The Journal of Systems and Software 167 (2020) 110617

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Uncertainty modeling and runtime verification for autonomous vehicles driving control: A machine learning-based approach

Dongdong An, Jing Liu*, Min Zhang, Xiaohong Chen, Mingsong Chen, Haiying Sun

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China

ARTICLE INFO

Article history:
Received 16 September 2019
Revised 22 April 2020
Accepted 27 April 2020
Available online 1 May 2020

Keywords:
Machine learning
Driving style classification
Uncertainty modeling
Autonomous driving control
Runtime verification
Statistical model checking
Intelligent decision & control

ABSTRACT

Intelligent Transportation Systems (ITS) are attracting much attention from the industry, academia, and government in staging the new generation of transportation. In the coming years, the human-driven vehicles and autonomous vehicles would co-exist for a long time in uncertain environments. How to efficiently control the autonomous vehicle and improve the interaction accuracy as well as the human drivers' safety is a hot topic for the autonomous industry. The safety-critical nature of the ITSs demands the system designers to provide provably correct guarantees about the actions, models, control, and performance. To model and recognize the drivers' behavior, we use machine learning classification algorithms based on the data we get from the uncertain environments. We define a parameterized modeling language stohChart(p) (parameterized stochastic hybrid statecharts) to describe the interactions of agents in ITSs. The learning result of the driver behavior classification is transferred to stohChart(p) as the parameters timely. Then we propose a mapping algorithm to transform stohChart(p) to NPTA (Networks of Probabilistic Timed Automata) and use the statistical model checker UPPAAL-SMC to verify the quantitative properties. So the run-time verification method can help autonomous vehicles make "more intelligent" decisions at run-time. We illustrate our approach by modeling and analyzing a scenario of the autonomous vehicle try to change to a lane occupied by a human-driven car.

1. Introduction

In recent years, the emergence of Intelligent Transporation Systems (ITS) as a novel paradigm has revolutionized the relationship between vehicles, humans, and the physical environment (McQueen and McQueen, 1999; Dimitrakopoulos and Demestichas, 2010; Sadigh et al., 2014). The autonomous vehicles do not drive in an isolated space, they need to interact with pedestrians and human-driven vehicles around them. For example, when the human driver notices an autonomous vehicle in the next lane trying to change to his lane, he might decide to slow down or to speed up. The behavior of the human-driven vehicle is based on the drivers' driving style and environment factor. Therefore, the action of the autonomous vehicle is based on its driving control model and interaction with the human driver. The autonomous vehicle may estimate that the human is a aggressive driver, so its safest strategy is not to change lanes. If there is a moderate driver, he may slow down to make some room for the autonomous car to merge in the front. So it is crucial for autonmous vehicle to understand the interaction between the human and the environment, which further requires driving style classification of the human drivers. Moreover, the uncertainties of human behaviors and physical environment usually result in unavoidable stochastic behaviors of ITS. How to model human and system behaviors within a congested and cluttered environment and how to guarantee the real-time performance specifications are becoming significant challenges in driving control of autonomous vehicle.

One of the key aspects for achieving safe controllers for ITS is analyzing the interaction between the human and AI. Driving style classification plays a vital role in human-centric vehicle control systems (Wang et al., 2014; Bengler et al., 2014; Wang et al., 2016) and ITS Bolovinou et al. (2014). This has motivated a large body of works on classifying the driving style (Bejani and Ghatee, 2019; Feng et al., 2019; Martinelli et al., 2018). However, their approaches usually classify the driving style of the own car instead of surrounding vehicles. In the scenario of lane changing in uncertain enviro,nment, it is essential for the autonomous vehicle to perceive the driving style of the surrounding vehicles (Jiang et al., 2019b; Qiao and Wu, 2019). In this work, we propose a trained model by using the classification learner of MATLAB to classify the driving style of surrounding human-driven vehicles.

For the design of safe controllers for autonomous vehicles, we further need to formally express the driving model and the desir-

* Corresponding author.
E-mail addresses: anndongdong@gmail.com (D. An), jliu@sei.ecnu.edu.cn (J. Liu).

https://doi.org/10.1016/j.jss.2020.110617
0164-1212/

able properties that the system should satisfy. A feasible approach is to develop a new formal modeling language which can model the interaction of autonomous vehicles and human-driven vehicles. The existing modeling languages (such as UML (Lodderstedt et al., 2002), MARTE (Faugere et al., 2007), and SysML (Weilkiens, 2011)) often lack concrete domain elements to model ITS and the semantics of their provided diagrams are weak Guan et al. (2018). To bridge the gap between stochastic modeling and quantitative analysis of ITS, we propose a stochastic behavior description and formal verification approach called parametrized stochastic hybrid statecharts (stohChart(p)). stohChart(p) is a novel framework based on *Statistical Model Checking (SMC) (*Legay et al., 2010) techniques. It is suitable for the approximate functional validation of complex ITS designs. So we use the statistical model-checker UPPAAL-SMC (David et al., 2015) as the engine of our approach. Using the proposed approach, we model and analyze the scenario of changing lane for the autonomous vehicle. The experimental results demonstrate the effectiveness and efficiency of the machine learning-based approach.

In summary, this paper makes the following contributions:

- We use a machine learning method to train a model which can classify the driving style of surrounding human-driven vehicles. The learning results are the input parameters of our modeling model.
- We present a new formal visual language, *stohChart(p)* which can model and analyze the driving safety, using the driving style classification result as parameters.
- We propose a mapping algorithm to automatically transform stohChart(p) into NPTA models. The classification results are transmitted periodically as parameters to the model. Then we can check the safety specifications at runtime with the statistical model checker UPPAAL-SMC.

The rest of this paper is organized as follows. Section II introduces classification algorithms in machine learning, NPTA and PCTL. Section III shows that our approach can be effectively applied to the quantitative analysis of stohMChart(p) designs. Sections IV shows a case study and experimental results. Section V discusses related work, and Section VI concludes the paper.

2. Preliminaries

In this section we introduce some preliminaries about machine learning and statistical model checking that are necessary to understand our work.

2.1. Classification algorithms in machine learning

Classification is a technique to categorize the data into desired classes where we can assign label to each class.

Decision Tree makes decision with tree-like model. It splits the sample into two or more homogeneous sets (leaves) based on the differentiators in our input variables (Mingers, 1989; Safavian and Landgrebe, 1991). To choose a predictor, the algorithm considers all features and does a binary split on them. It will then choose the one with the highest accuracy, and repeat recursively, until it successfully splits the data in all leaves.

Naive Bayes Classifiers: Naive Bayes classification algorithm is a supervised learning algorithm based on Bayes theorem in probability theory (Kotsiantis et al., 2007; Huang and Li, 2011).

$$P(b|a_1, a_2, \ldots, a_n) = \frac{P(a_1, a_2, \ldots, a_n|b)P(b)}{P(a_1, a_2, \ldots, a_n)}$$

$P(b|a_1, a_2, \ldots, a_n)$ represents the probability of event b occuring under the event condition that $a_1, a_2, \ldots, a_n$ have occurred. $P(a_1, a_2, \ldots, a_n|b)$ represents the probability of events $a_1, a_2, \ldots, a_n$ occuring under the event condition that b have occurred. The classification is conducted by deriving the maximum posteriori which is the maximal $P(b|a_i)$ with the above assumption applying to Bayes theorem.

Support Vector Machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible (Utkin, 2019; Jakkula, 2006). New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Nearest Neighbor Classifiers classify an object by a majority vote of the object's neighbours, in the space of input parameter. The object is assigned to the class which is most common among its nearest neighbour. It simply classifies the object based on feature similarity (input variables) (Weinberger and Saul, 2009; Denoeux, 1995).

2.2. NPTA and PCTL

Networks of Probabilistic Timed Automata (NPTA) is a widely-used model for formalizing stochastic systems. NPTA is created by composing PTAs via input and output actions (David et al., 2011).

Definition 1. A probabilistic timed automaton (PTA) is defined by a tuple $P = (L, l, C, Act, inv, enab, prob, F)$ Norman et al. (2013) where: 1) L is a finite set of locations and $l \in L$ is an initial location; 2) C is a finite set of clocks; 3) Act is a finite set of actions; 4) inv: $L \to CC(C)$ is an invariant condition, CC is the clock constraint function; 5) $enab$: $L \times Act \to CC(C)$ is an enabing condition; 6) $prob$: $L \times Act \to Dist(2^C \times L)$ is a probabilistic transition function; 7) F: $L \to 2^{AP}$ is a labelling function mapping each location to a set of atomic proposition.

Definition 2. Probabilistic computation tree logic (PCTL) is temporal logic for the formal specification of quantitative properties of PTAs. PCTL is a probabilistic extension of the logic CTL Ciesinski and Größer (2004). AP is a set of atomic propositions and $ap \in AP$, $p \in [0, 1]$, $k \in N$, $\bowtie \in \{ \leq , < , > , \geq \}$

The basic semantics of PCTL is defined in Llerena et al. (2017) and Zhao et al. (2019). The syntax of PCTL is defined by state formulae ϕ and path formulae ψ:

$$\phi := true|ap|\phi \wedge \phi|\neg\phi|P_{\bowtie p}(\psi)$$

$$\psi := \phi U^{\leq k}\phi|X\phi$$

where the temporal operator X is called *Next* and U is called *Until*. State formulae ϕ is evaluated to be either true or false in each state. Satisfaction relations for a state s are defined:

$s \vDash true, s \vDash ap \quad \text{iff} \quad ap \in inv(s)$

$s \vDash \neg\phi \quad \text{iff} \quad s \nvDash \phi$

$s \vDash \phi_1 \wedge \phi_2 \quad \text{iff} \quad s \vDash \phi_1 \quad \text{and} \quad s \vDash \phi_2$

$s \vDash P_{\bowtie p}(\psi) \quad \text{iff} \quad Pr(s \vDash \psi) \bowtie p$

We need to mention that P in $s \vDash P_{\bowtie p}(\psi)$ represents the probability of the set of paths begin in state s and satisfying ψ. Given a path ψ, the ith state and the initial state are denoted as $\psi[i]$ and $\psi[0]$ respectively. The satisfaction relation for a path ψ is defined as:

$\psi \vDash X\phi \quad \text{iff} \quad \psi[1] \vDash \phi$

$\psi \vDash \phi_1 U^{\leq k}\phi_2 \quad \text{iff} \quad \exists i, 0 \leq i \leq k$

$(\psi[i] \vDash \phi_2 (\forall i, 0 \leq i \leq k \quad \psi[i] \vDash \phi_1))$

3. Our approach

One of the critical aspects to achieve safe driving control for autonomous vehicles is the design of the interaction between surrounding human-driven cars and the autonomous vehicle itself.

