Controller Synthesis for Autonomous Systems With Deep-Learning Perception Components

Radu Calinescu, *Senior Member, IEEE*, Calum Imrie, Ravi Mangal, Genaína Nunes Rodrigues, Corina Păsăreanu, Misael Alpizar Santana, and Gricel Vázquez

***Abstract*—We present DeepDECS, a new method for the synthesis of correct-by-construction software 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 software 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 objectives. We evaluate the method in simulation by using it to synthesise controllers for mobile-robot collision limitation, and for maintaining driver attentiveness in shared-control autonomous driving.**

***Index Terms*—Discrete-event controller synthesis, Markov model, deep neural network, uncertainty quantification, neurosymbolic AI.**

I. Introduction

In a growing range of application domains, software-controlled systems use deep neural network (DNN) classifiers to perceive and respond to changes in their environment autonomously. In an example of such an *autonomous system* (AS) from healthcare, DNN classification and localisation of blood vessels has been used to develop a robotic device for introducing needles into deformable patient tissues to draw blood or deliver medication autonomously [19]. In autonomous driving, DNN classifiers are widely used for traffic-sign detection and recognition [82], for object sensing and classification [38], and for other perception tasks. In finance, the decision-making of autonomous trading agents relies on DNN classifiers that perceive or predict market trends [71].

This integration of DNN perception into the AS control loop poses major assurance challenges [3], [36]. In particular, the long-established methods for formal software verification [27] are not applicable to DNNs, and thus cannot be used to provide safety and performance guarantees for AS comprising both traditional software and deep-learning components. Furthermore, verification methods developed specifically for DNNs focus on verifying robustness to changes in DNN inputs [45], [56] or input clusters [37]. As such, DNN verification methods cannot be used to establish system-level properties for the software controllers of DNN-perception AS.

Our paper presents DeepDECS,[1] a method for the synthesis of *discrete-event controllers* (i.e., software components that control the response of a system to events in its environment) that addresses this significant limitation. As shown in Fig. 1, DeepDECS generates discrete-event controllers aware of the uncertainty induced by the DNN perception component of an AS in three stages.

First, in a *DNN uncertainty quantification* stage (shown at the top of Fig. 1), n DNN verification techniques (taken from the existing repertoire of such techniques—see Section III-E) are used to analyse the pre-trained DNN perception component over a test dataset representative for the operational design domain (ODD) of the AS. The verification results provide separate quantifications of the DNN prediction uncertainty for the inputs verified by each of the 2^n combinations of verification techniques. As shown by our theoretical and experimental results, the DNN predictions have higher accuracy for inputs verified by more techniques, enabling the controller to act confidently after such trustworthy predictions, and conservatively after predictions associated with inputs verified by fewer or no techniques. As an analogy, consider medical diagnosis by a doctor, and the questions “Are the patient’s symptoms familiar to the doctor? (true/false)” and “Is the doctor well rested? (true/false)?” Knowing the likelihood of a correct diagnosis for all combinations of answers to these questions, e.g., based on historical data, allows the doctor to decide when to trust his or her diagnosis, and when to ask a colleague for a second opinion and/or to request, for instance, a blood test before deciding a treatment for the patient.

Next, a *Model augmentation* stage (depicted in the middle of Fig. 1) uses the uncertainty quantification results—and a parametric discrete-time Markov chain (pDTMC) that models

Manuscript received 27 May 2023; revised 12 March 2024; accepted 26 March 2024. Date of publication 10 April 2024; date of current version 14 June 2024. This work was supported by the EPSRC under project EP/V026747/1 ‘UKRI Trustworthy Autonomous Systems Node in Resilience’, the UKRI Global Research and Innovation Programme, and the Assuring Autonomy International Programme. The work or Radu Calinescu was also supported by the Institute for Software Engineering and Software Technology “Jose María Troya Linero” at the University of Málaga. Recommended for acceptance by S. Nejati. *(Corresponding author: Radu Calinescu.)*

Radu Calinescu, Calum Imrie, Misael Alpizar Santana, and Gricel Vázquez are with the Department of Computer Science, University of York, YO10 5GH York, U.K. (e-mail: radu.calinescu@york.ac.uk).

Ravi Mangal and Corina Păsăreanu are with Carnegie Mellon University, Silicon Valley, Moffett Field, CA 94035 USA.

Genaína Nunes Rodrigues is with the Department of Computer Science, University of Brasília, Brasília 70910-900, Brazil (e-mail: genaina@unb.br).

Digital Object Identifier 10.1109/TSE.2024.3385378

[1] Deep-learning *aware* Discrete-Event Controller Synthesis

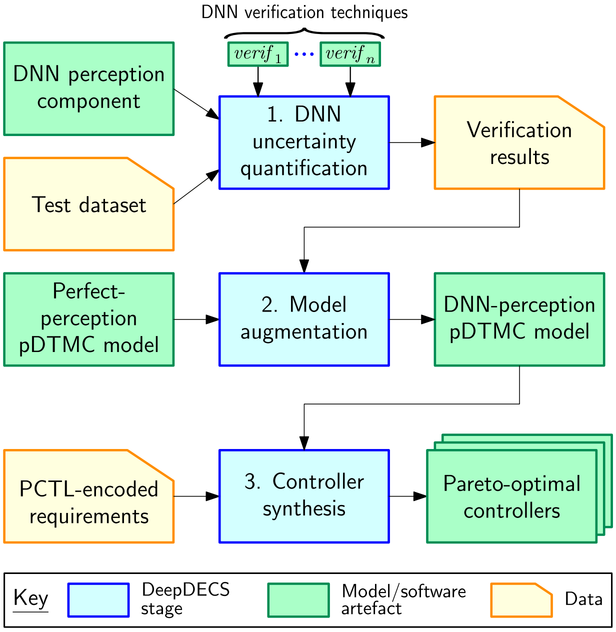

DNN verification techniques
verif_1 ··· verif_n
DNN perception component
Test dataset
1. DNN uncertainty quantification
Verification results
Perfect-perception pDTMC model
2. Model augmentation
DNN-perception pDTMC model
PCTL-encoded requirements
3. Controller synthesis
Pareto-optimal controllers
Key
DeepDECS stage
Model/software artefact
Data

Fig. 1. DeepDECS generation of discrete-event controllers aware of the uncertainty introduced by the DNN perception component of an autonomous system.

the AS behaviour assuming perfect perception—to assemble a pDTMC system model that takes the DNN-induced uncertainty into account. Finally, a *Controller synthesis* stage (shown at the bottom of Fig. 1) uses this uncertainty-aware pDTMC model to synthesise a set of discrete-event controllers guaranteed to satisfy the AS requirements (i.e., constraints and optimisation objectives) encoded in probabilistic computation tree logic (PCTL).

To the best of our knowledge, the hybrid verification approach underpinning the DeepDECS neuro-symbolic controller synthesis process is novel. As discussed in detail in our related work section, while other approaches that employ deep-learning classifiers for the discrete-event control of AS have been proposed, these approaches focus on the development of end-to-end DNN controllers for AS (e.g. [48], [65]), on quantifying the uncertainty of DNNs to support the probabilistic safety verification of autonomous systems (e.g., [5], [74]), and on verifying the safety of AS with DNN-based components and already implemented controllers (e.g., [22], [44], [46], [47], [49], [57], [72], [77]).

The main contributions of our paper are:

1) A theoretical foundation that integrates DNN uncertainty quantification, stochastic modelling and probabilistic model checking, to enable the synthesis of correct-by-construction neuro-symbolic controllers for autonomous systems with deep-learning perception components.
2) An open-source software tool that automates the augmentation of perfect-perception autonomous system models with constructs that capture the aleatory uncertainty introduced by the use of deep-learning perception components within such systems. The uncertainty that DeepDECS deals with is *aleatory* (i.e., it cannot be reduced) because it comes from an already trained DNN that does not learn at runtime. This differs from *epistemic uncertainty*, which is uncertainty due to insufficient knowledge, and therefore reducible by acquiring additional knowledge [26].
3) An extensive evaluation that shows the applicability of our method to the synthesis of discrete-event controllers for mobile-robot collision limitation, and for maintaining driver attentiveness in shared-control vehicles equipped with Level 3 (i.e., conditional automation) automated driving systems [70].

We structured the remainder of the paper as follows. Section II introduces a running example that we use to illustrate the application of DeepDECS. Section III provides a brief introduction to the formal modelling and verification paradigms integrated by DeepDECS. Sections IV and V present the DeepDECS theoretical foundation, and its evaluation for two autonomous systems from different application domains, respectively. We compare DeepDECS to related work, and discuss its merits and limitations in Section VI. Finally, Section VII provides a brief summary and discusses our plans for future work.

II. Running Example

To illustrate the DeepDECS theoretical foundation and its application, we will use a running example inspired by recent research on DNN-based collision avoidance for autonomous aircraft [54], [55], marine vehicles [85] and robots [28]. This running example involves the development of a collision-limitation controller for a mobile robot travelling between locations A and B, e.g., for infrastructure inspection, or to carry goods in a warehouse (Fig. 2). Within this environment, the robot may encounter and potentially collide with another moving agent. We assume that collisions are not catastrophic, but should be limited to reduce robot damage and delays. As such, the robot uses DNN perception at each waypoint, to assess if it is on a collision course. Based on the DNN output, its controller should decide if the robot will proceed to the next waypoint or needs to wait for a while at its current waypoint such that the following system-level requirements are achieved:

- The robot journey is completed without any collision with a probability of at least 0.9 (a *constraint*);
- An optimal trade-off is achieved between maximising the probability of completing the journey without collisions and minimising the duration of the journey (two conflicting *optimisation objectives*).

III. Preliminaries

A. Discrete-Event Controllers

Many computer and cyber-physical systems are used in applications in which they need to respond to events that occur at discrete points in time. Examples of such events include the arrival of a user request or reaching a predefined workload level for a web server, and encountering an obstacle or arriving at a waypoint for a mobile robot. More often than not, the systems can react to these events by selecting one of several possible

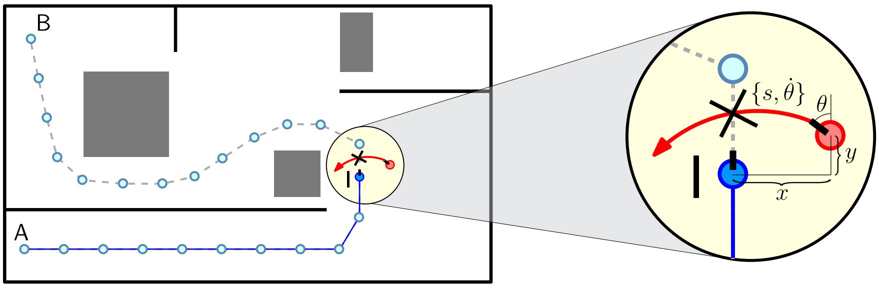


Fig. 2. Collision limitation for a mobile robot tasked with traversing a known environment through the use of waypoints. A mobile robot (darker blue) travelling between locations A and B may collide with another mobile agent (red) when moving from its current waypoint I to the next. A two-class DNN predicts 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.

responses. For instance, a web server may choose to process a user request with high or low priority, and the mobile robot may choose between several alternative routes around an obstacle. When this is the case, the selection of a suitable response is typically made by a system component termed a *discrete-event controller*.

Often implemented as software components, discrete-event controllers guide the way in which computer and cyber-physical systems respond to events, so that these systems meet their functional and non-functional requirements. Given their key role in many important systems, the formalisation, analysis and synthesis of these controllers—typically using state-transition models such as automata [76], Petri Nets [67] or Markov models [6], [35]—have received significant attention from the research community.

The synthesis of discrete-event controllers presented in this paper uses parametric discrete-event Markov chains. In this modelling paradigm, each combination of values for the parameters of a Markov chain corresponds to a different controller variant, and the controller synthesis problem involves determining the parameter value combinations that satisfy a set of system-level requirements. As such, we use the term *controller parameters* to refer to these parameters, which represent transition probabilities for the Markov chain, as well as the probabilities with which the controller selects between the possible system responses to events.

Before providing the required background on discrete-event Markov chains in the next section, we note that the discrete-event control that is the focus of our work differs fundamentally from continuous control techniques such as PID (proportional-integral-derivative) control [52], which involves the continuous adjustment of a control variable based on the error between a measured system parameter (e.g., web server workload, or room temperature) and its desired value, so that this value is maintained by the system.

B. Discrete-Event Markov Chains

DeepDECS uses discrete-event Markov chains (DTMCs), an established modelling paradigm for discrete-event controllers (e.g. [2], [29], [34], [61]), to capture the uncertainty affecting an autonomous system and its environment. DTMCs are finite state transition systems used to model the stochastic behaviour and to analyse the reliability, performance and other key properties of a wide range of real-world systems. DTMC states correspond to system configurations that are relevant for the properties under analysis, and are labelled with atomic propositions that hold in those states. State transitions model all possible transitions between states, and are annotated with probabilities. Finally, to allow the analysis of a broader set of properties, DTMC states and transitions can be annotated with nonnegative values termed *rewards*. These values are interpreted as “costs” (e.g., energy used by a robot) or “gains” (e.g. requests processed by a web server).

Definition 1: A reward-augmented discrete-time Markov chain over a set of atomic propositions AP is a tuple

$$\mathcal{M} = (S, s_0, P, L, R), \tag{1}$$

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 nonnegative values with the DTMC states (ρ) and transitions (ι):

$$\rho : S \to \mathbb{R}_{\geq 0}, \quad \iota : S \times S \to \mathbb{R}_{\geq 0}. \tag{2}$$

We note that the use of a set of reward structures R in (1) supports the analysis of multiple properties of the system modelled by the DTMC (e.g., resource use, time and risk). Furthermore, the association of rewards with both states and transitions enables the specification of different values for these properties both when in a state and when transitioning between states; as an example, a mobile robot may use energy e_1 while in a state s_1, energy e_{12} to transition to another state s_2, and then energy e_2 while in the new state s_2.

DeepDECS models the possible variants (i.e., the *design space*) of an AS controller as a reward-augmented *parametric* discrete-time Markov chain (pDTMC) [23].

Definition 2: A parametric discrete-time Markov chain is a DTMC (1) comprising transition probabilities and/or reward function values specified as rational functions over a set of continuous variables $\{x, y, \ldots\}$, i.e., functions that can be written as fractions whose numerators and denominators are polynomial expressions, e.g., $(1-x)/y$.

C. DTMC Modelling Language

We specify the discrete-time Markov models used by DeepDECS in the high-level modelling language of the PRISM model checker [62]. In this language, a system is modelled 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 commands that change these variables and have the generic form:

$$\begin{aligned}[][action]\ guard \rightarrow e_1 \!:\! update_1 +\ e_2 \!:\! update_2 + \ldots \\ + e_m \!:\! update_m; \end{aligned} \qquad (3)$$

where *guard* is a boolean expression over the variables of all modules, and e_i, $i \in [m]$, is an arithmetic expression that can only take values in the interval $[0,1]$ and is defined over the same variables such that $\sum_{i=1}^{m} e_i = 1$. If *guard* evaluates to true, the value of 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* must *synchronise* by performing one of these commands simultaneously.

