
Article:
Shin, Y.-J., Shin, D. orcid.org/0000-0002-0840-6449 and Bae, D.-H. (2024) Virtual environment model generation for CPS goal verification using imitation learning. ACM Transactions on Embedded Computing Systems, 23 (1). 13. pp. 1-29. ISSN 1539-9087

https://doi.org/10.1145/3633804

Virtual Environment Model Generation for CPS Goal Verification using Imitation Learning

YONG-JUN SHIN*, ETRI, South Korea
DONGHWAN SHIN†, University of Sheffield, United Kingdom
DOO-HWAN BAE, KAIST, South Korea

Cyber-Physical Systems (CPS) continuously interact with their physical environments through embedded software controllers that observe the environments and determine actions. Field Operational Tests (FOT) are essential to verify to what extent the CPS under analysis can achieve certain CPS goals, such as satisfying the safety and performance requirements, while interacting with the real operational environment. However, performing many FOTs to obtain statistically significant verification results is challenging due to its high cost and risk in practice. Simulation-based verification can be an alternative to address the challenge, but it still requires an accurate virtual environment model that can replace the real environment interacting with the CPS in a closed loop.

In this paper, we propose ENVI (ENVironment Imitation), a novel approach to automatically generate an accurate virtual environment model, enabling efficient and accurate simulation-based CPS goal verification in practice.To do this, we first formally define the problem of the virtual environment model generation and solve it by leveraging Imitation Learning (IL), which has been actively studied in machine learning to learn complex behaviors from expert demonstrations. The key idea behind the model generation is to leverage IL for training a model that imitates the interactions between the CPS controller and its real environment as recorded in (possibly very small) FOT logs. We then statistically verify the goal achievement of the CPS by simulating it with the generated model. We empirically evaluate ENVI by applying it to the verification of two popular autonomous driving assistant systems. The results show that ENVI can reduce the cost of CPS goal verification while maintaining its accuracy by generating accurate environment models from only a few FOT logs. The use of IL in virtual environment model generation opens new research directions, further discussed at the end of the paper.

CCS Concepts: • **Software and its engineering → Empirical software validation**; • **Computer systems organization → Embedded software**; • **Computing methodologies → Modeling methodologies**.

Additional Key Words and Phrases: Cyber-Physical System (CPS), Software controller, Simulation-based CPS goal verification, Environment modeling, Imitation Learning (IL)

ACM Reference Format:
Yong-Jun Shin, Donghwan Shin, and Doo-Hwan Bae. 2023. Virtual Environment Model Generation for CPS Goal Verification using Imitation Learning. *ACM Trans. Embedd. Comput. Syst.* 0, 0, Article 0 (2023), 30 pages. https://doi.org/XXXXXXX.XXXXXXX

*Part of this work was done while the author was affiliated with KAIST, South Korea.
†Corresponding author.

Authors' addresses: Yong-Jun Shin, ETRI, Daejeon, South Korea, yjshin@etri.re.kr; Donghwan Shin, University of Sheffield, Sheffield, United Kingdom, d.shin@sheffield.ac.uk; Doo-Hwan Bae, KAIST, Daejeon, South Korea, bae@se.kaist.ac.kr.

ACM 1539-9087/2023/0-ART0
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Cyber-Physical Systems (CPS) utilize both physical and software components deeply intertwined to control physical actuators at runtime continuously [7]. In particular, software controllers embedded in a CPS play a significant role in observing the CPS's environmental states and determining appropriate actions to achieve certain CPS goals based on the observation. For example, a lane-keeping system of a self-driving car continuously observes the car's current position on the lane through sensors and controls the steering wheel angle to keep the car at the lane center. Note that the CPS (including the software controller) and its environment interact with each other in a closed-loop mode where a current CPS action affects the next environmental states.

As the criticality of the CPS increases, Field Operational Test (FOT) becomes essential to verify to what extent the CPS under development can achieve its goals (e.g., the self-driving car should drive following the center of the lane) in the real operational environment [8]. Specifically, engineers involved in CPS software controller development could deploy a CPS equipped with the controller into the real environment and verify the CPS's goal achievement using the logs collected during the FOTs. However, the FOTs often contain uncertainties (e.g., nonuniform friction with the ground), and conducting many FOTs for achieving statistically significant verification results is expensive, time-consuming, and even dangerous.

An alternative is to use a *virtual environment model* that can simulate how the CPS controller interacts with the real environment, and repeat the simulation many times to statistically verify the goals. This simulation-based approach can significantly reduce the cost of the CPS goal verification compared to using FOTs [26, 28].

However, it is challenging to *accurately* make the virtual environment model so that the simulation-based verification result is the same as the FOT-based verification result, mainly due to the following three reasons. First, given the closed-loop interaction between the CPS and its environment, the environment model must be able to generate the *next* environmental state (i.e., the input of the software controller) considering the sequence of the *past* environmental states and CPS actions. Second, the environment model will be used to simulate a sequence of closed-loop interactions with the CPS where even small errors in individual interactions can be accumulated over time. This means that it is essential to reduce the accumulation of errors as much as possible. Third, even the uncertainties of the real environment must be taken into account in the virtual environment model.

To address the above challenges, we propose ENVI (ENVironment Imitation), a novel approach that generates a virtual environment model accurately mimicking the real environment. We recast the problem of virtual environment model generation as the problem of Imitation Learning (IL), which has been widely studied to mimic complex behaviors (e.g., climbing stairs) from expert's demonstrations [17]. ENVI leverages IL to generate the environment model that mimics how the real environmental state changes according to the CPS actions and uncertainties recorded in a set of log data collected from small FOTs as closely as possible. The generated environment model is then used to simulate the CPS software controller as often as needed. Using the simulation, engineers can efficiently assess the CPS goal achievement with well-known verification methods, such as statistical model checking [31] and model-based testing [3].

Specifically, we formally define the virtual environment model generation problem for the CPS goal verification. We provide a systematic process and user-configurable parameters to solve the problem by leveraging IL. We evaluate the feasibility of ENVI by verifying two popular autonomous driving assistant systems (i.e., lane-keeping systems and adaptive cruise control systems) of a robot vehicle. The empirical evaluation results show that software engineers can automatically and

efficiently generate an environment model that imitates the interaction between the CPS software controller and its real environment using our approach from small FOT logs.

In summary, below are the contributions of this paper:

(1) We shed light on the novel problem of virtual environment model generation with a formal problem definition.
(2) We propose ENVI, a novel data-driven approach for the virtual environment model generation utilizing IL.
(3) We assess the application of ENVI to the verification of two driving assistant systems of a robot vehicle.
(4) We discuss the research directions for the novel problem of the virtual environment model generation of the CPS controller.

The remainder of this paper is organized as follows. Section 2 illustrates a motivating example. Section 3 provides background on IL. Section 4 formalizes the problem of the virtual environment model generation. Section 5 proposes ENVI. Section 6 reports on the evaluation of ENVI. Section 7 discusses open issues with future research directions. Section 8 introduces related work. Section 9 concludes the paper.

2 MOTIVATING EXAMPLE

Consider a software engineer developing a lane-keeping system of an autonomous vehicle. The engineer aims to develop and verify the safety of vehicle's software controller (i.e., lane-keeping system) that continuously monitors the distance from the center of the lane and computes the steering angle that determines how much to turn to keep the distance as small as possible.

Once the lane-keeping controller is developed, the engineer must ensure that the vehicle equipped with the controller follows the center of the lane while driving in the real world. This holds true even if the controller has been tested in simulations during earlier phases of development, as there is an unavoidable gap between a fully simulated environment and the real world. To do this, the engineer deploys the vehicle on a safe road and collects an FOT log, including the distance d_t from the lane center, which is observed by sensors, and the steering angle a_t, which controls the actuators, at time $t = 1, \ldots, T$ where T is a pre-defined FOT duration. Based on the collected data, the engineer can quantitatively assess the safety of the lane-keeping system by calculating the maximum displacement from the lane center, i.e., $\max_{t \in \{1,\ldots,T\}} |d_t|$. The assessment is used to precisely verify the safety goal of the system, i.e., whether $\max_{t \in \{1,\ldots,T\}} |d_t| < \epsilon$ holds or not for a small threshold ϵ given from the industrial standard such as ISO 11270 [18]. Notice that, due to the uncertainties in FOTs, such as non-uniform friction between the tires and the ground, the same FOT and the assessment must be repeated multiple times for statistical verification of the goal.

Clearly, it takes a lot of time and resources to repeat many assessments of the FOTs to obtain statistically significant results. To address this issue, the engineer may decide to rely on simulations built on few FOT results. However, manually generating an accurate virtual environment model that can replace the real environment is also very challenging and expensive, especially for software engineers who do not have enough expertise in physics. For example, it is infeasible to manually make rules or functions defining d_{t+1} for all possible histories $\langle (d_1, a_1), \ldots, (d_t, a_t) \rangle$. Thus, the simulation-based goal verification results could be very different from the FOT-based results.

Our approach, ENVI, enables efficient CPS goal verification by generating the virtual environment model from a small amount of FOT data. The engineer can simply provide ENVI with the software controller (i.e., the lane-keeping system under analysis) and a few FOT logs, which is *far less* than the data required for statistically significant results using FOTs only. Then ENVI *automatically* generates an accurate virtual environment model that imitates the behavior of the real environment of the lane-keeping system in terms of how the real environment interacts with the

lane-keeping system; specifically, the virtual environment model generates d_{t+1} for given history $\langle (d_{t-l+1}, a_{t-l+1}), \ldots, (d_t, a_t) \rangle$ for $t = l, \ldots, T-1$, where l is the length of the history, such that $\max_{t \in \{l,\ldots,T\}} |d_t|$ calculated by the model is almost the same as the value calculated based on the FOTs (i.e., accurate). By utilizing the accurate virtual environment model, the engineer can verify the safety goal of the lane-keeping system while reducing the costs of FOTs by running simulations instead of repeating them.

The challenge for ENVI is automatically generating a virtual environment model that continuously behaves as similar as possible to the real environment using a limited amount of data. To address this, we leverage *imitation learning* detailed in Section 3.

3 BACKGROUND: IMITATION LEARNING

Imitation Learning (IL) is a learning method that allows an agent to mimic expert behaviors for a specific task by observing demonstrations of the expert [17]. For example, a humanoid robot (i.e., an agent) can learn how to walk by observing how a human walks (i.e., the expert's demonstration). IL assumes that an expert decides an action depending on only the current state that the expert encounters. Based on this assumption, an expert demonstration is a series of pairs of states and actions, and IL aims to extract the expert's internal decision-making function (i.e., a policy function that maps states into actions) from the demonstration [17]. We introduce two representative IL algorithms in the following subsections: Behavior Cloning (BC) and Generative Adversarial Imitation Learning (GAIL).

