly relevant, i.e., many perfect-perception controllers with no equivalent DNN-perception controllers are likely to be Pareto-optimal. For instance, the perfect-perception controller used for the mobile robot application from the next section decides that the robot should mostly or even always wait when a collision with another mobile agent would otherwise occur. This line of reasoning also implies that deterministic perfect-perception controllers are likely to not have equivalent (probabilistic or deterministic) DNN-perception controllers.

Stage 3: Controller synthesis. The *controller synthesis problem for the DNN-perception system* involves finding instantiations $\underline{\hat{x}}$ of the DNN-perception controller parameters for which the pDTMC $\hat{\mathcal{M}}$ from (17) satisfies the constraints (15) and is Pareto optimal with respect to the optimisation objectives (16). Solving the general version of this problem precisely is unfeasible. However, metaheuristics such as multi-objective genetic algorithms for probabilistic model synthesis[19,20] can be used to generate close approximations of the Pareto-optimal controller set. Alternatively, exhaustive search can be employed to synthesise the actual Pareto-optimal controller set for systems with deterministic controllers and small numbers of controller parameters, or—by discretising the search space—an approximate Pareto-optimal controller set for systems with probabilistic controllers. We demonstrate the synthesis of DEEPDECS controllers through the use of both metaheuristics and exhaustive search in the next section.

2 DEEPDECS Applications

Mobile-robot collision limitation. Recent research proposes the use of DNN perception in the collision avoidance systems of unmanned aircraft,[21,22] autonomous marine vehicles[23] and autonomous mobile robots.[24] We used DEEPDECS to develop a mobile robot collision-limitation controller inspired by these applications. As shown in Figure 3a, we considered a service robot tasked with travelling autonomously from location A to location B, e.g., for the purpose of carrying goods in a warehouse or hospital. Within this environment, the robot may encounter and potentially collide with another moving autonomous agent. We assume that these collisions are not catastrophic, but that they incur damage to the robot and slow it down. As such, the robot uses DNN perception to assess the risk of collision at each intermediate waypoint I, and decides whether to proceed to the next waypoint or to wait for a while at waypoint I based on the DNN output.

The logic underpinning the operation of the robot at any intermediate waypoint I is modelled by the perfect-perception pDTMC in Figure 3b. As shown by the MobileRobot pDTMC module, when reaching waypoint I the robot first uses its sensors (lidar, cameras, etc.) to look for the "collider" (state $z=0$). If the collider is present in the vicinity of the robot (which happens with probability $\mathsf{p}_{\mathsf{collider}}$, known from previous executions of the task), the robot performs a check action (state $z=1$). As defined in the module Collider, this leads to the execution of a monitor action to predict whether travelling to the next waypoint would place the robot on collision course with the other agent (which happens with probability $\mathsf{p}_{\mathsf{occ}}$, also known from historical data) or not. Each monitor action activates the controller, whose behaviour is specified by the PerfectPerceptionController module. A probabilistic controller with two parameters is used: the controller decides that the robot should wait with probability x_1 when no collision is predicted ($k=1$) and with probability x_2 if a collision is predicted ($k=2$). Depending on this decision, the robot will either retry after a short wait or proceed and travel to the next waypoint, reaching the end of the decision-making process. Finally, when the collider is absent (with probability $1-\mathsf{p}_{\mathsf{collider}}$ in the first line from the MobileRobot module), the robot can travel without going through these intermediate actions.

We used data from a simulator of the scenario in Figure 3a to train a collision-prediction DNN. We then applied DEEPDECS (Figure 1) to this DNN, a test dataset collected using the same simulator, the perfect-perception pDTMC model from Figure 3b, and a set of PCTL-encoded requirements demanding controllers that can (a) guarantee a collision-free journey with probability of at least 0.75:

$$C_1: \quad \mathcal{P}_{\geq 0.75}[\neg \text{collision} \,\mathrm{U}\, \text{done}] \tag{31}$$

and (b) achieve an optimal trade-off between maximising this probability and minimising the travel time:

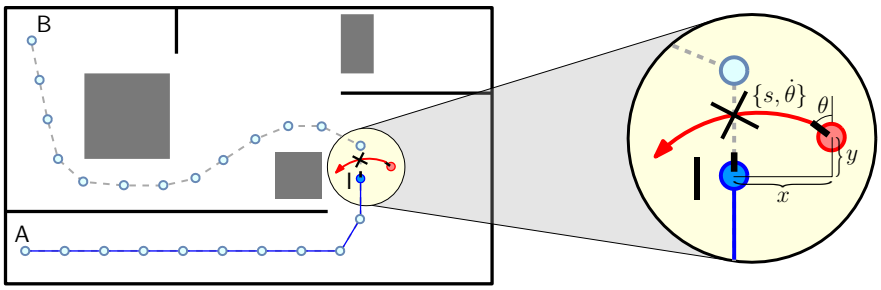
$$\begin{array}{ll} O_1: & \text{maximise } \mathcal{P}_{=?}[\neg \text{collision} \,\mathrm{U}\, \text{done}] \\ O_2: & \text{minimise } \mathcal{R}^{\text{time}}_{=?}[\mathrm{F}\, done] \end{array} \tag{32}$$

We applied DEEPDECS to these inputs four times, using different combinations of DNN verification methods in the DNN uncertainty quantification stage: (i) no DNN verification method; (ii) only $verif_1$ from (4); (iii) only $verif_2$ from (3); and (iv) both $verif_1$ and $verif_2$. Figure 3c shows DNN-perception pDTMC model produced by the model augmentation DEEPDECS stage for combination (ii), and Figure 4 presents the results from the other DEEPDECS stages and from the testing of the synthesised controllers for all combinations. Figure 4a presents the probability of the DNN making a correct prediction across the four setups after training. The DNN achieved high accuracy, and as verification methods are introduced it is observed that the "verified" predictions have a higher probability of being correct.

The controller design space was explored via discretising the parameter space, with each of the parameters $x_{1\mathsf{false}}$, $x_{1\mathsf{true}}$, $x_{2\mathsf{false}}$ and $x_{2\mathsf{true}}$ varied between 0 and 1 with a step size of 0.1, and every $(x_{1\mathsf{false}},x_{1\mathsf{true}},x_{2\mathsf{false}},x_{2\mathsf{true}})$ combination obtained in this way was analysed. The setup with perfect perception (Figure 3b) was analysed similarly for evaluation purposes. Pareto fronts were generated from the modelled controller parameter combinations, see Figure 4b, clearly showing an intuitive balance of collision probability and time to complete journey; to achieve a faster time the less likely it will be a safe journey. The optimal results, expectedly, are achieved with a perfect perception, and the front achieved when no verification DNN method is used is always outperformed by the perfect perception front, other than at two extreme points. When verification methods are introduced, the Pareto fronts improve and get closer to the perfect DNN Pareto front. There are three instances where the Pareto fronts achieve the same result; first when the parameters are all set to 0 (travel regardless of prediction), 0.5 (always flip a balanced coin regardless of prediction), or 1 (always wait when there is a collider present).

