Deep-Learning-Based Intrusion Detection for Autonomous Vehicle-Following Systems

Sheng-Li Wang[1], Sing-Yao Wu[1], Ching-Chu Lin[1], Srivalli Boddupalli[2],
Po-Jui Chang[1], Chung-Wei Lin[1], Chi-Sheng Shih[1], and Sandip Ray[2]

***Abstract*— Autonomous vehicle-following systems, including Adaptive Cruise Control (ACC) and Cooperative Adaptive Cruise Control (CACC), improve safety, efficiency, and string stability for a vehicle (the ego vehicle) following its leading vehicle. The ego vehicle senses or receives information, such as the position, velocity, acceleration, or even intention, of the leading vehicle and controls its own behavior. However, it has been shown that sensors and wireless channels are vulnerable to security attacks, and attackers can modify data sensed from sensors or received from other vehicles. To address this problem, in this paper, we design three types of *stealthy* attacks on ACC or CACC inputs, where the stealthy attacks can deceive a rule-based detection approach and impede system properties (collision-freeness and vehicle-following distance). We then develop two deep-learning models, a predictor-based model and an encoder-decoder-based model to detect the attacks, where the two models do not need attacker models for training. The experimental results demonstrate the respective strengths of different models and lead to a methodology for the design of learning-based intrusion detection approaches.**

I. Introduction

Intelligent vehicles have become a well-known concept that vehicles can assist human drivers or even behave autonomously with many advanced functions. Several Advanced Driver Assistance Systems (ADAS) aim to support vehicles to be safer and more efficient. Adaptive Cruise Control (ACC) is a representative one which controls a vehicle to maintain a safe gap to its leading vehicle. To realize the functionality, a vehicle equipped with ACC uses radars or other sensors to sense the relative position and velocity of its leading vehicle and combines the information to control its acceleration. Cooperative Adaptive Cruise Control (CACC) is a more advanced version of ACC. A vehicle equipped with CACC utilizes Vehicular Ad-hoc Networks (VANETs) such as Dedicated Short-Range Communications (DSRC) or Cellular V2X (C-V2X) and receives the information of its leading vehicle. Because of the connectivity between vehicles, more information, including the position, velocity, acceleration, or even intention of the leading vehicle, can be provided from the leading vehicle to further improve safety, efficiency, and string stability.

*This work is partially supported by National Science Foundation (NSF) under Grant Number CNS-1908549, Ministry of Education (MOE) in Taiwan under Grant Number NTU-109V0901, and Ministry of Science and Technology (MOST) in Taiwan under Grant Numbers MOST-109-2636-E-002-022 and MOST-110-2636-E-002-026.

*Email: cwlin@csie.ntu.edu.tw
[1]National Taiwan University, Taipei 10617, Taiwan
[2]University of Florida, Gainesville, FL 32611

However, research has shown the feasibility of hacking sensors such as GPS, radar, LIDAR, and camera [1], [2], [3]. The wireless channels of VANETs also have robustness and security concerns. They provide attackers the opportunities of intrusion and modifying data sensed from sensors or sent from other vehicles. If an attacker modifies ACC or CACC inputs, *i.e.*, a spoofing attack, it will impair the safety, efficiency, and string stability of ACC or CACC and can even create traffic jams and collisions between vehicles. Several existing works have analyzed the impact of attacks on CACC [4], [5], [6], which suggest that mitigation approaches should be provided to protect against those attacks.

One approach to protect against malicious modifications is cryptographic authentication. However, it can only verify the legitimacy of an identification, so it is difficult to protect against data-level attacks. For example, the sensors of an authenticated vehicle are compromised, or an authenticated vehicle itself provides false information. Besides, cryptographic authentication also has the concern of computational overhead.

For data-level protection, data-level Intrusion Detection Systems (IDS) aim to check the consistency of data in time series and detect false data. Recent research applies Hidden Markov Models (HMM) [6], Pearson correlation analysis [7], and Principal Components Analysis (PCA) [8] to the protection against insiders. The general principle of these works is to model and learn the normal behavior of vehicles and detect intrusion by checking if the data is anomalous to the pattern of the learned normal behavior. On the other hand, many deep-learning-based approaches are also proposed to solve general anomaly detection problems [9], [10], [11], [12]. Especially, deep learning is also applied to intrusion detection in the automotive domain [13], [14]. However, most of these deep-learning-based approaches require attacker models for training, which limits their performance of detecting attacks that are not seen or trained before. Therefore, we aim to develop deep-learning-based approaches which do not need attacker models for training.

In this paper, our main contributions include:

- We design three types of *stealthy* attacks on ACC or CACC inputs, where the stealthy attacks can deceive a rule-based detection approach and impede system properties (collision-freeness and vehicle-following distance).
- We develop two deep-learning models, a predictor-based model and an encoder-decoder-based model to

detect the attacks, where the two models do not need attacker models for training.
- The experimental results demonstrate the respective strengths of different models and lead to a methodology for the design of learning-based intrusion detection approaches.

The rest of this paper is organized as follows. Section II describes our problem formulation including the attacker models, detection goal, and detection evaluation. Section III presents our approaches including the predictor-based model and the encoder-decoder-based model. Section IV demonstrates and discusses the experimental results. Section V concludes this paper.