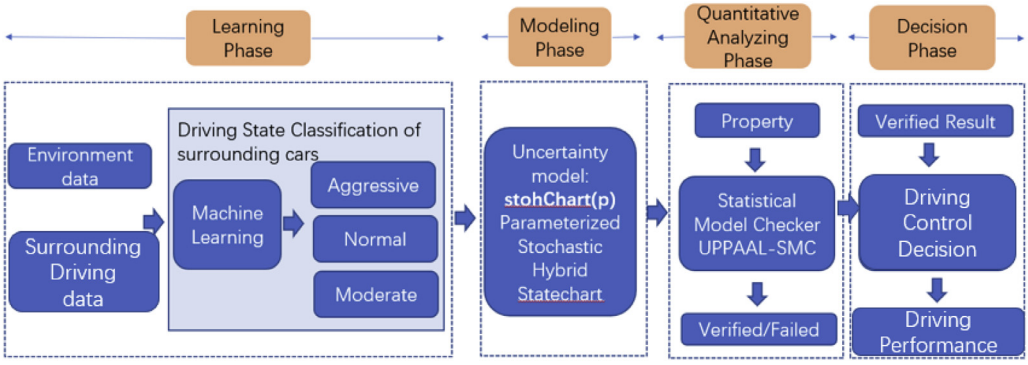

Fig. 1. The four phases of our approach: learning, modeling, quantitative analyzing and decision phase.

We attempt to estimate the environment and driving behaviors of the surrounding cars, then predict the driving style of the surrounding human-driven vehicles. We will evaluate the probability of the specific driving style based on a trained classification model. The results will be transferred to the uncertainty model stohChart(p) (Parameterized Stochastic Hybrid Statechart) as parameters. To achieve the goal, we propose an algorithm to mapping stohChart(p) to NPTA (Networks of Probabilistic Timed Automata). Moreover, it supports analyzing driving control with UPPAAL-SMC quantitatively. Our approach is shown in Fig. 1. It consists of four phases: Learning Phase, Modeling Phase, Quantitative Analyzing Phase, and Decision Phase.

1. **Learning phase:** In the first phase, the sensors of the autonomous vehicle collect the data from the uncertainty environment and driving behaviors of the surrounding cars. Our classification model is trained offline by using these data. At the end of the learning phase, we can get an offline trained model that can classify the driving style of the surrounding human-driven vehicles as *Aggressive, Normal, and Moderate*. When the autonomous car is driving on the road, it can collect and transfer the data to the trained classification model so that it can identify the driving style of the surrounding vehicles at runtime.
2. **Modeling phase:** We provide a hierarchical visual formal model stohChart(p) to model the driving states of the autonomous vehicle. The prediction results of the previous phase are transferred as parameters to the stohChart(p) model.
3. **Quantitative analyzing phase:** We propose a mapping algorithm to transfer stohChart(p) to NPTA models, in which the statistical model checker UPPAAL-SMC can verify the properties we define.
4. **Decision phase:** Based on the verification results, we can make driving control decisions to assist the autonomous vehicles make more intelligent actions.

Our approach integrates the learning and statistical model checking technologies to model the driving behaviors of surrounding vehicles. To implement our approach, we propose a technical framework, including seven modules shown in Fig. 2. The first module is *Classification*, which collects training data from the uncertain environment and surrounding driving cars. Then use the machine learning algorithm to generate the driving style classifier offline. So when the autonomous vehicles are driving on the road, the classifier can take new observation as input to get the prediction of the driving style of the surrounding human-driven cars. In the second module *Uncertainty Model Construction*, we input the prediction results as parameters to construct an uncertainty model stohChart(p). The blue gears represent the parameters and mapping rules are performed automatically. We use a mapping algorithm to transfer stohChart(p) to the NPTA model. In the third module *Runtime Verification*, the system designers (represent as the human head in Fig. 2) need to translate the informal design requirements to formal properties (represent as PCTL) by hand. The models of NPTA update the parameters automatically when new observations are classified during the driving of the autonomous vehicle. The model of NPTA and the formal properties are the input of the statistical model checker UPPAAL-SMC. So we can get quantitative analysis in runtime. In the module *Driving Control*, the comparison results of the runtime verification and the driving rules defined by system designers are the input of the driving control decision. At last, the physical part of the autonomous car will receive commands to perform new driving behaviors or not.

3.1. Learning phase: learning-based driving style classification

The autonomous vehicles are equipped with various sensors, and the sensors outputs are recorded. Fig. 3 shows the procedure of autonomous vehicle classifying the driving style of surrounding vehicles. The process of driving style classification comprises the following steps:

1. **Dataset generation:** Firstly, we need to obtain data from sensors. We divide the data which the sensors collect from the uncertain environment into three categories, as shown in Fig. 3, including *Relation to Surrounding Vehicles, Info of Surrounding vehicles* and *Surrounding Conditions*. In the left part of Fig. 3, the data in *Relation to Surrounding Vehicles* include the data related to surrounding vehicles such as relative distance, relative speed, and relative acceleration, etc. The data in *Info of Surrounding Vehicles* record the data like using car lamp correctly, braking time, and other wrong behavior, etc. The data in *Surrounding Conditions* contain pedestrians, cyclist obstacles, weather, road condition, traffic condition and day time, etc.
2. **Offline model training:** The *Classification Learner App* in MATLAB 2019b (ClassificationLearner, 2019) trains models to classify data which we obtained from previous step. So we can explore supervised machine learning based on various classifiers. By exploring our data, selecting features, specifying validation schemes, training models and assessing results, we can perform automated training to search for the best classification model type, including decision trees, discriminant analysis (Klecka et al., 1980), support vector machines, logistic regression (Dreiseitl and Ohno-Machado, 2002), nearest neighbors, naive Bayes, and ensemble classification.
We perform supervised machine learning by supplying a set of input data (observations) and responses to the data (e.g., labels or classes). We use the data to train a model that generates predictions for the response to new data. To learn about the programmatic classification and to use the model with new data,

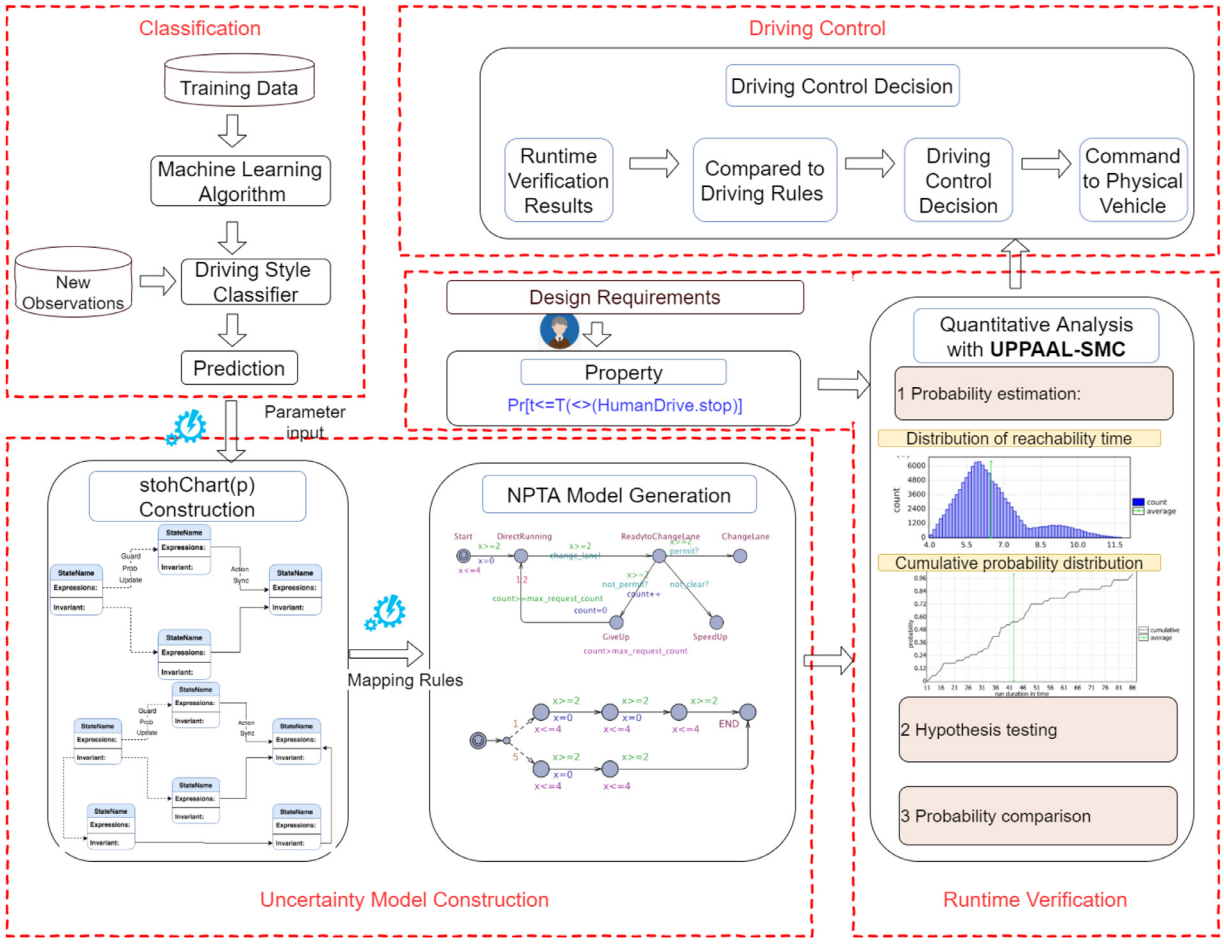

Fig. 2. The technical framework of our approach.