3.1 Behavior Cloning

Behavior Cloning (BC) infers the policy function of the expert using supervised learning [39]. Training data can be organized by pairing states and corresponding actions in the expert's demonstration. Then existing supervised learning algorithms can train the policy function that returns expert-like actions for given states. Due to the simplicity of the BC algorithm, BC can create a good policy function that mimics the expert quickly if there are sufficiently much demonstration data. However, if the training data (i.e., expert demonstration) does not fully cover the input state space or is biased, the policy function may not mimic the expert behavior correctly.

3.2 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) evolves the policy function using iterative competitions with a *discriminator* that evaluates the policy function [15]. Therefore, both the policy function and the discriminator are trained in parallel.

The policy function gets states in the expert demonstration and produces simulated actions. The discriminator then gets the policy function's input (i.e., states) and output (i.e., simulated actions) and evaluates how the policy function behaves like the real expert, as shown in the demonstration. The more similar the policy function's simulation is to the expert demonstration, the more rewarded the policy function is by the discriminator. The policy function is trained to maximize the reward from the discriminator. On the other hand, the discriminator is trained to discriminate the policy function's behavior as *fake* (i.e., the unreal expert) returning a low reward.

After numerous learning iterations of the policy function and the discriminator, the policy function finally mimics the expert well to deceive the advanced discriminator. GAIL uses both the expert demonstration data and the simulation trace data of the policy function generated internally, so it works well even with small demonstration data [15]. However, because of the internal simulation of the policy function, its learning speed is relatively slow [21].

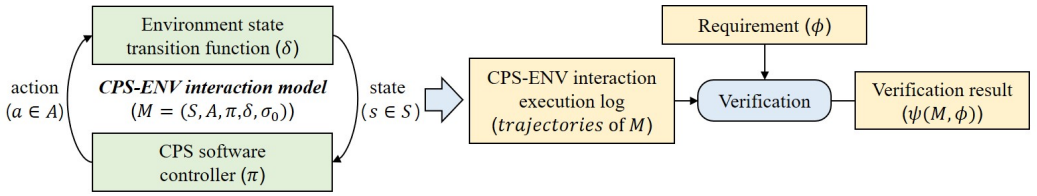


Fig. 1. Formal framework for CPS goal verification

4 PROBLEM DEFINITION

This section introduces a new mathematical framework for modeling how the CPS software controller under analysis *interacts* with its environment to achieve its goals. Based on the formal framework, we then define the environment model generation problem for CPS goal verification.

4.1 A Formal Framework for CPS Goal Verification

A CPS achieves its goals by interacting with its physical environment. Specifically, the CPS software controller first obtains the CPS's current state by observing the environment, and decides an appropriate action to maximize the likelihood of achieving the goals. Then, taking action causes a change in the environment for the next step, which the CPS will observe again to decide an action for the next step. We assume the CPS and the environment interact in a closed loop without interference by a third factor. To formalize this process, we present a novel *CPS-ENV interaction model* inspired by Markov Decision Process [48] that models an agent's sequential decision-making process under observation over its environmental states.

Specifically, a CPS-ENV interaction model is a tuple $M = (S, A, \pi, \delta, \sigma_0)$, as shown in the top of Figure 1, where S is a set of observable environmental states, A is a set of possible CPS actions, $\pi : S \rightarrow A$ is a policy function that captures the software controller of the CPS, $\delta : S \times A \rightarrow S$ is a transition function that captures the transitions of environmental states over time as a result of CPS actions and its previous states[1], and σ_0 is an initial input of δ. For example, starting from $\sigma_0 = (s_0, a_0)$, the controller observes its next state $s_1 = \delta(s_0, a_0)$, and decides a responding action $a_1 = \pi(s_1)$. Then the CPS again senses the changed environment $s_2 = \delta(s_1, a_1)$, and so on.

Note that M is under Markov assumption, which may not hold in practice, especially when an environmental state is partially observable due to the CPS's limited sensing capability. To address this, we extend the CPS-ENV interaction model M to consider the history of states and actions. Specifically, $\pi : S^l \rightarrow A$, $\delta : S^l \times A^l \rightarrow S$, and $\sigma_0 = \langle (s_0, a_0), \ldots, (s_{l-1}, a_{l-1}) \rangle$, where l is the length of history. Hereafter, we use the extended model to define the problem.

For a CPS-ENV interaction model $M = (S, A, \pi, \delta, \sigma_0)$, we can think of a sequence of transitions $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} s_n$ over n steps where $s_{t-1} \xrightarrow{a_{t-1}} s_t$ denotes a transition from a state s_{t-1} to another state s_t of the environment by taking an action a_{t-1} of the CPS. More formally, we define a *trajectory* of M over T time ticks as a sequence of tuples $tr(M, T) = \langle (s_0, a_0), \ldots, (s_T, a_T) \rangle$.

Since a trajectory of a CPS-ENV interaction model concisely captures the sequential interaction between the CPS software controller under analysis and its environment, one can easily assess a CPS goal achievement on a trajectory either quantitatively or qualitatively. Due to the uncertainties of the physical environment, the assessment of the CPS goal on different trajectories of the same model M may differ. Therefore, the CPS goal achievement is verified by statistically analyzing the multiple trajectories. Note that multiple trajectories are collected by independent executions of

[1]Though we use deterministic policy and transition functions for simplicity, they can be easily extended in terms of probability density, i.e., $\pi : S \times A \rightarrow [0, 1]$ and $\delta : S \times A \times S \rightarrow [0, 1]$, to represent stochastic behaviors if needed.

the CPS-ENV interaction model, and thus, individual trajectories are independent of each other. Figure 1 visualizes the CPS goal verification process. Specifically, let ϕ be a requirement that precisely specifies a goal under verification. For a CPS-ENV interaction model M, the verification result of ϕ for M, denoted by $\psi(M, \phi)$, is computed by *statistically* evaluating the achievement of ϕ on the trajectories of M. $\psi(M, \phi)$ can be a probability distribution of assessments of ϕ or a statistical test result. For example, the safety verification result of the lane-keeping system can be a distribution of the maximum displacements from the lane center obtained from many tests or a hypothesis test result based on a certain threshold of the maximum displacement.

4.2 Problem Statement

The problem of virtual environment model generation for simulation-based CPS goal verification is to find an accurate environment model that can replace the real environment while making the simulated verification result be same as the result achieved in the real environment. Specifically, for the same CPS controller under analysis, let a CPS-ENV interaction model $M_r = (S, A, \pi, \delta_r, \sigma_0)$ representing the interaction between the controller and its real environment (in FOT) and another model $M_v = (S, A, \pi, \delta_v, \sigma_0)$ representing the interaction between the same controller and its virtual environment (in simulations). Notice that we have the same S, A, π, and σ_o for both M_r and M_v since they are about the same CPS controller[2], whereas δ_r and δ_v are different since they represent how the corresponding environments react to the actions performed by the CPS. For a requirement ϕ, we aim to have δ_v that minimizes the difference between $\psi(M_r, \phi)$ and $\psi(M_v, \phi)$. Therefore, the problem of virtual environment model generation for CPS goal verification is to find δ_v such that $|\psi(M_r, \phi) - \psi(M_v, \phi)|$ is the minimum.

The virtual environment model generation problem has four major challenges. First, the number of possible states and actions is often very large, making it infeasible to build a virtual environment model (i.e., represented by a transition function $\delta_v : S^l \times A^l \rightarrow S$) by exhaustively analyzing individual states and actions. Second, since the virtual environment model continuously interacts with the CPS under analysis in a closed-loop, even a small difference between the virtual and real environments can significantly differ in verification results as it accumulates over time, the so-called compounding error problem. Therefore, simply having a transition function δ_v that mimics the behavior of δ_r in terms of individual input and output pairs, without considering the accumulation of errors for sequential inputs, is not enough. Third, the real CPS-ENV interaction with δ_r is nondeterministic, even if π under analysis is deterministic. Defining δ_v addressing various sources of δ_r's uncertainty, so that $\psi(M_v, \phi)$ is identical to $\psi(M_r, \phi)$, is challenging. Finally, generating δ_v should not be as expensive as using many FOTs; otherwise, there is no point in using simulation-based CPS goal verification. Recall that manually crafting virtual environment models requires a lot of expertise, which could take longer than doing FOTs many times for the verification.

To address the challenges mentioned above, we suggest leveraging IL to automatically generate virtual environment models from only a small amount of data. The data is the partial trajectory of M_r, which can be collected from a few FOTs for the CPS controller under test in its real application environment. Since IL can efficiently extract how experts make sequential actions for given states from a limited amount of demonstrations while minimizing the compounding errors, it is expected to be an excellent match to our problem. Therefore for our problem, IL will extract δ_v, instead of π (which is the original goal of IL), that can best reproduce given trajectories of M_r (i.e., FOT logs).

[2]Note that S and A can be the same for M_r and M_v because they are sets of *observable* environmental states and CPS actions from the perspective of the CPS software controller under analysis.

5 ENVIRONMENT IMITATION

This section provides ENVI, a novel approach to address the virtual environment model generation problem for CPS goal verification, defined in Section 4, by leveraging IL. The original IL aims to generate (train) the system's policy function π in a given environment, but we leverage IL for ENVI with the aim of generating the environment δ_v of the CPS software controller under analysis. Therefore, in our context, the real environment is considered an "expert", and FOT logs represent the expert's demonstrations.

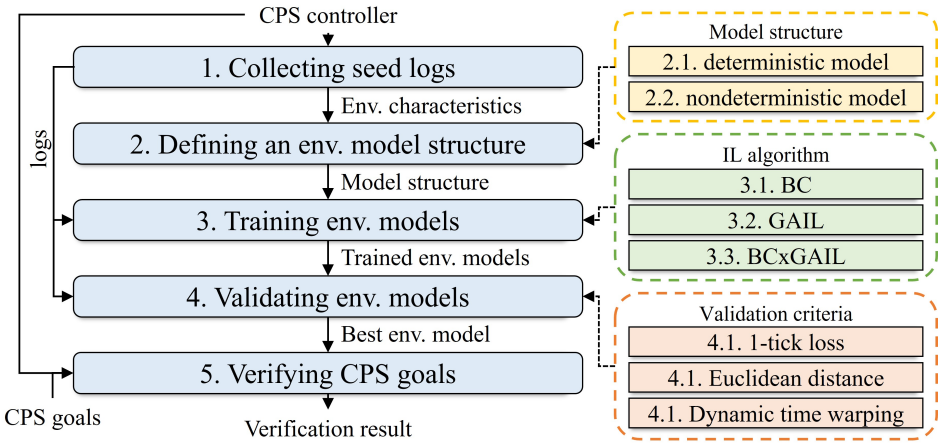


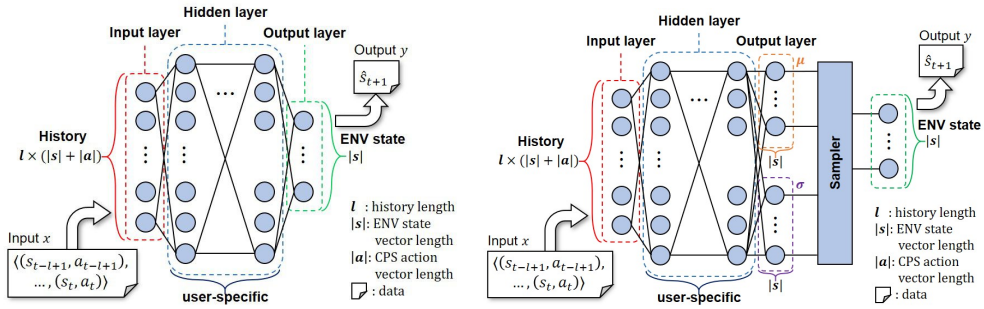
Fig. 2. ENVI: Overall process and parameters

Figure 2 shows the overview of the environment model generation and simulation-based CPS goal verification process using our approach. It is composed of five main stages: (1) collecting seed logs from FOTs, (2) defining an environment model structure based on environment characteristics, (3) training environment models from the seed logs using an IL algorithm, (4) validating the environment models and selecting the best one, and (5) verifying the given CPS goals using the best environment model. Each of the stages is detailed in the following subsections.