II. Problem Formulation

We consider a scenario that multiple vehicles driving on a lane from a platoon. Each vehicle follows its leading vehicle based on a vehicle-following model and a safe target velocity [15]. The model has at most 6 inputs including the positions, velocities and accelerations of the ego vehicle and its leading vehicle. Each vehicle is equipped with sensors and communication devices so that it can sense its own position, velocity, and acceleration and sense or receive the position, velocity, and acceleration of its leading vehicle. The position, velocity and acceleration of a vehicle are correlated by physics rules as the following equations [6]:

$$v_{min} \cdot \Delta + \frac{1}{2} a_{min} \cdot \Delta^2 - \epsilon_p \le p_{new} - p_{old}, \quad (1)$$

$$p_{new} - p_{old} \le v_{max} \cdot \Delta + \frac{1}{2} a_{max} \cdot \Delta^2 + \epsilon_p, \quad (2)$$

$$a_{min} \cdot \Delta - \epsilon_v \le v_{new} - v_{old} \le a_{max} \cdot \Delta + \epsilon_v, \quad (3)$$

where Δ is the time interval between two measurements, p_{old}, p_{new}, v_{old}, v_{new}, a_{old}, and a_{new} are respectively the old (right before a time interval) and new (right after the time interval) positions, velocities, and accelerations of the vehicle, $v_{min} = \min(v_{old}, v_{new})$, $a_{min} = \min(a_{old}, a_{new})$, and ϵ_p and ϵ_v are respectively the acceptable error ranges.

An attacker can modify any arbitrary input of a vehicle-following model. If the attacker only attacks one input, we can detect it by Equations (1–3). Therefore, we consider a stronger attacker which performs *stealthy* attacks and modifies multiple inputs (position, velocity, and acceleration) of the ego vehicle or the leading vehicle. The stealthy attacks are worth detection as we will show that they can impede system properties such as generating collisions and change vehicle-following distances.

However, we do not consider an attack which modify all inputs. This is because different attacking techniques are needed to compromise different sensors and messages, resulting in a higher attacking cost. Also, there is no data-level solution if all inputs are compromised, so this is out of the scope of this paper. In this paper, we design three stealthy attacks and introduce them in the following sections.

A. Attack 1: Shifting Stealthy Attack

Since a vehicle-following model is based on a safe target velocity which is usually the velocity of the leading vehicle. We design the first two attacks by modifying the velocity of the leading vehicle in a vehicle-following model. To make the attacks stealthy, the position and acceleration of the leading vehicle are also modified accordingly by physics rules so that the attacks can deceive the rule-based detection based on Equations (1–3). Note that the modifications are on the (sensed or received) inputs of a vehicle-following model, so the physical behavior of the leading vehicle is actually not modified.

A shifting stealthy attack adds a constant to the velocity of the leading vehicle. The false input $v^{0(t)}$ at time step t in a vehicle-following model can be written as:

$$v^{0(t)} = \begin{cases} \gamma + v^{(t)}, & \text{if } (t - t_{start} \mod T) < U; \\ v^{(t)}, & \text{otherwise,} \end{cases} \quad (4)$$

where $v^{(t)}$ is the true velocity of the leading vehicle, t_{start} is the time step that the attack starts, γ is the intensity of the attack, and T and U are respectively the period and the duration of the attack.

B. Attack 2: Scaling Stealthy Attack

A scaling stealthy attack multiplies a constant to the velocity of the leading vehicle. The false input $v^{0(t)}$ at time step t in a vehicle-following model can be written as:

$$v^{0(t)} = \begin{cases} \gamma \cdot v^{(t)}, & \text{if } (t - t_{start} \mod T) < U; \\ v^{(t)}, & \text{otherwise,} \end{cases} \quad (5)$$

where $v^{(t)}$ is the true velocity of the leading vehicle, t_{start} is the time step that the attack starts, γ is the intensity of the attack, and T and U are respectively the period and the duration of the attack. Similar to a shifting stealthy attack, the position and acceleration of the leading vehicle are also modified accordingly by physics rules to make the attacks stealthy.

C. Attack 3: Time-Variant Stealthy Attack

A time-variant stealthy attack adds offset functions $O_p(t)$, $O_v(t)$, and $O_a(t)$ to the real position, velocity, and acceleration to get false position p^0, velocity v^0, and acceleration a^0, respectively. Therefore, when the measurements at time t are received, the modified inputs can be written as:

$$p^0_{new} = p_{new} + O_p(t), \quad p^0_{old} = p_{old} + O_p(t-1), \quad (6)$$

$$v^0_{new} = v_{new} + O_p(t), \quad v^0_{old} = v_{old} + O_p(t-1), \quad (7)$$

$$a^0_{new} = a_{new} + O_p(t), \quad a^0_{old} = a_{old} + O_p(t-1). \quad (8)$$

Here we assume $O_p(t)$, $O_v(t)$, and $O_a(t)$ as continuous functions (use notation $O_p(t)$ instead of $O_p^{(t)}$) and use time steps to discretize them for convenience. Without loss the generality, we assume that the new values are larger than the old values, *i.e.*,

$$v^0_{min} = v^0_{old}, \; v^0_{max} = v^0_{new}, \; a^0_{min} = a^0_{old}, \; a^0_{max} = a^0_{new}. \quad (9)$$