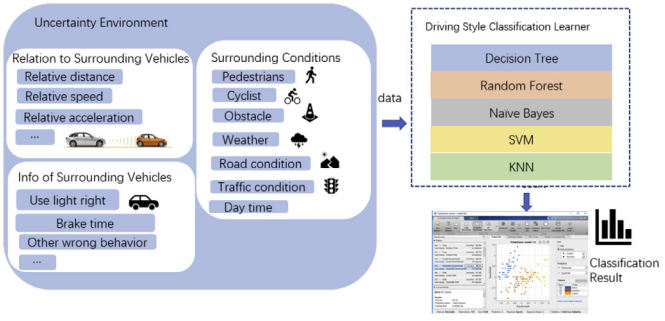

Fig. 3. Driving style classification based on machine learning.

we export the model to the workspace of MATLAB and generate code to recreate the trained model.

3. **Classification result:** The Classification Learner provides a series of classification algorithms, as mentioned in step 2. After training classifiers, based on accuracy scores, we can compare models, visualize results by plotting class predictions, and check performance using confusion matrix and ROC curve.
   - *Accuracy:* During training, we can plot different models and examine results of running training models with slightly lower accuracy maybe the most suitable classifier for our goal because some training is time-consuming. And for some data collection that is expensive or difficult, we may want to exclude that kind of predictors. When we finish training, the history list appears as selection of model types with a percentage accuracy score.
   - *Scatter Plot:* We use the scatter plot to investigate which variables are useful to predict the response. We can select different options on the various features under predictors to visualize the distribution of various driving styles. Based on the scatter plot, we can observe which variables separate the driving style most clearly.
   - *Confusion Matrix:* To find out how the classifier performed, we need to examine the *Confusion Matix*, as shown in Fig. 4. If the data points are misclassified, the cells in the plot are red. The green cells indicate the classifier has performed well, and the actual class and the predicted class match. On the right part of Fig. 4, it shows summaries per valid class in the last two columns. We can use this information to help us choose the best classifier for your goal.
   - *Parallel Coordinates Plot:* To investigate features to include or exclude, we use the parallel coordinates plot, as shown in Fig. 5. The parallel coordinates plot is useful to visualize matrix data with multiple columns. The columns of input data correspond to the axes in the plot. The data in Fig. 5 contains ten measurements on x-axis. We use a different color for each group as identified in driving styles, and label the horizontal axis using the variable names. The resulting plot contains one line for each observation (driving style). The color of each line indicates a different driving style. Different dots mean correct and incorrect observations, as shown in the right corner of Fig. 5.

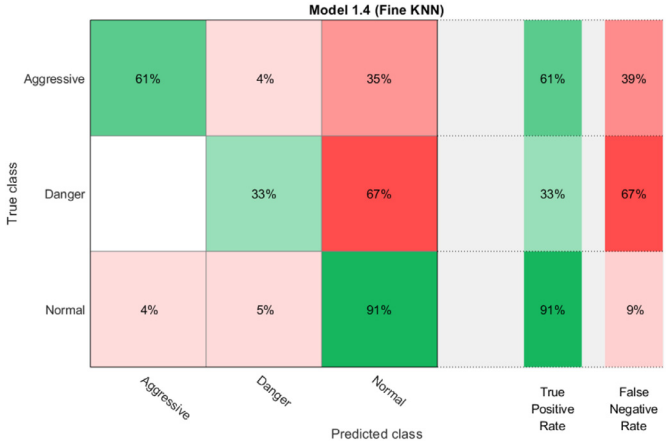

Fig. 4. Confusion matrix example.

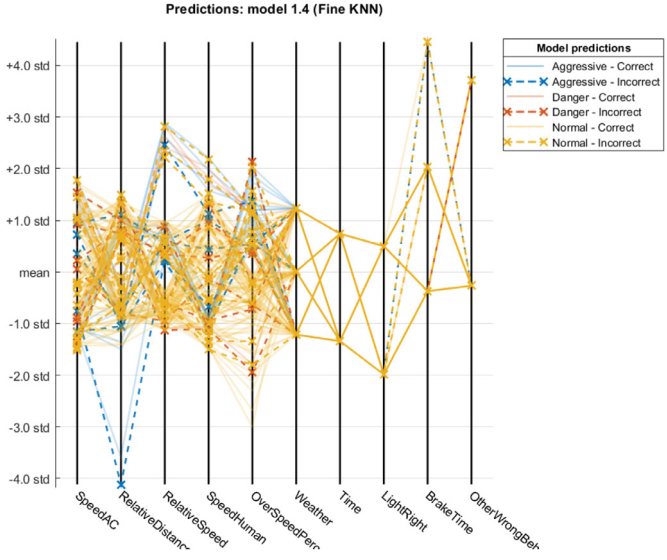

Fig. 5. Parallel coordinates plot example.

3.2. Modeling phase: uncertainty modeling language—stohChart(p)

stohChart(p) (parameterizsed stochastic hybrid statechart) is an extension of hybrid MARTE statecharts (Liu et al., 2013) and SHML (stochastic hybrid modeling language) (Du et al., 2019) based on NPTA (Networks of Probabilistic Timed Automata). Compared to hybrid MARTE statecharts, probabilistic transition are introduced to stohChart(p) to represent uncertain choice between different transitions. By introducing parameters as transition probability to SHML, stohChart(p) can use the learning results based on machine learning techniques to model the uncertain environments.

3.2.1. The syntax of stohChart(p)

A single stohChart(p) $\mathcal{S}$ is a tuple:

$$\mathcal{S} = (\mathbb{S}, \mathbb{T}, Cmd, Grd, Evt, Act, Var, C, Inv)$$

1. $\mathbb{S} = \{s_0, s_1, \ldots, s_m\}$ is a finite set of states. A state s is a tuple $(l, v, \exp, h)$ where l denotes the location, v denotes the value of the variables, *exp* denotes the expressions including ordinary differential equations (ODEs) which can model the continuous physical process. h denotes the hierarchy of the state.
2. $\mathbb{T} = \{t_0, t_1, \ldots, t_m\}$ is a finite set of transitions. $\mathbb{T} \subseteq 2^{\mathbb{S}} \times Cmd \times 2^{\mathbb{S}}$ for $(s, cmd, s') \in \mathbb{T},\ s \in \mathbb{S}, cmd \in Cmd, s' \in \mathbb{S}$.
3. $Cmd = \{cmd_0, cmd_1, \ldots, cmd_m\}$ is defined as $Cmd \subseteq Evt \times Grd \times Act \times Prb$ where Evt is the trigger events, Grd denotes the guards, Act denotes actions, P denotes the probabilities. The commands are used to identify transitions.
4. $Grd = \{g_0, g_1, \ldots, g_m\}$ is a finite set of guards. A guard is a boolean expression or boolean variable. The transition is triggered when the guard evaluates to ture. If the guard evaluates to false, then the event is consumed with no effect.
5. $Evt = \{e_0, e_1, \ldots, e_m\}$ is a finite set of events. The state of stohChart(p) $\mathcal{S}$ reacts to events such as signals, timeouts by triggering the transition and changing its current state to the target state if the guard is satisfied.
6. $Act = \{a_0, a_1, \ldots, a_m\}$ is a finite set of actions. The stohChart(p) $\mathcal{S}$ perform certain actions as a result of the state change when the transition is triggered.
7. Var is a finite set of variables. There are several types of variables in stohChart(p) $\mathcal{S}$.
8. C is a finite set of clocks variables. A clock valuation is a function $\mu : C \to R_+$. Given $d \in R_+$, $\mu + d$ denotes the valuation such as $(\mu + d)(c) = \mu(c) + d$, for all $\forall c \in C$.
9. Inv is a finite set of invariants, assigning to each $s \in \mathbb{S}$ a guard $Inv(s)$. State invariants are conditions attached to individual states and specify what has to hold in a state configuration (Sekerinski, 2008).

In the definition of state $(l, v, \exp, h)$, h denotes the hierarchy of the state ($h \in \{h_0, h_1\}$, h_0 denotes the super model hierarchy, h_1 denotes the submodel hierarchy). $\mathbb{S}_{h_0}$ represents the set of states in hierarchy h_0, $\mathbb{S}_{h_1}$ represents the set of states in hierarchy h_1. The function *Ode:* $s_i(v) \to s_i(v')$ denotes the calculation procedure of ODE which are defined in state s_i. The function *Sub:* $s_i \to P(s_i)$ denotes the set of children states of s_i. If $s' \in Sub(s_i)$, then s_i is the super state of s'. The function *Sup:* $s_i \to P(s_i)$ denotes the set of parent states of s_i. If $s' \in Sup(s_i)$, then s_i is the sub state of s'. The function *OutT:* $s_i \to P(t)$ denotes the set of transitions with the source state s_i. The function *Subl*: $s_i \to s_0$ denotes the initial state of a refined state s_i is s_0.