We used two established Pareto front indicators to analyse the quality of the Pareto fronts: the Inverse Generational Distance (IGD)[25] and the Hypervolume (HV).[26] IGD uses a reference frame and calculates, for each point on the reference frame, the distance to the closest point on the Pareto front, with an average then extracted. HV also uses a reference frame to generate a nadir point which is then used with the Pareto front to determine how much of a region is covered by said Pareto front. Smaller IGD and higher HV values indicate a better Pareto front. The reference frame used for both IGD and HV was the Pareto front generated from the perfect perception setup, since this is the ideal Pareto front that a 100% accurate DNN would achieve. From these indicators it is observed that the setups using verification methods produces higher quality Pareto fronts than the no verification DNN method, see Figure 4c. Both IGD and HV also indicate that using both verification methods led to a better Pareto front than just utilising one verification method.

The Pareto fronts generated from the models were validated by integrating the synthesised controllers with the original mobile-robot simulation. With each synthesised controller, the simulator conducted a number of journeys for the robot with the same probabilities of collider presence and resultant collision if the robot was to travel, with an average time and probability of robot collision extracted. As the number of waypoints across these journeys increases, the difference



(a) A mobile robot (darker blue) travelling between locations A and B may collide with another mobile agent (red) when moving from its current waypoint I to the next waypoint. A two-class DNN predicts whether the robot is on collision course based on the relative horizontal distance x and vertical distance y between the robot and the collider, and the speed s, angle θ and angular velocity $\dot{\theta}$ of the collider.

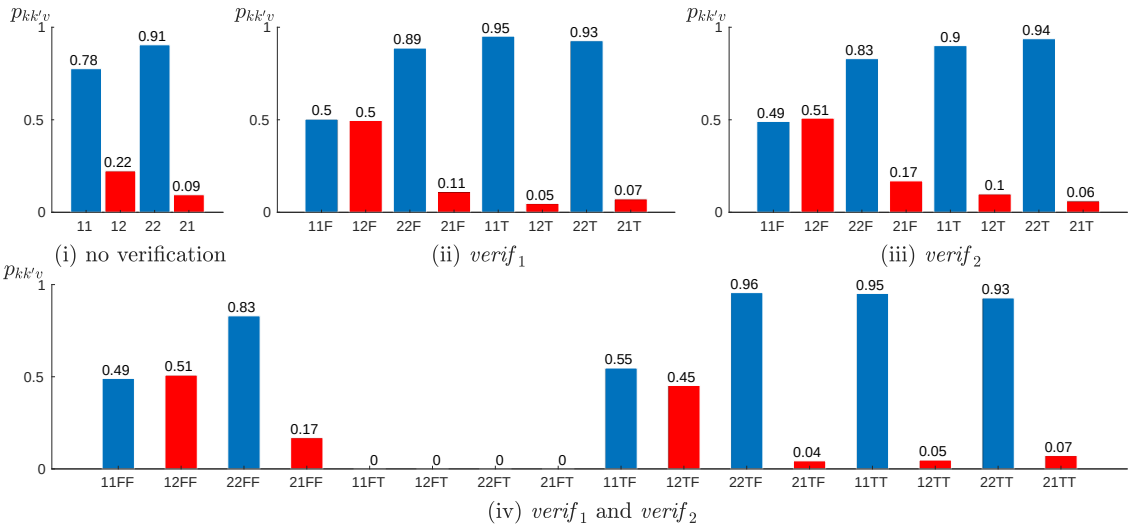
```
dtmc

const double p_collider = 0.8;

module MobileRobot // ManagedComponents
  z : [0..4] init 0; // 0:check collider, 1:collider detected,
                     // 2:check wait, 3:no collider, 4:done
  [look]     t=1 ∧ z=0 → p_collider:(z'=1) + (1-p_collider):(z'=3);
  [check]    t=1 ∧ z=1 → 1:(z'=2);
  [retry]    t=1 ∧ z=2 ∧ wait → 1:(z'=0);
  [proceed]  t=1 ∧ z=2 ∧ ¬wait → 1:(z'=3);
  [travel]   t=1 ∧ z=3 → 1:(z'=4);
  [end]      t=1 ∧ z=4 → 1:(z'=4);
endmodule

const double p_occ = 0.25;

module Collider // Environment
  k : [1..2] init 1; //1:not on collision course (occ), 2:occ
  [monitor] t=2 → (1-p_occ):(k'=1) + p_occ:(k'=2);
endmodule

const double x_1; // prob. of waiting when occ
const double x_2; // prob. of waiting when not occ

module PerfectPerceptionController
  wait : bool init false;
  [reaction] t=3 ∧ k=1 → x_1:(wait'=true) + (1-x_1):(wait'=false);
  [reaction] t=3 ∧ k=2 → x_2:(wait'=true) + (1-x_2):(wait'=false);
endmodule

module Turn
  t : [1..3] init 1;
  [check]    true → 1:(t'=2);
  [monitor]  true → 1:(t'=3);
  [decide]   true → 1:(t'=1);
endmodule

rewards "time"
  [travel]   true : 9.95;
  [proceed]  k=2 : 2.57;
  [retry]    true : 5;
endrewards

label "collision" = z=3 & k=2;
label "done" = z=4;
```

(b) Perfect-perception pDTMC model of the mobile robot journey between two successive waypoints. The model states are tuples $(z,k,t,wait) \in \{0,1,...,4\} \times [2] \times [3] \times \mathbb{B}$ with the semantics from (11). The reward structure from models the time taken by each robot actions: 9.95 time units to travel between adjacent waypoints (without collision), 2.57 additional time units when the robot decides to go despite being on collision course, and five time units when the robot decides to retry after a short wait; the other robot actions are assumed to take negligible time. Two atomic propositions are used by the labelling function at the end of the model: collision, for states in which the robot travels despite being on collision course, and done, for states that mark the end of the journey.