By plugging Equation (9) into Equations (1–3), a time-variant stealthy attack needs to satisfy all of the following equations:

$$v_{old}^0 \cdot \Delta + \frac{1}{2} a_{old}^0 \cdot \Delta^2 - \varepsilon_p \le p_{new}^0 - p_{old}^0 , \quad (10)$$

$$p_{new}^0 - p_{old}^0 \le v_{new}^0 \cdot \Delta + \frac{1}{2} a_{new}^0 \cdot \Delta^2 + \varepsilon_p , \quad (11)$$

$$a_{old}^0 \cdot \Delta - \varepsilon_v \le v_{new}^0 - v_{old}^0 \le a_{new}^0 \cdot \Delta + \varepsilon_v , \quad (12)$$

to deceive the rule-based detection. Then, by plugging Equations (6–8) into Equations (10–12), a time-variant stealthy attack needs to determine the offset functions to satisfy all of the following equations:

$$\begin{aligned} &(v_{old} + O_p(t-1))\Delta + \frac{1}{2}(a_{old} + O_a(t-1))\Delta^2 - \varepsilon_p \\ &\le p_{new} + O_p(t) - p_{old} - O_p(t-1) , \end{aligned} \quad (13)$$

$$\begin{aligned} &p_{new} + O_p(t) - p_{old} - O_p(t-1) \\ &\le (v_{new} + O_p(t))\Delta + \frac{1}{2}(a_{new} + O_a(t))\Delta^2 + \varepsilon_p , \end{aligned} \quad (14)$$

$$\begin{aligned} &(a_{old} + O_a(t-1))\Delta - \varepsilon_v \\ &\le v_{new} + O_v(t) - v_{old} - O_v(t-1) \\ &\le (a_{new} + O_a(t))\Delta + \varepsilon_v , \end{aligned} \quad (15)$$

to deceive the rule-based detection.

Similar to Equation (9), assuming that the new values are larger than the old values for the CACC inputs, *i.e.*,

$$v_{min} = v_{old} , v_{max} = v_{new} , a_{min} = a_{old} , a_{max} = a_{new} , \quad (16)$$

the real inputs must satisfy all of the following equations:

$$v_{old} \cdot \Delta + \frac{1}{2} a_{old} \cdot \Delta^2 - \varepsilon_p \le p_{new} - p_{old} , \quad (17)$$

$$p_{new} - p_{old} \le v_{new} \cdot \Delta + \frac{1}{2} a_{new} \cdot \Delta^2 + \varepsilon_p , \quad (18)$$

$$a_{old} \cdot \Delta - \varepsilon_v \le v_{new} - v_{old} \le a_{new} \cdot \Delta + \varepsilon_v . \quad (19)$$

By combining Equations (13–15) and Equations (17–19), a time-variant stealthy attack needs to determine the offset functions to satisfy all of the following equations:

$$O_v(t-1) \cdot \Delta + \frac{1}{2} O_a(t-1) \cdot \Delta^2 \le O_p(t) - O_p(t-1) , \quad (20)$$

$$O_p(t) - O_p(t-1) \le O_v(t) \cdot \Delta + \frac{1}{2} O_a(t) \cdot \Delta^2 , \quad (21)$$

$$O_a(t-1) \cdot \Delta \le O_v(t) - O_v(t-1) \le O_a(t) \cdot \Delta , \quad (22)$$

to deceive the rule-based detection.

To design a time-variant stealthy attack, we set $O_p(0) = O_v(0) = O_a(0) = 0$, and one solution which is built to satisfy Equations (20–22) is:

$$O_p(t) = \Delta^2 \gamma \left(\frac{1}{6} t^3 + \frac{1}{4} t^2 + \frac{1}{12} t \right) , \quad (23)$$

$$O_v(t) = \frac{\Delta \gamma}{2} t^2 , \quad (24)$$

$$O_a(t) = \gamma t , \quad (25)$$

where γ is the intensity of the time-variant stealthy attack. Therefore, the modified (false) inputs can be written as:

$$p^{0(t)} = p^{(t)} + O_p(t - t_{start}) , \quad (26)$$

$$v^{0(t)} = v^{(t)} + O_v(t - t_{start}) , \quad (27)$$

$$a^{0(t)} = a^{(t)} + O_a(t - t_{start}) . \quad (28)$$

The time-variant stealthy attack guarantees to deceive the rule-based detection when the actual inputs satisfy the assumption in Equation (16). The assumption is a sufficient but not necessary condition, *i.e.*, the time-variant stealthy attack can still deceive the rule-based detection if the assumption is not satisfied, especially when γ is large enough. Preliminary testing also demonstrates that the detection performance of the rule-based detection keeps decreasing as γ increases, and the time-variant stealthy attacker can totally deceive the rule-based detection when γ is large enough, regardless of the assumption.