D. Probabilistic Computation Tree Logic

DeepDECS uses *probabilistic computation tree logic* (PCTL) [8], [41] extended with rewards [1] to formalise AS requirements. Reward-augmented PCTL supports the specification of constraints such as 'the robot should not incur a collision until done with its mission with probability at least 0.9' ($\mathcal{P}[\neg\mathsf{collision}\ \mathrm{U}\ \mathsf{done}] \geq 0.9$) and optimisation objectives such as 'minimise the expected mission time' (minimise $\mathcal{R}^{\mathsf{time}}[\mathrm{F}\ done]$).

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 rwd reward structure (2) are defined by the grammar:

$$\begin{aligned} \Phi &::= true \mid \alpha \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}[\Psi] \sim p \\ \Psi &::= X\Phi \mid \Phi\ \mathrm{U}\ \Phi \mid \Phi\ \mathrm{U}^{\leq k}\ \Phi \\ \Phi_R &::= \mathcal{R}^{\mathsf{rwd}}[\mathrm{C}^{\leq k}] \sim r \mid \mathcal{R}^{\mathsf{rwd}}[\mathrm{F}\ \Phi] \sim r \end{aligned} \qquad (4)$$

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 [1], [8], [41] is defined using a satisfaction relation $\models$ over the states of a DTMC (1). Given a state s of this 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: $\mathcal{P}[\Psi] \sim p$ specifies that the probability that paths starting at state s satisfy a path property Ψ is $\sim p$; $\mathcal{R}^{\mathsf{rwd}}[C^{\leq k}] \sim r$ holds if the expected cumulated reward up to time-step k is $\sim r$; and $\mathcal{R}^{\mathsf{rwd}}[F\Phi] \sim r$ holds if the expected reward cumulated before reaching a state satisfying Φ is $\sim r$.

Removing '$\sim p$' (or '$\sim r$') from (4) 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 (using the established DTMC analysis algorithms implemented by model checking tools such as PRISM [62] and Storm [25]) for the initial state s_0 of $\mathcal{M}$.

E. Verification of DNN Classifiers

A K-class DNN classifier f is a function that maps a d-dimensional input to a *class* from the set $[K] = \{1, 2, \ldots, K\}$:

$$f : \mathbb{R}^d \rightarrow [K]. \qquad (5)$$

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. Each DNN verification technique used by DeepDECS (see Fig. 1) has the general form

$$verif : (\mathbb{R}^d \rightarrow [K]) \times \mathbb{R}^d \rightarrow \mathbb{B}, \qquad (6)$$

where $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$, such that, for a classifier $f \in \mathbb{R}^d \rightarrow [K]$ and an input $x \in \mathbb{R}^d$, $verif(f, x) = \mathsf{true}$ if the technique *deems* the DNN f likely to classify the input x correctly, and $verif(f, x) = \mathsf{false}$ otherwise. Two existing DNN verification techniques that can be used with DeepDECS are described below. However, DeepDECS can use any DNN verifier provided that it is fast enough to be used at runtime. The two verifiers described here were chosen because they verify two qualitatively different properties of DNNs and represent the state-of-the-art for these respective properties. For a list of other verifiers that could be used in conjunction with DeepDECS, the International Verification of Neural Networks competition [9], [68] conducted every year since 2020 serves as a useful resource.

1) Minimum Confidence Threshold: Given an input $x \in \mathbb{R}^d$, a K-class DNN classifier (5) operates by first computing a discrete probability distribution $\delta(x) = (p_1, p_2, \ldots, p_K)$ over the K classes, and then outputting the class $\mathrm{argmax}_{i=1}^{K} > p_i$ as its prediction. As $\delta(x)$ is typically a poor estimate of the true probability distribution of x, the temperature scaling mechanism introduced by Guo et al. [40] (and implemented by Kueppers et al. [60]) can be used to calibrate the DNN, allowing the definition of the following DNN verification method:

$$verif_1(f, x) = \begin{cases} \mathsf{true}, & \text{if } \max_{i=1}^{K} p_i \geq \tau \\ \mathsf{false}, & \text{otherwise} \end{cases} \qquad (7)$$

where τ is a threshold that we set to 0.8 in our experiments described later in the paper. The intuition behind this verification method is that if a calibrated DNN is very confident in its prediction, then the prediction is likely to be correct.

2) Local Robustness Certification: A DNN classifier (5) 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. A second verification technique can be defined using the GloRo Net framework of Leino et al. [63]. Given a DNN, this framework augments it with a local-robustness output by computing the Lipschitz constant of the function denoted by the original DNN and using it to verify local robustness. Adding this GloRo Net layer to the perception DNN of an AS, we can define the following verification method for the augmented DNN:

$$verif_2(f,x) = \begin{cases} \text{true}, & \text{if } \forall x' \in \mathbb{R}^d.\ ||x - x'||_2 \leq \epsilon \\ & \qquad\qquad \implies f(x) = f(x') \\ \text{false}, & \text{otherwise} \end{cases} \tag{8}$$

We use $\epsilon = 0.05$ for the experiments presented later in the paper. The intuition behind this verification method is that if a DNN is verified as locally robust at an input, i.e., a small change in the input does not change the classification output, then the prediction is likely to be correct.

IV. Theoretical Foundation

We model the design space of an AS controller under development as a pDTMC. The uncertainty introduced by the deep-learning perception and the one inherent to the environment are modelled by the probabilities of transition between the states of this pDTMC. The controller synthesis problem involves finding combinations of parameter values for which the pDTMC 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 PCTL formulae.

As shown in Fig. 1, DeepDECS derives the pDTMC underpinning its controller synthesis automatically from:

1) DNN verification results that quantify the uncertainty introduced by the DNN perception;
2) an "ideal" pDTMC that models the AS behaviour assuming perfect perception.

Given this pDTMC, Pareto-optimal DeepDECS controllers are then synthesised by applying a combination of probabilistic model checking and search techniques to this pDTMC.

As illustrated in Fig. 3, the cyber-physical components of an autonomous system managed by a DeepDECS controller (e.g., the software and hardware components involved in stopping the robot from our running example in Section II at a waypoint) monitor their environment (e.g., the surroundings of the robot from our running example and any nearby moving agents) 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 techniques 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. Thus, a DeepDECS controller operates by reacting to changes in the system, in the DNN outputs and, unique to DeepDECS, in the results of the online verification of the DNN classification.

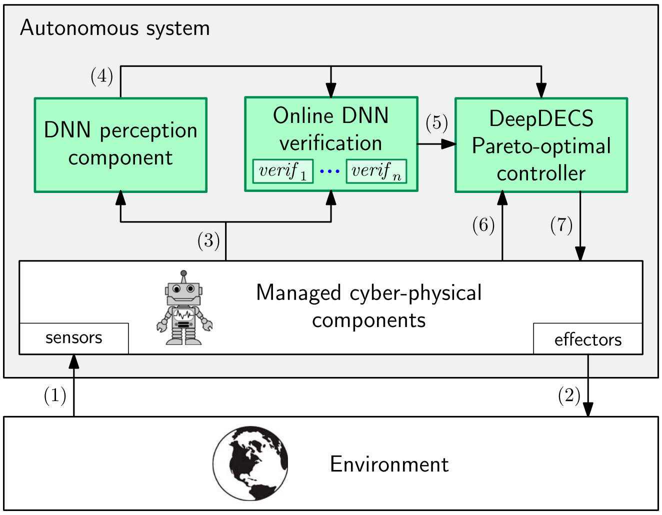


Fig. 3. DeepDECS controller deployment.

We detail the DeepDECS stages in the rest of this section.

A. Stage 1: DNN Uncertainty Quantification

DNN perception introduces aleatory uncertainty since DNNs cannot classify all inputs accurately. To quantify this uncertainty, DeepDECS uses $n \geq 0$ DNN verification techniques $verif_1$, $verif_2$,..., $verif_n$, and a test dataset $X \subset \mathbb{R}^d$ that represents a *statistically representative sample* of the inputs that the AS will encounter in operation. We note that X is one and the same with the testing dataset used in the established supervised machine learning practice [21], [53], [80]. The n verification techniques are used to partition X into 2^n subsets comprising inputs x with the same verification results. We note that using only a few verification techniques (e.g., $n \leq 3$) yields a small number of such subsets. Formally, given a DNN f, DeepDECS constructs the subset

$$X_v = \{x \in X \mid verif(f,x) = v\} \tag{9}$$

for every $v = (v_1, v_2, \ldots, v_n) \in \mathbb{B}^n$, where $verif(f,x) = (verif_1(f,x), verif_2(f,x), \ldots, verif_n(f,x))$. We use each subset (9) to obtain 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 $\mathcal{C}_v$ represents the number of class-k inputs from X_v that the DNN classifies as belonging to class k':

$$\mathcal{C}_v[k,k'] = \#\left\{x \in X_v \mid f^*(x) = k \wedge f(x) = k'\right\}, \tag{10}$$

where $f^*(x)$ represents the true class of x and, for any set A, $\#A$ denotes its cardinality.

As X is representative of the DNN inputs that the AS encounters in operation, we henceforth assume that the probability that a class-k input x is classified as belonging to class k' when it satisfies $\mathit{verif}(f,x)=v$ is given by:

$$\begin{aligned} p_{kk'v} &= Pr\left(f(x)=k' \wedge \mathit{verif}(f,x)=v \mid f^*(x)=k\right) \\ &= \frac{C_v[k,k']}{\sum_{v'\in\mathbb{B}^n}\sum_{k''\in[K]} C_{v'}[k,k'']} \\ &= \frac{C_v[k,k']}{\#\{x\in X \mid f^*(x)=k\}}. \end{aligned} \tag{11}$$

Formally, the law of large numbers [39, Ch. 8] implies that this result holds as $\#X\rightarrow\infty$. We note that

$$\sum_{(k',v)'\in[K]\times\mathbb{B}^n} p_{kk'v}=1. \tag{12}$$

We note that DeepDECS works seamlessly in the degenerate case when $n=0$, i.e., when no DNN verification technique is used. In this case, we have $v=()$ and $X_{()}=X$ in (9), $\mathcal{C}_{()}$ from (10) is simply the standard confusion matrix for the DNN, and the DeepDECS controller deployment from Fig. 3 uses no online DNN verification. This DeepDECS variant is useful (and thus included in the paper) because (i) it incurs no runtime DNN verification overheads, (ii) it requires no additional system coding/modifications, and (iii) it can be used when the DNN inputs cannot be accessed for the deployed system (e.g., due to licensing constraints).

Example 1: Consider the collision-prediction DNN $f_{\mathsf{collision}}$ used by the mobile robot from our running example in Section II, where, for any DNN input x, $f_{\mathsf{collision}}(x)=1$ predicts that the robot is on collision course, and $f_{\mathsf{collision}}(x)=2$ predicts that the robot is not on collision course. Suppose that a representative 49000-sample[2] test dataset X and the DNN verification technique verif_1 from (7) are used to quantify the uncertainty of $f_{\mathsf{collision}}$. Further assume that $X_{\mathsf{true}}=\{x\in X \mid \mathit{verif}_1(f_{\mathsf{collision}},x)=\mathsf{true}\}$ and $X_{\mathsf{false}}=\{x\in X \mid \mathit{verif}_1(f_{\mathsf{collision}},x)=\mathsf{false}\}$ contain 28843 and 20157 of the data samples from X, respectively (i.e., 28843 of the 49000 data samples from X are "verified"). In this DeepDECS stage, separate confusion matrices $\mathcal{C}_{\mathsf{true}}$ and $\mathcal{C}_{\mathsf{false}}$ are obtained for the test data samples from X_{true} and X_{false}, respectively. Assuming that the two matrices are

$$\mathcal{C}_{\mathsf{true}}=\begin{bmatrix}2302 & 180\\ 1266 & 25095\end{bmatrix} \text{ and } \mathcal{C}_{\mathsf{false}}=\begin{bmatrix}2786 & 353\\ 8447 & 8571\end{bmatrix},$$

the first DeepDECS stage terminates by using (11) to calculate the probabilities $p_{kk'v}$ that a class-k input x that satisfies $\mathit{verif}_1(f_{\mathsf{collision}},x)=v$ is classified as belonging to class k':

$$\begin{aligned} p_{11true} &= \frac{2302}{2482}=.93,\ p_{12true}=\frac{242}{2482}=.07,\\ p_{21true} &= \frac{1266}{26361}=.05,\ p_{22true}=\frac{25095}{26361}=.95,\\ p_{11false} &= \frac{2786}{3139}=.89,\ p_{12false}=\frac{353}{3139}=.11,\\ p_{21false} &= \frac{8447}{17018}=.50,\ p_{22false}=\frac{8571}{17018}=.50 \end{aligned}$$

B. Stage 2: Model Augmentation

1) Controller Synthesis Problem: DeepDECS organises each state s of the perfect-perception pDTMC model from Fig. 1 into a tuple

$$s=(z,k,t,c), \tag{13}$$

where $z\in Z$ models the 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 (i) partitions the state set into states in which the system can change ($t=1$), states in which the environment is observed ($t=2$) and states in which it is the controller's "turn" to act ($t=3$); and (ii) forces the pDTMC to visit these three types of states in order:

$$\begin{aligned} &\forall s=(z,k,t,c), s'=(z',k',t',c')\in S:\\ &\quad((t=1\wedge P(s,s')>0) \implies k'=k\wedge c'=c\wedge t'<3)\\ &\quad\wedge((t=2\wedge P(s,s')>0) \implies z'=z\wedge c'=c\wedge t'=3)\\ &\quad\wedge((t=3\wedge P(s,s')>0) \implies z'=z\wedge k'=k\wedge t'=1). \end{aligned} \tag{14}$$

We note that the use of a "turn" flag to distinguish between different types of model states is common in the research on discrete controller synthesis using probabilistic models (e.g., [20], [81]), which we drew inspiration from for the DeepDECS state partitioning. The outgoing transition probabilities from states $(z,k,3,c)\in S$ are controller parameters to be determined. We refer to them using the notation:

$$x_{zkcc'}=P((z,k,3,c),(z,k,1,c')) \tag{15}$$

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

We note that a perfect-perception Markov chain model is just a regular Markov chain model of a system with stochastic behaviour. As such, established practices can be used to obtain this model, e.g., through automatic generation from software artefacts such as activity diagrams [12], [30], or by following the tutorials and examples from the numerous case studies on the PRISM website https://prismmodelchecker.org/. In our experience, an engineer familiar with probabilistic model checking can develop one of these models within a few hours for a small to medium-sized system.

Example 2: We developed a perfect-perception pDTMC model for the mobile robot from our running example in the PRISM modelling language (Fig. 4(a)). This pDTMC models the logic underpinning the operation of the robot at a generic intermediate waypoint I from Fig. 2. The model states are tuples

$$(z,k,t,\mathit{wait})\in\{0,1,\ldots,4\}\times[2]\times[3]\times\mathbb{B} \tag{16}$$

with the semantics from (13), where $k=1$ and $k=2$ correspond to the mobile robot being on a collision course, and not being on a collision course, respectively.