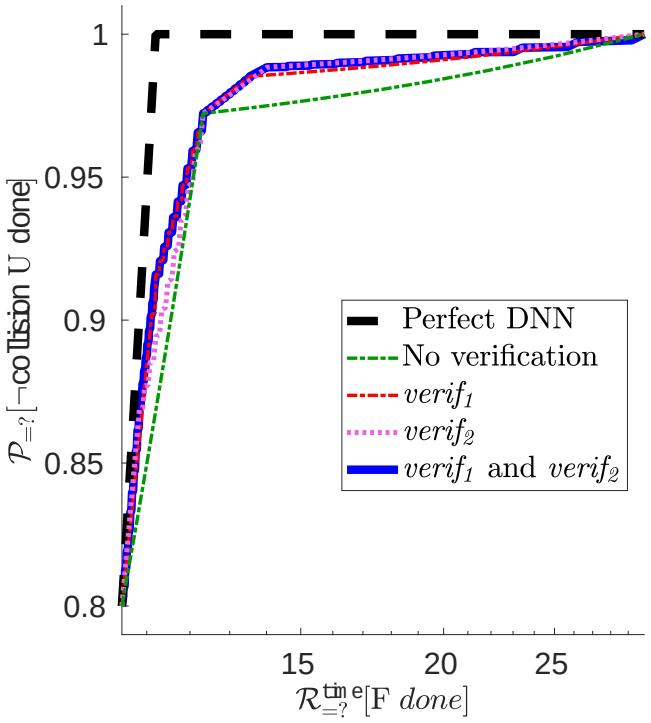
```
dtmc

const double p_collider = 0.8;

module MobileRobot // ManagedComponents
  z : [0..4] init 0;
  [look]     t=1 ∧ z=0 → p_collider:(z'=1) + (1-p_collider):(z'=3);
  [check]    t=1 ∧ z=1 → 1:(z'=2);
  [retry]    t=1 ∧ z=2 ∧ wait → 1:(z'=0);
  [proceed]  t=1 ∧ z=2 ∧ ¬wait → 1:(z'=3);
  [travel]   t=1 ∧ z=3 → 1:(z'=4);
  [end]      t=1 ∧ z=4 → 1:(z'=4);
endmodule

const double p_occ = 0.25;
const double p_11false = eq. (7)
...
const double p_22true = eq. (7)

module ColliderWithDNNPerception // EnvironmentWithDNNPerception
  k : [1..2] init 1; // 1:not occ, 2:occ
  k̂ : [1..2] init 1; // DNN predicts 1:not occ, 2:occ
  v_1 : bool init false;
  [monitor] t=2 → (1-p_occ)·p_11false:(k'=1)&(k̂'=1)&(v_1'=false)
                + (1-p_occ)·p_11true:(k'=1)&(k̂'=1)&(v_1'=true)
                + (1-p_occ)·p_12false:(k'=1)&(k̂'=2)&(v_1'=false)
                + (1-p_occ)·p_12true:(k'=1)&(k̂'=2)&(v_1'=true)
                + p_occ·p_21false:(k'=2)&(k̂'=1)&(v_1'=false)
                + p_occ·p_21true:(k'=2)&(k̂'=1)&(v_1'=true)
                + p_occ·p_22false:(k'=2)&(k̂'=2)&(v_1'=false)
                + p_occ·p_22true:(k'=2)&(k̂'=2)&(v_1'=true);
endmodule

const double x_1false; // DNN predicts not occ and v1 returns false
const double x_1true; // DNN predicts not occ and v1 returns true
const double x_2false; // DNN predicts occ and v1 returns false
const double x_2true; // DNN predicts occ and v1 returns true

module DNNPerceptionController
  wait : bool init false;
  [decide]   t=3 ∧ k̂=1 ∧ ¬v_1 → x_1false:(wait'=true) + (1-x_1false):(wait'=false);
  [decide]   t=3 ∧ k̂=1 ∧ v_1 → x_1true:(wait'=true) + (1-x_1true):(wait'=false);
  [decide]   t=3 ∧ k̂=2 ∧ ¬v_1 → x_2false:(wait'=true) + (1-x_2false):(wait'=false);
  [decide]   t=3 ∧ k̂=2 ∧ v_1 → x_2true:(wait'=true) + (1-x_2true):(wait'=false);
endmodule

module Turn
  t : [1..3] init 1;
  [check]    true → 1:(t'=2);
  [monitor]  true → 1:(t'=3);
  [decide]   true → 1:(t'=1);
endmodule
...
```

(c) DNN-perception pDTMC model of the mobile robot journey between two successive waypoints. The probabilities $p_{kk'v_1}$ from the ColliderWithDNNPerception module quantify the DNN accuracy for "verified" inputs ($v_1 = \text{true}$) and "unverified" inputs ($v_1 = \text{false}$), and are used to model the class $\hat{k}$ predicted by the DNN when the true class is k. The decisions of the four-parameter probabilistic controller depend on the DNN prediction $\hat{k}$ and the online verification result v_1.

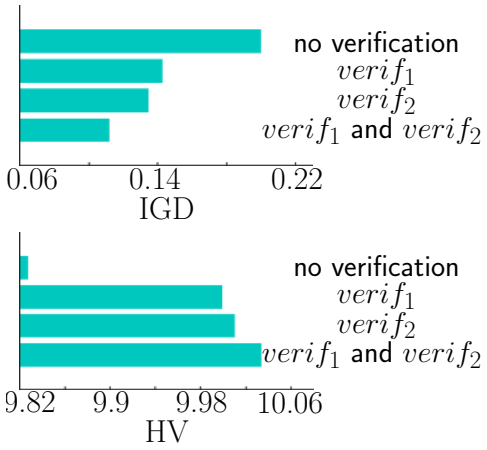
Figure 3: Collision limitation for a mobile robot tasked with traversing a known environment through the use of waypoints



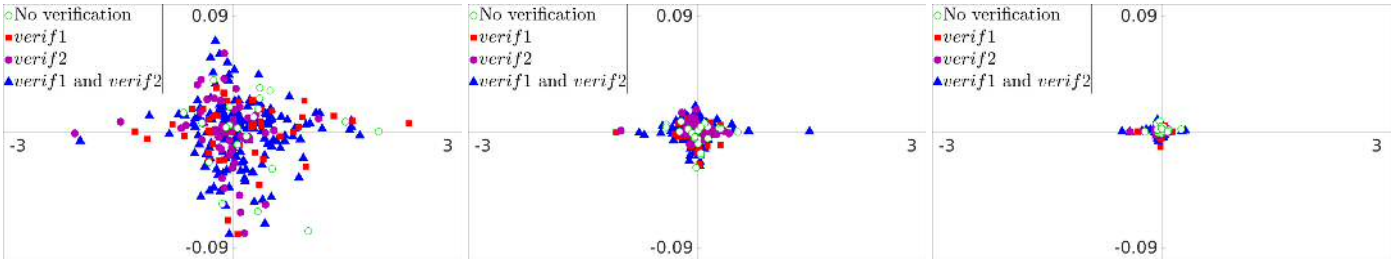
(a) Probability that a class-k DNN input is (mis)classified as class k' and satisfies $verif_i = (v_1, v_2, ..., v_n) \in \mathbb{B}^n$



(b) Pareto front associated with the set of Pareto-optimal DEEPDECS controllers



(c) Inverted Generational Distance (IGD) and hypervolume (HV) quality indicator values for the DNN-perception controller Pareto fronts: smaller IGD values and larger HV values indicate better Pareto fronts (i.e., Pareto fronts closer to the ideal-perception Pareto front).



(d) Difference between the model-based and simulation-based probability of collision-free travel and travel time for the Pareto-optimal DEEPDECS controllers from Figure 4b for 100 waypoints (left), 1000 waypoints (middle) and 10000 waypoints (right).

Figure 4: DEEPDECS controller synthesis and testing results for the mobile robot collision limitation

between the simulator and the model decreases, see Figure 4d, thus validating the models and controllers used.

Driver-attentiveness management. The recent adoption of the first United Nations regulation on vehicles with Level 3 automation[27] has paved the way for the safe introduction of shared-control passenger cars with Automated Lane Keeping Systems (ALKS). In certain traffic environments detailed in the regulation—and for as long as the driver is attentive—these cars will be able to drive entirely autonomously. However, ALKS can issue *transition demands* requesting the driver to take over the driving task when the car approaches traffic conditions outside its ODD. Transition demands will be issued timely, enabling an attentive driver to resume manual driving safely. If the driver is unresponsive or becomes inattentive and ALKS-issued alerts meant to restore driver attentiveness are ineffective, a *minimum-risk manoeuvre* (e.g., bringing the car to a standstill) will be performed.

We applied DEEPDECS to design a proof-of-concept driver-attentiveness management system for ALKS-enabled cars. Developed as part of our SafeSCAD project[f], this system uses (Figure 5a): (i) specialised sensors to monitor key car parameters (velocity, lane position, etc.) and driver's biometrics (eye movement, heart rate, etc.), (ii) a DNN to predict the driver's response to a transition demand, and (iii) a software controller to issue visual/acoustic/haptic alerts when the driver is inattentive.