D. Evaluation Metrics

A detector can be installed at a roadside unit or any vehicle in or nearby the platoon. The sensed or received inputs at time step t are written as a *trajectory vector* with dimension n:

$$\mathbf{r}^{(t)} = \left[r_1^{(t)}, r_2^{(t)}, \ldots, r_n^{(t)} \right] . \quad (29)$$

Here, $\mathbf{r}^{(t)}$ may consist of the following inputs: the position, velocity, and acceleration of the ego vehicle, and the position, velocity, and acceleration of its leading vehicle, and thus n is at most 6. We combine trajectory vectors at different time steps and form a time series of trajectory vectors:

$$\mathbf{R} = \left[\mathbf{r}^{(1)}, \mathbf{r}^{(2)}, \ldots, \mathbf{r}^{(t)} \right] . \quad (30)$$

We can further split R into smaller *trajectory windows* with a window size w:

$$\mathbf{w}^{(t)} = \left[\mathbf{r}^{(t-w+1)}, \mathbf{r}^{(t-w+2)}, \ldots, \mathbf{r}^{(t)} \right] . \quad (31)$$

The goal of intrusion detection is that, given a trajectory window $\mathbf{w}$, determine if it has false data.

We use the F_1 score to evaluate the detection performance. It is computed as follows:

$$\text{Precision} = \frac{TP}{TP + FP} , \quad (32)$$

$$\text{Recall} = \frac{TP}{TP + FN} , \quad (33)$$

$$F_\beta \text{ Score} = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} , \quad (34)$$

where TP (true positive) is the number of time steps that abnormal trajectory windows are correctly detected as false data; FP (false positive) is the number of time steps that normal trajectory windows are wrongly detected as false data; TN (true negative) is the number of time steps that normal trajectory windows are correctly classified as real data; FN (false negative) is the number of time steps that abnormal trajectory windows are wrongly classified as real

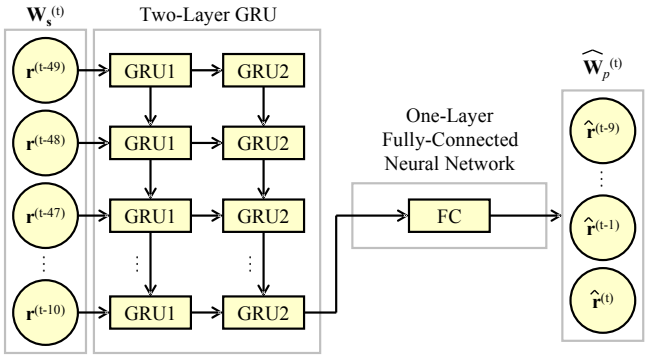

Fig. 1. The structure of an example predictor-based model with $w = 50$, $l_s = 40$, and $l_p = 10$.

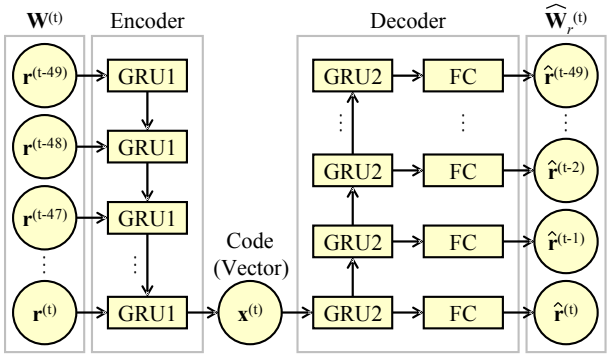

Fig. 2. The structure of an example encode-decoder-based model with $w = 50$.

data. We set $\beta = 1$, in which precision is considered the same important as recall.

If there is no false data, the precision and the recall will be meaningless, so we have the accuracy to observe the true-negative and false-positive rates in this case. It is computed as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}. \tag{35}$$

III. Proposed Approaches

In this section, we describe two developed deep-learning models for the intrusion detection. The deep-learning models do not need anomaly or intrusion data for training, as they are developed to learn the normal behavior and detect modified data even if the attacks are not seen or trained before.

A. Model 1: Predictor-Based Model

We apply the LSTM-AD [9] to develop our predictor-based model. The LSTM-AD utilizes the Long Short-Term Memory (LSTM) to predict the normal behavior of a time series. Here, we describe how we apply and modify the LSTM-AD to develop our predictor-based model.

At each time step t in a time series of trajectory vectors $\mathbf{R}$, we define a window size w to create a window using previous trajectory vectors:

$$\mathbf{W}^{(t)} = \left[\mathbf{r}^{(t-w+1)}, \mathbf{r}^{(t-w+2)}, \ldots, \mathbf{r}^{(t)}\right]. \tag{36}$$

Each window is further split into two parts, the input time series $\mathbf{W}_s^{(t)}$ with time step length l_s and the target time series $\mathbf{W}_p^{(t)}$ with time step length l_p:

$$\mathbf{W}_s^{(t)} = \left[\mathbf{r}^{(t-w+1)}, \mathbf{r}^{(t-w+2)}, \ldots, \mathbf{r}^{(t-w+l_s)}\right], \tag{37}$$

$$\mathbf{W}_p^{(t)} = \left[\mathbf{r}^{(t-l_p+1)}, \mathbf{r}^{(t-l_p+2)}, \ldots, \mathbf{r}^{(t)}\right], \tag{38}$$

where $w = l_s + l_p$. We use $\mathbf{W}_s^{(t)}$ as the input of our predictor-based model to predict the target time series $\mathbf{W}_p^{(t)}$, and the output of the predictor-based model is:

$$\widehat{\mathbf{W}}_p^{(t)} = \left[\hat{\mathbf{r}}^{(t-l_p+1)}, \hat{\mathbf{r}}^{(t-l_p+2)}, \ldots, \hat{\mathbf{r}}^{(t)}\right]. \tag{39}$$