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

[2] All numbers used in the example are actual values from our evaluation of DeepDECS for the mobile robot collision avoidance application (see also Section V).

```
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:on collision course (occ), 2:not occ
  [monitor] t=2 → p_occ:(k'=1) + (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;
  [decide]   t=3 ∧ k=1 → x_1:(wait'=true) + (1-x_1):(wait'=false);
  [decide]   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=1 : 2.57;
  [retry]    true : 5;
endrewards

label "collision" = z=3 & k=1;
label "done" = z=4;
```

(a) Perfect-perception pDTMC model of the mobile robot journey between two successive waypoints. The reward structure models the time taken by each mobile robot actions: 9.95 time units are required (on average) to travel between adjacent waypoints (without collision), 2.57 additional time units are required when the robot decides to proceed despite being on a collision course, and five time units are required 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 to the next waypoint.

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

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

(b) 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 = true) and "unverified" inputs (v_1 = 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.

Fig. 4. DeepDECS pDTMC models from the robot collision limitation application.

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 a collision is predicted ($k = 1$) and with probability x_2 if no 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.

Given a pDTMC with the characteristics described earlier, the *controller synthesis problem for the perfect-perception AS* is to find the combinations of values for parameters (15) for which the pDTMC satisfies $n_1 \geq 0$ PCTL-encoded *constraints*

$$C_i ::= \mathit{prop}_i \sim_i \mathit{bound}_i, \tag{17}$$

and achieves optimal trade-offs among $n_2 \geq 1$ PCTL-encoded *optimisation objectives*

$$O_j ::= \text{minimise } \mathit{prop}_{n_1+j} \mid \text{maximise } \mathit{prop}_{n_1+j} \tag{18}$$

where prop_1 to $\mathit{prop}_{n_1+n_2}$ are PCTL-encoded AS properties, $\sim_i \in \{<, \leq, \geq, >\}$, $\mathit{bound}_i \geq 0$, $i \in [n_1]$ and $j \in [n_2]$. The constraints (17) and optimisation objectives (18) taken together represent the *system-level requirements* considered by DeepDECS.

Example 3: The controller synthesis problem for the perfect-perception variant of the mobile robot from our running example involves finding the combinations of values for the parameters x_1 and x_2 of the pDTMC from Fig. 4(a) that ensure a collision-free journey with probability of at least 0.9:

$$C_1: \qquad \mathcal{P}[\neg\mathsf{collision}\ \mathsf{U}\ \mathsf{done}] \geq 0.9 \tag{19}$$

and an optimal trade-off between maximising this probability and minimising the travel time:

$$\begin{aligned} O_1 &: \ \text{maximise } \mathcal{P}[\neg\mathsf{collision}\ \mathsf{U}\ \mathsf{done}] \\ O_2 &: \ \text{minimise } \mathcal{R}^{\mathsf{time}}[\mathsf{F}\ \mathsf{done}] \end{aligned} \tag{20}$$

2) pDTMC Augmentation: The controller of an AS with deep-learning perception cannot access the true environment state k from (13). Instead, DeepDECS controllers need to operate with an estimate $\hat{k} \in [K]$ of k, and with the results $v = (v_1, v_2, \ldots, v_n) \in \mathbb{B}^n$ of the n verification techniques (6) applied to the DNN and its input that produced the estimate $\hat{k}$. The states $\hat{s}$ of a DeepDECS *DNN-perception pDTMC model*,

$$\hat{\mathcal{M}} = (\hat{S}, \hat{s_0}, \hat{P}, \hat{L}, \hat{R}) \tag{21}$$

are tuples that extend (13) with $\hat{k}$ and v:

$$\hat{s} = (z, k, \hat{k}, v, t, c). \tag{22}$$

To provide a formal definition for the derivation of this 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 $\hat{s} \in Z \times [K]^2 \times \mathbb{B}^n \times [3] \times C$. With this notation, the elements of $\hat{\mathcal{M}}$ are obtained from the perfect-perception pDTMC $\mathcal{M} = (S, s_0, P, L, R)$ of the AS and the probabilities (11) as follows:

$$\hat{S} = \{\hat{s} \in Z \times [K]^2 \times \mathbb{B}^n \times [3] \times C \mid s(\hat{s}) \in S\}; \tag{23}$$

$$\hat{s}_0 = (z_0, k_0, k_0, \mathsf{true}, \ldots, \mathsf{true}, t_0, c_0), \tag{24}$$

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

$$\hat{P}(\hat{s}, \hat{s}') = \begin{cases} P(s(\hat{s}), s(\hat{s}')), & \text{if } t = 1 \wedge (\hat{k}', v') = (\hat{k}, v) \\ P(s(\hat{s}), s(\hat{s}')) \cdot p_{k'\hat{k}'v'}, & \text{if } t = 2 \\ x_{z\hat{k}vcc'}, & \text{if } t = 3 \wedge (z', k', \hat{k}', v', t') \\ & \qquad = (z, k, \hat{k}, v, 1) \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

where

$$x_{z\hat{k}vcc'} = \hat{P}((z, k, \hat{k}, v, 3, c), (z, k, \hat{k}, v, 1, c')) \tag{26}$$

are controller parameters 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 $\hat{s} \in \hat{S}$,

$$\hat{L}(\hat{s}) = L(s(\hat{s})), \tag{27}$$

and

$$\begin{aligned} \hat{R} = \{(\hat{\rho}, \hat{\iota}) \in (\hat{S} \to \mathbb{R}_{\geq 0}) \times (\hat{S} \times \hat{S} \to \mathbb{R}_{\geq 0}) \mid \\ \exists (\rho, \iota) \in R : \big(\forall \hat{s} \in \hat{S} : \hat{\rho}(\hat{s}) = \rho(s(\hat{s}))\big) \\ \wedge \big(\forall \hat{s}, \hat{s}' \in \hat{S} : \hat{\iota}(\hat{s}, \hat{s}') = \iota(s(\hat{s}), s(\hat{s}'))\big)\}. \end{aligned} \tag{28}$$

An alternative encoding of the controller design space using a partially observable Markov decision process (POMDP) is possible [17], [18]. However, we opted for the pDTMC formalisation because current POMDP-enabled probabilistic model checkers [42], [69] do not support policy synthesis for combinations of requirements as complex as (17), (18).

Example 4: Fig. 4(b) illustrates the PRISM-encoded DNN-perception pDTMC model obtained for the mobile robot from our running example when a single (generic) verification method is used in the DNN uncertainty quantification stage of DeepDECS.

We end this section with a series of theorems that demonstrate the correctness, and show several key properties of our approach. First, Theorem 1 demonstrates that the pDTMC augmentation method from Section IV-B2 yields – by construction – a valid pDTMC (21) in which the controller actions are oblivious of the true environment state k, which is unknown to the controller. Second, Theorem 2 shows that the set $\mathit{Ctrl}_{\mathsf{DNN}}$ of DNN-perception controllers for an autonomous system is included in the set $\mathit{Ctrl}_{\mathsf{perf}}$ of perfect-perception controllers for that system (i.e., $\mathit{Ctrl}_{\mathsf{DNN}} \subset \mathit{Ctrl}_{\mathsf{perf}}$). This is a relevant property, as it provides a bound for what can be achieved using DNN-perception controllers; in particular, if no perfect-perception controller is adequate for the needs of an application, there is no point in seeking a DNN-perception controller that meets those needs, as no such controller exists. Next, Theorem 3 demonstrates that the set of DNN-perception controllers $\mathit{Ctrl}_{\mathsf{perf}}$ contains controllers not present in the DNN-perception controller set $\mathit{Ctrl}_{\mathsf{DNN}}$ (i.e., $\mathit{Ctrl}_{\mathsf{perf}} \setminus \mathit{Ctrl}_{\mathsf{DNN}} \neq \emptyset$). This property is relevant because it shows that one cannot use DNN-perception controllers to match the performance of every single perfect-perception controller. Finally, Theorem 4 shows that increasing the number of DNN verification techniques used by DeepDECS is never detrimental, and may yield a better set of DNN-perception controllers. As further detailed in the discussion provided after Theorem 4, this last property is relevant as it confirms that using additional DNN verification techniques may produce better controllers.

The following result shows that the DeepDECS module augmentation produces a valid pDTMC in which the probabilities of control-parameter changes are independent of the true environment state k.

Theorem 1: The tuple (21) with the elements defined by (23)–(28) is a valid pDTMC that satisfies:

$$\begin{aligned}&\forall \hat{s}=(z,k,\hat{k},v,t,c), \hat{s}'=(z',k',\hat{k}',v',t',c')\in\hat{S}:\\&\quad(c'\neq c\wedge \hat{P}(\hat{s},\hat{s}')>0)\implies\big((z',k',\hat{k}',v')=(z,k,\hat{k},v)\\&\qquad\wedge\ t=3\wedge t'=1\wedge\forall k''\in[K]:\hat{P}((z,k'',\hat{k},v,t,c),\\&\qquad\quad(z,k'',\hat{k},v,t',c'))=\hat{P}(\hat{s},\hat{s}')\big).\end{aligned}\tag{29}$$

Proof: To demonstrate that (21) 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 the following variant of (14) (which is required for the subsequent proofs):

$$\begin{aligned}&\forall \hat{s}=(z,k,\hat{k},v,t,c), \hat{s}'=(z',k',\hat{k}',v',t',c')\in\hat{S}:\\&\big((t{=}1\wedge P(\hat{s},\hat{s}'){>}0)\\&\qquad\qquad\implies((k',\hat{k}',v',c'){=}(k,\hat{k},v,c)\wedge t'{<}3)\big)\\&\wedge\big((t{=}2\wedge P(\hat{s},\hat{s}'){>}0)\\&\qquad\qquad\implies((z',c'){=}(z,c)\wedge t'{=}3)\big)\\&\wedge\big((t{=}3\wedge P(\hat{s},\hat{s}'){>}0)\\&\qquad\qquad\implies((z',k',\hat{k}',v'){=}(z,k,\hat{k},v)\wedge t'{=}1)\big)\end{aligned}\tag{30}$$

for each possible value of $t\in\{1,2,3\}$.

For $t=1$, (25) implies that

$$\sum_{\hat{s}'\in\hat{S}}\hat{P}(\hat{s},\hat{s}')=\sum_{\hat{s}'\in\hat{S}}P(s(\hat{s}),s(\hat{s}'))=1$$

because the last sum adds up all outgoing transition probabilities of state $s(\hat{s})$ from the perfect-perception pDTMC $\mathcal{M}$. Consider now any $\hat{s}'=(z',k',\hat{k}',v',t',c')\in\hat{S}$ such that $\hat{P}(\hat{s},\hat{s}')>0$. According to (25), this requires $\hat{k}'=\hat{k}\ \wedge\ v'=v$ when $t=1$. Additionally, since $\hat{P}(\hat{s},\hat{s}')=P((z,k,1,c),(z',k',t',c'))$, (14) implies that $k'=k\ \wedge\ c'=c\ \wedge\ t'<3$, as required by (30).

For $t=2$, we have

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

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

$$\hat{P}(\hat{s},\hat{s}')=P((z,k,2,c),(z',k',t',c'))\cdot p_{k'\hat{k}'v'},$$

(14) 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 (30) is explicitly stated in (25). To show now that (29) holds, we note that, according to definition (25), both transition probabilities from this relation (i.e., $\hat{P}(\hat{s},\hat{s}')$ and $\hat{P}((z,k'',\hat{k},v,t,c),(z,k'',\hat{k},v,t',c'))$) are equal to $x_{z\hat{k}vcc'}$. □

Proving the next results requires the following lemma.

Lemma 1: Let $\underline{x}$ and $\underline{\hat{x}}$ be valid instantiations of the perfect-perception controller parameters

$$\left\{x_{zkcc'}\in[0,1]\,\middle|\,(\exists k\in[K].(z,k,3,c)\in S)\ \wedge\ c'\in C\right\}$$

from (15) and of the DNN-perception controller parameters

$$\left\{x_{z\hat{k}vcc'}\in[0,1]\,\middle|\,(\exists k\in[K].(z,k,\hat{k},v,3,c)\in\hat{S})\ \wedge\ c'\in C\right\}$$

from (25), 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

$$pmc(\Phi,\hat{\mathcal{M}}_{\underline{\hat{x}}})=pmc(\Phi,\mathcal{M}_{\underline{x}}),\tag{31}$$

and

$$pmc(\Phi_R,\hat{\mathcal{M}}_{\underline{\hat{x}}})=pmc(\Phi_R,\mathcal{M}_{\underline{x}}),\tag{32}$$

for any (quantitative) PCTL state formula Φ and reward state formula Φ_R if and only if the elements of $\underline{x}$ and $\underline{\hat{x}}$ satisfy

$$x_{zkcc'}=\sum_{\hat{k}\in[K]}\sum_{v\in\mathbb{B}^n}p_{k\hat{k}v}x_{z\hat{k}vcc'}\tag{33}$$

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 (31) and (32) hold iff, for any path $\pi=s_0s_1s_2\ldots\in Paths^{\mathcal{M}_{\underline{x}}}(s_0)$, set of associated paths $\hat{\Pi}=\left\{\hat{s}_0\hat{s}_1\hat{s}_2\ldots\in Paths^{\hat{\mathcal{M}}_{\underline{\hat{x}}}}(\hat{s}_0)\,\middle|\,\forall i\geq 0\,.\,s(\hat{s}_i)=s_i\right\}$, and $i\geq 0$, the following property holds:

$$P(s_i,s_{i+1})=\sum_{\hat{s}_0\hat{s}_1\hat{s}_2\ldots\in\hat{\Pi}}\hat{P}(\hat{s}_i,\hat{s}_{i+1}).\tag{34}$$

This is required because, according to (27) and (28), 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 (28), 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 (31) (or for (32)).

Given the definition of $\hat{P}$ from (25), property (34) 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

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

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 (34) 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 (33) holds, which completes the proof. □

The next two theorems show that for each controller that satisfies constraints (17) and Pareto-optimises objectives (18) for the DNN-perception AS there is an equivalent controller for the perfect-perception AS, but the converse does not hold.

Theorem 2: For any AS requirements (17), (18) for which there exists a DNN-perception controller that satisfies the constraints (17), there exists also a perfect-perception controller that satisfies the same constraints and yields the same values for the PCTL properties from the optimisation objectives (18).

Proof: We prove this result by showing that the application of (33) 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

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

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

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

which completes the proof. □

Theorem 3: If a confusion matrix $\mathcal{C}_{v_0}$ from (10) satisfies $\mathcal{C}_{v_0}[k_1, k_0] > 0 \wedge \mathcal{C}_{v_0}[k_2, k_0] > 0$ for a combination of verification results $v_0 \in \mathbb{B}^n$, two classes $k_1 \neq k_2$ and a class k_0, then there is an infinite number of AS requirements (17), (18) for which there exists a perfect-perception controller that satisfies the constraints (17), and no DNN-perception controller exists that satisfies the constraints and yields the same values as the perfect-perception controller for the PCTL properties from the optimisation objectives (18).