We divide transitions into four categories: Normal Transition $\mathbb{T}_n$ denoted as ($\to$), Stochastic Delay Transition $\mathbb{T}_s$ denoted as ($\rightsquigarrow$), Probabilistic Parametric Transition $\mathbb{T}_p$ denoted as ($\dashrightarrow$), Stochastic Probabilistic Parametric Transition $\mathbb{T}_{sp}$ denoted as ($\rightsquigarrow \bullet \dashrightarrow$). The function Type: $\mathbb{T}_i \to \{\mathbb{T}_n, \mathbb{T}_s, \mathbb{T}_p, \mathbb{T}_{sp}\}$ denotes the type of transitions $\mathbb{T}_i$. For all $\forall t = (s, cmd, s')$, we define the function *Source:* $t \to s$, *Target:* $t \to s'$ to get the source and target state of transition t. The stochastic delay transition is defined as: $s \overset{\texttt{after(F)}}{\rightsquigarrow} s',\ s, s' \in \mathbb{S}$, where `after(F)` is the measurement of the stochastic delay function `F`, `F` is defined as three types of probability density functions (pdf): $\texttt{Normal}(\mu, \delta), \texttt{Exp}(\theta), \texttt{Unif(a, b)}$. `after(F)` describes the probability distribution of the waiting time until the timeout happens. We define κ is a valuation of the delay function `after(F)`, if $j \in R_+, j = \kappa(\texttt{after(F)})$, then $\forall j' \in [0, j], (l, v, c + j', h_0) \in s$. The probabilistic parametric transition is defined as: $s \overset{\texttt{p}}{\dashrightarrow} s',\ s, s' \in \mathbb{S}$, where `p` denotes the transition probability. A parameter valuation τ is a function $\tau : \texttt{p} \to [0, 1]$. There are two types of transition probability, one is set the value during design procedure e.g.$s \overset{0.5}{\dashrightarrow} s'$, the other cannot get the exact value until at the runtime e.g.$s \overset{\texttt{p}}{\dashrightarrow} s',\ \tau(\texttt{p}) \in [0, 1]$. The stochastic probabilistic parametric transition is defined as the combination of stochastic delay transition and probabilistic parametric transiton: $s \overset{\texttt{after(F)}}{\rightsquigarrow} \bullet \overset{\texttt{p}}{\dashrightarrow} s',\ s, s' \in \mathbb{S}$, the system will stay in state s for time $j, j = \kappa(\texttt{after(F)})$ then it will get to state s' with the probability $\tau(\texttt{p})$, if `p` is unknown.

For all $\forall t = (s, cmd, s')$, we define the function $Source(t) = s, Target(t) = s', Guard(t) = g, Action(t) = a, Prb(t) = p$. The boolean function *Trg*: $e \to \{T, F\}$ denotes the event is triggerred or

Table 1
The operational semantic rules of stohChart(p).

(1) Self-updating with ODE

Rule 1.1 $\frac{s\wedge\exists e|Trg(e)\wedge(\nexists t\in\mathbb{T}, OutT(s)\wedge(v\vDash g\wedge c\vDash g))\wedge Sub(s)==\varnothing\wedge\exists ode\in s, v'=s.ode(v)}{(l,v,c,h_0)\xrightarrow{e,g,a}(l,v',c',h_0)}$

Rule 1.2 $\frac{s\wedge\exists e|Trg(e)\wedge(\nexists t\in\mathbb{T}, OutT(s)\wedge(v\vDash g\wedge c\vDash g))\wedge(\nexists t\in\mathbb{T}, OutT(Sup(s)\wedge v\vDash g\wedge c\vDash g)\wedge\exists ode\in s, v'=s.ode(v)}{(l,v,c,h_1)\xrightarrow{e,g,a}(l,v',c',h_1)}$

(2) $\mathbb{T}_n$ Activation

Rule 2.1 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_n,\ OutT(s)\wedge(v\vDash g\wedge c\vDash g))\vdash s'\in S_{h_0}\wedge Sub(s)==\varnothing}{(l,v,c,h_0)\xrightarrow{e,g,a}(l',v',c',h_0)}$

Rule 2.2 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_n,\ OutT(s)\wedge(v\vDash g\wedge c\vDash g))\vdash s'\in S_{h_0}\wedge Sub(s)\neq\varnothing\wedge(\nexists t, OutT(s')\wedge(v'\vDash g'\wedge c'\vDash g')\wedge s'=Sub_I(s)}{(l,v,c,h_0)\xrightarrow{e,g,a}(l',v',c',h_1)}$

Rule 2.3 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_n,\ OutT(Sup(s))\wedge(v\vDash g\wedge c\vDash g))\vdash s'\in S_{h_0}}{(l,v,c,h_1)\xrightarrow{e,g,a}(l',v',c',h_0)}$

(3) $\mathbb{T}_s$ Activation

Rule 3.1 $\frac{s\wedge(\exists t\in\mathbb{T}_s,\ OutT(s)\wedge(c':=\mu(c+\kappa(\texttt{after(F)})))\vdash s'\in S_{h_0}\wedge Sub(s)==\varnothing}{(l,v,c,h_0)\overset{\texttt{after(F)}}{\rightsquigarrow}(l',v,c',h_0)}$

Rule 3.2 $\frac{s\wedge(\exists t\in\mathbb{T}_s,\ OutT(s)\wedge(c':=\mu(c+\kappa(\texttt{after(F)})))\vdash s'\in S_{h_0}\wedge Sub(s)\neq\varnothing\wedge(\nexists t, OutT(s')\wedge(c'\vDash g'))\wedge s'=Sub_I(s)}{(l,v,c,h_0)\overset{\texttt{after(F)}}{\rightsquigarrow}(l',v,c',h_1)}$

Rule 3.3 $\frac{s\wedge(\exists t\in\mathbb{T}_s,\ OutT(Sup(s))\wedge(c':=\mu(c+\kappa(\texttt{after(F)})))\vdash s'\in S_{h_0}}{(l,v,c,h_1)\overset{\texttt{after(F)}}{\rightsquigarrow}(l',v,c',h_0)}$

(4) $\mathbb{T}_p$ Activation

Rule 4.1 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_p, OutT(s)\wedge(v\vDash g\wedge c\vDash g)\wedge p':=\tau(p))\vdash s'\in S_{h_0}\wedge Sub(s)==\varnothing}{(l,v,c,h_0)\overset{e,g,a,p'}{\dashrightarrow}(l',v',c',h_0)}$

Rule 4.2 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_p, OutT(s)\wedge(v\vDash g\wedge c\vDash g)\wedge p':=\tau(p))\vdash s'\in S_{h_0}\wedge Sub(s)\neq\varnothing\wedge(\nexists t, OutT(s')\wedge(v'\vDash g'\wedge c'\vDash g')\wedge s'=Sub_I(s)}{(l,v,c,h_0)\overset{e,g,a,p'}{\dashrightarrow}(l',v',c',h_1)}$

Rule 4.3 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_p, OutT(Sup(s))\wedge(v\vDash g\wedge c\vDash g)\wedge p':=\tau(p))\vdash s'\in S_{h_0}}{(l,v,c,h_1)\overset{e,g,a,p'}{\dashrightarrow}(l',v',c',h_0)}$

(5) $\mathbb{T}_{sp}$ Activation

Rule 5.1 $\frac{s\wedge\exists e|Trg(e)\wedge(\exists t\in\mathbb{T}_{sp}, OutT(s)\wedge(v\vDash g\wedge c':=\mu(c+\kappa(\texttt{after(F)}))\wedge p':=\tau(p))\vdash s'\in S_{h_0}\wedge Sub(s)==\varnothing}{(l,v,c,h_0)\overset{\texttt{after(F)}}{\rightsquigarrow}{}^{\bullet}_{\bullet}\overset{e,g,a,p'}{\dashrightarrow}(l',v',c',h_0)}$

not. The function $Trigger_s$: $s\rightarrow P(e)$ denotes the triggered events of state s. The function $Trigger_t$: $t\rightarrow P(e)$ denotes the triggered events of transition t. The function $Action_t$: $t\rightarrow P(a)$ denotes the actions of transition t. P is a finite set of parameters, i.e., unkown rational-valued variables. A parameter valuation τ is a function τ: $P\rightarrow[0, 1]$. Given a stohChart(p) $\mathcal{S}$ and a parameter valuation τ, we denote by $\tau(\mathcal{S})$ the non-parametric stohChart where all occurences of a parameter p_i have been replaced by $\tau(p_i)$, for each $p_i\in P$.

3.2.2. The semantics of stohChart(p)

We show the formal operational semantics rules of stohChart(p) in Table. 1.

1. **Self-updating with ODE:**
   Rule 1.1 shows that the current state is in the super layer $s(l, v, c, h_0)$, when a trigger event occurs ($\exists e|Trg(e)$), there does not exist a transition in the outgoing transiton set of state s ($\nexists t\in\mathbb{T}OutT(s)$) which satisfy the guard ($v\vDash g\wedge c\vDash g$). Besides there is no sub state of s $Sub(s)==\varnothing$, the ODE equation of state s is *ode* and the computing result is $v'=s.ode(v)$. The next state becomes (l, v', c', h_0).
   Rule 1.2 shows that when a trigger event happens in the sub layer $s(l, v, c, h_1)$, when a trigger event occurs ($\exists e|Trg(e)$), there does not exist a transition in the outgoing transiton set of state s ($\nexists t\in\mathbb{T}OutT(s)$) as well as in super state $\nexists t\in\mathbb{T}OutT(Sup(s))$. Then s calls ODE if $v'=s.ode(v)$ The next state becomes (l, v', c', h_1)
2. $\mathbb{T}_n$ **Activation:**
   Rule 2.1 shows the normal transition activation in super state $s(l, v, c, h_0)$, when a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transition set of state s ($\exists t\in\mathbb{T}_nOutT(s)$) which satisfy the guard ($v\vDash g\wedge c\vDash g$). S_{h_0} represents the set of states in hierarchy h_0, we have $s'\in S_{h_0}$ and there is no sub state of s $Sub(s)==\varnothing$. Then the next state becomes (l', v', c', h_0).
   Rule 2.2 shows the normal transition activation from super state to sub state. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transition set of state s ($\exists t\in\mathbb{T}_nOutT(s)$) which satisfy the guard ($v\vDash g\wedge c\vDash g$). S_{h_0} represents the set of states in hierarchy h_0, we have $s'\in S_{h_0}$ and there exists sub state of s $Sub(s)\neq\varnothing$ and s' is the initial state of sub state of s described as $s'=Sub_I(s)$. Besides, there do not exist a transition in the outgoing transition set of state s' ($\nexists t\in\mathbb{T}_nOutT(s')$) which satisfy the guard ($v'\vDash g'\wedge c'\vDash g'$). Then the next state becomes (l', v', c', h_1).
   Rule 2.3 shows the normal transition activation from sub state to super state. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transiton set of super state of s ($\exists t\in\mathbb{T}_nOutT(Sup(s))$ which satisfy the guard ($v\vDash g\wedge c\vDash g$). S_{h_0} represents the set of states in hierarchy h_0, we have $s'\in S_{h_0}$. Then the next state becomes (l', v', c', h_0).
3. $\mathbb{T}_s$ **Activation:**
   Rule 3.1 shows the stochastic delay transition activation in super state $s(l, v, c, h_0)$, when a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transiton set of state s ($\exists t\in\mathbb{T}_sOutT(s)$), the new clock c' update to $\mu(c+\kappa(\texttt{after(F)}))$ where `after(F)` is the measurement of the stochastic delay function `F`, μ is the clock valuation function and κ is a valuation of delay function. S_{h_0} represents the set of states in hierarchy h_0, we have $s'\in S_{h_0}$ and there is no sub state of s $Sub(s)==\varnothing$. Then the next state becomes (l', v, c', h_0).
   Rule 3.2 shows the stochastic delay transition activation from super state to sub state. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transiton set of state s ($\exists t\in\mathbb{T}_sOutT(s)$), the new clock c' updates to $\mu(c+\kappa(\texttt{after(F)}))$, we have $s'\in S_{h_0}$ and there exists sub state of s $Sub(s)\neq\varnothing$ and s' is the initial state of sub state of s described as $s'=Sub_I(s)$. Besides, there does not exist a transition in the outgoing transiton set of state s' ($\nexists t\in\mathbb{T}_sOutT(s')$) which satisfy the guard ($v'\vDash g'\wedge c'\vDash g'$). Then the next state becomes (l', v, c', h_1).
   Rule 3.3 shows the stochastic delay transition activation from sub state to super state. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transition set of su-