The structure of an example predictor-based model with $w = 50$, $l_s = 40$, and $l_p = 10$ is shown in Figure 1, consisting of two-layer Gated Recurrent Units (GRUs) and a one-layer fully-connected neural network (FC in the figure). The GRU, as a kind of Recurrent Neural Networks (RNNs), is similar to LSTM, but it has fewer parameters than LSTM. We also add dropout to each layer of GRU and FC to reduce overfitting. Here, we predict the distance between the ego vehicle and its leading vehicle instead of directly predicting the positions of the ego vehicle and its leading vehicle. We observe that this setting can increase the detection performance of the predictor-based model. This is because the distance can provide a better connection between the position of the ego vehicle and that of the leading vehicle, and the distance is the information that directly influences the decision of CACC. Meanwhile, the range of distances is usually smaller than that of positions, so the predictor-based model is more sensitive to the distance modifications, compared with the corresponding position modifications.

After the predictor-based model outputs $\widehat{\mathbf{W}}_p^{(t)}$, for any time step t^0, where $t - l_p + 1 \le t^0 \le t$, we compute the prediction error $\mathbf{e}^{(t^0)}$ between the target trajectory vector $\mathbf{r}^{(t^0)}$ and the corresponding output $\hat{\mathbf{r}}^{(t^0)}$ as:

$$\mathbf{e}^{(t^0)} = \hat{\mathbf{r}}^{(t^0)} - \mathbf{r}^{(t^0)}. \tag{40}$$

Note that $\mathbf{e}^{(t^0)}$ is a vector with dimension 3 since we only predict the distance between two vehicles and the velocities and accelerations of the ego vehicle. We then sum up each dimension and get the anomaly score. If the anomaly score is higher than a predefined threshold (based on observations from testing), we determine that the corresponding trajectory window has false data. An alternative of the predefined threshold is dynamic thresholding [12].

B. Model 2: Encoder-Decoder-Based Model

We apply the EncDec-AD [10] to develop our encoder-decoder-based model. The EncDec-AD utilizes an LSTM-based encoder-decoder to compress and reconstruct a time series. Here, we describe how we apply and modify the EncDec-AD to develop our encoder-decoder-based model.

For our encoder-decoder-based model, we also define a window size w to create a window using previous trajectory vectors. However, we do not split each window into the input time series and the target time series. Instead, we directly use

the whole window $\mathbf{W}_r^{(t)}$ as the input of our encoder-decoder-based model to reconstruct the window, and the output of the encoder-decoder-based model is:

$$\hat{\mathbf{W}}_r^{(t)} = \left[\hat{\mathbf{r}}^{(t-w+1)}, \hat{\mathbf{r}}^{(t-w+2)}, \ldots, \hat{\mathbf{r}}^{(t)}\right]. \quad (41)$$

The structure of an example encoder-decoder-based model with $w = 50$ is shown in Figure 2. consisting of one-layer GRUs for the encoder and one-layer GRUs connected with a fully-connected neural network (FC in the figure) for the decoder. We also add dropout to each layer of GRU and FC to reduce overfitting. Besides, based on the observations from testing, we add an additional dimension which is the distance between the ego vehicle and its leading vehicle. Unlike the predictor-based model, we observe that reconstructing the positions of the ego vehicle and its leading vehicle can increase the detection performance of the encoder-decoder-based model. This is because the positions of one vehicle at different time steps can reflect the variations of the velocity and the acceleration, which assists the encoder-decoder-based model to encode the trajectory vectors better. Therefore, we reconstruct the whole window with the positions of the ego vehicle and its leading vehicle in the encoder-decoder-based model.

After the encoder-decoder model outputs $\hat{\mathbf{W}}_r^{(t)}$, for any time step t^0, where $t - w + 1 \le t^0 \le t$, we compute the reconstruction error $\mathbf{e}^{(t^0)}$ between the target trajectory vector $\mathbf{r}^{(t^0)}$ and the corresponding output $\hat{\mathbf{r}}^{(t^0)}$ as:

$$\mathbf{e}^{(t^0)} = \hat{\mathbf{r}}^{(t^0)} - \mathbf{r}^{(t^0)}. \quad (42)$$

Note that $\mathbf{e}^{(t^0)}$ here is a vector with dimension 7 since we reconstruct the positions, velocities, and accelerations of the ego vehicle and its leading vehicle and their distance. We then sum up each dimension and get the anomaly score. If the anomaly score is higher than a predefined threshold (based on observations from testing), we determine that the corresponding trajectory window has false data. An alternative of the predefined threshold is dynamic thresholding [12].