Proof: Consider two perfect-perception controller parameters $x_{zk_1cc'}$ and $x_{zk_2cc'}$ corresponding to a configuration c' being selected by the controller when the environment state is k_1 and k_2, respectively. Since $\mathcal{C}_{v_0}[k_1, k_0] > 0 \wedge \mathcal{C}_{v_0}[k_2, k_0] > 0$, definition (11) implies that $p_{k_1k_0v_0} > 0 \wedge p_{k_2k_0v_0} > 0$, and we consider the infinite set of (probabilistic) perfect-perception controllers with $x_{zk_1cc'} = 1$ and $x_{zk_2cc'} \in [0, p_{k_2k_0v_0})$. For any such controller, consider the instantiation of equality (33) for $x_{zk_1cc'}$. The parameters of any equivalent DNN-perception controller that are multiplied by non-zero probabilities $p_{k\hat{k}v}$ on the right-hand side of this instantiation must have value 1, or otherwise the terms of the double sum from (33) will add up to a value below 1 and the equality cannot hold. In particular, we must have $x_{zk_0v_0cc'} = 1$ because this parameter is multiplied by $p_{k_1k_0v_0} > 0$. However, according to (33), this means that the DNN-perception controller can only be equivalent to a perfect-perception controller whose parameter $x_{zk_2cc'}$ satisfies

$$\begin{aligned}x_{zk_2cc'} &= \sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^n} p_{k_2\hat{k}v} x_{z\hat{k}vcc'} \geq p_{k_2k_0v_0} \cdot x_{zk_0v_0cc'} \\ &= p_{k_2k_0v_0} \cdot 1 = p_{k_2k_0v_0}.\end{aligned}$$

This inequality is not satisfied by any of the perfect-perception controllers from the infinite set we considered. As such, no equivalent DNN-perception controller exists for any of these perfect-perception controllers. □

Theorem 3 demonstrates that the decision-making capabilities of infinitely many perfect-perception controllers cannot be replicated by DNN-perception controllers (unless, exceptionally, applying the n DNN verification techniques resolves the uncertainty introduced by the DNN). Finally, the following result shows that increasing the number of DNN verification techniques is never detrimental and may yield better DeepDECS controllers.

Theorem 4: For any AS requirements (17), (18) and DNN-perception controller generated using n DNN verification techniques $\mathit{verif}_1, \mathit{verif}_2, \ldots, \mathit{verif}_n$ such that the constraints (17) are satisfied, the DeepDECS pDTMC obtained using any verification technique verif_{n+1} in addition to $\mathit{verif}_1, \mathit{verif}_2, \ldots, \mathit{verif}_n$ can be used to generate a controller that satisfies the constraints and yields the same values for the PCTL properties from the optimisation objectives.

Proof: Let $\hat{\mathcal{M}}^n$ and $\hat{\mathcal{M}}^{n+1}$ be the DNN-perception pDTMCs obtained using the DNN verification techniques $\mathit{verif}_1, \mathit{verif}_2, \ldots, \mathit{verif}_n$, and the DNN verification techniques $\mathit{verif}_1, \mathit{verif}_2, \ldots, \mathit{verif}_{n+1}$, respectively. Consider any instantiation $\underline{\hat{x}}^n$ of the controller parameters (25) for $\hat{\mathcal{M}}^n$ so that the constraints (17) are satisfied, and let $\underline{\hat{x}}^{n+1}$ be the instantiation of the controller parameters (25) for $\hat{\mathcal{M}}^{n+1}$ such that the elements of this instantiation satisfy

$$\begin{aligned}\underline{x}^{n+1}_{z\hat{k}(v_1,v_2,\ldots,v_n,\mathsf{false})cc'} &= \underline{x}^{n+1}_{z\hat{k}(v_1,v_2,\ldots,v_n,\mathsf{true})cc'} \\ &= \underline{x}^{n}_{z\hat{k}(v_1,v_2,\ldots,v_n)cc'}. \end{aligned} \tag{35}$$

We will show that the controller defined by $\underline{\hat{x}}^{n+1}$ over $\hat{\mathcal{M}}^{n+1}$ is equivalent to the controller defined by $\underline{\hat{x}}^n$ over $\hat{\mathcal{M}}^n$ (and therefore must satisfy the constraints (17) and yield the same values for the PCTL properties from the optimisation objectives (18)).

According to Lemma 1, the former controller is equivalent to the perfect-perception controller whose parameters (15) satisfy:

$$x_{zkc c'} = \sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^{n+1}} p_{k\hat{k}v} \underline{x}^{n+1}_{z\hat{k}vcc'}. \tag{36}$$

Taking into account (35), we obtain:

$$\begin{aligned} x_{zkcc'} =& \sum_{\hat{k}\in[K]} \sum_{(v_1,\ldots,v_n)\in\mathbb{B}^n} \Big(p_{k\hat{k}(v_1,\ldots,v_n,\mathsf{false})} \underline{x}^{n+1}_{z\hat{k}(v_1,\ldots,v_n,\mathsf{false})cc'} \\ &\qquad + p_{k\hat{k}(v_1,\ldots,v_n,\mathsf{true})} \underline{x}^{n+1}_{z\hat{k}(v_1,\ldots,v_n,\mathsf{true})cc'}\Big) \\ =& \sum_{\hat{k}\in[K]} \sum_{(v_1,\ldots,v_n)\in\mathbb{B}^n} \Big[\Big(p_{k\hat{k}(v_1,\ldots,v_n,\mathsf{false})} + p_{k\hat{k}(v_1,\ldots,v_n,\mathsf{true})}\Big) \\ &\qquad \cdot \underline{x}^{n}_{z\hat{k}(v_1,\ldots,v_n)cc'}\Big] = \sum_{\hat{k}\in[K]} \sum_{v\in\mathbb{B}^n} p_{k\hat{k}v} \underline{x}^{n}_{z\hat{k}vcc'} \end{aligned}$$

because, according to (11),

$$\begin{aligned} & p_{k\hat{k}(v_1,\ldots,v_n,\mathsf{false})} + p_{k\hat{k}(v_1,\ldots,v_n,\mathsf{true})} \\ &\quad = \frac{C_{(v_1,\ldots,v_n,\mathsf{false})}[k,k'] + C_{(v_1,\ldots,v_n,\mathsf{true})}[k,k']}{\sum_{v'\in\mathbb{B}^{n+1}} \sum_{k''\in[K]} C_{v'}[k,k'']} \\ &\quad = \frac{C_{(v_1,\ldots,v_n)}[k,k']}{\sum_{v'\in\mathbb{B}^{n}} \sum_{k''\in[K]} C_{v'}[k,k'']} = p_{k\hat{k}(v_1,\ldots,v_n)}. \end{aligned}$$

According again to Lemma 1, this result implies that the perfect-perception controller induced by (36) is also equivalent to the controller defined by $\underline{\hat{x}}^n$ over $\hat{\mathcal{M}}^n$. Hence, using (35) to select the parameters of the controller obtained for $n+1$ DNN verification technique yields a controller equivalent to that obtained using only the first n verification techniques. □

The result from Theorem 4 (i.e., that including any additional verification technique cannot yield worse DeepDECS controllers) may appear counterintuitive. To understand why this is the case, consider a poor verification technique which consistently tells that the DNN output is wrong when the DNN classifies its input correctly, and the other way around. If this happened, then the uncertainty quantification from the first DeepDECS stage would simply indicate to our stage 3 controller synthesis that the DNN should be trusted more when the result of this verification is ‘false’. This scenario is similar to a weather forecasting service that keeps predicting dry weather on rainy days (and the other way around): its regular users will know to take their umbrellas with them on days predicted to be dry, and to leave their umbrellas at home on days predicted to be rainy. Another type of “poor” verification technique is one that selects its true or false output completely randomly. For such a technique, the uncertainty quantification from the first DeepDECS stage will show that the technique provides no useful information, and therefore the DeepDECS controller synthesis will automatically ignore its output and only rely on the other verification techniques. Adding this technique will neither help nor harm the outcome of the DeepDECS synthesis. To return to our analogy, the regular users of a weather forecasting service that makes random predictions will learn to ignore that service, and to base their decisions on other services they have access to.

C. Stage 3: Controller Synthesis

The *controller synthesis problem for the DNN-perception system* involves finding instantiations for the controller parameters for which the pDTMC $\hat{\mathcal{M}}$ from (21) satisfies the constraints (17) and is Pareto optimal with respect to the optimisation objectives (18). Solving the general version of this problem precisely is unfeasible. However, metaheuristics such as multi-objective genetic algorithms for probabilistic model synthesis [14], [32] 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 AS with deterministic controllers and small numbers of parameters, or—by discretising the search space—an approximate Pareto-optimal controller set for AS with probabilistic controllers.

Example 5: Consider again the DNN-perception pDTMC model of the mobile robot from our running example (Fig. 4(b)). One option for searching its controller design space for parameter combinations $(x_{1\mathsf{false}}, x_{1\mathsf{true}}, x_{2\mathsf{false}}, x_{2\mathsf{true}}) \in [0,1]^4$ that satisfy the constraint (19) and achieve optimal trade-offs with respect to the optimisation objectives (20) is via discretising the four controller parameters, with each parameter varied between 0 and 1 with a step size of 0.1. The DNN-perception pDTMC instance for every parameter combination obtained in this way can then be analysed using the probabilistic model checker PRISM, so that the combinations which violate the constraint (19) are discarded, and the remaining combinations are used to assemble a Pareto-optimal set of controllers.

We demonstrate the synthesis of DeepDECS controllers through the use of both metaheuristics and exhaustive search in the next section.

V. Evaluation

A. Evaluation Methodology

We carried out experiments to answer the research questions (RQs) summarised below.

RQ1 (Uncertainty quantification effectiveness): *Are the DNN input subsets (9) “endorsed” by verification techniques a sizeable part of the inputs encountered in the operational design domain of autonomous systems?* Given a DNN classifier f, DeepDECS distinguishes between “endorsed” DNN inputs x, i.e., inputs for which $\mathit{verif}_i(f,x) = \mathsf{true}$ for one or more of the verification techniques verif_1, verif_2,..., verif_n, and inputs that do not have this property. The envisaged benefits of using DNN verification techniques in DeepDECS can only be achieved if the endorsed inputs are (i) associated with higher accuracy levels than unendorsed inputs, and (ii) encountered frequently by the autonomous system. By using established DNN verification techniques within DeepDECS, we know that prerequisite (i) is going to hold. Therefore, we assessed whether prerequisite (ii) is also met.

RQ2 (Controller synthesis effectiveness): *How do the controllers synthesised by DeepDECS compare to those obtained without using DeepDECS, and are they achieving better trade-offs between the optimisation objectives (18) when more DNN*

TABLE I
KEY CHARACTERISTICS OF THE APPLICATIONS USED FOR THE DEEPDECS EVALUATION

	Collision Limitation	**Driver-Attentiveness Management**
Application Domain	Infrastructure Inspection/Goods Transportation	Autonomous Driving
System type	Mobile robot	Embedded autonomous system
System-level properties†	Probability of collision-free journey Journey time	Risk level Driver nuisance
DNN perception component	Binary DNN classifier	Three-class DNN classifier
DNN training/testing data	Obtained through simulation	Obtained from user study with human drivers

†These are the properties that appear in the system-level requirements (i.e., constraints and optimisation objectives) guiding the DeepDECS controller synthesis.

verification techniques are used? For the first part of this research question, we compare DeepDECS to a baseline approach in which controllers are synthesised based solely on the output of the perception DNN used by an autonomous system. For the second part of this research question, Theorem 4 shows that using $n+1$ DNN verification techniques yields controllers at least as good as the controllers obtained using only n of those techniques, but does not guarantee that the former controllers are actually better than the latter. As such, we examined experimentally if using additional verification techniques produces better controllers.

RQ3 (Overheads): *What are the development-time and run-time computational overheads to synthesise DeepDECS controllers and to use online DNN verification techniques within an autonomous system, respectively?* We assessed the execution time for the synthesis of the DeepDECS Pareto-optimal controllers, and for the online verification of DNN inputs.

To answer these research questions, we used DeepDECS to synthesise discrete-event controllers for two autonomous systems from different application domains. First, we considered the autonomous mobile robot from our running example and the synthesis of its collision-limitation controller. Second, we used our method to synthesise an attentiveness-management controller for drivers of level 3 autonomous vehicles, i.e., vehicles whose drivers must retain situational awareness at all times, so that they can resume manual driving when needed. Table I summarises the significant differences between the key characteristics of these systems.

In each of the two case studies, we considered four DNN uncertainty quantification setups. These setups correspond to using every subset of the two DNN verification techniques from Section III-E in the uncertainty quantification stage of DeepDECS:

(i) no verification technique;
(ii) only $verif_1$ from (7);
(iii) only $verif_2$ from (8);
(iv) both $verif_1$ and $verif_2$.

We synthesised separate sets of Pareto-optimal DeepDECS controllers for each of these setups, as well as a fifth set of Pareto-optimal controllers corresponding to the perfect-perception variant of the autonomous system. Finally, we used the following Pareto front quality metrics to compare the five sets of Pareto optimal controllers, and to evaluate their quality:

1) *Inverted Generational Distance (IGD) [84]*, which measures the distance between the analysed Pareto front and a *reference frame* (e.g., the true Pareto front, the best known approximation of the true Parero front, or an “ideal” Pareto front) by calculating, for each point on the reference frame, the distance to the closest point on the Pareto front. The IGD measure for the front is then computed as the mean of these distances. Smaller IGD values indicate better Pareto fronts. The IGD values from our case studies were computed using the perfect-perception Pareto front as the reference frame.
2) *Hypervolume (HV) [87]*, which captures the proximity of the analysed Pareto front to a reference frame and the diversity of its points (where higher diversity is better) by measuring the volume (or area for two-dimensional Pareto fronts) delimited by these points and a reference point defined with respect to the reference framework. The HV values from our case studies were obtained using the perfect-perception Pareto front as the reference frame and, in line with common practice [86], its *nadir* (i.e., the point corresponding to the worst values for each optimisation objective) as the reference point.

To ease the application of DeepDECS and support its adoption, we implemented a Python software tool that automates the DeepDECS model augmentation process. The tool takes as input a perfect-perception pDTMC model and the confusion matrices (10) and outputs a DNN-perception pDTMC model. The tool is executed as

```
pythondeepDECSAugment.py perfect-pDTMC.pm
confusion_matrices.txt DNN-pDTMC.pm
```

where:

- `perfect-pDTMC.pm` is a file containing the perfect-perception pDTMC model;
- `confusion_matrices.txt` is a file containing the confusion matrix elements $\mathcal{C}_v[k,k']$, $v\in\mathbb{B}^n$, $k\in[K]$, $k'\in[K]$, from (10);
- `DNN-pDTMC.pm` is the name of the file in which the DNN-perception pDTMC model will be generated.