5.1 Stage 1: Collecting Seed Logs

The first stage of ENVI is to collect the interaction data between the CPS controller and its real environment, which will be used as the "demonstrations" for IL to generate the virtual environment later. For a CPS-ENV interaction model $M_r = (S, A, \pi, \delta_r, s_0)$ defined in Section 4, the data collected over time T can be represented as the trajectory of M_r over T steps, i.e., $\langle (s_0, a_0), (s_1, a_1), \ldots, (s_T, a_T) \rangle$ where $s_{t+1} = \delta_r(s_t, a_t)$ and $a_t = \pi(s_t)$ for $t \in \{0, 1, \ldots, T-1\}$. The trajectory can be easily collected from an FOT, since it is common to record the interaction between the CPS controller and its real environment as an FOT log [55]. The environmental state s_t and CPS action a_t are vectors of observable environmental state features and CPS action features, respectively, recorded at t. For example, the lane-keeping system records time-series data of the distances the vehicle deviated from the center of the lane d_t and the steering angles a_t over $t = 0, 1, \ldots, T$ during an FOT.

In practice, the trajectories of the same M_r are not necessarily the same due to the uncertainty of the physical environment, such as the non-uniform surface friction. Therefore, it is recommended to collect a few FOT logs for the same M_r. Since the virtual environment model generated by IL will mimic the given trajectories as much as possible, the uncertainty of the real environment recorded



(a) Deterministic environment model (b) Nondeterministic environment model

Fig. 3. The neural network structure of environment model

in the trajectories will also be imitated. A set of collected FOT logs is then divided into training and validation sets, which are used for Stages 3 and 4, respectively.

5.2 Stage 2: Defining Environment Model Structure

Before we train a virtual environment model δ_v, implemented as a neural network, using an IL algorithm and the seed logs collected in the previous stage, it is required to define the structure of δ_v in the following aspects: the input history length l, the hidden layer design of δ_v, and the determinism of δ_v (i.e., either deterministic or not).

The history length l affects the information captured in environmental states; the larger l, the more information. However, having more information decreases the training and execution time of δ_v. To better balance between the amount of information and the computation cost, one can investigate the seed logs to obtain environment characteristics; for example, if there is a cyclic pattern in the seed logs, l can be the length of the cycle.

The design of hidden layers in δ_v specifies how the output variables of δ_v are calculated from the input variables of δ_v. It is specific to a domain, but general guidelines of the neural network design exist for practitioners [37, 40].

The determinism of δ_v is about the choice between simplicity and realism; a deterministic model, which returns the same output for a given input deterministically, is simpler than a nondeterministic model, which may return different outputs for the same input, whereas the latter is more realistic than the former considering the uncertainty of real environments. Specifically, Figure 3 shows the structures of (a) deterministic and (b) nondeterministic models, respectively. As defined in Section 4, $\delta_v : S^l \times A^l \rightarrow S$ takes as input $\langle (s_{t-l+1}, a_{t-l+1}), \ldots, (s_t, a_t) \rangle$ and returns $\hat{s}_{t+1}$ in both structures. However, the deterministic model shown in Figure 3a is trained to predict $\hat{s}_{t+1}$ directly, whereas the nondeterministic model shown in Figure 3b is trained to predict $\hat{s}_{t+1}$ by inferring parameter values for a given probability distribution of s_{t+1}.

For example, if we assume that s_{t+1} follows a normal distribution $\mathcal{N}(\mu, \sigma)$ with unknown μ (mean) and σ (standard deviation) values, a nondeterministic model is trained to predict $\hat{s}_{t+1}$ by inferring $\hat{\mu}$ and $\hat{\sigma}$, which are internally utilized by a sampler to generate a sample (i.e., $\hat{s}_{t+1}$) following $\mathcal{N}(\hat{\mu}, \hat{\sigma})$. Figure 3b illustrates this process. Note that a sampler is essential in a nondeterministic model to mimic the randomness of the real environment. Users can select an appropriate probability distribution (e.g., uniform distribution, normal distribution, and Bernoulli distribution) based on their observations in the real environment. This makes a nondeterministic model more realistic but also more complex than a deterministic model.

5.3 Stage 3: Training Environment Model using IL

Once the structure of δ_v is determined, we can train δ_v using an IL algorithm with the training part of seed logs. The seed logs for training are used as a proper set of training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, where n is the number of FOT logs for training, $X_i = \langle x_1, x_2, \ldots \rangle$ is the sequence of δ_v's inputs collected from i-th FOT log, and $Y_i = \langle y_1, y_2, \ldots \rangle$ is the corresponding sequence of outputs (i.e., the expected value of $\delta_v(x_j)$ is y_j for all $j \in \{1, \ldots, |X_i|\}$ and $|X_i| = |Y_i|$ for $i \in \{1, \ldots, n\}$)[3]. Since $x \in X$ is an l-length sequence of state-action pairs, we can generate $X = \langle x_1, x_2, \ldots \rangle$ from an FOT log using a sliding window of length l. Specifically, for an FOT log $\langle (s_0, a_0), (s_1, a_1), \ldots, (s_T, a_T) \rangle$, we generate $x_j = \langle (s_j, a_j), \ldots, (s_{j+l-1}, a_{j+l-1}) \rangle$ (i.e., a sequence of pairs of the previous environment states and CPS actions) and $y_j = s_{j+l}$ (i.e., the subsequent environment state) for $j \in \{0, \ldots, T - l - 1\}$. This allows us to directly train δ_v to mimic (the state transition of) the real environment, as recorded in seed logs using IL algorithms.

We leverage specific IL algorithms for the environment model generation problem and run the algorithm to train δ_v using D. In the following subsections, we explain how each of the representative IL algorithms, i.e., BC, GAIL, and the combination of BC and GAIL (BCGAIL), can be used for training δ_v.

Note that we only present how BC, GAIL, and BCGAIL can be extended for ENVI as representative examples since they are the most widely used IL algorithms. Nevertheless, all IL algorithms can be extended for ENVI in general, as long as an IL algorithm is modified for training the environmental state transition function $\delta : S \times A \rightarrow S$ from training policy function $\pi : S \rightarrow A$, as described in the following.

5.3.1 Algorithm 1: BC. As described in Section 3.1, BC trains an environment model δ_v using supervised learning. Pairs of the input and output of the real environment recorded in FOT logs are given to δ_v as training data, and δ_v is trained to learn the real environment state transition shown in the training data.

Specifically, the BC algorithm (whose pseudocode is shown in Algorithm 1) takes as input a randomly initialized environment model δ_v and a training dataset D; it returns a set of trained environment models M.

Algorithm 1: ENVI BC algorithm

Input : ENV model (randomly initialized) δ_v,
Training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$
Output : Set of trained ENV models M

```
1  Set of trained ENV models M ← ∅
2  while not(stoping_condition) do
3      foreach (X, Y) ∈ D do
4          Sequence of model outputs Y' ← δ_v(X)
5          Float loss_BC ← getLoss(Y, Y')
6          δ_v ← update(δ_v, loss_BC)
7      end
8      M ← append(M, δ_v)
9  end
10 return M
```

[3]In D, the data from the i-th FOT log (i.e., $X_i = \langle x_1, x_2, \ldots \rangle$ and $Y_i = \langle y_1, y_2, \ldots \rangle$) is kept separated from the other data from the other FOT logs, since the specific IL algorithms (e.g., GAIL) need to process a sequential demonstration collected in a single log (e.g., see lines 9-15 in Algorithm 2).

The algorithm initializes a set of trained environment models M (line 1). The algorithm then iteratively trains δ_v using D until a stopping condition (e.g., a fixed number of iterations) is met (lines 2–9). For each $(X, Y) \in D$, the algorithm repeats the following (lines 3–7): (1) executing δ_v on X to predict a sequence of outputs Y' (line 4), (2) calculating the training loss $loss_{BC}$ based on the difference between Y' and Y (line 5), and (3) updating δ_v to minimize $loss_{BC}$ using optimization algorithms such as well-known Adam [24] (line 6). For every iteration of the training the copy of current δ_v is saved in M (line 8). The algorithm ends by returning the trained δ_vs collected in M (line 10). Notice that this algorithm explicitly guides δ_v to closely resemble the real environment based on $loss_{BC}$, i.e., the difference between the real (Y) and virtual (Y') environment states, i.e., the real and virtual environment states.

Algorithm 1 is intuitive and easy to implement. In addition, the model's loss converges fast because it is a supervised learning approach. However, if the training data does not fully cover the input space or is biased, the model may not accurately imitate the real environment.

5.3.2 Algorithm 2: GAIL. As described in Section 3.2, GAIL iteratively trains not only δ_v but also the discriminator ζ that evaluates δ_v in terms of the CPS controller π. Specifically, for a state s, ζ evaluates δ_v with respect to δ_r (captured by D) by comparing $\delta_v(s, \pi(s))$ and $\delta_r(s, \pi(s))$. The evaluation result is given to δ_v as a reward; the better δ_v mimics the sequence of states starting from s recorded in D, the higher the reward. To do this, ζ is trained using D by supervised learning[4], and δ_v is trained using the rewards calculated by ζ.

Algorithm 2 shows the pseudocode of GAIL. Similar to Algorithm 1, it takes as input a randomly initialized environment model δ_v and a training dataset $D = (X, Y)$; however, it additionally takes as input a randomly initialized discriminator ζ and the CPS controller under analysis π. It returns a set of trained virtual environment models M.