per state of s ($\exists t \in \mathbb{T}_s OutT(Sup(s))$ the new clock c' update to $\mu(c+\kappa(\texttt{after(F)}))$, we have $s' \in S_{h_0}$. Then the next state becomes (l', v, c', h_0).

4. $\mathbb{T}_p$ **Activation:**
   Rule 4.1 shows the probabilistic parametric transition activation in super state $s(l, v, c, h_0)$, when a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transition set of state s ($\exists t \in \mathbb{T}_p OutT(s)$) which satisfy the guard ($v \vDash g \wedge c \vDash g$). S_{h_0} represents the set of states in hierarchy h_0, we have $s' \in S_{h_0}$ and there is no sub state of s $Sub(s) == \varnothing$. The transition probability p' is $\tau : \mathrm{p} \rightarrow [0, 1]$. Then the next state becomes (l', v', c', h_0).
   Rule 4.2 shows the probabilistic parametric transition activation from super state to sub state. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transiton set of state s ($\exists t \in \mathbb{T}_p OutT(s)$) which satisfied the guard ($v \vDash g \wedge c \vDash g$). S_{h_0} represents the set of states in hierarchy h_0, we have $s' \in S_{h_0}$ and there exists sub statefof s $Sub(s) \neq \varnothing$ and s' is the initial state of sub state of s described as $s' = Sub_I(s)$. Besides, there does not exist a transition in the outgoing transiton set of state s' ($\nexists t \in \mathbb{T} OutT(s')$) which satisfy the guard ($v' \vDash g' \wedge c' \vDash g'$). The transition probability p' is $\tau : \mathrm{p} \rightarrow [0, 1]$. Then the next state becomes (l', v', c', h_1).
   Rule 4.3 shows the probabilistic parametric transition activation from sub state to super state. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transiton set of super state of s ($\exists t \in \mathbb{T}_p OutT(Sup(s))$ which satisfy the guard ($v \vDash g \wedge c \vDash g$). S_{h_0} represents the set of states in hierarchy h_0, we have $s' \in S_{h_0}$. The transition probability p' is $\tau : \mathrm{p} \rightarrow [0, 1]$. Then the next state becomes (l', v', c', h_0).
5. $\mathbb{T}_{sp}$ **Activation:**
   We can note the stochastic probabilistic parametric transition as the combination of a stochastic transition with a probabilistic parametric transition. To make it more clear, we only provide this type of transition in super state.
   Rule 5.1 shows the stochastic probabilistic parametric transition activation in super state $s(l, v, c, h_0)$. When a trigger event occurs ($\exists e|Trg(e)$), there exists a transition in the outgoing transiton set of state s ($\exists t \in \mathbb{T}_s OutT(s)$), the new clock c' updates to $\mu(c+\kappa(\texttt{after(F)}))$ where $\texttt{after(F)}$ is the mesurement of the stochastic delay function F, μ is the clock valuation function and κ is a valuation of delay function. S_{h_0} represents the set of states in hierarchy h_0, we have $s' \in S_{h_0}$ and there is no sub state of s $Sub(s) == \varnothing$. The transition probability p' is $\tau : \mathrm{p} \rightarrow [0, 1]$. Then the next state becomes (l', v, c', h_0).

3.3. Quantitative analyzing phase: tansform stohChart(p) to networks of probabilistic timed automata

After we defined the syntax and semantics of stohChart(p), we can transform the model to networks of probabilistic hybrid automata which UPPAAL-SMC is based on. In Algorithm 1, we give the outline of the transformation of stohChart(p) to networks of probabilistic timed automata. To make it easier, first, we need to flatten the hierarchical model to one-layer statechart (line 5). Then we generate new state based on the set of states in stohChart(p) (line 6). Based on different kinds of transitions, we need to generate new transitions and updating the rate of exponential and probability weight (line 7–10). In the end, we need to link the new states and transitions together (line 11). The NPTA, which we get from the output of the algorithm, and the specifications are the input of the statistical model checker UPPAAL-SMC. We can get the quantitative analysis of the driving control for autonomous vehicles.

Algorithm 1 Transform stohChart(p) to networks of probabilistic timed automata.

Input:
1: The model of stohChart(p), $\mathcal{S}$;
2: The machine learning results– the set of valuation of parameters, p;
Output:
3: Networks of probabilistic timed automata (NPTA);
4: *Algorithm Procedure:*
5: Flatten the $\mathcal{S}$ to $\mathcal{S}'$, $\mathcal{S}' = Flatten(\mathcal{S})$;
6: Generate new states in NPTA based on the set of states in $\mathcal{S}'$;
7: Classify the transitions based on different types, $\mathbb{T} = Type(T)$. $T = OutT(s_i), s_i \in S$ the set of transitions with the source state s_i;
8: Generate new transitions with source state and target state.
9: Update the rate of exponential with the function of $\texttt{after(F)}$ stochastic transition.
10: Update the probability weight with the parameters p in transition.
11: Link the new transitions with new states.
12: **return** NPTA;

3.4. Decision phase

In the Driving Control section, we need to input the verification result and environment data to the Driving Rules. Based on different driving environment, the autonomous vehicle can decide to change lane or not depending on verification results.

1. When the vehicle is driving in the highway and the speed of the vehicle is more than 80km/h, the vehicle can change lane with a probability higher than 95%.
2. When the vehicle is driving in the downtowm and the speed of the vehicle is less than 30km/h, the vehicle can change lane with a probability higher than 80%.

4. Case study

Change Lane Decision (CLD) is an important feature in automated driving systems, where the goal is to provide correct, timely, and reliable decisions to the autonomous vehicle. To achieve the goal, vehicles are equipped with forward-facing vision and radar sensors. Sensor fusion is required to increase the probability of accurate decision and minimize the probability of false decision. To illustrate our approach, we present a case study: an autonomous vehicle (in blue) interacting with a human-driven vehicle (in red) on a shared road, as shown in Fig. 6. In the upper part of Fig. 6, scenario 1 shows the case when the human-driven vehicle slows down to make room for the autonomous vehicle to nudge in front of it. In scenario 2 of Fig. 6, it indicates that the red car refuses to

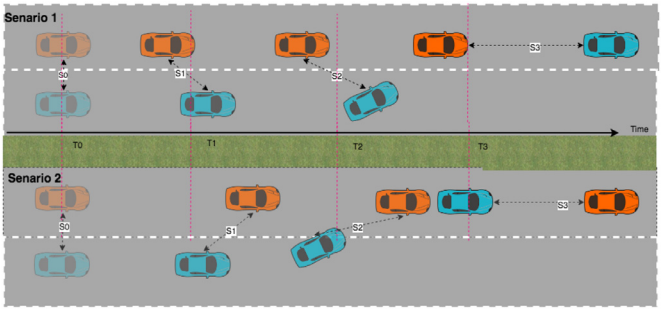

Fig. 6. A scenario of self-driven car and human-driven car.

Table 2
The dataset input to classification learner.

Categories	Data
Relation to Surrounding Vehicles	
Speed of autonomous vehicle	30–90 km/h
Relative distance	1–10 m
Relative speed	−15–15 km/h
Speed of surrounding vehicle	30–90 km/h
Overspeed percentage	$<-10\%$, -10–10%, $>10\%$
Info of Surrounding Vehicles	
Use light right	0–right,1–wrong for one time
Brake time	0–no brake, 1–break once
Other wrong behavior	0–no, 1–yes
Surrounding Conditions	
Weather	0–rainy, 1–sunny, 2–foggy
Time	0–night, 1–day
Driving Style Classification	Normal, Aggressive, Moderate

let the blue car cut in by speeding up. We define the problem as a small ITS which can be modeled by stohChart(p).

Due to the lack of experiment equipment and of data available online, we have generated an arbitrary dataset based on our own knowledge. As shown in Table 2, our data set is a matrix (150*14). 14 columns, the first 13 columns are features (predictor), and the last column is label. 150 is the number of observations. The dataset can be found in https://github.com/DongdongAn/DrivingStyleClassification.git.