We used an existing DNN trained and validated with driver data from a SafeSCAD user study carried out within a driving simulator.[28] The test dataset used for our DNN uncertainty quantification came from the same study, and the perfect-perception pDTMC model provided to DEEPDECS (shown in Figure 5b) is a significantly revised version of a model we proposed in preliminary SafeSCAD work.[29] Finally, the controller requirements comprise two constraints that limit the maximum expected risk and driver nuisance accrued over a 45-minute driving trip, and two optimisation objectives requiring that the same two measures are minimised:

$$\begin{aligned} C_1\colon&\quad \mathcal{R}^{\mathsf{risk}}_{\leq 100}[\mathrm{C}^{\leq 2000}]\\ C_2\colon&\quad \mathcal{R}^{\mathsf{nuisance}}_{\leq 6\times 10^3}[\mathrm{C}^{\leq 2000}]\\ O_1\colon&\quad \text{minimise } \mathcal{R}^{\mathsf{risk}}_{=?}[\mathrm{C}^{\leq 2000}]\\ O_2\colon&\quad \text{minimise } \mathcal{R}^{\mathsf{nuisance}}_{=?}[\mathrm{C}^{\leq 2000}] \end{aligned} \tag{33}$$

where each occurrence of the PCTL reward operator $\mathcal{R}$ is annotated with the name of the reward structure from Figure 5b it refers to (i.e., 'risk' of 'nuisance'). The 2000 time-steps from the PCTL cumulative reward properties correspond to the 45 minutes of the journey: verifying the driver state every 4s requires 667 verifications over 2667s, and each verification is modelled by three pDTMC time-steps, one for the monitoring of the driver state, one for the controller to decide the appropriate alerts for the observed state, and one for the decided alerts to be issued in order to warn the driver.

The DNN verification methods $verif_1$ and $verif_2$ from (3) and (4) were used in all possible combinations (i.e., alone, together, and neither) in the DEEPDECS DNN uncertainty quantification stage. Figure 6a shows the DNN-perception pDTMC obtained in the model augmentation stage using verification results produced when $verif_1$ was used alone, and Figures 6b and 6c compare the controller Pareto fronts obtained for all these combinations to the Pareto front associated with the perfect-perception model from Figure 5b.

The search space in this case study (8^{12} controller parameter combinations when both DNN verification methods are used) is significantly larger than for the discretised robot collision avoidance scenario. As such, an exhaustive search to determine the Pareto-optimal controllers is infeasible. Therefore EvoChecker,[20] which adopts multiobjective genetic algorithms, was employed to generate close approximations of the Pareto-optimal controllers. The Pareto fronts, see Figure 6b, convey a similar relationship to that displayed by the Pareto fronts in the robot collision avoidance case; the inclusion of verification methods achieves Pareto-optimal controllers closer to the perfect-perception Pareto fronts. Furthermore, the knee point of the $verif_1$ and $verif_1$ and $verif_2$ fronts are closest to the knee point of the perfect DNN front. These two fronts share a similar frontier in general. This is reflected further in the quantitative analysis, Figure 6c, with the IGD values of these fronts being the smallest out of the four setups.

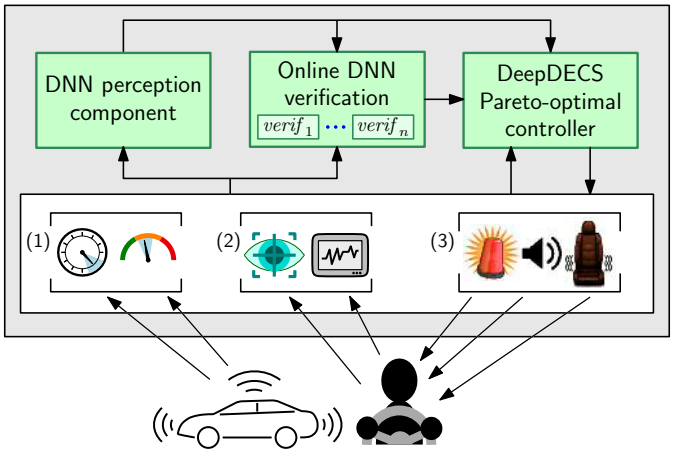
The HV indicator also supports inclusion of verification methods, as it strongly shows the no verification setup performs the worse. The two best setups were, again, $verif_1$ on its own and the $verif_1$ and $verif_2$ setup, with HV values of 8.91×10^5 and 8.89×10^5 respectively. This is still quite a large gap between the two fronts, though this is not surprising. The $verif_1$ Pareto front has the extremes significantly further apart, thus resulting in a significantly higher HV value. It is perhaps best to note that the two absolute extremes can be achieved in all setups; for minimised risk, perform the most cautious action of issuing all alerts regardless of driver's attention, and vice versa for minimising nuisance. The state space for a controller integrated with two verification methods is twice the size of using only one, which therefore requires a significantly larger evaluation time to properly search and find the Pareto-optimal controller. We expect that if the population size and evaluation are of substantial size, the two extremes would be found (in all setups) and the HV would in turn favour the $verif_1$ and $verif_2$ setup.

Another interesting outcome is that while the same verification methods were applied to both studies, the different scenarios dictated which method was more valuable; $verif_2$ in the robot collision avoidance study and $verif_1$ in the SafeSCAD scenario. This is indicative of the appropriateness of creating the confusion matrices and models with respect to the specific verification methods outcomes rather than using a collective count of verification methods satisfied.

3 Discussion

To operate safely and effectively, autonomous systems need to perceive, and respond to, changes in their environment. Increasingly, this perception involves some form of machine learning—often a deep-learning component that maps sensor data to predefined classes of environmental states or events. Autonomous system controllers, typically implemented as tra-

[f] Safety of Shared Control in Autonomous Driving (`cutt.ly/Safe-SCAD`)



(a) SafeSCAD driver-attentiveness management system. Data from car sensors (1) and driver biometric sensors (2) are supplied to a DNN perception component that classifies the driver state as attentive, semi-attentive or inattentive. The DEEPDECS controller decides when optical, acoustic and/or haptic alerts (3) should be used to increase the driver's attentiveness.

```
1   dtmc
2
3   module Alerts // ManagedComponents
4     z : [0..7] init 0;
5
6     [warn]     t=1 → 1:(z'=c);
7   endmodule
8
9   // probabilities pd_{kk'c} that driver attentiveness changes from level k∈{1,2,3} to level k'∈{1,2,3} given alerts z∈{0,1,...,7}
10  const double  pd110 = 0.99775;
    ...
81  const double  pd337 = 0.809;
82
83  module Driver // Environment
84    k : [1..3] init 1; // driver status: attentive (k = 1); semi-attentive (k = 2); or inattentive (k = 3)
85
86    // driver attentiveness changes from level k∈{1,2,3} to level k'∈{1,2,3} given alerts z∈{0,1,...,7}
87    [monitor] t=2 ∧ k=1 ∧ z=0 → pd110:(k'=1) + pd120:(k'=2) + pd130:(k'=3);
      ...
110   [monitor] t=2 ∧ k=3 ∧ z=7 → pd317:(k'=1) + pd327:(k'=2) + pd337:(k'=3);
111 endmodule
112
113 const int  x1; // alerts to be issued when driver is found attentive (k = 1)
114 const int  x2; // alerts to be issued when driver is found semi-attentive (k = 2)
115 const int  x3; // alerts to be issued when driver is found inattentive (k = 3)
116
117 module PerfectPerceptionController
118   c : [0..7] init 0;
119
120   [decide]   t=3 ∧ k=1 → 1:(c'=x1);
121   [decide]   t=3 ∧ k=2 → 1:(c'=x2);
122   [decide]   t=3 ∧ k=3 → 1:(c'=x3);
123 endmodule
124
125 module Turn
126   t : [1..3] init 1;
127
128   [warn]     true → 1:(t'=2);
129   [monitor]  true → 1:(t'=3);
130   [decide]   true → 1:(t'=1);
131 endmodule
```