A set of trained environment models M is first initialized (line 1). The algorithm then iteratively trains both δ_v and ζ using D and π until a stopping condition is met (lines 2–20). To train ζ, for each $(X, Y) \in D$ (lines 3–18), the algorithm executes δ_v on X to predict a sequence of outputs Y' (line 4), calculates the discriminator loss $loss_d$ indicating how well ζ can distinguish Y and Y' for X (line 5), and updates ζ using $loss_d$ (line 6). Once ζ is updated, the algorithm trains δ_v using ζ and π (lines 7–17). Specifically, the algorithm initializes a sequence of rewards R (line 7) and a model input x' (line 8), collects $r \in R$ for each x' using δ_v, π, and ζ (lines 9–15), calculates the environment model loss $loss_{GAIL}$ by aggregating R (line 16), and updates δ_v using $loss_{GAIL}$ using optimization algorithms in reinforcement learning [41, 49] (line 17). To collect $r \in R$ for each x' (lines 9–15), the algorithm executes δ_v on x' to predict an output y' (line 10), executes ζ on x' and y' to get a reward r (line 11), appends r at the end of R (line 12), executes π on y' to decide a CPS action a (line 13), and updates $x' = \langle (s_1, a_1), (s_2, a_2) \ldots, (s_l, a_l) \rangle$ as $x' = \langle (s_2, a_2) \ldots, (s_l, a_l), (y', a) \rangle$ by removing (s_1, a_1) and appending (y', a) (line 14). A copy of δ_v is temporarily saved in M for each iteration (line 19) and the algorithm ends by returning M, the set of trained δ_vs (line 21).

Notice that, to train δ_v, GAIL uses the input-output pair (x', y') simulated by π and ζ, in addition to the real input-output pair (x, y) in D. This is why it is known to work well even with a small amount of training data [15, 21]. However, the algorithm is more complex to implement than BC, and the environment model converges slowly or sometimes fails to converge depending on hyperparameter values.

5.3.3 Algorithm 3: Combination of BC and GAIL (BCGAIL). Notice that BC trains δ_v using the training data only, but GAIL trains δ_v using the simulated data as well; BC and GAIL can be combined to use both training and simulated data without algorithmic conflict. This idea is suggested by Ho

[4]The structure of ζ is similar to δ_v, but the input of ζ is $(s, \delta_v(s, \pi(s))$ and the output of ζ is a reward value r.

Algorithm 2: ENVI GAIL algorithm

Input : ENV model (randomly initialized) δ_v,
Discriminator (randomly initialized) ζ,
Function of CPS decision-making logic π,
Training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$
Output : Set of trained ENV models M

1 Set of trained ENV models $M \leftarrow \emptyset$
2 **while** *not*(*stoping_condition*) **do**
3 **foreach** $(X, Y) \in D$ **do**
  `// Discriminator training`
4 Sequence of model outputs $Y' \leftarrow \delta_v(X)$
5 Float $loss_d \leftarrow getDisLoss(\zeta, X, Y, Y')$
6 $\zeta \leftarrow update(\zeta, loss_d)$
  `// Environment model training`
7 Sequence of model rewards $R \leftarrow \emptyset$
8 Model input $x' \leftarrow X[0]$
9 **for** $|X| - 1$ **do**
10 Model output $y' \leftarrow \delta_v(x')$
11 Reward $r \leftarrow \zeta(x', y')$
12 $R \leftarrow append(R, r)$
13 CPS action $a \leftarrow \pi(y')$
14 $x' \leftarrow updateInput(x', y', a)$
15 **end**
16 Float $loss_{GAIL} \leftarrow aggregate(R)$
17 $\delta_v \leftarrow update(\delta_v, loss_{GAIL})$
18 **end**
19 $M \leftarrow append(M, \delta_v)$
20 **end**
21 **return** M

and Ermon [15] to improve learning performance, and Jena et al. [21] later implemented the idea as an algorithm BCGAIL.

The BCGAIL algorithm is the same as GAIL in terms of its input and output, and it also trains both δ_v and ζ similar to GAIL. In particular, ζ is updated as the same as in GAIL. However, δ_v is updated using both $loss_{BC}$ (line 4 in Algorithm 1) and $loss_{GAIL}$ (line 15 in Algorithm 2). By doing so, BCGAIL can converge fast (similar to BC) with a small amount of training data (similar to GAIL).

5.4 Stage 4: Validating Environment Models

The IL algorithms return a set of trained environment models, and ENVI's validation stage validates the performance of the models and selects the best environment model that mimics the actual environment state transition well using seed logs that were left unused for training. This is because many IL algorithms, especially those based on GAIL, suffer from the convergence difficulty problem; the model's loss slowly converges or fails to converge [13]. Thus, we cannot guarantee the latest model to be the best model, and validation is required to evaluate and select the best model from candidate models stored during the training procedure. In original IL, human experts usually observe the simulation traces of the trained model for validation [34]. However, the physical environment is the target of imitation of ENVI, so it is challenging to validate environment models

manually. To address this and automatically evaluate trained models, we suggest using three domain-agnostic metrics: (1) 1-tick loss, (2) Euclidean distance, and (3) Dynamic Time Warping (DTW). The key idea behind the validation metrics is to assess the similarity between the virtual and real environments using the validation part (i.e., not used for training) of the seed logs. If needed, domain experts are also allowed to define domain-specific metrics (e.g., safety and passenger comfort for autonomous driving systems; see Section 6.1 for more details) to ensure the performance of the models additionally. Using the metrics, the best model can be automatically selected from the candidate models generated by the IL algorithm from the previous stage. The following paragraphs detail the three domain-agnostic metrics.

1-tick loss (exact matching of the 1-step execution). The first metric evaluates the 1-step execution of δ_v. This expects that if a single environmental state transition mimics the real environment well, the simulation result, which is the sum of accumulated state transitions, will also be realistic [39]. This rationale is the same as that of the BC algorithm. Therefore, the same loss function is also used here. Specifically, all possible model inputs collected from the validation FOT logs are given to δ_v, and δ_v's outputs are compared to the expected outputs collected from the validation dataset to calculate the environment model's validation loss.

Euclidean Distance (exact matching of the T-step executions). The second metric verifies that the model's T-step simulation results exactly match the FOT logs. It expects that given the same starting point, FOT and simulation will proceed the same. Specifically, the model is simulated from the initial states extracted from validation FOT logs, as described in the GAIL algorithm. The simulation logs are compared to the validation FOT logs by the Euclidean distance. Euclidean distance compares ith point of simulation log to the ith point of FOT log (so-called lock-step alignment) [1], so it captures whether the δ_v's simulations are precisely the same with FOTs well.

Dynamic Time Warping (pattern matching of the T-step executions). The third metric quantifies the similarity of the patterns of T-step simulations and FOTs. This assumes that it is almost impossible for the simulation to be exactly the same as the FOT in the multi-step simulation, so it at least seeks to find an environmental model whose simulation pattern is similar to the FOT. Specifically, it compares the simulation logs and FOT logs by Dynamic Time Warping (DTW). DTW is a time-series distance metric that compares a point in a source series to many points in a target series (so-called elastic alignment) and finally quantifies the similarity of the patterns of two time-series [1]. Therefore, it measures how similar the behavior pattern of the virtual environment is to the real environment.

5.5 Stage 5: Verifying CPS Goals

The last stage of ENVI is to verify the CPS controller under analysis using the simulation with the virtual environment model δ_v generated from the previous stages. This is decoupled from the previous stages that leverage IL, so engineers can use any simulation-based methods with δ_v to get the CPS goal verification result $\psi(M_v, \phi)$ for a given goal ϕ. Specifically, an engineer can test the controller π based on δ_v that provides realistic inputs (i.e., observable states) to π by simulating the virtual CPS-ENV interaction model $M_v = (S, A, \pi, \delta_v, \sigma_0)$ to collect many execution trajectories instead of FOTs. The simulation can be repeated to accumulate as many assessments of ϕ as needed by statistical verification methods, such as statistical model checking (SMC) [25]. Indeed, an SMC algorithm (e.g., Sequential Probability Ratio Test [25]) may require thousands of trajectories to verify the CPS goal depending on the given confidence interval. Therefore, the simulation using generated δ_v allows engineers to perform the CPS goal verification with little cost in such cases. Although the initial input σ_0 is required for simulating M_v, having σ_0 is much cheaper than having

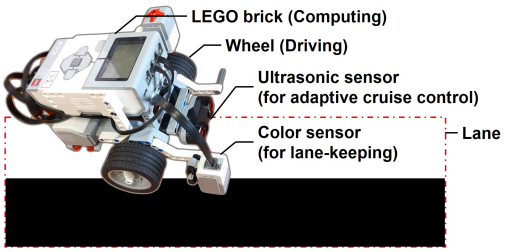


Fig. 4. Case study subject CPS: an autonomous robot vehicle

full FOT logs for FOT-based CPS goal verification since l (i.e., the length of σ_0) is much shorter than T (i.e., the entire FOT duration). Furthermore, only one σ_0 would be enough for a nondeterministic δ_v since it returns different simulation results for the same σ_0.

It is worth noting the differences between ENVI and (statistical) model checking. While both model checking and ENVI serve verification purposes, they have different scopes and focuses. Model checking, as a formal verification approach, can accommodate various formal specifications for any systems that can be modelled. On the other hand, ENVI is designed explicitly for CPS goal verification, focusing on the interaction between the CPS and its environment. Model checking does not involve an automated model generation, whereas ENVI proposes an automated method to create an accurate (virtual) environment model that can interact with the CPS. Last but not least, ENVI internally incorporates statistical model checking [25, 44], as mentioned above, to provide statistical results based on sampling (i.e., generating execution logs in our context). Therefore, ENVI mainly serves as a tool for building virtual environment models, while model checking is aimed at the formal verification of system specifications.

6 EMPIRICAL EVALUATION

This section is to evaluate the accuracy and efficiency of *ENVI-based verification* applied to the real CPS software controller. We will call the simulation-based CPS goal verification using ENVI-generated models *ENVI-based verification* in this section. Specifically, we first investigate the impact of using different ENVI parameters (i.e., model determinism, IL algorithms, and model validation criteria) on CPS goal verification and obtain a guide for setting optimal ENVI. We then analyze how similar the environment models generated by the optimized ENVI are to the real environment and how accurate the CPS goal verification results using the models are. Last but not least, we analyze ENVI's environment model generation efficiency for efficient CPS goal verification in terms of the cost of collecting FOT logs for the model generation. To summarize, we answer the following three research questions:

RQ1: What is the impact of ENVI parameters on the simulation-based CPS goal verification accuracy?
RQ2: How accurate is the simulation-based CPS goal verification using ENVI?
RQ3: Can ENVI efficiently generate environment models with a small amount of FOTs?

6.1 Evaluation Subjects

We implement a simplified autonomous vehicle equipped with autonomous driving assistant systems (ADAS) to answer the research questions in the context of a real CPS goal verification. We utilize our open physical experimental environment [45] that abstracts an autonomous vehicle as a

programmable LEGO robot and a road as a white and black paper lane, as shown in Figure 4. Like many other CPS, the autonomous robot vehicle comprises three parts: sensors, controllers, and actuators. Sensors (e.g., the color and ultrasonic sensors) give data observing the CPS environment to the controllers. Controllers (e.g., a Python program in a LEGO brick) control actuators (e.g., motors of the wheels) that make CPS act. The robot is capable of making 20 actions in a second. As the controllers under analysis, we develop two popular ADAS, a lane-keeping system (LKS), and an adaptive cruise control system (ACCS), for the robot vehicle and verify their goals using ENVI. The following subsections introduce the two software controllers and their goals under verification.