The two applications of DeepDECS are detailed in the remainder of this section. To enable the reproducibility of our results, the DeepDECS project website [24] contains all the data, models and code from our experiments, as well as the software tool that automates the model augmentation stage of DeepDECS, and a tutorial that provides step-by-step instructions for the use of our tool-supported controller synthesis method.

B. Application 1: Mobile-Robot Collision Limitation

Many of the steps from the DeepDECS synthesis of collision-limitation controllers for the mobile robot from our running example were already summarised in Examples 1 through 5 from Section IV. The additional details provided below complement the information presented in those examples.

1) DeepDECS Inputs: The four inputs required for the application of DeepDECS (see Fig. 1) were:

- a collision-prediction DNN trained using data from a simulator we implemented for the scenario in Fig. 2;
- a test dataset collected using our mobile robot simulator;
- a perfect-perception pDTMC model of the robot journey;
- controller requirements specifying the minimum acceptable probability of a collision-free journey, and demanding an optimal trade-off between maximising this probability and minimising the travel time.

Further details about these four inputs are provided below.

DNN perception component. The data for training the DNN and quantifying its uncertainty were obtained using the 2D particle simulator *Box2D* (https://box2d.org/), with the robot and collider simulated by circular particles of 0.5-unit radius. We ran simulations with the robot starting at the origin $(0, 0)$ with a heading of $\frac{\pi}{2}$ radians, and travelling in a straight line to a goal destination (x_{goal}, y_{goal}), with a speed of 1 unit/s. A journey was deemed completed when the robot reached a goal area defined by $(x_{goal} \pm \epsilon, y_{goal} \pm \epsilon)$ for a small $\epsilon > 0$. The robot advanced with an angular velocity

$$\dot{\theta}_r = \alpha \cdot \arctan\left(\frac{v_x \cdot y_{goal} - v_y \cdot x_{goal}}{v_x \cdot x_{goal} + v_y \cdot y_{goal}}\right)$$

where v_x and v_y are the horizontal and vertical velocities of the robot, respectively, and $\alpha > 0$ is a constant. When the difference between the robot's and the target heading exceeded $\frac{\pi}{36}$, the robot's linear speed was reduced to 0.1 unit/s, allowing it time to correct its course. The collider had a random initial position

$$(x,\ y,\ \theta) = (\mathcal{U}(-x_{lim}, x_{lim}),\ \mathcal{U}(0, y_{lim}),\ \mathcal{U}(-\pi, \pi))$$

where $\mathcal{U}$ is the uniform distribution function, and random linear and angular speeds given by

$$(s,\ \dot{\theta}_c) = (\mathcal{U}(0, s_{lim}),\ \mathcal{U}(-\dot{\theta}_{lim}, \dot{\theta}_{lim})).$$

The parameter values used for the experimental setup are: $\alpha = 0.5, x_{goal} = 0, y_{goal} = 10, \epsilon = 0.05, x_{lim} = 10, y_{lim} = 10,$ $s_{lim} = 2$ units/s, and $\dot{\theta}_{lim} = \frac{\pi}{4}$ rads/s. Each collected datapoint was a tuple

$$(x_{diff}, y_{diff}, s, \theta, \dot{\theta}_c, occ),$$

where x_{diff} and y_{diff} are the relative horizontal and vertical distances between the robot and the collider, and occ is the label specifying whether the two agents are on collision course ($occ = 2$) or not ($occ = 1$). The datapoints were normalised such that $x_{diff}, \theta, \dot{\theta}_c \in [-1, 1]$ and $y_{diff} \in [0, 1]$. Multiple simulations were performed to collect 10000 collision datapoints and 10000 no-collision datapoints, and the mean times to complete a journey between two successive waypoints with and without collision were recorded.

We used 80% of the collected datapoints to train a two-class DNN classifier with the architecture proposed by Ehlers [28]. This architecture comprises a fully-connected linear layer with 40 nodes, followed by a MaxPool layer with pool size 4 and stride size 1, a fully-connected ReLU layer with 19 nodes, and a final fully-connected ReLU layer with 2 nodes. The DNN was implemented and trained using TensorFlow in Python, with a cross-entropy loss function, the Adam optimisation algorithm [59], and the following hyperparameters: 100 epochs, batch size 128, and initial learning rate 0.005 set to decay to 0.0001.

Test dataset. We assembled the test dataset using the 20% of the datapoints collected from the mobile robot simulations that were not used for training the collision-prediction DNN.

Perfect-perception pDTMC model. See Example 2.

PCTL-encoded requirements. See Example 3.

2) DeepDECS Application

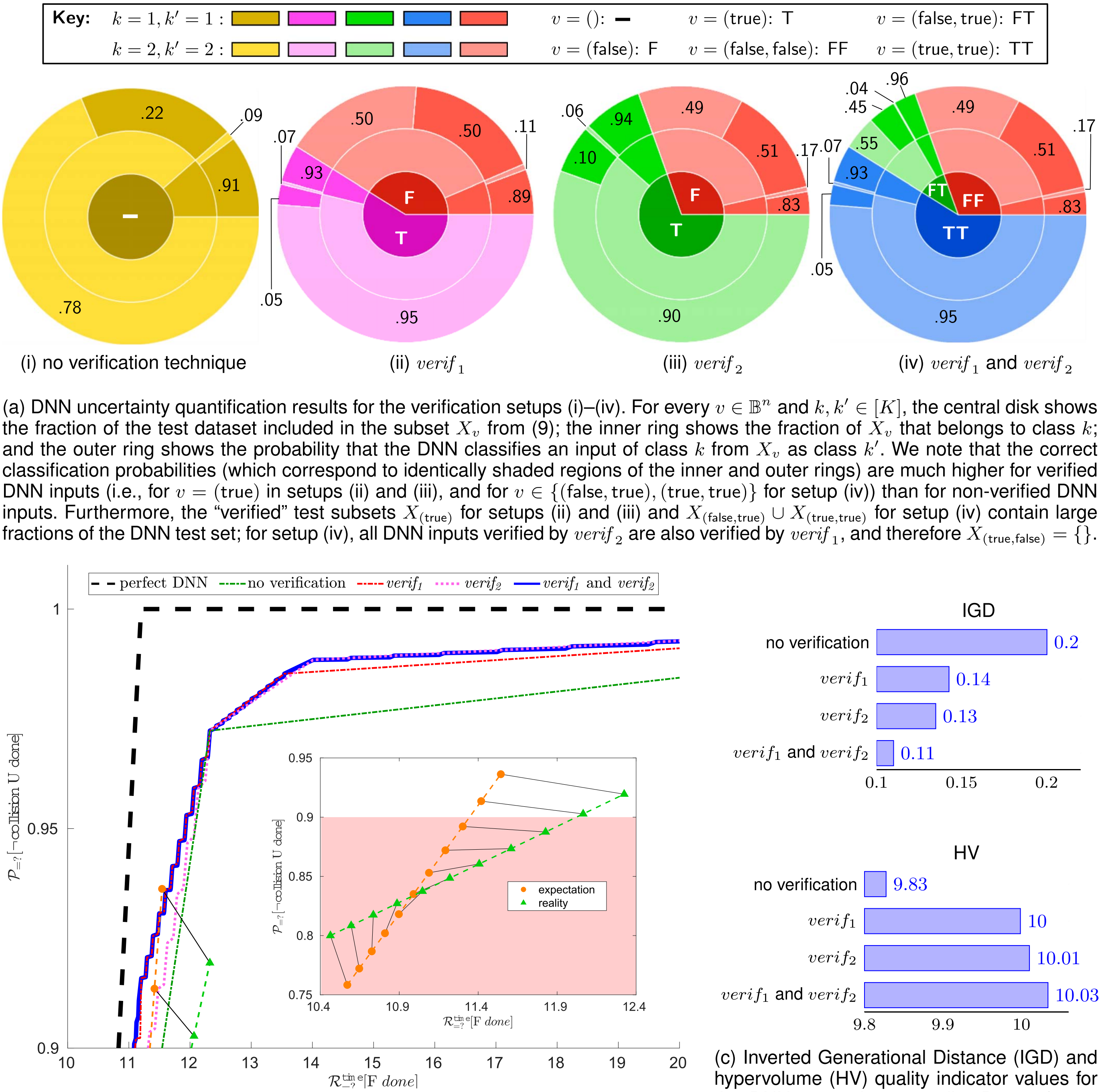
Stage 1: DNN uncertainty quantification. We obtained four sets of DNN uncertainty quantification probabilities (11) using the test dataset mentioned in Section V-B1 and each possible subset of DNN verification techniques from Section III-E. The probabilities of the DNN classifying class-k inputs associated with every verification result v as class k' are summarised in Fig. 5(a), which shows that "verified" classifications (i.e., those associated with $v = (\mathtt{true})$ for setups (ii) and (iii), and $v = (\mathtt{true}, \mathtt{true})$ for setup (iv)) are obtained for large percentages of DNN inputs, and have a much higher probability of being correct than "unverified" classifications.

Stage 2: Model augmentation. We used our model augmentation tool to obtain the DNN-perception pDTMC model for each combination of the verification methods (7) and (8) from the perfect-perception pDTMC and the DNN uncertainty quantification probabilities (11) obtained in stage 1. Example 4 presents one such DNN-perception model; the models generated for all four uncertainty quantification setups explored in our experiments are provided on the DeepDECS website [24].

Stage 3: Controller synthesis. The controller parameters synthesised by DeepDECS were the probabilities x_{1v} and x_{2v} for the robot to wait at its current waypoint when the DNN predicts it is on collision course (class 1) and not on collision course (class 2), respectively, where $v = ()$ for setup (i), $v \in \mathbb{B}$ for setups (ii) and (iii), and $v \in \mathbb{B}^2$ for setup (iv).

As mentioned in Example 5, the controller design space was explored via discretising the controller parameters x_{1v}, x_{2v}, with each parameter varied between 0 and 1 with a step size of 0.1. The DNN-perception pDTMC instance for every parameter combination obtained in this way was analysed using the probabilistic model checker PRISM.

The Pareto fronts for the controllers that satisfied constraint (19) for each setup are presented in Fig. 5(b), together with



(a) DNN uncertainty quantification results for the verification setups (i)–(iv). For every $v \in \mathbb{B}^n$ and $k, k' \in [K]$, the central disk shows the fraction of the test dataset included in the subset X_v from (9); the inner ring shows the fraction of X_v that belongs to class k; and the outer ring shows the probability that the DNN classifies an input of class k from X_v as class k'. We note that the correct classification probabilities (which correspond to identically shaded regions of the inner and outer rings) are much higher for verified DNN inputs (i.e., for $v = (\mathsf{true})$ in setups (ii) and (iii), and for $v \in \{(\mathsf{false}, \mathsf{true}), (\mathsf{true}, \mathsf{true})\}$ for setup (iv)) than for non-verified DNN inputs. Furthermore, the "verified" test subsets $X_{(\mathsf{true})}$ for setups (ii) and (iii) and $X_{(\mathsf{false},\mathsf{true})} \cup X_{(\mathsf{true},\mathsf{true})}$ for setup (iv) contain large fractions of the DNN test set; for setup (iv), all DNN inputs verified by $verif_2$ are also verified by $verif_1$, and therefore $X_{(\mathsf{true},\mathsf{false})} = \{\}$.

(b) Pareto fronts associated with the Pareto-optimal DeepDECS controllers; the inset plot shows the expected Pareto front and actual outcome for the baseline approach, with shading used to indicate the region disallowed by constraint (19).

(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 Pareto fronts that are better (i.e., closer to the ideal-perception Pareto front).

	perfect DNN	**no verification**	$verif_1$	$verif_2$	$verif_1$ **and** $verif_2$
Controller synthesis time	0.84s	0.85s	18.78s	18.76s	2998s
Online DNN verification time	–	–	0.2ms	50ms	50.2ms

(d) DeepDECS controller synthesis and mean online DNN verification times

Fig. 5. DeepDECS controller synthesis for the mobile robot collision limitation.

the Pareto front for the perfect-perception setup, which we analysed for comparison purposes. Expectedly, the best results are achieved in the perfect-perception setup, and the worst when no DNN verification technique is used. The use of verification methods yields Pareto fronts located closer to the perfect-perception Pareto front, with the best DNN-perception Pareto front obtained when both verification methods are used. These findings from the visual inspection of Fig. 5(b) are confirmed by the analysis (Fig. 5(c)) of the Pareto fronts using the two established Pareto-front quality indicators (IGD and HV) mentioned in Section V-A.

For comparison purposes, we also considered a baseline approach in which the Pareto-optimal controllers were synthesised based on the DNN outputs alone, i.e. by assuming these outputs to be always correct. We note that this is the only option when the aleatory uncertainty introduced by the DNN is not quantified as done in the first stage of DeepDECS. The *expected* Pareto front for this baseline approach is shown as an inset plot in Fig. 5(b). For each point on this expected Pareto front, the plot also shows the *actual* outcome delivered by the controller associated with that point. Because the DNN uncertainty is not considered by this baseline, the expected and actual outcomes are typically different. Furthermore, because the DNN accuracy is worse when the mobile robot is not on collision course than when it is on collision course (i.e., 78% versus 91%, cf. Fig. 5(a)(i)), the expected Pareto front is overly pessimistic, and only two of its points satisfy constraint (19).

The synthesis of these Pareto-optimal controller sets was performed on a HP Elitebook 840 G7 Laptop with i5 Intel 10th generation processor and 16GB memory, and the synthesis times are reported in Fig. 5(d), alongside the mean execution time for the online DNN verification technique(s) used by DeepDECS.

C. Application 2: Driver-Attentiveness Management

We used DeepDECS to design a proof-of-concept driver-attentiveness management system for shared-control autonomous cars. Developed as part of our SafeSCAD project [4], [13] and inspired by the first United Nations regulation on vehicles with Level 3 automation [83], this system uses (Fig. 6): (i) specialised sensors to monitor key car parameters (velocity, lane position, etc.) and driver's biometrics (eye movement, heart rate, etc.), (ii) a three-class DNN to predict the driver's response to a request to resume manual driving, and (iii) a deterministic controller to issue visual/acoustic/haptic alerts when the driver is insufficiently attentive.

1) DeepDECS Inputs: The four inputs required for the application of DeepDECS (see Fig. 1) were:

- an existing DNN trained and validated with driver data from a SafeSCAD user study performed within a driving simulator [73];
- a test dataset obtained also from the study in [73];
- a perfect-perception pDTMC model of the decision process used for driver-attentiveness management;
- controller requirements that place constraints on, and require the minimisation of, the journey risk and the driver nuisance caused by the use of alerts.