IV. Experimental Results

A. Experimental Setup

We used Simulation of Urban MObility (SUMO) [16] to generate trajectories for training and testing data. We built one straight lane with length 1,000 meters in SUMO and simulated dense traffic (which triggers vehicle-following systems). For the shifting stealthy attack in Section II-A and the scaling attack in Section II-B, we directly modified the velocity of the leading vehicle in the Krauss vehicle-following model in SUMO to create false data. For the time-variant stealthy attack in Section II-C, we first used the CACC vehicle-following model in SUMO to generate data and then modified the data after the SUMO simulation (note that we cannot directly modify the position, velocity, and acceleration at the same time in the SUMO simulation due to its setting) to create false data. The training data and testing data were collected in 500,000 simulation steps and 10,000 simulation steps, respectively, and the interval between each simulation step is 0.1 second. The neural network structures were implemented using PyTorch library [17]. All the experiment were run on the desktop with Intel®Core i7-9700 CPU, and NVIDIA-2080Ti GPU.

TABLE I
Experimental results (accuracy) without attacks. D is the average distance between vehicles.

D (m)	PHY	HMM-L	HMM-M	PDT	EDC
31.93	1.00	0.55	0.99	0.97	0.85

We compare the predictor-based (PDT) model and the encoder-decoder (EDC) model as follows with the rule-based detection approach (PHY) based on physics rules [6] as well as the Hidden Markov Model (HMM) model [6].

- For the comparative PHY detection, we check if the inputs satisfy Equations (1–3), and set $_v = 10^{-4}$ (m/s) and $_p = 10^{-4}$ (m).
- For the comparative HMM model, we build it with a Gaussian model by hmmlearn, a toolkit separated from scikit-learn library [18]. We use the distance, the relative velocity, and the relative acceleration between the ego vehicle and its leading vehicle, as the inputs of the HMM model. The number of hidden states is 2, same as that in [6]. To compare with the HMM model which has better detection performance, the HMM model is experimented with two thresholds (335 and 550), denoted as HMM-L and HMM-H, respectively. They are two settings performing well when there is false data and when there is no false data, respectively.
- The window size w is set as 50 to split the time series of trajectory vectors $\mathbf{R}$ into trajectory windows for the HMM model, the PDT model, and the EDC model. Two consecutive trajectory windows have 40 overlapping simulation steps, *i.e.*, the windows shift 10 simulation steps.
- For the PDT model, the length of the input time series l_s is 40, and the length of the target time series l_p is 10. The number of the hidden states of every GRU layer is 120. We train the model with batch size 128, and the optimizer is Adam with learning rate 10^{-5}.
- For the ENC model, the number of the hidden states of every GRU layer is 200. We train the model with batch size 128, and the optimizer is Adam with learning rate 10^{-5}.

Besides the accuracy (used if there is no false data) and the F_1 score defined in Section II-D, we also consider scenarios that collisions happen and report the average distances between vehicles to demonstrate the impacts of attacks on vehicle-following systems.

B. Preliminary Experiment without Attacks

It is important for an approach not to have a high false-positive rate when there is no false data, so we start from this experiment without attacks and report the accuracy. The experimental results are listed in Table I. When there is no false data, the PHY detection, the HMM-M model, and the

TABLE II

EXPERIMENTAL RESULTS (F_1 SCORES) WITH SHIFTING STEALTHY ATTACKS. γ IS THE INTENSITY OF AN ATTACK, AND D IS THE AVERAGE DISTANCE BETWEEN VEHICLES.

γ	D (m)	PHY	HMM-L	HMM-M	PDT	EDC
−10	40.09	< 0.01	0.69	0.01	0.87	0.78
−5	32.83	< 0.01	0.71	0.01	0.85	0.82
2	33.21	< 0.01	0.73	0.77	0.79	0.73
5	37.38	< 0.01	0.64	0.11	0.67	0.60

TABLE III

EXPERIMENTAL RESULTS (F_1 SCORES) WITH SHIFTING STEALTHY ATTACKS GENERATING COLLISIONS. γ IS THE INTENSITY OF AN ATTACK, AND D IS THE AVERAGE DISTANCE BETWEEN VEHICLES.

γ	D (m)	PHY	HMM-L	HMM-M	PDT	EDC
−10	No Collision					
−5	No Collision					
2	21.62	< 0.01	0.73	0.94	0.90	0.79
5	22.97	< 0.01	0.59	0.02	0.85	0.63

PDT model achieve high accuracy. However, we will see that the F_1 scores of the PHY detection and the HMM-M model are lower or much less stable when there is false data in the following experiments. Also, the average execution times of the PHY detection, the HMM models, the PDT model, and the ENC model to check 1-second CACC inputs are 3.7 (ms), 1.3 (ms), 4.6 (ms), and 8.1 (ms), respectively. The execution times in the following experiments also fall in this range, showing that all of them are applicable in real time, even on an embedded platform (the minimum requirement is to check 1-second CACC inputs in 1 second).

C. Experiment with Shifting Stealthy Attacks

The shifting stealthy attacks are defined in section II-A. In this experiment, we evaluate different intensities $\gamma = -10, -5, 2, 5$ of the attacks. A positive γ means that the false data make a vehicle overestimating the velocity of its leading vehicle, and a negative γ means that the false data make a vehicle underestimating the velocity of its leading vehicle. For γ larger than 5, the experiments results are similar to those with $\gamma = 5$.