Case Study 1: Lane-keeping System (LKS). The goal of the LKS is to keep the center of the lane, indicated by the border between white and black areas while driving. As already described in Section 2, the LKS observes the distance from the center of the lane and decides the steering angle for turning to the lane center. Our LEGO-lized robot vehicle monitors how far the vehicle deviated from the lane center through a color sensor facing down. The observed color indicates the vehicle's displacement from the lane center because the lane's left/center/right is sensed as white/gray/black by the color sensor. The color value ranges from 0, meaning the darkest, to 100, meaning the brightest. The LKS returns the vehicle's turning rate (degree per second), given the observed color value. Positive/negative rate means turning right/left, respectively. In this case study, we developed three different versions of the LKS showing different performances and verified them using ENVI.

The LKS controller is verified in terms of *passenger comfort* and *safety*. The controller shall maximize passenger comfort. The comfort is measured by a jerk, the rate of acceleration change of the vehicle. The LKS's jerk is calculated as the rate of acceleration change in the vertical directions of the vehicle. Simultaneously, the controller shall maximize the degree of safety. It is measured by the maximum displacement from the lane center.

Case Study 2: Adaptive Cruise Control System (ACCS). The ACCS aims to keep the distance from a moving front vehicle. The system observes the front distance using an ultrasonic sensor and decides the driving speed to keep the distance between cars set by users (200 *mm* in our experiments). An input of the system is the distance to the front vehicle (*mm*), and an output is a driving speed (mm/s). If the front vehicle is far, the system accelerates, and the system decelerates if the front vehicle is close. We also develop and verify three versions with different performances.

Like the LKS, the ACCS is also verified in terms of passenger comfort and safety. A jerk also measures the comfort of the ACCS by the rate of acceleration change in the horizontal directions of the vehicle, and the minimum displacement from the target safe distance to the front vehicle measures the safety.

6.2 Evaluation metric

It is essential to assess the accuracy of the ENVI-based verification for all RQs. To do this, we measure the (dis)similarity between the FOT-based verification and ENVI-based verification results. The more similar the ENVI-based and FOT-based verification results, the better ENVI accurately mimics the real environment, when the same CPS controller is verified.

Specifically, we define an *imitation score* (i.e., *the smaller the better*) of an environment model δ_v as

$$ImitationScore(\delta_v) = D_{KL}(\psi(M_v, \phi) \| \psi(M_r, \phi))$$

where $\psi(M_v, \phi)$ is a simulation-based verification result of CPS controller goals ϕ using δ_v generated by the environment model generation method under analysis (e.g., ENVI), and $\psi(M_r, \phi)$ is an FOT-based verification result on the same goals as a reference. Note that executions of M_v and M_r are nondeterministic as discussed in Section 4, so we define $\psi(M_v, \phi)$ and $\psi(M_r, \phi)$ as joint distributions

of the passenger comfort and safety assessments obtained from *multiple* simulation and FOT logs, respectively. Though the distributions of goal assessment results can be further analyzed to get a boolean or numeric verification result by statistical verification methods (e.g., SMC) as described in Section 5.5, the distributions of the goal assessment results are directly compared to evaluate ENVI more rigorously at a lower level in our experiments. The dissimilarity of $\psi(M_v, \phi)$ and $\psi(M_r, \phi)$ is quantified by Kullback–Leibler divergence (KL divergence, D_{KL}) [23]. $D_{KL}(P\|Q)$ is a measure of divergence (i.e., relative entropy) of a probability distribution P from a reference distribution Q, widely used in imitation learning [52, 60]. If P is identical to Q, $D_{KL}(P\|Q)$ is zero; the divergence increases as their dissimilarity increases. Thus, the better ENVI mimics the real environment so that $\psi(M_v, \phi)$ is identical to $\psi(M_r, \phi)$, the smaller the KL divergence $D_{KL}(\psi(M_v, \phi)\|\psi(M_r, \phi))$, which is the imitation score.

We interpret the experiment results based on the imitation score to answer the three research questions. In RQ1, we compare different ENVI configurations based on the imitation score. In RQ2, we evaluate the accuracy of ENVI-based verification based on the imitation score. In RQ3, we also analyze the change of the imitation score according to the number of training FOT logs to evaluate how efficient the data-driven model generation is.

Note that, since KL divergence (D_{KL}) measures the *relative entropy* of a probability distribution from a reference distribution, directly interpreting its values or establishing "acceptable" threshold values for specific case studies is inappropriate due to its restiveness. To address this issue, many IL studies [15, 16] additionally define domain-specific metrics for evaluating the performance of trained models, and we did the same (i.e., the mean verification errors in terms of *safety* and *passenger comfort*, as described in Section 6.1). This allows us to interpret the verification errors directly. Although we cannot define acceptable thresholds for the verification errors in terms of *safety* and *passenger comfort* due to a lack of domain knowledge, defining such a threshold value for domain experts would be straightforward as long as the AV specifications and metrics, such as *safety* and *passenger comfort*, are clearly defined.

6.3 ENVI Experimental Setup

As described in Section 5, the CPS goal verification using ENVI follows five main stages. Remind the second, third, and fourth stages have user-configurable parameters, so we make total 18 ENVI versions for all possible combinations (2 model structures, 3 IL algorithms, and 3 validation criteria) for empirical analysis. In the following subsections, we explain our experimental setup for each stage in detail.

6.3.1 Stage 1: Collecting Seed Logs. For case study 1, we run a robot vehicle with an LKS on a straight road for about 5 seconds for an FOT. At 20 Hz, the following information is recorded in the logs: (1) a lane color value c_t as an environmental state observed by the vehicle's color sensor and (2) a turning rate r_t as a CPS action decided by the vehicle's controller. Therefore, an FOT log is a sequence of state-action pairs $\langle (c_0, r_0), \ldots, (c_T, r_T) \rangle$ where T is the FOT duration.

For case study 2, we run an ego vehicle equipped with an ACCS and another moving front vehicle, one meter apart at the beginning, on a three-meter straight road until the front vehicle reaches the end of the road. One FOT takes about 10 seconds. The ego vehicle records (1) a distance to the front vehicle d_t observed by the distance sensor and (2) a driving speed s_t as a CPS action at 20 Hz. Therefore, an FOT log is a sequence $\langle (d_0, s_0), \ldots, (d_T, s_T) \rangle$ where T is the FOT duration.

For each of the three versions of the two systems, we conduct 50 FOTs, so we collect a total of 150 logs (3 software versions and 50 FOTs) for each subject system. Using the 50 FOT logs for each version of the controllers, we conduct a 5-fold cross-validation to evaluate ENVI's CPS goal verification accuracy. Specifically, the 50 logs are randomly partitioned into 5 non-overlapping

Table 1. Deterministic environment model structure

id	input id	layer	output shape
0		input	(2, 10)
1	0	1D convolution layer	(16, 8)
2	1	Max pooling layer	(16, 4)
3	2	Flatten layer	(64)
4	3	Fully connected layer	(512)
5	4	Fully connected layer	(512)
6	5	Fully connected layer	(1)

Table 2. Nondeterministic environment model structure

id	input id	layer	output shape
0		input	(2, 10)
1	0	1D convolution layer	(16, 8)
2	1	Max pooling layer	(16, 4)
3	2	Flatten layer	(64)
4	3	Fully connected layer	(512)
5	4	Fully connected layer	(512)
6	5	Fully connected layer (μ)	(1)
7	5	Fully connected layer (σ)	(1)
8	6, 7	Normal distribution sampler	(1)

folds, so each fold (10 logs) is used for evaluating the accuracy while the remaining folds (40 logs) are used as seed logs (20 randomly selected logs for training in Stage 3 and the remaining 20 logs for validating in Stage 4). The procedure is repeated five times until all folds have been considered precisely once as the evaluation set.

6.3.2 Stage 2: Defining Environment Model Structure. As for the model structure, we set the length of history l as 10, meaning that the input of a virtual environment model is a 20-dimensional vector (i.e., a sequence of 10 state-action pairs). Tables 1 and 2 summarizes structures of the deterministic and nondeterministic environment model structures, respectively. We tried to design the hidden layers straightforward. We decided to use a 1D convolution layer for reducing noise and selecting important features of the time series data and fully connected layers for forward propagation. The deterministic model directly calculates the next environmental state from the historical data. On the other hand, the nondeterministic model calculates a mean and standard deviation of the next state and randomly selects a state based on the normal distribution. We used a normal distribution since it best represents the uncertainties appearing in the seed logs.

6.3.3 Stage 3: Training Environment Model. We implement IL algorithms using PyTorch library [32]. BC uses the ADAM optimizer [24] to update environment models. Since GAIL needs a policy gradient algorithm to update models, we use a state-of-the-art Proximal Policy Optimization (PPO) algorithm [41]. As for the hyperparameters of the IL algorithms, we use default values from the original papers [15, 41]. Table 3 shows the hyperparameter values used in our evaluation.

Table 3. Hyperparameter values for IL algorithms

Algorithm	Hyperparameter	Value
BC	Number of training iteration	500
	Learning rate	0.00005
GAIL	Number of training iteration	500
	Model & discriminator learning rate	0.00005
	PPO num. policy iteration	10
	PPO num. discriminator iteration	10
	PPO reward discount γ	0.99
	PPO GAE parameter λ	0.95
	PPO clipping ϵ	0.2

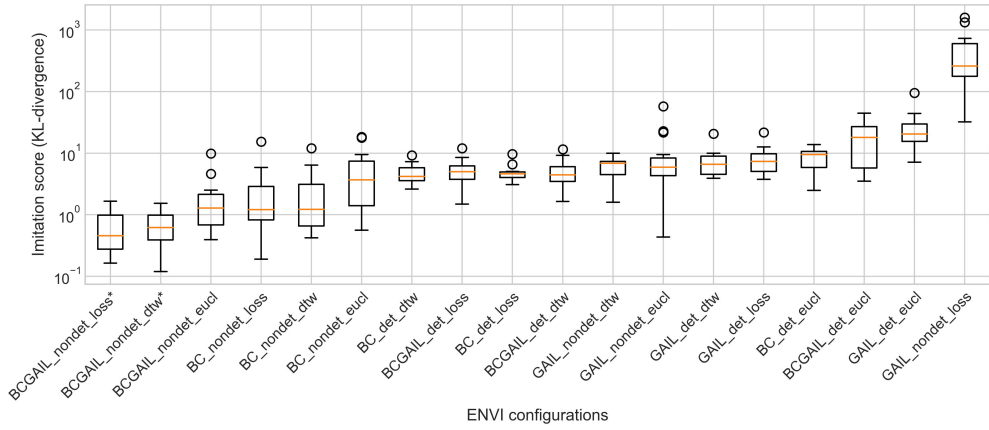
6.3.4 Stage 4: Validating Environment Models. When the IL algorithm is finished, trained environment models are evaluated based on the validation logs. For each validation log, only the initial environment state is given to the models to generate a following sequence of environmental state transitions by interacting with the CPS controller. The generated transitions are compared with the rest of the validation log to measure the accuracy of the models in replicating the real environment. Based on the validation results, the best environment model is chosen for each metric proposed in Section 5.4.