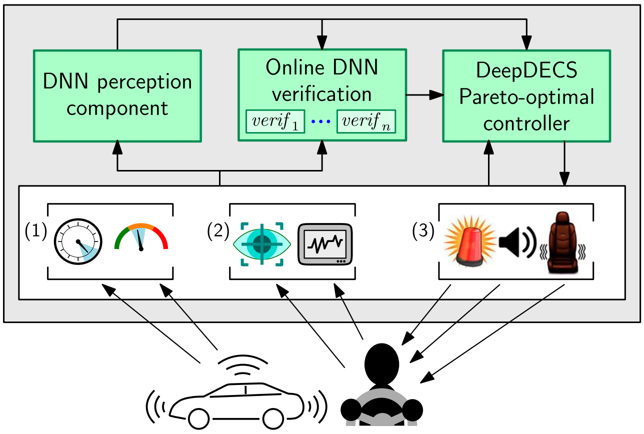


Fig. 6. Driver-attentiveness management for shared-control autonomous driving. Data from car sensors (1) and driver biometric sensors (2) are supplied to a DNN perception component that classifies the driver state as attentive, semi-attentive or inattentive. The DeepDECS controller decides when optical, acoustic and/or haptic alerts (3) should be used to increase the driver's attentiveness.

Further details about these four inputs are provided below.

DNN perception component. The datasets for training the DNN and quantifying its uncertainty were taken from a user study [73] conducted as part of our SafeSCAD project on the safety of shared control in autonomous driving [4]. Each datapoint included: (i) driver biometrics (eye movement, heart rate, and galvanic skin response); (ii) driver gender; (iii) driver perceived workload and psychological stress (estimated using established metrics); (iv) driver engagement in non-driving tasks while not in control of the car (e.g., using a mobile phone, or reading); and (v) vehicle data (distances to adjacent lanes and to any potential hazard, steering wheel angle, velocity, and gas and break pedal angles). We used 60% of the collected data for training a three-class DNN classifier with the architecture proposed by Pakdamanian et al. [73], and 15% for its calibration and validation.

Test dataset. We used a test dataset comprising the 25% of the data mentioned above that were not used for the DNN training, calibration and validation.

Perfect-perception pDTMC model. We modelled the operation of the driver attentiveness management system from Fig. 6 using a perfect-perception pDTMC (available in our GitHub repository [24]) whose states are tuples

$$(z,k,t,c) \in \{0,1,\ldots,7\} \times [3] \times [3] \times \{0,1,\ldots,7\} \quad (37)$$

with the semantics from (13). In this tuple, the system state $z \in \{0,1,\ldots,7\}$ is a binary encoding of the alerts currently activated, e.g., $z = 5 = 101_{(2)}$ corresponds to a scenario in which the optical alert is active, the acoustic alert is inactive, and the haptic alert is active; the classes $k=1$, $k=2$ and $k=3$ correspond to the driver being attentive, semi-attentive and inattentive, respectively;[3] and the control variable $c \in \{0,1,\ldots,7\}$

[3] We used a three-class DNN classifier as recommended by the authors of the user study [73] that this application is based on. Advantageously, this allowed the evaluation of DeepDECS for non-binary DNN classifiers in addition to its evaluation for binary classifiers in Section V-B.

is the binary encoding of the alerts to be activated in response to a new DNN prediction of the driver's attentiveness level.

PCTL-encoded requirements. The system-level requirements comprise two constraints that limit the maximum expected risk and driver nuisance cumulated over a 45-minute driving trip, and two optimisation objectives requiring that the same two measures are minimised:

$$\begin{array}{ll} C_1: & \mathcal{R}^{\mathsf{risk}}[\mathrm{C}^{\leq ntrans(45)}] \leq 100 \\ C_2: & \mathcal{R}^{\mathsf{nuisance}}[\mathrm{C}^{\leq ntrans(45)}] \leq 6000 \\ O_1: & \text{minimise } \mathcal{R}^{\mathsf{risk}}[\mathrm{C}^{\leq ntrans(45)}] \\ O_2: & \text{minimise } \mathcal{R}^{\mathsf{nuisance}}[\mathrm{C}^{\leq ntrans(45)}] \end{array} \tag{38}$$

where $\mathcal{R}^{\mathsf{rwd}}[\mathrm{C}^{\leq ntrans(45)}]$ denotes the reward rwd cumulated over the number of DNN-perception pDTMC transitions corresponding to a 45-minute journey.

2) DeepDECS Application

Stage 1: DNN uncertainty quantification. Fig. 7(a) shows the four sets of DNN uncertainty quantification probabilities (11) obtained using the test dataset from Section V-C1 for each DNN verification setup. Similar to the robot collision-limitation controller, setups (ii)–(iv), which use the DNN verification techniques $verif_1$ from (7) and/or $verif_2$ from (8), led to large fractions of the test dataset being verified, and to higher DNN accuracy for these subsets compared to the no-verification setup. Furthermore, the "verified" DNN predictions have a much higher probability of being correct than "unverified" ones.

Stage 2: Model augmentation. We used our DeepDECS model augmentation tool to derive the DNN-perception pDTMC model for each setup (i.e., set of DNN uncertainty quantification probabilities (11) from stage 1). The DNN-perception pDTMC models for all uncertainty quantification options explored in our experiments are provided on the DeepDECS website [24].

Stage 3: Controller synthesis. The controller parameters synthesised by DeepDECS were the encodings $x_{1v}, x_{2v}, x_{3v} \in \{0, 1, \ldots, 7\}$ of the alert combinations to be issued when the driver is attentive, semi-attentive and inattentive, respectively, where $v = ()$ for setup (i), $v \in \mathbb{B}$ for setups (ii) and (iii), and $v \in \mathbb{B}^2$ for setup (iv). As the controller design space was too large for exhaustive exploration, we used the EvoChecker probabilistic model synthesis tool [32] to generate close approximations of the Pareto-optimal controllers. EvoChecker performs this synthesis using a multi-objective genetic algorithm (MOGA) whose fitness function is computed with the help of a probabilistic model checker. For all setups, we configured EvoChecker to use the NSGA-II MOGA with a population size of 1000 and a maximum number of evaluations set to 20×10^4, and the model checker PRISM. The result of the DeepDECS controller synthesis is presented in Fig. 7(b). A visual inspection of the (approximate) Pareto fronts from this figure indicates that the setups that employed verification techniques achieved Pareto-optimal controllers closer to the perfect-perception Pareto front. In particular, the knee points of the Pareto fronts from setups (ii) and (iv) are much closer to the knee point of the perfect-perception front than those from the other setups. These findings are confirmed by the Pareto-front analysis that we conducted using the quality metrics IGD and HV (Fig. 7(c)), which shows that the quality metrics for these two fronts are the best out of the four setups. We further note that the two best Pareto fronts are almost indistinguishable visually, with the front obtained using both verification techniques having only a marginally better HV score than the one obtained using only $verif_1$. This result is in line with Theorem 4, which states that using more verification techniques is never detrimental but is not guaranteed to yield better trade-offs between the optimisation objectives.

As for the mobile robot application, we also considered the baseline approach in which the Pareto-optimal controllers were synthesised based on the DNN outputs alone, i.e., without quantifying the DNN uncertainty. As a result, we obtained an *expected* Pareto front whose points are associated with synthesised controllers which yield *actual* outcomes different from the expected ones, as shown by the inset plot from Fig. 7(b). Furthermore, none of the points on the expected Pareto front meets the two constraints from 38.

The EvoChecker executions used to generate the Pareto-optimal controller sets from Fig. 7(b) were carried out using five CPUs and 8GB of memory on the University of York's Viking high-performance cluster (https://www.york.ac.uk/it-services/services/viking-computing-cluster), with a set time of five hours. This result is shown in Fig. 7(d), which also reports the mean time required to execute the online DNN verification technique(s) on the regular computer with the specification mentioned at the end of Section V-B2.

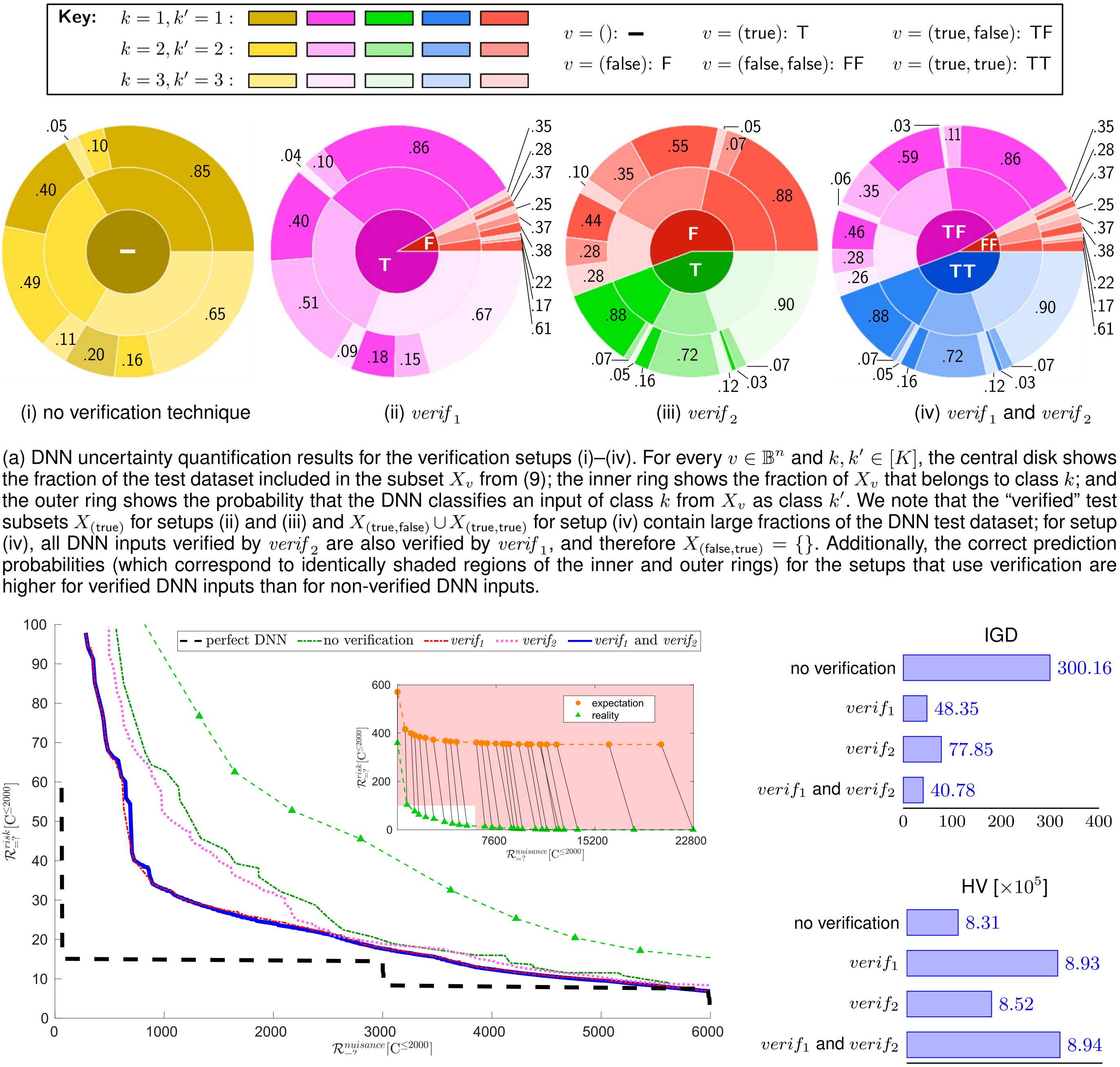
D. Discussion

Having applied our approach to two systems taken from different application domains and exhibiting the different characteristics summarised in Table I, we can now provide answers to the research questions from Section V-A.

RQ1 (Uncertainty quantification effectiveness): As shown in Figs. 5(a) and 7(a), for both DeepDECS applications and across all setups that used DNN verification techniques, the DNN inputs verified by at least one verification technique amounted to well over half of test datasets representative for the operational design domains of the two autonomous systems, with the exception of the setup from Fig. 7(a)(iii), for which a still large fraction of slightly under half of the test dataset was verified. These results show that a significant percentage of DNN inputs encountered within the ODDs of the two applications are verified, confirming that the key second prerequisite from the RQ1 description in Section V-A holds.[4]

RQ2 (Controller synthesis effectiveness): As shown by the inset plots from Figs. 5(b) and 7(b), the baseline approach synthesised inferior controllers whose expected outcomes: (a) barely met the required constraints for our first application, and did not meet the constraints for our second application; *and* (b) differed considerably from the actual outcomes produced

[4] Gven our use of established DNN verification techniques, the other prerequisite from the RQ1 description also holds: the DNNs from both applications exhibited much higher accuracies for their verified inputs, with differences as significant as 50% versus 95% accuracy for the correct classification of class 2 unverified and verified inputs, respectively, in the setup from Fig. 5(a)(ii).



(i) no verification technique (ii) $verif_1$ (iii) $verif_2$ (iv) $verif_1$ and $verif_2$

(a) DNN uncertainty quantification results for the verification setups (i)–(iv). For every $v \in \mathbb{B}^n$ and $k, k' \in [K]$, the central disk shows the fraction of the test dataset included in the subset X_v from (9); the inner ring shows the fraction of X_v that belongs to class k; and the outer ring shows the probability that the DNN classifies an input of class k from X_v as class k'. We note that the "verified" test subsets $X_{(\text{true})}$ for setups (ii) and (iii) and $X_{(\text{true},\text{false})} \cup X_{(\text{true},\text{true})}$ for setup (iv) contain large fractions of the DNN test dataset; for setup (iv), all DNN inputs verified by $verif_2$ are also verified by $verif_1$, and therefore $X_{(\text{false},\text{true})} = \{\}$. Additionally, the correct prediction probabilities (which correspond to identically shaded regions of the inner and outer rings) for the setups that use verification are higher for verified DNN inputs than for non-verified DNN inputs.

(b) Pareto front associated with the set of Pareto-optimal SafeSCAD controllers; the inset plot shows the expected Pareto front and actual outcome for the baseline approach, with shading used to indicate the region disallowed by the constraints in (38).

(c) Evaluation of Pareto front quality using the established IGD and HV metrics.

	perfect DNN	no verification	$verif_1$	$verif_2$	$verif_1$ and $verif_2$
Controller synthesis time	5 hours	5 hours	5 hours	5 hours	5 hours
Online DNN verification time	–	–	0.2ms	110ms	110.2ms

(d) DeepDECS controller synthesis and mean online DNN verification times

Fig. 7. DeepDECS controller synthesis results for the driver-attentiveness management system.

by these controllers. Additionally, the controller Pareto fronts from Figs. 5(b) and 7(b), and their quantitive evaluations from Figs. 5(c) and 7(c) show that increasing the number of DNN verification techniques used by DeepDECS yields controllers with better trade-offs between the optimisation objectives (18). Using either of the two verification techniques considered in our evaluation produced significantly better results than the no-verification setup, with verif_2 performing slightly better than verif_1 for the collision limitation controller (Fig. 5(c)), and verif_1 leading to much better results than verif_2 for driver-attentiveness management controller (Fig. 7(c)). Using both verification techniques at the same time produced even better controllers in both DeepDECS applications, although the improvements over the setups with a single verification technique were much smaller than those achieved by moving from no verification to one verification technique. This result indicates that including larger numbers of verification techniques yields diminishing returns.