4.1. Learning result as parameters

We divide the data which the sensors collect from uncertainty environment into three categories as shown in Fig. 3, including *Relation to Surrounding Vehicles, Info of Surrounding vehicles* and *Surrounding Conditions*. Table 2 shows the range of our sample numbers.

To investigate features to include or exclude, use the parallel coordinates plot. We can visualize high dimensional data on a single plot to see 2D patterns as shown in Fig. 7. This plot can help us understand relationships between features and identify useful predictors for separating classes. We can visualize training data and misclassified points on the parallel coordinates plot as shown in Fig. 5 the misclassified points show dashed lines.

Table 3 shows the classification learner result with different models and accuracy. We can see the optimizable tree, weighted KNN and subspace discriminant methods can get high accuracy with 85.5%. For the training time, the coarse tree and medium tree are most timing saving methods with 0.16574 and 0.17223 second. Given their much faster execution time and slightly lower accuracy, we choose the coarse tree and medium tree as the learning model

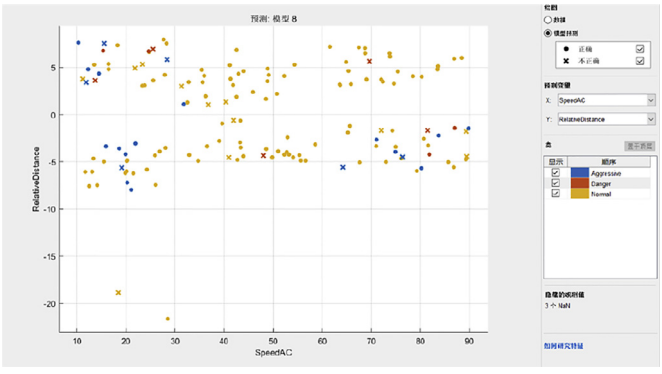

Fig. 7. Scatter Plot.

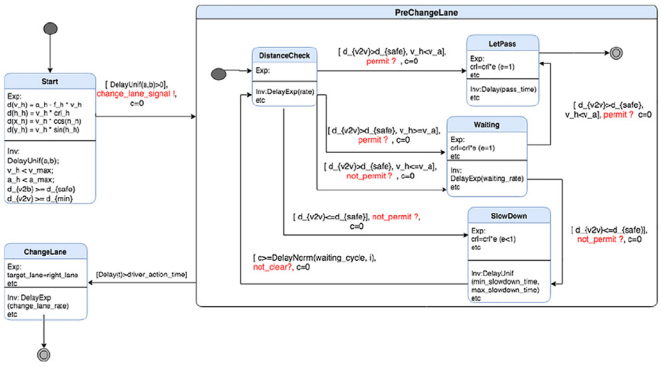

Fig. 8. The stohChart(p) model: the scenario of autonomous vehicle.

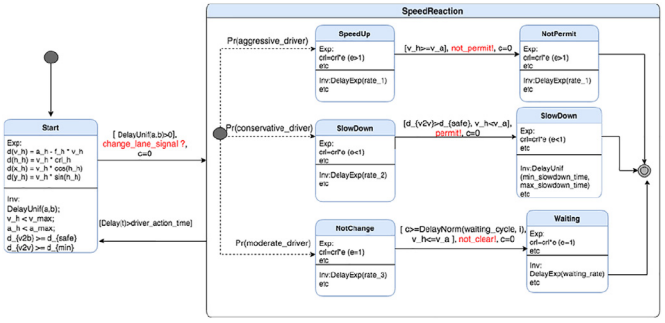

Fig. 9. The stohChart(p) model: the scenario of human-driven vehicle.

to analyze the dataset. Table 4 shows the probabilities of the three runs of learning results. We input the results as parameters to the stohChart(p) model.

4.2. Build stohMChart for the system

The model of stohChart(p) shows the autonomous car and human-driven car in Figs. 8 and 9, respectively. In Fig. 8, the autonomous car is driving along the road before it sends a signal *change_lane_signal*, after a time delay which follows a uniform distribution, it entered *DistanceCheck* state which is the start state of *PreChangeLane*. Based on the distance and velocity from the human-driven car, it can enter one of the *LetPass, Waiting* or *SlowDown* state. After it gets to the *LetPass* state, it can *ChangeLane* successfully. Fig. 9 shows the model of human-driven car. After receive the signal *change_lane_signal* and a reaction time which follows a distribution based on different human reaction time, it enters *SpeedUp, SlowDown* and *NotChange* state with a probability transition distribution according to different driving styles. Based on different control rates, it emits different signals to the autonomous car.

4.3. Mapping to UPPAAL-SMC model

After building the stohCharts, we use the transformation algorithm to transform the model to NPTA in UPPAAL-SMC in Fig. 10 and Fig. 11. The scenarios of the autonomous vehicle and human-driven are shown in Fig. 10 and Fig. 11, respectively.

4.4. Define queries

To focus on the quantitative analysis of the scenario influenced by uncertain factors such as different driving styles, we illustrate two queries of the model.

1. *Pr[<=20] (< > HumanDrive.LET_PASS)*

Table 3
Classification learner result.

Model	Accuracy	Misclassification	Prediction speed	Training time
Decision Trees				
Fine Tree	82.2%	27	12,000 obs/s	0.30234 s
Medium Tree	82.2%	27	10,000 obs/s	0.17223 s
Coarse Tree	84.2%	24	8300 obs/s	0.16574 s
Optimizable Tree	85.5%	22	10,000 obs/s	30.933 s
Naive Bayes Classifiers				
Kernel Naive Bayes	85.5%	22	2700 obs/s	1.6162 s
Optimizable Naive Bayes	86.2%	21	3200 obs/s	71.631 s
Support Vector Machines				
Linear SVM	80.9%	29	4100 obs/s	1.8722 s
Qudratic SVM	83.6%	29	5900 obs/s	0.59292 s
Cubic SVM	82.2%	27	5600 obs/s	0.57337 s
Fine Guassian SVM	70.5%	38	5700 obs/s	0.64042 s
Medium Guassian SVM	81.6%	28	5800 obs/s	0.58794 s
Coarse Guassian SVM	75.0%	38	6000 obs/s	0.57512 s
Optimizable SVM	83.6%	25	5900 obs/s	123.92 s
Nearest Neighbor Classifiers				
Fine KNN	81.6%	28	5900 obs/s	0.79366 s
Medium KNN	82.2%	27	8300 obs/s	0.24149 s
Coarse KNN	77%	35	8200 obs/s	0.2123 s
Cosine KNN	78.3%	33	7200 obs/s	0.29478 s
Cubic KNN	80.3%	30	7500 obs/s	0.25397 s
Weighted KNN	85.5%	22	7800 obs/s	0.22393 s
Optimizable Tree	87.5%	19	5400 obs/s	40.782 s
Ensemble				
Boosted Trees	78.3%	33	5300 obs/s	1.06755 s
Bagged Trees	84.2%	24	910 obs/s	2.4637 s
Subspace Discriminant	85.5%	22	780 obs/s	2.7029 s
Subspace KNN	77.6%	34	530 obs/s	2.4307 s
RUSBoosted Trees	73.0%	41	1000 obs/s	2.5364 s

Table 4
The probability distribution of different driving style.

Driving style	First time	Second time	Third time
Aggressive Driver	0.120	0.821	0.094
Conservative Driver	0.101	0.078	0.793
Moderate Driver	0.779	0.101	0.113

The first query means the probability distribution of the human-driven vehicle would let the autonomous vehicle pass in 20-time units.

2. *Pr[< =20](< > AutoDrive.CHANGE_LANE)*
The second query means the probability distribution of the autonomous vehicle would change the lane in 20-time units.

4.5. Experiment results and analysis

We set the statistical parameters of UPPAAL-SMC with $\alpha = 0.01, \varepsilon = 0.05, bucketwidth = 0.02, bucketcount = 50$. The quantitative result of the property 1 [Pr[<=20] (<> Human-Drive.LET_PASS)] is shown in Fig. 12 and the left part of Fig. 14. The x-axis denotes the time limit, and the y-axis indicates the probability density distribution. The quantitative result of the property 2 [Pr[<=20](<>AutoDrive. CHANGE_LANE)] is shown in Fig. 13 and the right part of Fig. 14.

These quantitative analysis results and the traffic information (i.e., road condition is a highway or urban road) are the input of the driving control decision procedure. The designers of the system define the driving rules. For example, suppose that the settings for change lane behavior are set as 95% (highway condition) and 90% (urban road condition), respectively. If the quantitative analysis result of the probability of the autonomous vehicle successfully changes lane in the 20-time unit is 93%, in highway condition, the safe driving control would deny the request to change lane. However, in urban road condition, it will permit the autonomous vehicle to change lane.

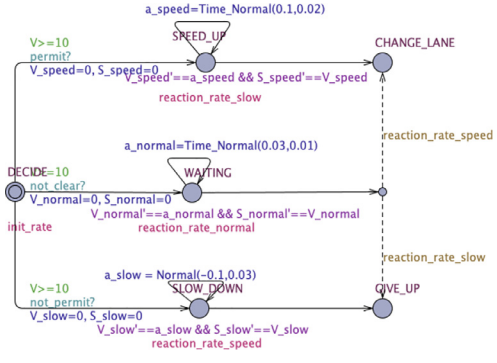

Fig. 10. The UPPAAL-SMC model: the scenario of autonomous vehicle.

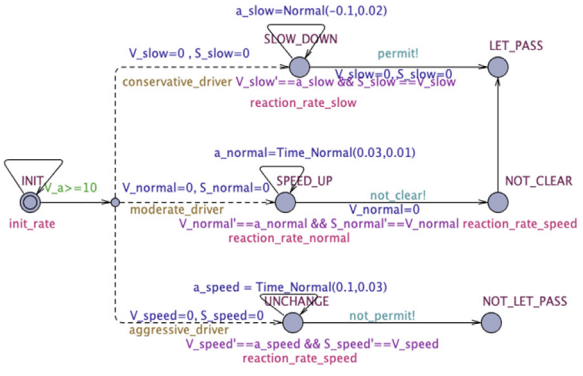

Fig. 11. The UPPAAL-SMC model: the scenario of human-driven vehicle.