6.3.5 Stage 5: Verifying CPS Goals. Each version of LKS and ACCS is simulated multiple times with the environment models. For the simulation, the initial inputs of the environment models are given from the testing dataset described in Section 6.3.1. Each simulation log is then used to assess both passenger comfort and safety. As described in Section 6.2, the joint distribution of the passenger comfort and safety assessment results based on the multiple simulation logs is the verification result $\psi(M_v, \phi)$. In contrast, the verification result based on the full testing FOT logs is $\psi(M_r, \phi)$.

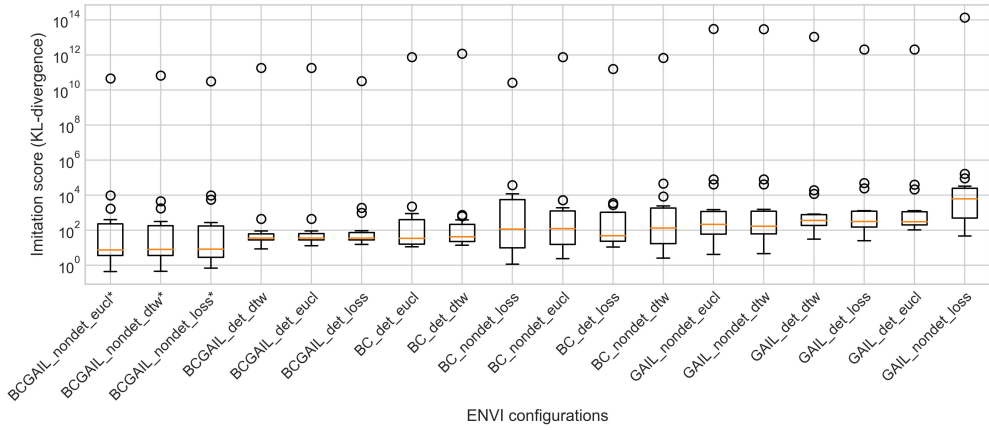
6.4 Comparison Baseline

In RQ2 and RQ3, we compare ENVI with two alternative data-driven environment model generation approaches using Machine Learning (ML) techniques other than IL. In terms of ML, the environment model generation problem defined in this paper can be seen as a regression problem that infers the future based on the past data. Therefore, engineers can generate the environment model δ_v using regression models without IL. We consider two well-known regression models, i.e., Polynomial Regression (PR) [29] and Random Forest regression (RF) [42]. We used pre-defined PR and RF APIs in Scikit-learn library [33]. All experimental settings except for the parts related to the learning method (e.g., the volume of training data) are the same as ENVI.

In addition to PR and RF, we make a random environment model. The random environment model does not require data or domain knowledge for modeling but changes the environmental state randomly regardless of the previous CPS actions. Injecting random environmental state observation to CPS controllers is often used to verify the possibility of unknown malfunctions of the controllers [12, 56]. However, it does not represent the continuous interaction of the CPS and its operational environment, making the verification result imprecise or rarely reproduced in reality [56]. Therefore, we use the random environment model as another baseline ignoring the CPS-ENV interaction defined in Section 4.



(a) Case study 1



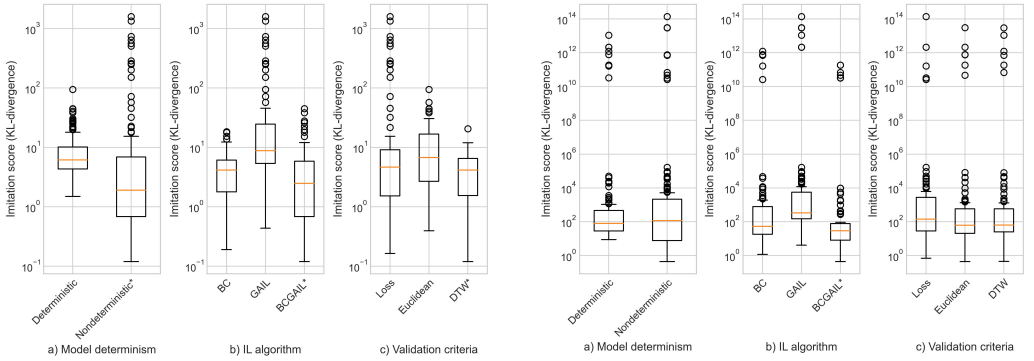
(b) Case study 2

Fig. 5. Imitation scores of all possible configurations of ENVI. The asterisk (*) highlights a set of optimal configurations with no statistically significant differences.

6.5 Experiment Results

6.5.1 RQ1: ENVI Parameters. RQ1 aims to investigate the effect of ENVI parameters on the imitation score and suggest optimal configurations of ENVI. To answer RQ1, we make all possible configurations of ENVI parameter settings and compare them statistically in the imitation score.

Figure 5 shows the comparison of 18 ENVI configurations in terms of the imitation score in the verification of LKSs (Figure 5a) and ACCSs (Figure 5b). ENVI is configured by an IL algorithm (*BC*, *GAIL*, or *BCGAIL*), a model structure (deterministic (*det*) or nondeterministic (*nondet*)), and a model validation criterion (1-tick loss (*loss*), Euclidean distance (*eucl*), or *DTW*). For each case study, an ENVI configuration is evaluated 15 times (3 different controller versions and 5-fold cross-validation). The spread of imitation scores of the ENVI configurations is shown on the boxplot



(a) Case study 1 (b) Case study 2

Fig. 6. Comparison of imitation scores achieved by different ENVI parameter settings. The asterisk (*) highlights parameter settings that are statistically significantly better than the others.

Table 4. Confidence level (p-value) of effect of the ENVI parameters on the imitation score and rank of the influence. p-value is highlighted in bold when it is smaller than 0.05.

ENVI parameter	Case study 1	Case study 2	Rank of the influence on the imitation score
Model determinism	**3.201*e*-10**	4.661*e*-01	2
IL algorithm	**3.770*e*-16**	**5.192*e*-12**	1
Validation criteria	**3.823*e*-04**	4.688*e*-01	3

in the log scale. For each case study, the ENVI configurations are sorted by the average imitation score. A configuration achieving the smallest imitation score is the optimal.

Figure 5 highlights optimal configurations with asterisks (*) that outperform the others in each case study. The Kruskal test selects a set of best configurations that do not have a statistically significant difference in terms of the imitation score. *BCGAIL_nondet_loss* and *BCGAIL_nondet_dtw* are optimal configurations in case study 1. In case study 2, three configurations using *BCGAIL* and *nondeterministic* model structure are optimal regardless of model validation criteria (*loss*, *eucl*, and *dtw*). *BCGAIL_nondet_loss* and *BCGAIL_nondet_dtw* configurations are common optimal in both case studies. It means that ENVI with these configurations could generate the most accurate environment models in both cases. Interesting is that ENVI versions that train *nondeterministic* environment model structure using *BCGAIL* algorithm are the common top-3 configurations in both case studies. It seems that the environment model structures and the IL algorithms greatly influence accuracy of the model, but the model validation criteria have less impact on the score.

To further analyze the effect of each ENVI parameter on the model generation, we analyzed the variance of each parameter's imitation score. Figure 6 shows the spreads of the imitation score achieved by each ENVI parameter setting, and Table 4 summarizes the statistical test (Kruskal test) results of the ENVI parameters' effects on the imitation score. The test's null hypothesis is that the distributions of the imitation score achieved by different ENVI parameter settings are identical. Suppose the p-value of the test is smaller than 0.05. In that case, we can reject the null hypothesis, so the imitation score is highly affected by different ENVI parameter settings. It means that users should carefully configure the ENVI parameter.

Both Figure 6 and Table 4 show magnitude of the effect of ENVI parameters on the imitation score and the optimal parameter settings. First, the IL algorithm affects the imitation score most significantly in both case studies. Especially, the BCGAIL algorithm generates the environment model most accurately. Following the IL algorithm, the model structure influences the imitation score. In particular, it is generally better to train the nondeterministic environment model in case study 1. In case study 2, the two model structures do not have a significant difference in the average imitation score, but the nondeterministic environment model could achieve a much lower minimum imitation score than the deterministic model. The validation criteria have the smallest effect on the imitation score in both case studies. All three criteria do not make a difference in the ENVI's performance in case study 2, while using DTW is slightly better in case study 1. This seems to be because DTW can most flexibly compare the noisy FOT and the simulation logs [1].

The RQ1 results first show that the IL algorithm significantly impacts the performance of ENVI. We found BCGAIL algorithm outperforms the other algorithms to generate accurate environment models. Indeed, the BCGAIL algorithm is known to mimic expert behavior better than BC and GAIL [21]. We also confirm that it effectively solves the virtual environment model generation problem. In addition, the nondeterministic environment model structure is more suitable for mimicking the real FOT environment. The real environment state transition is also nondeterministic because CPS FOTs suffer from uncertainties such as sensor noise or non-static road friction. Therefore, the nondeterministic environment model seems appropriate to mimic the uncertain environment. Finally, we can say that the three validation criteria introduced in Section 5.4 have little impact on ENVI's performance, while DTW is recommended in case study 1. However, it also implies that the rationales behind the three criteria are all reasonable.

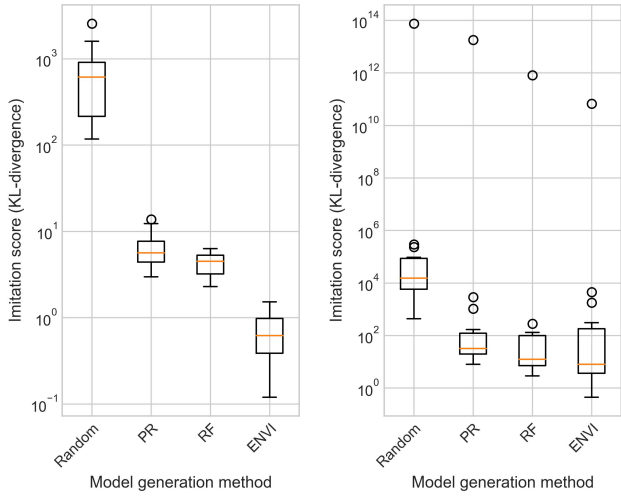
In RQ1, we empirically suggest a guide to optimize ENVI regarding the imitation score based on two case studies. Although the guide is based on our limited case studies and all the parameters introduced in Section 5 are still meaningful in IL, we provide a starting point for the novel use of IL in the environment model generation for CPS goal verification. The following subsections examine how accurate and efficient CPS goal verification using ENVI optimized by *BCGAIL_nondet_dtw* is.

> The answer to RQ1 is that ENVI's imitation score is most influenced by the IL algorithms, followed by the model structures and validation criteria in both case studies. Statistically, *BCGAIL* algorithm, the *nondeterministic* model structure, and *DTW* validation criterion are first recommended, based on our empirical evaluation.