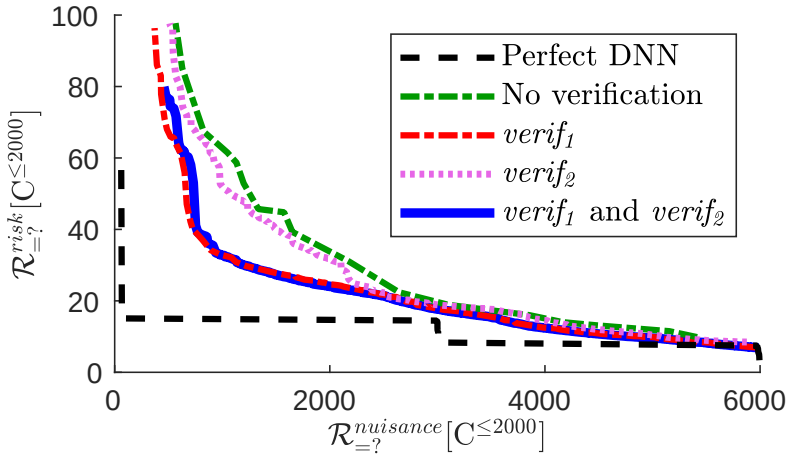
```
132
133 // risk when driver is not attentive
134 rewards "risk"
135   [monitor] k=1 : 0; // no risk
136   [monitor] k=2 : 1; // low risk
137   [monitor] k=3 : 4; // high risk
138 endrewards
139
140 // driver nuisance caused by alerts
141 rewards "nuisance"
142   [monitor] z=1 : (k=1)?6:2;
143   [monitor] z=2 : (k=1)?3:1;
144   [monitor] z=3 : (k=1)?8:3;
145   [monitor] z=4 : (k=1)?10:3;
146   [monitor] z=5 : (k=1)?16:5;
147   [monitor] z=6 : (k=1)?11:4;
148   [monitor] z=7 : (k=1)?20:6;
149 endrewards
```

(b) Perfect-perception pDTMC model of the SafeSCAD system. The model states are tuples $(z,k,t,c) \in [7] \times [3]^2 \times \{0,1,...,7\}$ with the semantics from (11). The Alerts module is responsible for warning the driver by "implementing" the controller-decided alerts c. The Driver module models the driver attentiveness level k, which is monitored every 4s; the probabilities of transition between attentiveness levels depend on the combination of alerts z in place. The control parameters $x_1,x_2,x_3 \in \{0,1,...,7\}$ are binary encodings of the alerts to be activated for each of the three driver attentiveness levels, e.g., $x_3 = 5 = 101_{(2)}$ corresponds to a deterministic-controller decision to have the optical alert active, the acoustic alert inactive, and the haptic alert active when the driver is inattentive. The reward structures from lines 134–138 and 141–149 quantify the risk and driver nuisance associated with the different driver attentiveness levels and alert combinations, respectively. The expressions '$(k=1)?value_1 : value_2$' from lines 142–148 evaluate to the larger $value_1$ if the driver is attentive (i.e., $k=1$), and $value_2$ otherwise.

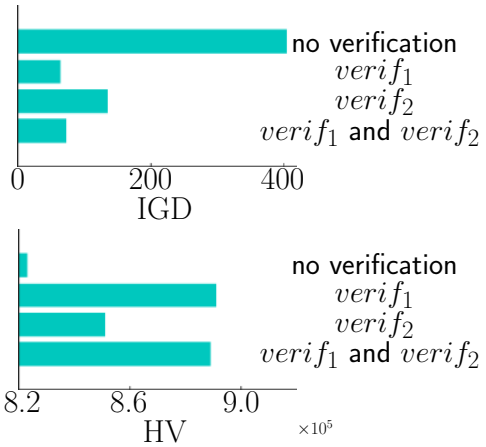
Figure 5: Driver-attentiveness management for shared-control autonomous driving

```
1   dtmc
2
3   module Alerts // ManagedComponents
4     z : [0..7] init 0;
5
6     [warn]     t=1 → 1:(z'=c);
7   endmodule
8
9   // probabilities pd_{kk'c} that driver attentiveness changes from level k∈{1,2,3} to level k'∈{1,2,3} given alerts z∈{0,1,...,7}
10  const double  pd110 = 0.99775;
    ...
81  const double  pd337 = 0.809;
82
83  // probabilities p_{kk̂v1} that DNN (mis)classifies the driver state k as k̂ when the online verification result is v1
84  const double  p11false = eq. (7)
    ...
101 const double  p33true = eq. (7)
102
103 module DriverWithDNNPerception // EnvironmentWithDNNPerception
104   k : [1..3] init 1; // driver status: attentive (k = 1); semi-attentive (k = 2); or inattentive (k = 3)
105   k̂ : [1..3] init 1; // DNN-predicted driver status: attentive (k̂ = 1); semi-attentive (k̂ = 2); or inattentive (k̂ = 3)
106   v1 : bool init false;
107
108   // driver attentiveness changes from level k to true level k' and DNN-predicted level k̂' given alerts z
109   [monitor] t=2 ∧ k=1 ∧ z=0 → pd110·p11false:(k'=1)&(k̂'=1)&(v1'=false) + pd110·p11true:(k'=1)&(k̂'=1)&(v1'=true) +
110                               pd110·p12false:(k'=1)&(k̂'=2)&(v1'=false) + pd110·p12true:(k'=1)&(k̂'=2)&(v1'=true) +
111                               pd110·p13false:(k'=1)&(k̂'=3)&(v1'=false) + pd110·p13true:(k'=1)&(k̂'=3)&(v1'=true) +
112                               pd120·p21false:(k'=2)&(k̂'=1)&(v1'=false) + pd120·p21true:(k'=2)&(k̂'=1)&(v1'=true) +
113                               pd120·p22false:(k'=2)&(k̂'=2)&(v1'=false) + pd120·p22true:(k'=2)&(k̂'=2)&(v1'=true) +
114                               pd120·p23false:(k'=2)&(k̂'=3)&(v1'=false) + pd120·p23true:(k'=2)&(k̂'=3)&(v1'=true) +
115                               pd130·p31false:(k'=3)&(k̂'=1)&(v1'=false) + pd130·p31true:(k'=3)&(k̂'=1)&(v1'=true) +
116                               pd130·p32false:(k'=3)&(k̂'=2)&(v1'=false) + pd130·p32true:(k'=3)&(k̂'=2)&(v1'=true) +
117                               pd130·p33false:(k'=3)&(k̂'=3)&(v1'=false) + pd130·p33true:(k'=3)&(k̂'=3)&(v1'=true);
    ...
325 endmodule
326
327 const int  x1false; // alerts to be issued when driver is classified attentive (k̂ = 1) and verification result is false
...
332 const int  x3true; // alerts to be issued when driver is classified inattentive (k̂ = 3) and verification result is true
333
334 module DNNPerceptionController
335   c : [0..7] init 0;
336
337   [decide]   t=3 ∧ k̂=1 ∧ ¬v1 → 1:(c'=x1false);
    ...
342   [decide]   t=3 ∧ k̂=3 ∧ v1 → 1:(c'=x3true);
343 endmodule
344
345 module Turn
    ...
351 endmodule
```

(a) DNN-perception pDTMC model of the SafeSCAD driver-attentiveness management system for the scenario when a single DNN verification method is used to distinguish between "verified" ($v_1 = \text{true}$) and "unverified" ($v_1 = \text{false}$) DNN predictions of the driver's attentiveness level. Lines 103–111 show how all combinations of true (k') and DNN-predicted ($\hat{k}'$) driver attentiveness levels can be reached from the attentive driver state ($k = 1$) when no alerts are used ($c = 0$). The six-parameter deterministic controller decides a combination of alerts c for each pair of DNN-predicted driver atentiveness level $\hat{k}$ and online DNN verification result v_1 (lines 339-344). The Switch module and the two reward structures from the pDTMC in Figure 5b are omitted for brevity.