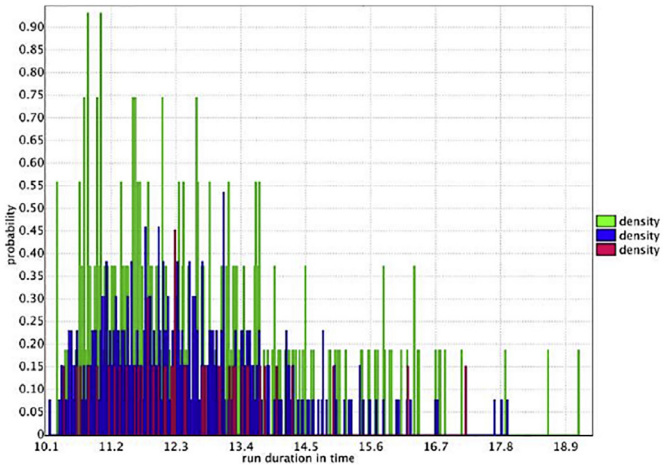

Fig. 12. The probability distribution of human-driven vehicle change lane within 20 time units based on different driving styles.

5. Related work

Many researchers and communities have put effort into recognizing and classifying the driving style. In the context of Aljaafreh et al. (2012), the fuzzy logic are used to recognize the driving style. In Han et al. (2019), Bayesian probability with kernel density estimation based on statistical analyzing method was investigated in driving style recognition. Different from Van Ly et al. (2013) using inertial sensors, Wang et al. (2017) and Zhang et al. (2015) used semi-supervised support vector machine to classify the driving patterns. Some other learning-based methods are proposed in Bai et al. (2019) and Jiang et al. (2019a) which modeled the uncertainty of driver behavior and predict driver's state using CNN and Naive Bayes classifier. In Jia and Zhang (2019) and Akintunde et al. (2019), KNN feature augmentation and RNN feature augmentation are proposed respectively. Therefore, some researchers have developed Convex Markov Chains to predict quantitative information about the driver behavior (Sadigh et al., 2014). In order to improve human-machine interactions in intelligent systems, Lin (2015) integrated control algorithms with computational models of human decision making in dynamic contexts. In contrast to our current work, most of those approaches didn't pay much attention to classify the driving style of surrounding cars. Based on the classification method we mentioned above, we use the classification learner App of MATLAB, which including various learning-based classification algorithms. We choose the most efficient algorithm binary tree to classify the driving style of the surrounding vehicles. Bai et al. (2019) modeled the uncertainty of driver behavior and predict driver's state using CNN and Naive Bayes classifier, but did not consider the surrounding vehicles driving behavior uncertainty environment. Different from Bai et al. (2019), we use data-driven classification algorithm to classigy the driving style of surrounding vehicles.

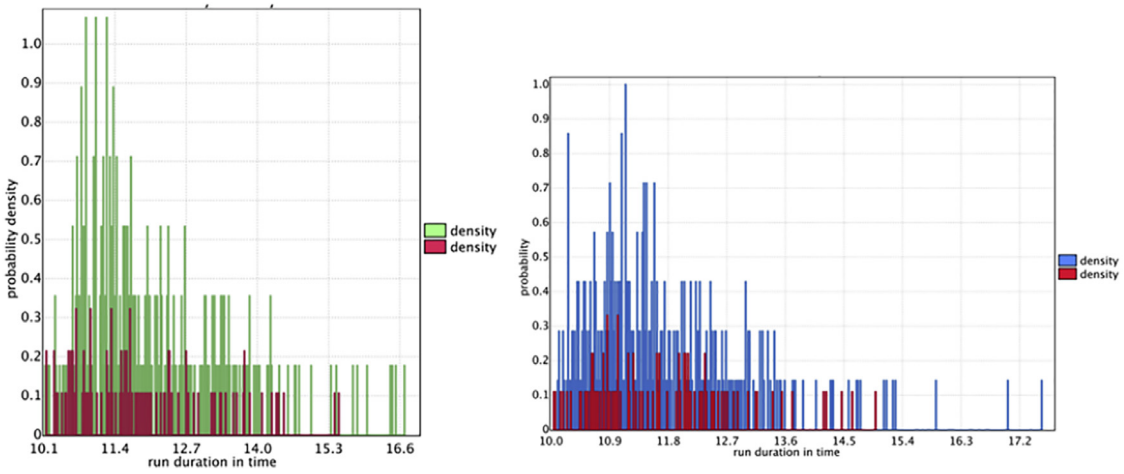

Fig. 13. The probability distribution of autonomous vehicle change lane within 20 time units based on different driving styles.

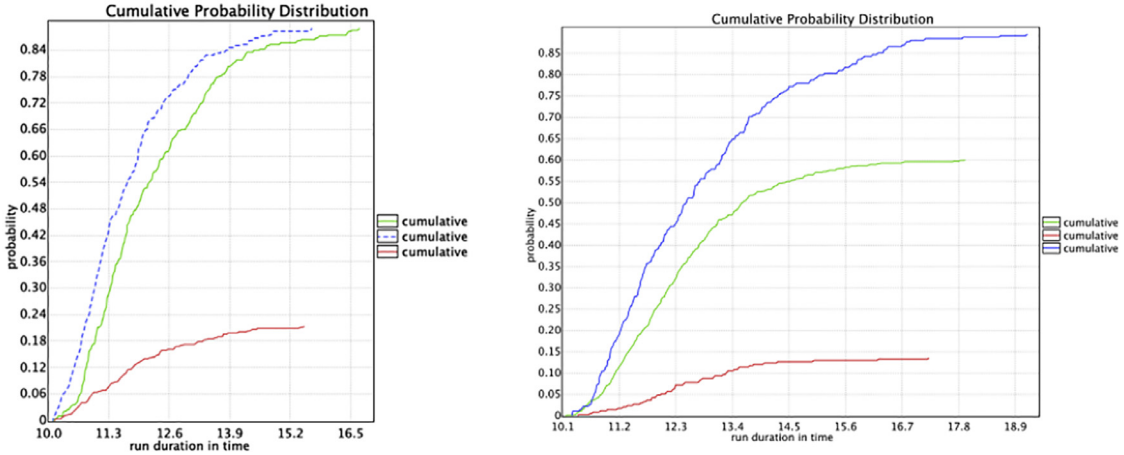

Fig. 14. The cumulative probability distribution of different driving styles.

As for modeling the stochastic continuous behaviors of the systems, Du et al. (2019) and Guan et al. (2018) proposed the modeling languages SHML and xSHS respectively to model stochastic and continuous behaviors of Cyber-Physical Systems. Some communities have developed direct and optimization based methods to model semi-autonomous systems (Lin, 2015; Shia et al., 2014; Urmson et al., 2008). Compared to the existing work, our modeling language stohChart(p) extended these languages with the parameters which can help us to analyze the ITS at runtime.

How should autonomous systems be verified is a challenging problem along with their increasing applications (Fisher et al., 2013). Zhao et al. (2019) proposed a framework for probabilistic model checking on a layered Markov model to verify the safety and reliability requirements of robots. They used Markov Decision Processes as the model to perform probabilistic model checking (Bai et al., 2019; Seshia et al., 2015; Sadigh et al., 2014). Sadigh et al. (2014) used data-driven probabilistic modeling and verification method to analyze human driver behavior. Zhao et al. (2019) proposed a framework for probabilistic model checking on a layered Markov model to verify the safety and reliability requirements of robots. The approaches in Bai et al. (2019), Fisher et al. (2013), Song et al. (2019) and Zhao et al. (2019) used the probabilistic model checker PRISM to verify the properties. In contrast, our approach uses statistical model checker UPPAAL-SMC to verify the properties.

6. Contribution and conclusion

In this paper, we introduced an approach to integrate machine learning and statistical model checking technologies to help autonomous vehicles perform safe control during lane changing. We used machine learning-based classification algorithms to get the driving style classification results by collecting the data from the uncertain environment. The learning results were transferred as parameters to the visual formal model stohChart(p). We proposed and implemented a mapping algorithm to transfer hierarchical stohChart(p) to NPTA automatically. In the end, the verification results and decision rules were used to help the autonomous vehicle make safe and intelligent decisions based on the changing environment. We also showed experimental results and verification results to demonstrate the feasibility of our approach.

Our work is still imperfect. We will unfold our future work in two directions. Firstly, we intend to build a more accurate learning classifier to help the autonomous vehicle detects the dangerous driving behavior of surrounding human drivers so that it can react to the uncertain driving environment. Secondly, we intend to consider the more complex driving condition, for example, interact with pedestrians and more surrounding vehicles.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Dongdong An: Software, Methodology, Conceptualization.

Acknowledgements

This work is partially supported by the projects funded by the National Key Research and Development Project 2019YFA0706404, NSFC Project 61972150, and Shanghai Knowledge Service Platform Project ZF1213.