6.5.2 RQ2: Accuracy. RQ2 aims to investigate how well environment models generated by ENVI mimic the real environments and how accurate the ENVI-based verification is. To answer RQ2, ENVI is configured by an optimal setting found in RQ1 (*BCGAIL_nondet_dtw*). We analyze the ENVI-based verification compared to the baselines.

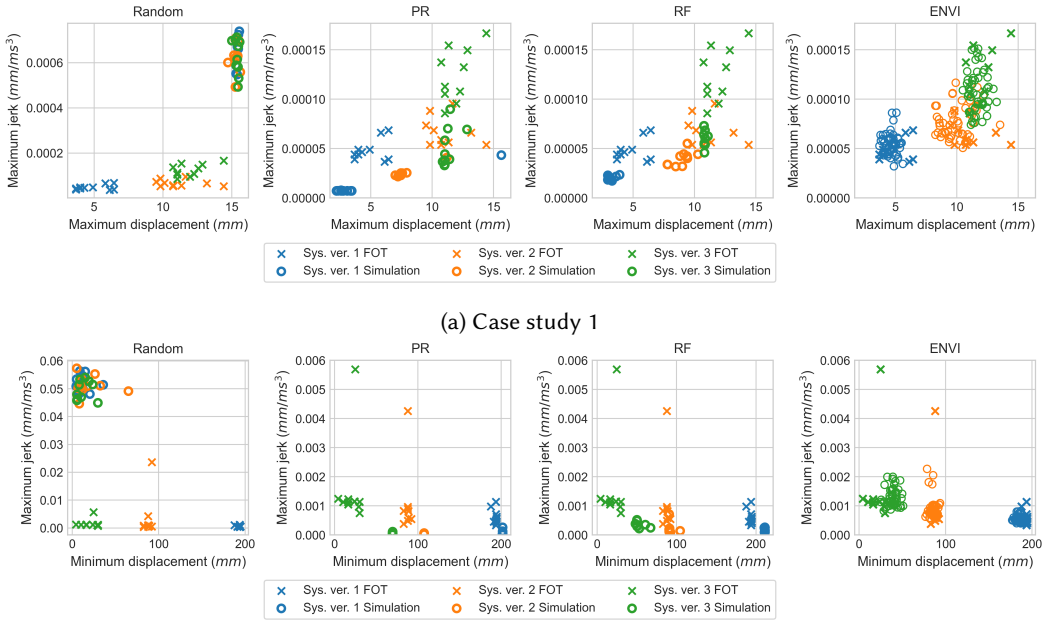
The accuracy of the models can be compared in terms of the imitation score considering the two verification goals together. The more the simulation-based verification result accurately resembles the FOT-based result, the lower the imitation score. Figure 7 shows the log scale spreads of the imitation score of ENVI and the baselines. The theoretically lowest (best) imitation score is zero. As already confirmed in the previous results, ENVI achieves the best imitation scores in overall in both case studies. PR and RF are better than the random model but not as effective as ENVI to mimic the real environment well.

Although we can compare different model generation methods in terms of their imitation scores, it is difficult to interpret the absolute value of the imitation scores. To supplement this, we additionally analyze the CPS goal verification accuracy in terms of safety and passenger comfort. Figure 8



(a) Case study 1 (b) Case study 2

Fig. 7. Comparison of the verification accuracy of ENVI and baselines in terms of imitation score



(a) Case study 1

(b) Case study 2

Fig. 8. Comparison of FOT-based and simulation-based passenger comfort and safety verification results

visualizes multiple passenger comfort and safety assessment results obtained from simulations and FOTs. The x-axis of a scatter diagram is the safety measure (i.e., maximum displacement (*mm*)

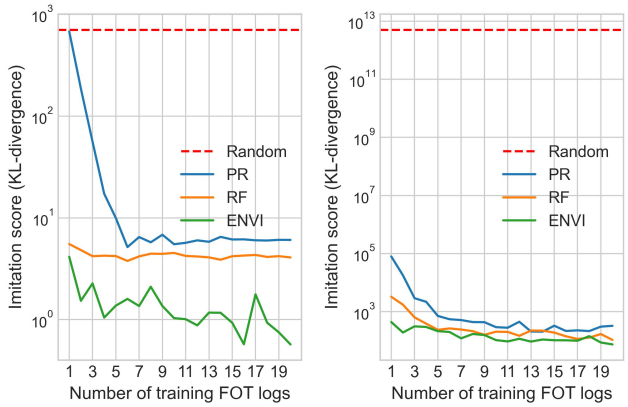
Table 5. Comparison of mean verification errors of ENVI and baselines. The lowest error for each case study and verification goal is highlighted in bold.

		Mean verification error	
		safety (mm)	passenger comfort (mm/ms^3)
Case 1	Random	6.04010	0.00055
	PR	2.36947	0.00005
	RF	1.66293	0.00003
	ENVI	**1.18543**	**0.00002**
Case 2	Random	86.06980	0.04900
	PR	28.33626	0.00233
	RF	18.76562	0.00159
	ENVI	**10.49110**	**0.00121**

from the lane center for the LKS and minimum displacement (mm) from the front safety distance for the ACCS), and the y-axis is the passenger comfort measure (i.e., maximum jerk (mm/ms^3) for both case studies). A FOT/simulation-based verification result is visualized as X/O-shaped dots, respectively. As already described in Section 6.3.1, 10 FOT logs for each controller version are used for testing, so there are 10 X-shaped dots of the same color distinguishing the controller version in each scatter diagram. Based on the testing FOT logs, the environment models under comparison simulate the CPS controllers, marked as 10 O-shaped dots of each color. However, ENVI, used here, trains the nondeterministic environment model, so we repeat the simulation five times using the same testing FOT logs to show the results mitigating the nondeterminism, 50 O-shaped dots are shown in the ENVI diagrams. Remind the virtual environment model generation goal is to make the simulation-based verification result similar to the FOT-based results. Therefore, the closer the distributions of the O-shaped dots and the X-shaped dots, the more accurate and realistic the simulation-based verification.

In Figure 8, we can see that the distribution of ENVI's verification results more overlapped with the distribution of FOT verification results than the baselines. This shows that ENVI-made environment models mimics the real environment well, so the verification results based on the simulations using the model are also realistic compared to the baselines. On the other hand, the vehicle was evaluated as unrealistically unsafe and uncomfortable by the random model, since the environmental state oscillates randomly regardless of the CPS actions. PR and RF models change the verification results depending on the controller versions. However, the distribution of verification results of the two models rarely overlap with the FOT-based results. In particular, the PR and RF models do not properly imitate the uncertainty that emerges in the real world, so even if the test is repeated, many simulation results are mostly the same, which is not the case in reality.

Table 5 shows the mean verification error of the simulation-based verification to interpret the accuracy of the simulation-based verification. Simulation-based comfort and safety assessment results, O-shaped dots in Figure 8, are compared with corresponding FOT-based assessment results, X-shaped dots in Figure 8. As already visually confirmed in Figure 8, Table 5 shows that ENVI's verification errors are smaller than the baselines for all CPS goals and case studies; it means that the ENVI-based verification is the most accurate. For example, it shows that ENVI's mean verification error of safety distance between two robot vehicles in Case Study 2 is about 10mm. Considering the noise of the ultrasonic distance sensor mounted on the robot, this verification error is acceptably small. In other words, the verification results obtained through simulation using ENVI accurately



(a) Case study 1 (b) Case study 2

Fig. 9. Comparison of training data efficiency of ENVI and baselines

replicated the results obtained by repeating many FOTs with small errors in safety and passenger comfort evaluation metrics.

RQ2 results show ENVI can generate virtual environment models that can perform accurate simulation-based verification. When verifying the passenger comfort and safety of the two ADAS using the ENVI-made environment model, the simulation-based verification results were similar to the FOT-based results visually and quantitatively compared to the baselines. Therefore, engineers can accurately verify CPS controllers at a low cost with simulation using ENVI instead of FOT.

> The answer to RQ2 is that ENVI can generate accurate environment models from the seed logs. Specifically, the ENVI-based verification results achieves smaller verification error and lower imitation scores than the baselines for all case studies and verification goals. Thus, ENVI makes CPS goal verification efficient by replacing the real environment with the virtual environment model while keeping the verification result similar to reality.

6.5.3 RQ3: Efficiency. RQ3 aims to investigate the efficiency of ENVI in terms of the number of FOTs required for collecting the training data for environment model generation. Collecting seed logs is a bottleneck in the ENVI process, which is laborious and challenging to accelerate. Therefore ENVI aims to generate an environment model with as small seed logs as possible. To answer RQ3, we reduce the number of FOT logs given to ENVI as training data from 20 to 1 and analyze the change of imitation score compared to the baselines. Remind 20 training and validation FOT logs of each controller version were given in RQ1 and RQ2 as described in Section 6.3.1. However, the number of FOT logs for training the environment models varies in RQ3, while the testing logs are the same. ENVI in RQ3 also uses the optimal configuration found in RQ1.

Figure 9 shows the change in the average imitation score according to the number of FOT logs for training and validation in two case studies. Since the random environment model does not require training data, the score does not vary depending on the number of training FOT logs.

In both case studies, ENVI achieves smaller imitation scores than baselines even when the number of training FOT logs is small overall. In addition, even if ENVI uses only one FOT log for the model generation, it achieves the imitation scores similar to the baselines with 20 FOT logs. It means

that the verification accuracy of ENVI with only one FOT log could be similar to the accuracy of PR and RF shown in RQ2. It implies that IL is very efficient in inferring the real environmental behavior from the small data. From this, we can see that even when small FOT logs are available, ENVI can mimic the real environment well. However, using PR- or RF-made environment models is still better than using random models for verifying CPS controllers.

RQ3 results show that ENVI can generate more accurate environment models with a small amount of seed log data than the baselines. Therefore, ENVI is promising to perform accurate simulation-based CPS goal verification using only a small amount of data when the FOT is costly. It will significantly reduce the cost of CPS goal verification. In addition, the baselines are not as efficient as ENVI, even in our simplified case studies, so ENVI is more applicable in practice. However, applying ENVI to the verification of more complex CPS is still one significant future work.

> The answer to RQ3 is that ENVI can generate environment models with a small number of seed logs compared to the alternative data-driven environment model generation techniques. Therefore, engineers can reduce the cost of FOTs for collecting training data for the environment model by using ENVI.

6.6 Threats to Validity

In terms of external validity, our LEGO-lized autonomous vehicle and two driving assistance systems under verification are simplified for representing the real CPS and software controllers. To mitigate the threat, our case studies were carefully designed to incorporate real CPS controllers, reflecting real-world scenarios. Moreover, these studies vary in their CPS goals, with Case Study 1 focusing on single-vehicle lane-keeping and Case Study 2 addressing adaptive cruise control between two moving vehicles. Although they may differ from the real CPS (e.g., autonomous vehicle), it represents CPS software controllers in practice in terms of continuous interaction with the environment. Applying ENVI to more complex CPS could show different results, but the applicability of ENVI for the simulation-based verification shown in this paper is still valid for CPSs with such software controllers. However, additional case studies with more complex CPS are required to improve our results' generalizability.