(b) Pareto front associated with the set of Pareto-optimal SafeSCAD controllers



(c) Evaluation of Pareto front quality using the established IGD and HV metrics.

Figure 6: DEEPDECS controller synthesis model and results for the driver-attentiveness management system

ditional-software components, can then consider these states and events in their decision making. However, deep-learning components can never be 100% accurate. This limitation poses a major challenge to established controller development methods. The DEEPDECS method introduced in our paper addresses this challenge through several key contributions.

First, we devised a new approach to quantifying aleatory DNN uncertainty within the ODD of an autonomous system. Using a suite of n DNN verification techniques, this approach identifies 2^n categories of DNN outputs, each of which is associated with a different trustworthiness level. The uncertainty of each category of DNN outputs is then separately quantified using a unique combination of development-time and online DNN verification. This enables the controllers of autonomous systems to consider each category of DNN outputs differently. In particular, these controllers can react confidently to highly trustworthy DNN outputs, and conservatively to DNN outputs associated with low trustworthiness levels. Importantly, we showed experimentally that the vast majority of DNN outputs fell into the former category for the DNN-perception components of two autonomous systems from different application domains. Our approach to quantifying DNN uncertainty opens up the opportunity to leverage the broad range of recently devised DNN verification techniques[8–10,12,30–32] that certify DNN properties like local robustness and confidence.

Second, we developed a theoretical foundation for the integration of DNN-perception uncertainty into discrete-time stochastic models describing the behaviour of autonomous systems. The new theory enables the formal analysis of safety, dependability, performance and other key properties of autonomous systems with necessarily imperfect deep-learning perception components. Furthermore, it supports the formal modelling of autonomous systems that use any combination of online DNN verification techniques to quantify their environment perception uncertainty.

Third, we showed how the DEEPDECS parametric DTMC models of DNN-perception autonomous systems can be used to synthesise both deterministic and probabilistic discrete-event controllers for these systems. Given n_1 constraints and n_2 optimisation objectives that formalise in probabilistic temporal logic the safety, dependability and performance requirements of an autonomous system, this synthesis yields controllers guaranteed to satisfy the n_1 constraints and that are Pareto-optimal with respect to the n_2 optimisation objectives. Importantly, we showed how using larger numbers of DNN verification techniques produces better sets of Pareto-optimal controllers.

Finally, we demonstrated the applicability of DEEPDECS within two case studies from different application domains. For the first case study, we synthesised probabilistic collision-limitation controllers for a mobile robot travelling in an environment where another moving autonomous agent may also be present. To obtain these controllers, we discretised the controller parameter space, and then performed an exhaustive search over all possible combinations of discretised parameter values. For the second case study, we used multi-objective genetic algorithms to synthesise deterministic controllers for a driver-attentiveness management system.

While our paper focuses on controller synthesis for autonomous systems with DNN perception components, DEEPDECS is not prescriptive about the type of machine learning (ML) components that introduce uncertainty into autonomous systems. We therefore envisage that DEEPDECS is equally applicable to additional such systems, as long as analogous ML verification methods exist to enable the quantification of the aleatory uncertainty introduced by their ML components. Examples of other ML techniques that utilise confidence measures to quantify the uncertainty of their predictions include support vector machines and Gaussian processes.

While the design of autonomous systems that use DNN classifiers for perception in combination with discrete-event controllers for decision-making has been studied,[33–35] synthesizing safe and optimal discrete-event controllers that account for the uncertainty in the DNN outcomes is a novel contribution of this work. Additionally, our DNN uncertainty quantification mechanism, which uses the outcomes of off-the-shelf DNN verifiers in a black-box manner, is also new.

Related work includes the approach of Jha et al.[33] that synthesize correct-by-construction controllers for autonomous systems with noisy sensors, i.e., with perception uncertainty. Unlike our approach, they only consider systems using linear models (i.e., not DNNs) for perception where the uncertainty quantities are already known. Moreover, while we formulate the control problem as a pDTMC, Jha et al. consider the simpler setting of deterministic linear systems. Michelmore et al.[34] analyze the safety of autonomous driving control systems that use DNNs in an end-to-end manner for both perception and control, i.e., the DNN consumes sensor readings and outputs control actions. They use Bayesian methods for calculating the uncertainty in the control actions predicted by the DNN, and in case the DNN uncertainty is higher than pre-determined thresholds, the system defaults to executing fail-safe actions. In contrast, we synthesize controllers that can use the quantified uncertainty of DNN perception in order to select optimal yet safe actions. Cleaveland et al.[35] study verification of autonomous systems with machine learning-based perception. They are interested in situations where the controller has already been constructed and the uncertainty in the perception outcomes is known, so the only goal is to verify if the autonomous system satisfies a required probabilistic specification of safety.

There is a large body of work on quantifying the uncertainty in DNN classifiers. One major approach is to consider K-class DNN classifiers as functions that map an input $x \in \mathbb{R}^d$ to a discrete probability distribution over K classes. The probability associated with a class is then interpreted as the probability that the class is the *true* label of x. For these probability estimates to be useful for downstream decision-making, it is essential that the DNN is well-calibrated, i.e., the predicted probabilities are close to the true probabilities. Formally, a DNN f is perfectly calibrated if the following holds,

$$\forall\, p \in [0,1].\ \mathop{\mathbb{P}}_{x \sim D}\left[y = f^*(x) \mid \hat{p} = p\right] = p, \tag{34}$$

where $y = \text{argmax}_{i \in [K]}\{f(x)_i\}$, i.e., y is the predicted label, and $\hat{p} = f(x)_y$, i.e., $\hat{p}$ is the estimated probability associated with the predicted label. It has been shown that modern DNNs are not well-calibrated.[12] Even with carefully designed interventions to ensure well-calibration, there is no guarantee that the proba-

Table 1: Robot collision avoidance parameters

Parameter	Value
α	0.5
x_{goal}	0
y_{goal}	10
ϵ	0.05
x_{lim}	10
y_{lim}	10
s_{lim}	2 units/s
$\dot{\theta}_{lim}$	$\frac{\pi}{4}$ rads/s

bility estimates are close to the true values. Another approach to quantifying the uncertainty of DNN prediction is to use Bayesian techniques. In particular, Bayesian Neural Networks (BNNs) have been proposed as a Bayesian extension of DNNs. As opposed to DNNs where the weights θ associated with a DNN are fixed, BNNs consider a distribution over the weights, and the BNN prediction is the expected outcome with respect to the weights distribution. While a BNN can produce an estimate of the prediction uncertainty, even these estimates are not necessarily well-calibrated due to the possibility of model mis-specification.[36] Alternately, techniques based on conformal prediction[37,38] can be used to construct prediction sets,[39] i.e., a set of predicted values instead of a single predicted value for a given input, such that the true label is guaranteed to be in the prediction set with a user-controllable probability.