The experimental results are listed in Table II. As the attacks are stealthy, the PHY detection cannot detect the attacks. The PDT model has the best detection performance, and the HMM-L model and the EDC model have acceptable and stable detection performance. The detection performance of the HMM-M model is generally not good. It should be mentioned that we do not have anomaly or intrusion data for training, as our goal is to detect false data even if the attacks are not seen or trained before. As a result, it is expected that the detection performance against stealthy attacks is not up to, for example, 90% or higher. It should also be noted that a larger γ does not imply a shorter average distance between vehicles. This is because it results in the ego vehicle "oscillating" behind its leading vehicle. Also, a larger γ does not imply a higher detection performance. This is because it results in the ego vehicle shorting its distance to the leading vehicle rapidly, and thus the intrusion detection approaches have much less trajectory windows to detect attacks.

TABLE IV

EXPERIMENTAL RESULTS (F_1 SCORES) WITH SCALING STEALTHY ATTACKS. γ IS THE INTENSITY OF AN ATTACK, AND D IS THE AVERAGE DISTANCE BETWEEN VEHICLES.

γ	D (m)	PHY	HMM-L	HMM-M	PDT	EDC
0.8	30.55	< 0.01	0.72	0.00	0.52	0.59
1.1	32.37	< 0.01	0.74	0.79	0.81	0.74
1.3	33.74	< 0.01	0.72	0.76	0.79	0.72
1.5	36.95	< 0.01	0.65	0.19	0.69	0.61
1.7	37.41	< 0.01	0.64	0.11	0.67	0.60

TABLE V

EXPERIMENTAL RESULTS (F_1 SCORES) WITH SCALING STEALTHY ATTACKS GENERATING COLLISIONS. γ IS THE INTENSITY OF AN ATTACK, AND D IS THE AVERAGE DISTANCE BETWEEN VEHICLES.

γ	D (m)	PHY	HMM-L	HMM-M	PDT	EDC
0.8	No Collision					
1.1	20.26	< 0.01	0.83	0.92	0.91	0.83
1.3	21.41	< 0.01	0.75	0.94	0.90	0.80
1.5	22.61	< 0.01	0.61	0.17	0.86	0.66
1.7	23.21	< 0.01	0.59	0.02	0.84	0.63

Besides the average distance between vehicles, we are especially interested in the scenarios that collisions happen (the ego vehicle collides with its leading vehicle), as our first priority is to detect attacks (or false data) generating collisions. The experimental results are listed in Table III which only includes the time series of trajectory vectors having a collision at some time step t (after a collision happens, the corresponding vehicles will not generate any new trajectory vector). We can observe that the PDT model and the EDC model have better and more stable detection performance, compared with the HMM-L model and the HMM-H model. Besides, the PDT model and the EDC model also have better detection performance against shifting stealthy attacks generating collisions, compared with themselves against shifting stealthy attacks not generating collisions. This is a good feature as detecting attacks generating collisions is more critical from the system perspective.

D. Experiment with Scaling Stealthy Attacks

The scaling stealthy attacks are defined in section II-B. In this experiment, we evaluate different intensities $\gamma = 0.8, 1.1, 1.3, 1.5, 1.7$ of the attacks. A γ larger than 1 means that the false data make a vehicle overestimating the velocity of its leading vehicle, and a γ smaller than 1 means that the false data make a vehicle underestimating the velocity of its leading vehicle. For γ larger than 1.7, the experiments results are similar to those with $\gamma = 1.7$.

The experimental results are listed in Table IV. Similar to the shifting stealthy attacks, as the attacks are stealthy, the PHY detection cannot detect the attacks. The PDT model has the best detection performance, except with $\gamma = 0.8$, and the HMM-L model has comparable and stable detection performance. The EDC model has a little lower detection performance, and the detection performance of the HMM-

TABLE VI

EXPERIMENTAL RESULTS (F_1 SCORES) WITH TIME-VARIANT STEALTHY ATTACKS. γ IS THE INTENSITY OF AN ATTACK, AND D IS THE AVERAGE DISTANCE BETWEEN VEHICLES.

γ	D (m)	PHY	HMM-L	HMM-M	PDT	EDC
0.001	29.81	0.00	0.59	0.02	0.81	0.57
0.002	29.84	0.00	0.54	0.15	0.86	0.62
0.004	30.01	0.00	0.44	0.32	0.83	0.65
0.008	30.23	0.00	0.31	0.49	0.81	0.58

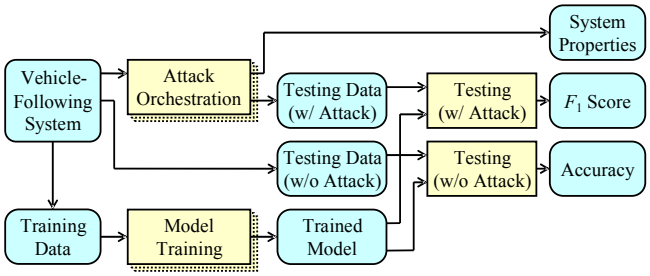

Fig. 3. The methodology implies that a good intrusion detection approach should have high accuracy (when there is no attack) and high F_1 scores (when there are attacks), especially for the attacks which have significant impacts on system properties.