References

Akintunde, M.E., Kevorchian, A., Lomuscio, A., Pirovano, E., 2019. Verification of RNN-based neural agent-environment systems. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33, pp. 6006–6013.
Aljaafreh, A., Alshabatat, N.T., Aldin, M.S.N., 2012. Driving Style Recognition Using Fuzzy Logic, pp. 460–463.
Bai, X., Xu, C., Ao, Y., Chen, B., Du, D., 2019. Learning-based Probabilistic Modeling and Verifying Driver Behavior using MDP, pp. 152–159.
Bejani, M.M., Ghatee, M., 2019. Convolutional neural network with adaptive regularization to classify driving styles on smartphones. IEEE Trans. Intell. Transp. Syst..
Bengler, K., Dietmayer, K., Farber, B., Maurer, M., Stiller, C., Winner, H., 2014. Three decades of driver assistance systems: review and future perspectives. IEEE Intell. Transp. Syst. Mag. 6 (4), 6–22.
Bolovinou, A., Amditis, A., Bellotti, F., Tarkiainen, M., 2014. Driving style recognition for co-operative driving: asurvey. In: The Sixth International Conference on Adaptive and Self-Adaptive Systems and Applications, pp. 73–78.
Ciesinski, F., Größer, M., 2004. On probabilistic computation tree logic. In: Validation of Stochastic Systems. Springer, pp. 147–188.
ClassificationLearner, 2019. Matlab Classification Learner App.
David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., 2015. Uppaal smc tutorial. Int. J. Softw. Tools Technol. Trans. 17 (4), 397–415.
David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Van Vliet, J., Wang, Z., 2011. Statistical model checking for networks of priced timed automata. In: International Conference on Formal Modeling and Analysis of Timed Systems. Springer, pp. 80–96.
Denoeux, T., 1995. A k-nearest neighbor classification rule based on dempster-shafer theory. IEEE Trans. Syst. Man Cybern. 25 (5), 804–813.
Dimitrakopoulos, G., Demestichas, P., 2010. Intelligent transportation systems. IEEE Veh. Technol. Mag. 5 (1), 77–84.
Dreiseitl, S., Ohno-Machado, L., 2002. Logistic regression and artificial neural network classification models: a methodology review. J. Biomed. Inform. 35 (5–6), 352–359.
Du, D., Guo, T., Wang, Y., 2019. Shml: stochastic hybrid modeling language for CPS behavior. In: 2019 26th Asia-Pacific Software Engineering Conference (APSEC). IEEE, pp. 220–227.
Faugere, M., Bourbeau, T., De Simone, R., Gerard, S., 2007. Marte: aso an uml profile for modeling aadl applications. In: Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on. IEEE, pp. 359–364.
Feng, Y., Pickering, S., Chappell, E., Iravani, P., Brace, C., 2019. A support vector clustering based approach for driving style classification. Int. J. Mach. Learn.Comput. 9 (3), 344–350.
Fisher, M., Dennis, L.A., Webster, M., 2013. Verifying autonomous systems. Commun. ACM 56 (9), 84–93.
Guan, C., Ao, Y., Du, D., Mallet, F., 2018. xshs: An executable domain-specific modeling language for modeling stochastic and hybrid behaviors of cyber-physical systems. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE, pp. 683–687.
Han, W., Wang, W., Li, X., Xi, J., 2019. Statistical-based approach for driving style recognition using bayesian probability with kernel density estimation. IET Intel. Transport Syst. 13 (1), 22–30.
Huang, Y., Li, L., 2011. Naive bayes classification algorithm based on small sample set. In: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems. IEEE, pp. 34–39.
Jakkula, V., 2006. Tutorial on Support Vector Machine (SVM), 37. School of EECS, Washington State University.
Jia, B.-B., Zhang, M.-L., 2019. Multi-dimensional classification via knn feature augmentation. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33, pp. 3975–3982.
Jiang, B., Wu, X., Yu, K., Chen, H., 2019. Joint semi-supervised feature selection and classification through Bayesian approach. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33, pp. 3983–3990.
Jiang, S., Chen, J., Shen, M., 2019. An interactive lane change decision making model with deep reinforcement learning. In: 2019 7th International Conference on Control, Mechatronics and Automation (ICCMA). IEEE, pp. 370–376.
Klecka, W.R., Iversen, G.R., Klecka, W.R., 1980. Discriminant Analysis, vol. 19. Sage.
Kotsiantis, S.B., Zaharakis, I., Pintelas, P., 2007. Supervised machine learning: a review of classification techniques. Emerg. Artif. Intell. Appl.Comput. Eng. 160, 3–24.
Legay, A., Delahaye, B., Bensalem, S., 2010. Statistical model checking: an overview. In: International Conference on Runtime Verification. Springer, pp. 122–135.
Lin, T., 2015. Modeling of Human Decision Making via Direct and Optimization-based Methods for Semi-Autonomous Systems. UC Berkeley.
Liu, J., Liu, Z., He, J., Mallet, F., Ding, Z., 2013. Hybrid marte statecharts. Front. Comput. Sci. 7 (1), 95–108.
Llerena, Y.R.S., Su, G., Rosenblum, D.S., 2017. Probabilistic model checking of perturbed MDPs with applications to cloud computing. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, pp. 454–464.
Lodderstedt, T., Basin, D., Doser, J., 2002. Secureuml: A uml-based modeling language for model-driven security. In: International Conference on the Unified Modeling Language. Springer, pp. 426–441.
Martinelli, F., Mercaldo, F., Orlando, A., Nardone, V., Santone, A., Sangaiah, A.K., 2018. Human behavior characterization for driving style recognition in vehicle system. Comput. Electric. Eng..

McQueen, B., McQueen, J., 1999. Intelligent Transportation Systems Architectures.
Mingers, J., 1989. An empirical comparison of pruning methods for decision tree induction. Mach. Learn. 4 (2), 227–243.
Norman, G., Parker, D., Sproston, J., 2013. Model checking for probabilistic timed automata. Formal Methods Syst. Design 43 (2), 164–190.
Qiao, B., Wu, X., 2019. Real time trajectory re-planning for autonomous vehicle lane changing in uncertain traffic. In: 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE). IEEE, pp. 1524–1529.
Sadigh, D., Driggs-Campbell, K., Puggelli, A., Li, W., Shia, V., Bajcsy, R., Sangiovanni-Vincentelli, A., Sastry, S.S., Seshia, S., 2014. Data-driven probabilistic modeling and verification of human driver behavior. 2014 AAAI Spring Symposium Series.
Safavian, S.R., Landgrebe, D., 1991. A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. 21 (3), 660–674.
Sekerinski, E., 2008. Verifying statecharts with state invariants. In: 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008). IEEE, pp. 7–14.
Seshia, S. A., Sadigh, D., Sastry, S., 2015. Formal methods for semi-autonomous driving, 148.
Shia, V.A., Gao, Y., Vasudevan, R., Campbell, K.D., Lin, T., Borrelli, F., Bajcsy, R., 2014. Semiautonomous vehicular control using driver modeling. IEEE Trans. Intell. Transp. Syst. 15 (6), 2696–2709.
Song, F., Zhang, Y., Chen, T., Tang, Y., Xu, Z., 2019. Probabilistic alternating-time μ-calculus. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33, pp. 6179–6186.
Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al., 2008. Autonomous driving in urban environments: boss and the urban challenge. J. Field Rob. 25 (8), 425–466.
Utkin, L.V., 2019. An imprecise extension of SVM-based machine learning models. Neurocomputing 331, 18–32.
Van Ly, M., Martin, S., Trivedi, M. M., 2013. Driver classification and driving style recognition using inertial sensors, 1040–1045.
Wang, W., Xi, J., Chen, H., 2014. Modeling and recognizing driver behavior based on driving data: a survey. Math. Problems Eng. 2014.
Wang, W., Xi, J., Chong, A.Y.K., Li, L., 2017. Driving style classification using a semisupervised support vector machine. IEEE Trans. Hum. Mach. Syst. 47 (5), 650–660.
Wang, W., Xi, J., Liu, C., Li, X., 2016. Human-centered feed-forward control of a vehicle steering system based on a driver's path-following characteristics. IEEE Trans. Intell. Transp. Syst. 18 (6), 1440–1453.
Weilkiens, T., 2011. Systems Engineering with SysML/UML: Modeling, Analysis, Design. Elsevier.
Weinberger, K.Q., Saul, L.K., 2009. Distance metric learning for large margin nearest neighbor classification. J. Mach. Learn. Res. 10 (Feb), 207–244.
Zhang, X., Wu, G., Dong, Z., Crawford, C., 2015. Embedded feature-selection support vector machine for driving pattern recognition. J. Frankl. Inst.-Eng.Appl. Math. 352 (2), 669–685.
Zhao, X., Robu, V., Flynn, D., Dinmohammadi, F., Fisher, M., Webster, M., 2019. Probabilistic model checking of robots deployed in extreme environments. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33, pp. 8066–8074.

Dongdong An received the B.S. degree in software engineering from East China Normal University, Shanghai, China, in 2013. She is currently finishing her Ph.D. degree in software engineering at East China Normal University, Shanghai, China. From 2016.10 to 2018.4, she got a scholarship from China Scholarship Council (CSC) to work as a joint Ph.D. student in KAIROS team in the French Institute for Research in Computer Science and Automation (INRIA), France. Her research interests include model-driven architecture, machine learning, formal methods and statistical model checking techniques.

Jing Liu is currently a professor of computer science in software engineering institute at East China Normal University, Shanghai, China. In recent years, she has been involved in the area of model-driven architecture. Currently, she focuses on the design of safety-critical systems and cyber-physical systems.

Min Zhang received his B.S. degree in computer science from Shandong Normal University, Jinan, China, in 2005, the M.S. degree in software theory from Shanghai Jiao Tong University, Shanghai, China, in 2008, and the Ph.D. degree in software science from Japan Advanced Institute of Science and Technology (JAIST), Nomi, Japan, in 2011. From 2011 to 2014, he was a Postdoctoral Researcher at JAIST. Afterward, he joined East China Normal University (ECNU), Shanghai, China, as an Associate Professor. He is currently the director of the Department of Software Science and Technology in ECNU. His research interests include formal methods, programming languages, and software engineering.

Xiaohong Chen received the Ph.D. degree from the Academy of Mathematics and System Science, Chinese Academy of Sciences, in 2010. She is currently an Associate Professor at East China Normal University. Her research interests include requirements engineering, model-driven development, and cyber-physical systems.

Mingsong Chen received the B.S. and M.E. degrees from Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2003 and 2006 respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2010. He is currently a Professor with the Software Engineering Institute at East China Normal University. His research interests are in the area of design automation of cyber-physical systems, cloud computing, parallel and distributed systems, and formal verification techniques.

Haiying Sun was born in 1976. She received the master's degree in computer science and technology from the NanJing University Science and Technology in 2001. She earned her Ph.D. in Formal Method From East China Normal University in 2017. She is now a lecturer at East China Normal University. Her research interests include formal method, automatic test generation, system simulation and model-driven engineering.