RQ3 (Overheads): For all four DNN verification setups, the synthesis of the DeepDECS controller sets was completed in under one hour on a standard laptop computer for the mobile robot collision limitation (Fig. 5(d)), and controller sets of significantly better quality than those obtained for the baseline were generated in five hours on a modest five-CPU computer cluster for the driver-attentiveness management (Fig. 7(d)). As the DeepDECS controller synthesis is a one-off, development-time activity, all of these computational overheads are perfectly acceptable. As for the online computational overheads, carrying out the verification of one DNN prediction on a standard laptop computer took between 0.2ms (when only the lightweight DNN verification technique verif_1 was used) and 110.2ms (when both verification techniques were used for the driver-attentiveness management system). The former overhead is clearly acceptable, whereas the latter is likely to be acceptable for many practical applications; for instance, the United Nations regulation on vehicles with Level 3 automation [83] (which inspired our second DeepDECS application) specifies that driver unavailability should be established within 30s after the driver lost concentration—a time interval that is over 270 times larger than the 110.2ms DNN verification overhead.

The answers to our research questions show the effectiveness of the hybrid, neuro-symbolic approach to controller synthesis employed by DeepDECS, whose combined use of DNN verification techniques and probabilistic model checking provides an effective quantification of DNN classification uncertainty, and guarantees that the synthesised controllers meet system-level requirements, respectively.

Furthermore, DeepDECS is generalisable in two important ways. First, our approach to quantifying DNN uncertainty opens up the opportunity to leverage the broad range of recently devised DNN verification techniques, [37], [40], [45], [56], [63], [75], [78] that certify DNN properties like local robustness and confidence, for the purpose of uncertainty quantification. Second, DeepDECS is not prescriptive about the type of machine learning that introduces uncertainty into autonomous systems. As such, we envisage that it is equally applicable to autonomous systems with other types of machine learnt components for which local verification techniques exist to enable the quantification of their aleatory uncertainty. Such machine learning techniques that utilise confidence measures to quantify the uncertainty of their predictions include support vector machines and Gaussian processes. Finally, the case studies presented in the paper indicate that DeepDECS supports autonomous system controller synthesis for different application domains.

DeepDECS also has a number of limitations. A key factor for its successful application is the test dataset used for quantifying the uncertainty of the DNN used for perception by the autonomous systems. The DeepDECS theoretical foundation relies on the assumption that this test dataset is representative of the operational design domain (ODD) of the autonomous system. The invalidation of this assumption will cause the DeepDECS process to generate suboptimal controllers, or controllers that do not meet the required constraints (17). If the ODD evolves slowly over time, this will also cause the initial test dataset to no longer be representative. An example is an outdoors vision system trained during summer, and subsequently operating in winter, when snow has visually changed the landscape. Thus, DeepDECS only provides guarantees with respect to the assumed ODD. Future investigation into DeepDECS should incorporate an online monitoring component to instigate the DeepDECS process with new data if the system operates outside the ODD.

Another potential limitation of DeepDECS is its scalability. The uncertainty quantification and model augmentation stages of DeepDECS are scalable, as the former stage applies established, efficient DNN verification techniques independently to each sample within a dataset, and the latter stage only needs to parse and expand 2^n-fold certain commands from the PRISM-encoded perfect-perception pDTMC model, typically for $n \leq 2$ DNN verification techniques. However, the controller synthesis stage uses (probabilistic) model checking, which is known to experience scalability problems. While the synthesis of the Pareto-optimal sets of controllers from our two case studies only took between a few seconds and several hours, the controller design spaces of autonomous systems grow linearly with the numbers of DNN classes K and system states $\#Z$, and exponentially with the number of DNN verification techniques n. Encouragingly, the EvoChecker probabilistic model synthesis tool [32] (which we used in the second case study) was shown to generate close Pareto front approximations for search spaces comprising over 10^{86} parameter-value combinations [33]. Nevertheless, further experimentation, in particular with DNNs predicting larger numbers of classes, will be needed to confirm the scalability of DeepDECS.

E. Threats to Validity

Construct validity threats may arise due to assumptions made about the autonomous systems used in our evaluation, or about their deep-learning perception components. To limit these threats, the two autonomous systems used in the paper are adapted from existing research on collision avoidance for autonomous vehicles/robots [28], [54], [55], [85] and driver-attentiveness management [13]. Furthermore, their DNN perception components have the architecture and outputs recommended in the research literature on deep-learning classifiers for collision prediction [28] and by a user study on deep-learning classifiers for monitoring driver attentiveness [73].

Internal validity threats may stem from bias in establishing cause-effect relationships in the experiments from our DeepDECS applications. To mitigate these threats, we assessed DeepDECS for four combinations of two existing DNN verification techniques, and we used two established Pareto front quality metrics to evaluate the sets of Pareto-optimal controllers produced by our approach. Furthermore, we have enabled replication by making all our models, code and experimental data available in the project's GitHub repository [24].

External validity threats could affect the applicability of DeepDECS to other autonomous systems than those used in our evaluation. As summarised in Table I, we mitigated these threats by evaluating the use of DeepDECS for two applications drawn from different application domains, involving different types of systems with requirements based on different properties of those systems. Additionally, the DNN perception components used by the two systems have different characteristics. Furthermore, DeepDECS employs established probabilistic model checking methods and tools that have been successfully used across a wide range of application domains, and we can expect their generality to extend to our approach. Nevertheless, additional experiments are needed to establish the applicability and feasibility of DeepDECS in domains and for autonomous systems with characteristics different from those used in our evaluation.

Another external validity threat may arise if the modelling of other autonomous systems requires the use of much larger perfect-perception pDTMC models and/or these systems use DNN classifiers with many more classes than for the two applications from our evaluation of DeepDECS. As our approach uses established probabilistic model checking and synthesis tools, evaluating their scalability is beyond the scope of our paper. Such evaluations are already available, e.g., in [50] and [32], [33], respectively, and (as explained at the end of Section V-D) they suggest that DeepDECS should scale well to larger systems. As always, further case studies are needed to examine this hypothesis.

VI. Related Work

The design of autonomous systems that use DNN classifiers for perception in combination with discrete-event controllers for decision-making has been studied before. The approach of Jha et al. [51] synthesises correct-by-construction controllers for autonomous systems with noisy sensors, i.e., with perception uncertainty. Unlike DeepDECS, this approach only considers systems that use linear models (i.e., not DNNs) for perception, and assumes already known uncertainty quantities. Moreover, while we formulate the control problem as a pDTMC, Jha et al. consider the simpler setting of deterministic linear systems.

Michelmore et al. [65] 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 DNN control actions, and, when this uncertainty exceeds pre-determined thresholds, the system defaults to executing fail-safe actions. In contrast, we synthesise controllers that use the quantified uncertainty of DNN perception to select optimal yet safe actions.

Similarly, Ivanov et al. [48] present a technique for learning end-to-end DNN controllers for an autonomous system such that the autonomous system satisfies non-probabilistic specifications of safety. In contrast to our work where the autonomous system is verified with respect to an abstract, uncertainty-aware, probabilistic model of the DNN, Ivanov et al. verify the autonomous system with respect to the actual DNN. This makes verification extremely expensive; applying their technique to an autonomous system with a complex DNN (thousands or millions of parameters) is practically infeasible.

Recent work by ourselves [74] and others [5] also proposes to use confusion matrices for quantifying the aleatory uncertainty of DNNs and to verify the probabilistic safety of the autonomous system in an uncertainty-aware manner. Unlike DeepDECS, these approaches assume that the controller has already been synthesized. A number of other approaches [22], [44], [46], [47], [49] [57], [72], [77] have been proposed in recent years for verifying the closed-loop safety of autonomous systems with DNN-based components and already synthesized controllers. These approaches differ in the manner in which they model the environment and perception components but, in general, scalability is a challenge.

The related field of reinforcement learning (RL) is a paradigm for generating policies to solve a task, and have incorporated deep-learning to solve complex problems [31], [66]. In recent years the deep-RL community has been interested in ensuring the safety during both exploration and exploitation phases [10]. Berkenkamp et al. [7] implemented a safe model-based RL algorithm with verification in the loop, which provides assurances that the system would not enter unsafe states while learning. Control barrier functions have been studied [64] for safe RL which penalises the system if it approaches an unsafe state. RL in general, however, produces one controller compared to multiple correct-by-construction controllers generated by DeepDECS. The multiple controllers with their respective trade-offs yields flexibility to the user/stakeholders to choose a controller that best satisfy their objectives. There is also a higher level of explainability with DeepDECS, as the controller is modelled via pDTMC rather than learned with a DNN. Related to the idea of having DeepDECS operate in an online setting, these safe RL approaches could be exploited. If the deployed system detects it is outside the ODD, then it will need to gather new data while maintaining safety, which these algorithms are designed to provide.

Handling uncertainty in autonomous systems has been studied extensively by the recent research on self-adaptive software, e.g., [15], [16], [43], [79], including in the context of controller synthesis, e.g., [11], [58]. However, the approaches proposed by this research focus on uncertainty due to failures, changes in the environment, interactions with users, etc. In contrast, DeepDECS tackles the uncertainty introduced by the deep-learning perception components of autonomous systems.

In conclusion, synthesising safe and optimal controllers that account for the uncertainty in the DNN outcomes is a novel contribution of DeepDECS. Additionally, our DNN uncertainty quantification mechanism, which uses the outcomes of off-the-shelf DNN verifiers in a black-box manner, is also new.

VII. Conclusion

We introduced DeepDECS, a new method for the synthesis of correct-by-construction controllers for autonomous systems with deep-learning perception. The new method uses (i) a suite of techniques for the verification of deep neural networks to quantify the aleatory uncertainty associated with DNN perception, and (ii) a mathematically based stochastic modelling paradigm to synthesise discrete-event controllers that take this uncertainty into account. The controllers synthesised using our method are guaranteed to satisfy the safety, dependability and performance requirements of the autonomous system within its operational design domain. Furthermore, they are guaranteed to achieve optimal trade-offs between a set of pre-specified optimisation objectives for the autonomous system.

To evaluate DeepDECS, we presented its application to the synthesis of discrete-event controllers for mobile-robot collision limitation, and for maintaining driver attentiveness in shared-control autonomous driving. To ensure the reproducibility of our experiments, we made all the software, datasets, models and results from these experiments publicly accessible on our GitHub project website [24].

In future work, we will explore several opportunities for extending the applicability, effectiveness and usability of DeepDECS. First, we will assess the possibility to use DeepDECS controller synthesis for autonomous systems with other types of machine learnt components (e.g., support vector machines, reinforcement learning agents, and Gaussian processes). Second, we plan to assemble a broad repertoire of DNN verification techniques that can be used in the DeepDECS uncertainty quantification stage, and to examine the usefulness and limitations of these techniques when used for this purpose. Third, we will explore options for complementing the current DeepDECS capabilities with monitoring the environment of a deployed autonomous system in order to identify changes from its operational design domain, and to dynamically update the synthesised controllers in line with such changes. Fourth, we will investigate ways in which DeepDECS controllers can be augmented with the ability to detect out-of-distribution DNN inputs (e.g. through worsening DNN verification results over time) and to mitigate their occurrence. Last but not least, we aim to expand the evaluation of DeepDECS to additional application domains and types of autonomous systems.

Acknowledgment

The authors are grateful to the developers of the DeepTake deep neural network [73] 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] S. Andova, H. Hermanns, and J.-P. Katoen, "Discrete-time rewards model-checked," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, Berlin, Germany: Springer-Verlag, 2003, pp. 88–104.

[2] A. Arapostathis, R. Kumar, and S.-P. Hsu, "Control of Markov chains with safety bounds," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 4, pp. 333–343, Oct. 2005.

[3] R. Ashmore, R. Calinescu, and C. Paterson, "Assuring the machine learning lifecycle: Desiderata, methods, and challenges," *ACM Comput. Surv.*, vol. 54, no. 5, pp. 1–39, 2021.

[4] Assuring Autonomy International Programme, "Safe-SCAD: Safety of shared control in autonomous driving," Univ. of York, York, U.K., 2022. [Online]. Available: https://www.york.ac.uk/assuring-autonomy/demonstrators/autonomous-driving/

[5] A. Badithela, T. Wongpiromsarn, and R. M. Murray, "Leveraging classification metrics for quantitative system-level analysis with temporal logic specifications," in *Proc. 60th IEEE Conf. Decis. Control (CDC)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 564–571.

[6] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski, "Controller synthesis for probabilistic systems," in *Proc. Exploring New Frontiers Theor. Inform., IFIP 18th World Comput. Congr. TC1 3rd Int. Conf. Theor. Comput. Sci. (TCS),* Toulouse, France. Boston, MA, USA: Springer-Verlag, 2004, pp. 493–506.

[7] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30.

[8] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Proc. Int. Conf. Found. Softw. Technol. Theor. Comput. Sci.*, Berlin, Germany: Springer-Verlag, 1995, pp. 499–513.

[9] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First three years of the international verification of neural networks competition (VNN-COMP)," *Int. J. Softw. Tools Technol. Transfer*, vol. 25, pp. 329–339, Jun. 2023.

[10] L. Brunke et al., "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 5, pp. 411–444, May 2022.

[11] R. D. Caldas, A. Rodrigues, E. B. Gil, G. N. Rodrigues, T. Vogel, and P. Pelliccione, "A hybrid approach combining control theory and AI for engineering self-adaptive systems," in *Proc. IEEE/ACM 15th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst.*, 2020, pp. 9–19.

[12] R. Calinescu, K. Johnson, and Y. Rafiq, "Developing self-verifying service-based systems," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 734–737.

[13] R. Calinescu, N. Alasmari, and M. Gleirscher, "Maintaining driver attentiveness in shared-control autonomous driving," in *Proc. Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst. (SEAMS)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 90–96.

[14] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Efficient synthesis of robust models for stochastic systems," *J. Syst. Softw.*, vol. 143, pp. 140–158, Sep. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218300967

[15] R. Calinescu, R. Mirandola, D. Perez-Palacin, and D. Weyns, "Understanding uncertainty in self-adaptive systems," in *Proc. IEEE Int. Conf. Autonomic Comput. Self-Organizing Syst. (ACSOS)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 242–251.

[16] J. Cámara et al., "The uncertainty interaction problem in self-adaptive systems," *Softw. Syst. Model.*, vol. 21, no. 4, pp. 1277–1294, 2022.

[17] K. Chatterjee, M. Chmelík, R. Gupta, and A. Kanodia, "Qualitative analysis of POMDPs with temporal logic specifications for robotics applications," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2015, pp. 325–330.

[18] K. Chatterjee, M. Chmelík, R. Gupta, and A. Kanodia, "Optimal cost almost-sure reachability in POMDPs," *Artif. Intell.*, vol. 234, pp. 26–48, May 2016.

[19] A. I. Chen, M. L. Balter, T. J. Maguire, and M. L. Yarmush, "Deep learning robotic guidance for autonomous vascular access," *Nature Mach. Intell.*, vol. 2, pp. 104–115, Feb. 2020.