M model is very diverse. Similarly to the shifting stealthy attacks, a larger γ does not imply a shorter average distance between vehicles nor a higher detection performance.

The experimental results with collisions are listed in Table V. Again, we can observe that the PDT model and the EDC model have better and more stable detection performance, compared with the HMM-L model and the HMM-H model. Again, the PDT model and the EDC model also have better detection performance against scaling stealthy attacks generating collisions, compared with themselves against scaling stealthy attacks not necessarily generating collisions.

E. Experiment with Time-Variant Stealthy Attacks

The scaling stealthy attacks are defined in section II-C. In this experiment, we evaluate different intensities $\gamma = 0.001, 0.002, 0.004, 0.008$ of the attacks. The experimental results are listed in Table VI. As the attacks are stealthy, the PHY detection cannot detect *any* of the attacks. The PDT model has the best detection performance with F_1 scores over 80%, and its advantages over the other approaches are significant. The HMM-L model and the HMM-M model have poor detection performance. Even if we change the threshold to 400 (a good value based on testing), the F_1 scores of the HMM model are 0.67, 0.74, 0.80, and 0.78 for the different γ values, still lower than the PDT model.

F. Discussion

The experimental results suggest that the PDT model has the best and most stable detection performance. Although we admit that the claim is only applicable to the stealthy attacks designed in this paper, the methodology is applicable to different attacks and different intrusion detection approaches, as shown in Figure 3. The methodology considers accuracy (when there is no attack), F_1 scores (when there are attacks), and system properties. A good intrusion detection approach should (at least) have the following three features:

- High accuracy when there is no attack (97% for the PDT model in Table I).
- High F_1 scores when there are attacks (67%–86% for the PDT model in the most scenarios in Table II, Table IV, and Table VI).
- Higher F_1 scores when the attacks have significant impacts on system properties (84%–91% for the PDT model against attackers generating collisions in Table III and in Table V).

In this paper, we do not have anomaly or intrusion data for training, as our goal is to detect false data even if the attacks are not seen or trained before. If there is a specific attack to be detected, the attacked data should be included in the training data, and it is expected to have better detection performance against the specific attack — a trade-off between the detection performance and generality.

V. CONCLUSIONS

In this paper, we designed three types of *stealthy* attacks on ACC or CACC inputs, where the stealthy attacks can deceive a rule-based detection approach and impede system properties (collision-freeness and vehicle-following distance). We then developed two deep-learning models, a predictor-based model and an encoder-decoder-based model to detect the attacks, where the two models do not need attacker models for training. The experimental results demonstrated the respective strengths of different models and led to a methodology for the design of learning-based intrusion detection approaches. Future work includes the integration of multiple approaches, more complicated driving scenarios, and system reaction following intrusion detection.

REFERENCES

[1] J. Petit, B. Stottelaar, M. Feiri, and F. Kargl, "Remote attacks on automated vehicles sensors: Experiments on camera and lidar," *Black Hat Europe*, vol. 11, 2015.

[2] C. Yan, W. Xu, and J. Liu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle," *DEF CON*, vol. 24, 2016.

[3] S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, vol. 18, no. 11, pp. 2898–2915, 2017.

[4] M. Amoozadeh, A. Raghuramu, C.-N. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. Levitt, "Security vulnerabilities of connected vehicle streams and their impact on cooperative driving," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 126–132, 2015.

[5] R. van der Heijden, T. Lukaseder, and F. Kargl, "Analyzing attacks on cooperative adaptive cruise control (CACC)," in *IEEE Vehicular Networking Conference (VNC)*. IEEE, 2017, pp. 45–52.

[6] M. Jagielski, N. Jones, C.-W. Lin, C. Nita-Rotaru, and S. Shiraishi, "Threat detection for collaborative adaptive cruise control in connected cars," in *ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 2018, pp. 184–189.

[7] P. Sharma, J. Petit, and H. Liu, "Pearson correlation analysis to detect misbehavior in VANET," in *IEEE Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–5.

[8] H. Liang, M. Jagielski, B. Zheng, C.-W. Lin, E. Kang, S. Shiraishi, C. Nita-Rotaru, and Q. Zhu, "Network and system level security in connected vehicle applications," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–7.

[9] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[10] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[11] T. Kieu, B. Yang, and C. S. Jensen, "Outlier detection for multidimensional time series using deep neural networks," in *IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2018, pp. 125–134.

[12] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 387–395.

[13] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, vol. 11, no. 6, 2016.

[14] E. Khanapuri, T. Chintalapati, R. Sharma, and R. Gerdes, "Learning-based adversarial agent detection and identification in cyber physical systems applied to autonomous vehicular platoon," in *IEEE/ACM International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*. IEEE, 2019, pp. 39–45.

[15] J. Song, Y. Wu, Z. Xu, and X. Lin, "Research on car-following model based on SUMO," *IEEE International Conference on Advanced Infocomm Technology, IEEE/ICAIT*, pp. 47–55, 01 2015.

[16] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Fötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using SUMO," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018. [Online]. Available: ¡https://elib.dlr.de/124092/

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NeurIPS Autodiff Workshop*, 2017.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.