4 Methods

Dataset collection. We obtained the dataset for the collision avoidance study using the 2D particle simulator *Box2D* (`https://box2d.org/`). The robot and collider were circular particles with radius of 0.5 units. The robot was initialised at the origin with a heading of $\frac{\pi}{2}$ radians, however the robot only considers the local coordinate frame with itself as the reference, i.e. the initial heading is also 0 radians. The robot would travel in a straight line to the goal destination at coordinate (0,10) with a speed of 1 unit per second. The robot is considered to have successfully completed the journey if the robot is within the goal area which is defined as

$$(\hat{x},\hat{y}) = (x_{goal} \pm \epsilon, y_{goal} \pm \epsilon)$$

The robot will have an angular velocity as

$$\dot{\theta}_r = \alpha \cdot arctan\left(\frac{v_x \cdot y_{goal} - v_y \cdot x_{goal}}{v_x \cdot x_{goal} + v_y \cdot y_{goal}}\right)$$

where v_x and v_y is the velocity of the robot in the x-axis and y-axis respectively, and α is a scaling constant. If the difference between the robot's and the target heading is greater than $\frac{\pi}{36}$ then the robot's linear speed is reduced to 0.1, allowing time to correct its course.

The collider's initial position is

$$(x, y, \theta) = (\mathcal{U}(-x_{lim}, x_{lim}), \mathcal{U}(0, y_{lim}), \mathcal{U}(-\pi, \pi))$$

where $\mathcal{U}$ is the uniform distribution function. The collider's linear speed and angular speed were determined using the following function

$$(s, \dot{\theta}_c) = (\mathcal{U}(0, s_{lim}), \mathcal{U}(-\dot{\theta}_{lim}, \dot{\theta}_{lim}))$$

Table 1 provides the values used for the experimental setup.

Table 2: Hyperparameters for training DNN models

Hyperparameter	Collision Detection DNN	Driver Attentiveness Levels DNN
# of epochs	100	100
batch size	32	128
initial learning rate	0.005	0.01
optimizer	Adam	Adam
ϵ	0.05	0.01
τ	0.8	0.7

A collected datapoint was of the form $(x_{diff}, y_{diff}, s, \theta, \dot{\theta_c}, c)$, where x_{diff} and y_{diff} is the relative distance in the x and y axes between the robot and collider, and c is the label for collision/no collision. The datapoints were normalised for the DNN; [0,1] for y_{diff} and s and [-1,1] for x_{diff}, θ and $\dot{\theta}_c$.

The training data for the collision detection DNN was gathered through repeated simulations until 6×10^3 instances of collisions and 6×10^3 instances of no collisions occurred, with the collider's setup normalised and recorded. The time to complete a journey with a collision, and the time to complete without a collision was calculated from averages of 10×10^3 simulations for both collision and no collision instances. We used 80% of this dataset for training the DNN and 20% for calibration and validation. To gather the test dataset, 50×10^3 simulations were conducted which had a split of 5843 collisions and 44157 no collisions. This was passed through the trained DNN with the verification methods and the resultant confusion matrices were used to generate the model's probabilities of the DNN, see Figure 4a.

The dataset for the driver attentiveness management DNN was obtained from a user study[28] conducted as part of our SafeSCAD project. The data was normalised in the range [0,1]. We used 60% of the dataset for training the DNN and 15% for calibration and validation. The remaining 25% of the dataset was used for testing the DNN model and constructing the confusion matrices.

DNN training. The two autonomous systems considered in our evaluation, namely, a mobile robot navigation system and a system for maintenance of driver attentiveness level, use DNN-based classifiers for perception. In the first case, a DNN is used to detect if a mobile robot is on a collision path with another mobile robot in the second environment. In the second case, a DNN helps gauge a car driver's attentiveness levels.

The DNN for collision detection has the architecture prescribed by Ehlers[24]- the network has a fully-connected linear layer with 40 nodes, followed by a MaxPool layer with pool size 4 and stride size 1, followed by a fully-connected ReLU layer with 19 nodes, and a final fully-connected ReLU layer with 2 nodes. The DNN for gauging driver attentiveness level has an architecture with 10 layers, with eight fully-connected layers and two MaxPool layers (with additional Reshape layers as needed) . The 10 layers from start to end are as follows - fully-connected ReLU layer with 23 nodes, fully-connected ReLU layer with 50 nodes, fully-connected ReLU layer with 80 nodes, MaxPool layer with pool size 4 and stride size 1, fully-connected ReLU layer with 40 nodes, MaxPool layer with pool size 4 and stride size 1, fully-connected ReLU layer with 20 nodes, fully-connected ReLU layer with 14 nodes, fully-connected ReLU layer with 8 nodes, and a fully-connected linear layer with 3 nodes.

We train these models with a cross-entropy loss function (implicitly assuming a final softmax layer for both the DNNs) using the Adam optimization algorithm.[40] Table 2 lists the hyperparameters used for training the DNN models. Note that the learning rate in both cases is set to decay to 0.0001. We implement our DNN models and their training in Python, using TensorFlow.

DNN verification. We use DNN verifiers that check if a DNN satisfies specific properties of interest to augment a DNN's outcome. For every input, in addition to the DNN's prediction, the controller has access to the boolean outcomes of the DNN verifiers to aid in decision-making. As described in Section 1.3, we also use the DNN verification outcomes to construct fine-grained confusion matrices from the validation dataset. These confusion matrices help us compute empirical estimates of the probability of an incorrect prediction, conditioned on the correct prediction and the verification outcomes. These estimates are then used to synthesize a discrete-event controller that can account for the uncertainty in DNN predictions in it's decision-making.

The specific DNN properties that we consider for verification are the local robustness of the DNN at an input, and the probability assigned by the DNN classifier to the predicted class. For local robustness verification, we use the GloRo Net framework of Leino et al.[30] Given a DNN, the GloRo Net framework adds a new final layer to the network that augments the DNN outcome with an additional outcome indicating the local robustness of the DNN. This new layer computes the Lipschitz constant of the function denoted by the original DNN and uses it to verify local robustness. Instead of training the original DNN, the GloRo Net framework recommends training the modified DNN to help improve the robustness of the network. We follow this recommendation for both the systems that we study, and update the DNN architectures described earlier by adding the layer provided by the GloRo Net framework as the final layer. For the collision detection DNN, we verify local robustness of the within radius $\epsilon = 0.05$ (where ϵ is as defined in 4) whereas we use $\epsilon = 0.01$ for the DNN that gauges driver attentiveness levels.

For verifying if the DNN predictions meets a minimum confidence threshold, we look at the softmax output of the DNN classifier that assigns a probability to each class. To ensure that the DNN softmax outputs are well-calibrated, we use the simple temperature scaling mechanism presented by Guo et al.[12] as implemented by Kueppers et al.[41] We use the probability threshold τ of 0.8 for the collision detection DNN and threshold τ of 0.7 for the DNN that predicts driver attentiveness levels (where τ is as defined in 3).