[20] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, "Automatic verification of competitive stochastic systems," in *Proc. 18th Int. Conf. Tools Algorithms Construction Anal. Syst. (TACAS)*, C. Flanagan and B. König, Eds., vol. 7214. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 315–330.

[21] F. Chollet, *Deep Learning with Python*. New York, NY, USA: Simon and Schuster, 2021.

[22] M. Cleaveland, I. Ruchkin, O. Sokolsky, and I. Lee, "Monotonic safety for scalable and data-efficient probabilistic safety analysis," in *Proc. ACM/IEEE 13th Int. Conf. Cyber-Physical Syst. (ICCPS)*, Piscataway, NJ, USA: IEEE Press, 2022, pp. 92–103.

[23] C. Daws, “Symbolic and parametric model checking of discrete-time Markov chains,” in *Proc. Int. Colloq. Theor. Aspects Comput.*, 2005, pp. 280–294.

[24] “DeepDECS project website.” GitHub. [Online]. Available: https://ccimrie.github.io/DeepDECS/

[25] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, “A STORM is coming: A modern probabilistic model checker,” in *Proc. 29th Int. Conf. Comput. Aided Verification (CAV)*, 2017, pp. 592–600.

[26] A. Der Kiureghian and O. Ditlevsen, “Aleatory or epistemic? Does it matter?” *Structural Saf.*, vol. 31, no. 2, pp. 105–112, Mar. 2009.

[27] V. D’Silva, D. Kroening, and G. Weissenbacher, “A survey of automated techniques for formal software verification,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1165–1178, Jul. 2008.

[28] R. Ehlers, “Formal verification of piece-wise linear feed-forward neural networks,” in *Proc. Automated Technol. Verification Anal.*, D. D’Souza and K. Narayan Kumar, Eds., Cham, Switzerland: Springer-Verlag, 2017, pp. 269–286.

[29] A. Filieri et al., “Software engineering meets control theory,” in *Proc. IEEE/ACM 10th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst.*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 71–82.

[30] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, “Quality prediction of service compositions through probabilistic model checking,” in *Proc. Int. Conf. Qual. Softw. Archit.*, Berlin, Germany: Springer-Verlag, 2008, pp. 119–134.

[31] J. Garcia and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[32] S. Gerasimou, R. Calinescu, and G. Tamburrelli, “Synthesis of probabilistic models for quality-of-service software engineering,” *Automated Softw. Eng.*, vol. 25, no. 4, pp. 785–831, 2018.

[33] S. Gerasimou, G. Tamburrelli, and R. Calinescu, “Search-based synthesis of probabilistic models for quality-of-service software engineering,” in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 319–330.

[34] M. Gleirscher and R. Calinescu, “Safety controller synthesis for collaborative robots,” in *Proc. 25th Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 83–92.

[35] M. Gleirscher et al., “Verified synthesis of optimal safety controllers for human-robot collaboration,” *Sci. Comput. Program.*, vol. 218, Jun. 2022, Art. no. 102809.

[36] M. Gleirscher, N. Johnson, P. Karachristou, R. Calinescu, J. Law, and J. Clark, “Challenges in the safety-security co-assurance of collaborative industrial robots,” in *The 21st Century Industrial Robot: When Tools Become Collaborators. Intelligent Systems, Control and Automation: Science and Engineering*. Cham, Switzerland: Springer-Verlag, 2022, pp. 191–214.

[37] D. Gopinath, G. Katz, C. S. Pasareanu, and C. Barrett, “DeepSafe: A data-driven approach for assessing robustness of neural networks,” in *Proc. Int. Symp. Automated Technol. Verification Anal. (ATVA)*, 2018, pp. 3–19.

[38] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, Apr. 2020.

[39] C. M. Grinstead and J. L. Snell, *Introduction to Probability*. Rhode Island: American Mathematical Soc., 1997.

[40] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proc. 34th Int. Conf. Mach. Learn.*, JMLR.org, 2017, vol. 70, pp. 1321–1330.

[41] H. Hansson and B. Jonsson, “A logic for reasoning about time and reliability,” *Formal Aspects Comput.*, vol. 6, no. 5, pp. 512–535, Sep. 1994.

[42] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, “The probabilistic model checker Storm,” *Int. J. Softw. Tools Technol. Transfer*, vol. 24, pp. 589–610, Aug. 2022.

[43] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin, “Uncertainty in self-adaptive systems: A research community perspective,” *ACM Trans. Auton. Adaptive Syst.*, vol. 15, no. 4, pp. 1–36, 2021.

[44] C. Hsieh, Y. Li, D. Sun, K. Joshi, S. Misailovic, and S. Mitra, “Verifying controllers with vision-based perception using safe approximate abstractions,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4205–4216, Nov. 2022.

[45] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, “Safety verification of deep neural networks,” in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, R. Majumdar and V. Kuncak, Eds., Cham, Switzerland: Springer-Verlag, 2017, pp. 3–29.

[46] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, “Verisig 2.0: Verification of neural network controllers using taylor model preconditioning,” in *Proc. Int. Conf. Comput. Aided Verification*, Cham, Switzerland: Springer-Verlag, 2021, pp. 249–262.

[47] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verifying the safety of autonomous systems with neural network controllers,” *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 1, pp. 1–26, 2020.

[48] R. Ivanov, K. Jothimurugan, S. Hsu, S. Vaidya, R. Alur, and O. Bastani, “Compositional learning and verification of neural network controllers,” *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5s, pp. 1–26, 2021.

[49] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verisig: Verifying safety properties of hybrid systems with neural network controllers,” in *Proc. 22nd ACM Int. Conf. Hybrid Syst.,* Comput. Control, 2019, pp. 169–178.

[50] D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelinga, and I. Zapreev, “How fast and fat is your probabilistic model checker? An experimental performance comparison,” in *Proc. Haifa Verification Conf.*, Berlin, Germany: Springer-Verlag, 2007, pp. 69–85.

[51] S. Jha, V. Raman, D. Sadigh, and S. A. Seshia, “Safe autonomy under perception uncertainty using chance-constrained temporal logic,” *J. Automated Reasoning*, vol. 60, no. 1, pp. 43–62, 2018.

[52] M. A. Johnson and M. H. Moradi, *PID Control*. London, U.K.:Springer-Verlag, 2005.

[53] V. R. Joseph, “Optimal ratio for data splitting,” *Statist. Anal. Data Mining, ASA Data Sci. J.*, vol. 15, no. 4, pp. 531–538, 2022.

[54] K. D. Julian and M. J. Kochenderfer, “Reachability analysis for neural network aircraft collision avoidance systems,” *J. Guid., Control, Dyn.*, vol. 44, no. 6, pp. 1132–1142, 2021.

[55] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, “Deep neural network compression for aircraft collision avoidance systems,” *J. Guid., Control, Dyn.*, vol. 42, no. 3, pp. 598–608, 2019.

[56] G. Katz et al., “The Marabou framework for verification and analysis of deep neural networks,” in *Proc. Int. Conf. Comput. Aided Verification (CAV)*, Cham, Switzerland: Springer-Verlag, 2019, pp. 443–452.

[57] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer, “Verification of image-based neural network controllers using generative models,” *J. Aerosp. Inf. Syst.*, vol. 19, no. 9, pp. 574–584, 2022.

[58] M. Keegan, V. Braberman, N. D’Ippolito, N. Piterman, and S. Uchitel, “Control and discovery of environment behaviour,” *IEEE Trans. Softw. Eng.*, vol. 48, no. 6, pp. 1965–1978, Jun. 2022.

[59] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. 3rd Int. Conf. Learn. Representations (ICLR)*, San Diego, CA, USA, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[60] F. Küppers, J. Kronenberger, A. Shantia, and A. Haselhoff, “Multivariate confidence calibration for object detection,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, Jun. 2020, pp. 326–327.

[61] M. Kwiatkowska, G. Norman, and D. Parker, “Controller dependability analysis by probabilistic model checking,” *Control Eng. Pract.*, vol. 15, no. 11, pp. 1427–1434, Nov. 2007.

[62] M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in *Proc. 23rd Int. Conf. Comput. Aided Verification*, vol. 6806. Berlin, Germany: Springer-Verlag, 2011, pp. 585–591.

[63] K. Leino, Z. Wang, and M. Fredrikson, “Globally-robust neural networks,” in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 6212–6222.

[64] Z. Marvi and B. Kiumarsi, “Safe reinforcement learning: A control barrier function optimization approach,” *Proc. Int. J. Robust Nonlinear Control*, vol. 31, no. 6, pp. 1923–1940, 2021.

[65] R. Michelmore, M. Wicker, L. Laurenti, L. Cardelli, Y. Gal, and M. Kwiatkowska, “Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control,” in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2020, pp. 7344–7350.

[66] V. Mnih et al., “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[67] J. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets* (The International Series on Discrete Event Dynamic Systems), vol. 8. New York, NY, USA: Springer Science & Business Media, 1998.

[68] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson, “The third international verification of neural networks competition (VNN-COMP 2022): Summary and results,” 2022, *arXiv:2212.10376*.

[69] G. Norman, D. Parker, and X. Zou, “Verification and control of partially observable probabilistic systems,” *Real-Time Syst.*, vol. 53, no. 3, pp. 354–402, 2017.

[70] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, SAE International Standard J3016_202104, 2018. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/preview/

[71] A. M. Ozbayoglu, M. U. Gudelek, and O. B. Sezer, "Deep learning for financial applications: A survey," *Appl. Soft Comput.*, vol. 93, Aug. 2020, Art. no. 106384.

[72] P. Habeeb, N. Deka, D. D'Souza, K. Lodaya, and P. Prabhakar, "Verification of camera-based autonomous systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 42, no. 10, pp. 3450–3463, Oct. 2023.

[73] E. Pakdamanian, S. Sheng, S. Baee, S. Heo, S. Kraus, and L. Feng, "DeepTake: Prediction of driver takeover behavior using multimodal data," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2021, pp. 1–14.

[74] C. S. Pasareanu et al., "Closed-loop analysis of vision-based autonomous systems: A case study," in *Computer* Aided Verification, C. Enea and A. Lal, Eds., Cham, Switzerland: Springer Nature Switzerland, 2023, pp. 289–303.

[75] C. Paterson, R. Calinescu, and C. Picardi, "Detection and mitigation of rare subclasses in deep neural network classifiers," in *Proc. IEEE Int. Conf. Artif. Intell. Testing*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 9–16.

[76] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[77] U. Santa Cruz and Y. Shoukry, "NNLander-VeriF: A neural network formal verification framework for vision-based autonomous aircraft landing," in *Proc. NASA Formal Methods Symp.*, Cham, Switzerland: Springer-Verlag, 2022, pp. 213–230.

[78] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "An abstract domain for certifying neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 1–30, 2019.

[79] G. F. Solano, R. D. Caldas, G. N. Rodrigues, T. Vogel, and P. Pelliccione, "Taming uncertainty in the assurance process of self-adaptive systems: A goal-oriented approach," in *Proc. IEEE/ACM 14th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst. (SEAMS)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 89–99.

[80] E. Stevens, L. Antiga, and T. Viehmann, *Deep Learning with PyTorch*. New York, NY, USA: Manning Publications, 2020.

[81] M. Svorenova and M. Kwiatkowska, "Quantitative verification and strategy synthesis for stochastic games," *Eur. J. Control*, vol. 30, pp. 15–30, Jul. 2016.

[82] D. Tabernik and D. Skocaj, "Deep learning for large-scale traffic-sign detection and recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 4, pp. 1427–1440, Apr. 2020.

[83] "ECE/TRANS/WP.29/2020/81—United Nations Regulation on Uniform provisions concerning the approval of vehicles with regard to Automated Lane Keeping Systems." UNECE. [Online]. Available: https://undocs.org/ECE/TRANS/WP.29/2020/81

[84] D. A. V. Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations," Ph.D. dissertation, Air Force Institute of Technology, Wright-Patterson AFB, OH, USA, 1999.

[85] Q. Xu, Y. Yang, C. Zhang, and L. Zhang, "Deep convolutional neural network-based autonomous marine vehicle maneuver," *Int. J. Fuzzy Syst.*, vol. 20, no. 2, pp. 687–699, 2018.

[86] E. Zitzler, D. Brockhoff, and L. Thiele, "The hypervolume indicator revisited: On the design of Pareto-compliant indicators via weighted integration," in *Evolutionary Multi-Criterion Optimization*, S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, Eds., Berlin, Germany: Springer-Verlag, 2007, pp. 862–876.

[87] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms — A comparative case study," in *Parallel Problem Solving from Nature — PPSN* V, A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds., Berlin, Germany: Springer-Verlag, 1998, pp. 292–301.

Radu Calinescu (Senior Member, IEEE) is a Professor in computer science with the University of York, U.K. His research interests include formal methods for self-adaptive, autonomous, secure and dependable software, cyber-physical and AI systems, and in performance and reliability software engineering. He is an Active Promoter of formal methods at runtime as a way to improve the integrity and predictability of self-adaptive, autonomous and AI systems and processes.

Calum Imrie is a Researcher with the Centre for Assuring Autonomy, the University of York, U.K. He investigates robotics and autonomous systems with a particular focus on the safety and assurance of deploying these systems. This includes both the learning phases, such as reinforcement learning, and at runtime particularly for AI components, and self-adaptive mechanisms. He has a special interest in robotics and autonomous systems being utilized for managing and protecting the environment.

Ravi Mangal received the Ph.D. degree in computer science from Georgia Institute of Technology, in 2020. He is a Postdoctoral Researcher with Carnegie Mellon University, the Security and Privacy Institute (CyLab). He is interested in all aspects of designing and applying formal methods for assuring the correctness and safety of software systems. His research interests include developing algorithms and methodologies for formally analyzing the safety and trustworthiness of learning-enabled systems.

Genaína Nunes Rodrigues received the Ph.D. degree from the University College London. She is an Associate Professor with the University of Brasilia. Her research interests include the mutual collaboration between smart autonomous systems engineering and software engineering, mainly through model checking, runtime verification, and goal-oriented requirements engineering.

Corina Păsăreanu is an ACM Fellow and an IEEE ASE Fellow, working at NASA Ames. She is affiliated with KBR and Carnegie Mellon University's CyLab. Her research interests include model checking, symbolic execution, compositional verification, probabilistic software analysis, autonomy, and security. She is on the steering committees for the ICSE, TACAS and ISSTA conferences, and is currently an Associate Editor for IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and for *STTT*, Springer Nature.

Misael Alpizar Santana received the Ph.D. degree from the University of York. He is currently a Postdoctoral Research Associate with the Department of Engineering, Durham University. His research interests include machine learning, particularly deep neural networks, secure and resilient autonomous and AI systems, self-adaptation, and formal verification.

Gricel Vázquez received the M.Sc. degree in computational intelligence and robotics with the University of Sheffield with distinction. She is a Ph.D. Student and a Research Associate in computer science with the University of York, U.K. Her research interests include model-driven engineering, formal methods, task allocation and planning, and self-adaptive and critical systems.