SafeSCAD DNN-perception controller synthesis. The probabilistic model synthesis tool EvoChecker[20] was required for generating the Pareto fronts in the SafeSCAD study due to the large search space. For all setups the population size for the multi-objective genetic algorithm employed by EvoChecker was set to 1000 and the maximum number of evaluations was 20×10^4. EvoChecker utilised the Viking Cluster (see here for full technical details `https://www.york.ac.uk/it-services/services/viking-computing-cluster/#tab-1`), and specifically used 5 CPUs and 8GB of memory, with a set maximum time of five hours.

Pareto front hypervolume evaluation. HV for both case studies was calculated using the package *PyGMO* (`https://esa.github.io/pygmo/`). For HV a nadir point is required, which is usually found via the extrema of the reference frame scaled by a constant. The Pareto fronts of the setup with the perfect DNN were used as the reference frames, with a scaling factor of 1.5 in the collision avoidance scenario and 1.75 for the SafeSCAD study.

PyGMO for HV calculation requires the system's goal to minimise all objectives. Therefore for the collision avoidance scenario a Pareto front was generated to no longer maximise $\mathcal{P}_{=?}[\neg \text{collision} \,\mathrm{U}\, \text{done}]$ but instead minimise $\mathcal{P}_{=?}[\text{collision} \,\mathrm{U}\, \text{done}]$, i.e. maximise probability of being safe to minimise probability of being at risk. This was achieved by using the existing Pareto front and calculating the complementary probability; $1 - \mathcal{P}_{=?}[\neg \text{collision} \,\mathrm{U}\, \text{done}]$.

Validation of collision-mitigation controllers. The results presented in Figure 4d were achieved by first acquiring data buckets for setups resulting in collision and no collision. This was to ensure that the probability of encountering a collider that will result in collision in the model was replicated in the simulator. Simulations were conducted until 50×10^3 collision instances and 50×10^3 no collision instances were recorded. To ensure that these were guaranteed setups to achieve the desired result each setup was tested through 100 simulations to assure that the outcome was consistent. For the simulations producing the results in Figure 4d the setup for the collider (providing a collider was present determined by probability $p_{collider}$) used the weighted probability used in the model, p_{occ} see Figure 3. The robot used the trained DNN and verification methods to decide which parameter of the synthesised controller to use. If the robot decided to wait the wait time was added and the simulation restarted, i.e. the old collider was removed with a new collider. Once the robot had completed it's journey the journey time was recorded along with whether a collision occurred. To compare with the model, the model's results were subtracted from the simulator's averaged values to acquire the difference plotted in Figure 4d.

Acknowledgements

This project has received funding from the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience', the UKRI Global Research and Innovation Programme, and the Assuring Autonomy International Programme. The authors are grateful to the developers of the DeepTake deep neural network[28] for sharing the DeepTake data sets, and to the University of York's Viking research computing cluster team for providing access to their systems.

References

[1] Alvin I. Chen, Max L. Balter, Timothy J. Maguire, and Martin L. Yarmush. Deep learning robotic guidance for autonomous vascular access. *Nat Mach Intell*, 2:104—115, 2020.

[2] Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, et al. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature medicine*, 24(9):1342–1350, 2018.

[3] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[4] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[5] Domen Tabernik and Danijel Skočaj. Deep learning for large-scale traffic-sign detection and recognition. *IEEE Transactions on Intelligent Transportation Systems*, 21(4):1427–1440, 2019.

[6] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys*, 54(5):1–39, 2021.

[7] Vijay D'silva, Daniel Kroening, and Georg Weissenbacher. A survey of automated techniques for formal software verification.

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(7):1165–1178, 2008.

[8] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The Marabou framework for verification and analysis of deep neural networks. In *CAV*, pages 443–452. Springer, 2019.

[9] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *CAV*, pages 3–29, 2017.

[10] Divya Gopinath, Guy Katz, Corina S Păsăreanu, and Clark Barrett. DeepSafe: A data-driven approach for assessing robustness of neural networks. In *ATVA*, pages 3–19, 2018.

[11] On-Road Automated Driving (ORAD) committee. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Standard J3016_201806, SAE International, 2018.

[12] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1321–1330. JMLR.org, 2017.

[13] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*, pages 39–57. IEEE, 2017.

[14] Conrado Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *International Colloquium on Theoretical Aspects of Computing*, pages 280–294, 2005.

[15] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[16] Andrea Bianco and Luca De Alfaro. Model checking of probabilistic and nondeterministic systems. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.

[17] Suzana Andova, Holger Hermanns, and Joost-Pieter Katoen. Discrete-time rewards model-checked. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 88–104. Springer, 2003.

[18] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of the 23rd Int. Conf. on Computer Aided Verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[19] Radu Calinescu, Milan Ceska, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software*, 143:140 – 158, 2018.

[20] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. Synthesis of probabilistic models for quality-of-service software engineering. *Automated Software Engineering*, 25(4):785–831, 2018.

[21] Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3):598–608, 2019.

[22] Kyle D Julian and Mykel J Kochenderfer. Reachability analysis for neural network aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 44(6):1132–1142, 2021.

[23] Qingyang Xu, Yiqin Yang, Chengjin Zhang, and Li Zhang. Deep convolutional neural network-based autonomous marine vehicle maneuver. *International Journal of Fuzzy Systems*, 20(2):687–699, 2018.

[24] Rüdiger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In Deepak D'Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[25] David A. Van Veldhuizen. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. PhD thesis, Ph. D. thesis, 1999.

[26] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[27] UNECE. ECE/TRANS/WP.29/2020/81—United Nations Regulation on Uniform provisions concerning the approval of vehicles with regard to Automated Lane Keeping Systems, June 2020.

[28] Erfan Pakdamanian, Shili Sheng, Sonia Baee, Seongkook Heo, Sarit Kraus, and Lu Feng. DeepTake: Prediction of driver takeover behavior using multimodal data. In *2021 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2021.

[29] Radu Calinescu, Naif Alasmari, and Mario Gleirscher. Maintaining driver attentiveness in shared-control autonomous driving. In *16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 90–96. IEEE, 2021.

[30] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning (ICML)*, 2021.

[31] Colin Paterson, Radu Calinescu, and Chiara Picardi. Detection and mitigation of rare subclasses in deep neural network classifiers. In *2021 IEEE International Conference on Artificial Intelligence Testing*, pages 9–16. IEEE, 2021.

[32] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[33] Susmit Jha, Vasumathi Raman, Dorsa Sadigh, and Sanjit A. Seshia. Safe autonomy under perception uncertainty using chance-constrained temporal logic. *Journal of Automated Reasoning*, 60(1):43–62, 2018.

[34] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, and Marta Kwiatkowska. Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7344–7350, 2020.

[35] Matthew Cleaveland, Ivan Ruchkin, Oleg Sokolsky, and Insup Lee. Monotonic safety for scalable and data-efficient probabilistic safety analysis, 2021.

[36] Edwin Fong and Christopher C. Holmes. Conformal bayesian computation. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[37] Volodya Vovk, Alexander Gammerman, and Craig Saunders. Machine-learning applications of algorithmic randomness. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 444–453, 1999.

[38] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

[39] Stephen Bates, Anastasios Angelopoulos, Lihua Lei, Jitendra Malik, and Michael Jordan. Distribution-free, risk-controlling prediction sets. *J. ACM*, 68(6), September 2021.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[41] Fabian Küppers, Jan Kronenberger, Amirhossein Shantia, and Anselm Haselhoff. Multivariate confidence calibration for object detection. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

ASSURING
AUTONOMY
INTERNATIONAL PROGRAMME