In terms of internal validity, the goal verification results based on specific autonomous driving goals (e.g., passenger comfort and safety) could be a potential threat since the evaluation of the driving assistance controller's goal could be biased to a specific aspect of driving. To mitigate this threat, in our evaluation, we chose two popular and important goals motivated by industrial standards such as ISO 11270 for LKS [18] and ISO 15622 for ACCS [19] specifying acceptable safety and comfort. We then aggregated the results on both goals to comprehensively understand whether the subject controllers work well or not. Hyperparameter value settings for IL (e.g., number of iterations, learning rates, Etc.) could be another potential threat to the internal validity since the performance of machine learning can largely depend on hyperparameter values. We used the values recommended in the original studies [15, 41]. Nevertheless, hyperparameter tuning is an important research field, so it remains an interesting future work.

7 DISCUSSION

One of our main contributions is bringing attention to the novel problem of virtual environment model generation using imitation learning in CPS goal verification. This section discusses various characteristics and limitations of ENVI, as well as open challenges and future research directions.

Efficiency. Sample efficiency is essential to ENVI. This is because conducting FOTs to collect logs is the most expensive task in the data-driven approach. As the cost of generating accurate virtual environment models increases, the incentive to replace field operational testing (FOT) with simulation diminishes. In our experiments, ENVI, especially using BCGAIL algorithm, was the most efficient environment model generation approach than the other baselines in most cases. Using state-of-the-art IL algorithms for increasing sample efficiency [21, 59] could further help.

Scalability. The scalability of ENVI also largely depends on the IL algorithms used. Recent studies have demonstrated that IL algorithms can handle input variables with dimensions of up to $30,000$ (i.e., three channels of images size of 100×100) [46]. Generative IL algorithms, such as GAIL, have also proven effective in generating high-dimensional data (e.g., images and videos) from demonstration examples [22]. Therefore, ENVI can handle a much larger number of environment variables. It is important to note that ENVI does not prescribe any specific IL algorithm to use; instead, it is a general approach, as shown in Figure 2, that aims to address the problem of virtual environment model generation.

Generalizability. ENVI is domain-agnostic by serving formal problem definition of the virtual environment model generation for CPS goal verification and a multifaceted solution using imitation learning. Therefore, ENVI approach is deliberately designed to be agnostic to CPS specifics, making it suitable for broader CPS goal verification applications. While we have validated our idea in the automotive domain, optimizing ENVI for specific domains is an interesting future work.

Explainability. As with any deep learning-based approach, ENVI also lacks explainability for trained environment models. However, ENVI includes a validation stage (Stage 4; Section 5.4), where trained models are evaluated using a portion of seed logs collected in Stage 1 (Section 5.1) before being used for CPS verification in Stage 5 (Section 5.5). In addition, domain experts can further validate the performance of virtual environment models meet certain pre-defined criteria. Furthermore, with the recent advances in explainable IL algorithms [9, 30, 58], the explainability of ENVI would also be explicitly improved.

Robustness. ENVI should be robust to noise in FOT logs. Many IL studies assume the correctness of the expert demonstration [2, 34]. However, the *expert* in our problem is the real environment, so some level of noise is inevitable in the demonstration data (e.g., due to sensor noise). Though we used noisy data collected by the real CPSs in the experiments, systematically investigating the impact of noise was not in the scope of our work. Nevertheless, as many studies have already considered the noise issue in machine learning [14, 57], they could better guide how to address noisy FOT logs in ENVI.

Environment Abstraction. Finding a proper level of abstraction for the complex environment is important. We abstracted the environment as a state-transition function in a closed-loop simulation and recast the model generation problem as the IL problem (see Section 4). This is a typical level of abstraction for the environment modeling [36, 38, 43]. However, this simple representation may not be sufficient for some domains. Therefore, an extension of the environment model is an interesting future work, and we can also refer to some IL studies that imitate complex expert behaviors (e.g., multi-task or concurrent behavior) [4, 46].

Data-Driven vs. Knowledge-Based. A hybrid of data-driven and knowledge-based environment modeling can make the model further effective. If a high-fidelity simulation engine is based on well-known principles in the CPS domain, engineers could manually create an accurate virtual environment in the simulator. In contrast to such knowledge-based environment modeling, ENVI is a data-driven approach in which only a few seed logs are required to generate an accurate virtual

environment model automatically. This is a huge advantage in inferring complex environmental behavior from data. Therefore, ENVI can complement the knowledge-based approach depending on the application domain.

8 RELATED WORK

Modeling the change of observable environment according to the CPS operation is widely used to verify to what extent the CPS (controller) under analysis can achieve its goals or requirements.

In the formal verification perspective, the dynamics of the environmental state affected by the controller actions are embedded in the system model under verification [20, 53]. For example, Ding et al. [11] modeled the continuous environmental state transition as a continuous place in an extension of Petri nets, and model checking verified the system properties. Tran et al. [51] specified the vehicle's dynamics according to the control actions and performed safety verification. Wang et al. [54] modeled the system and its environmental dynamics as a probabilistic model and verified the model using statistical model checking. On the other hand, Cámara et al. [10] and Moreno et al. [27] modeled the environment decoupled from the system model in Markov Decision Process (MDP) and verified the combination of the system and the environment models. Our work is close to [10, 27, 54] in terms of using statistical verification and generating decoupled environment model, but it is differentiated in focusing on automated model generation.

The environment model also guides the CPS testing on simulation. For example, Püschel et al. [35] modeled the environment configuration (e.g., location of the obstacle) variability under analysis in the CPS simulation. Qin et al. [36] and Reichstaller and Knapp [38] explicitly modeled the interaction between the CPS and environment as a closed-loop similar to our CPS-ENV interaction model for test case generation. Sood et al. [47] recently also proposed a testing and validation approach with a closed-loop model of the controller and its surrounding dynamics of the CPS. Arrieta et al. [6] optimized the CPS test suite that manipulates the virtual environment to improve CPS testing coverages. These are close to our work in terms that the virtual environment models specify the environmental situations under analysis that the CPS may encounter in the real world. Still, the simulation-based analysis result could be inconsistent with the real CPS operation depending on engineer knowledge. Our work attempts to reduce the inconsistency of the simulation and field test as much as possible based on the data.

Several studies have also proposed data-driven environment modeling. Ding et al. [11] learned the environment state variables in the Petri net model from the runtime data. Aizawa et al. [5] and Sykes et al. [50] inferred the environment model as a labeled transition system (LTS), and a logic program from the CPS execution traces, respectively. Moreno et al. [27] forecasted the future environmental change and represented the change in a probability tree model. Unfortunately, the existing approaches revise the initial environment model made by an expert or infer some part of the environment model from data, so sufficient domain knowledge is still necessary for the accurate environment-model-based analysis of the CPS. However, our work automatically generates the virtual environment model from small CPS FOT logs without significant domain knowledge by abstracting the environment model as a black-box function. In addition, to the best of our knowledge, this is the first work to formally define the problem of virtual environment model generation and propose a solution leveraging IL.

9 CONCLUSION

This paper presents ENVI, a novel data-driven environment imitation approach that efficiently generates accurate virtual environment models for CPS goal verification. Instead of conducting expensive FOTs many times, ENVI requires only a few FOTs to train a virtual environment model. By leveraging IL, an accurate virtual environment model can be generated automatically from the

collected seed logs. Specifically, we provided (1) the formal framework of the CPS goal verification and problem definition of the virtual environment model generation, (2) the process of ENVI with user-configurable parameters, (3) an empirical evaluation of the ENVI with two case studies using robot vehicles, and (4) discussions on the research direction on the virtual environment model generation. We suggested optimal configurations of the ENVI based on the empirical evaluations. The evaluation results finally show that ENVI can generate the virtual environment models efficiently with a few seed logs, and the simulation-based CPS goal verification using ENVI was more accurate than the alternative techniques.

In future work, we plan to apply ENVI in the evolutionary development and verification of the CPS software controller. By this, ENVI can further reduce the need for laborious FOTs for the CPS goal verification. In addition, we expect that ENVI is not limited to the purpose of CPS controller verification, so we also plan to suggest a new application of ENVI, such as an optimal CPS control predicting the environmental reaction.

ACKNOWLEDGMENTS

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2023-2020-0-01795) and (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System (No. 2015-0-00250) supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation). This work was supported by Electronics and Telecommunications Research Institute(ETRI) grant funded by the Korean government. [23ZS1300, Research on High Performance Computing Technology to overcome limitations of AI processing]

REFERENCES

[1] Amaia Abanda, Usue Mori, and Jose A Lozano. 2019. A review on distance based time series classification. *Data Mining and Knowledge Discovery* 33, 2 (2019), 378–412.

[2] Mohammed Abdou, Hanan Kamal, Samah El-Tantawy, Ali Abdelkhalek, Omar Adel, Karim Hamdy, and Mustafa Abaas. 2019. End-to-End Deep Conditional Imitation Learning for Autonomous Driving. In *2019 31st International Conference on Microelectronics (ICM)*. 346–350. https://doi.org/10.1109/ICM48031.2019.9021288

[3] Arend Aerts, Michel Reniers, and Mohammad Reza Mousavi. 2017. Model-based testing of cyber-physical systems. In *Cyber-Physical Systems*. Elsevier, 287–304.

[4] Shailen Agrawal and Michiel van de Panne. 2016. Task-Based Locomotion. *ACM Trans. Graph.* 35, 4, Article 82 (jul 2016), 11 pages. https://doi.org/10.1145/2897824.2925893

[5] Kazuya Aizawa, Kenji Tei, and Shinichi Honiden. 2018. Identifying safety properties guaranteed in changed environment at runtime. In *2018 IEEE International Conference on Agents (ICA)*. 75–80. https://doi.org/10.1109/AGENTS.2018.8460083

[6] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2017. Search-based test case generation for cyber-physical systems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 688–697.

[7] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *The impact of control technology* 12, 1 (2011), 161–166.

[8] Yvonne Barnard, Satu Innamaa, Sami Koskinen, Helena Gellerman, Erik Svanberg, and Haibo Chen. 2016. Methodology for field operational tests of automated vehicles. *Transportation research procedia* 14 (2016), 2188–2196.

[9] Tom Bewley, Jonathan Lawry, and Arthur Richards. 2020. Modelling agent policies with interpretable imitation learning. In *International Workshop on the Foundations of Trustworthy AI Integrating Learning, Optimization and Reasoning*. Springer, 180–186.

[10] Javier Cámara, Wenxin Peng, David Garlan, and Bradley Schmerl. 2018. Reasoning About Sensing Uncertainty in Decision-Making for Self-adaptation. In *Software Engineering and Formal Methods*, Antonio Cerone and Marco Roveri (Eds.). Springer International Publishing, Cham, 523–540.

[11] Zuohua Ding, Yuan Zhou, and Mengchu Zhou. 2016. Modeling Self-Adaptive Software Systems With Learning Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46, 4 (2016), 483–498. https://doi.org/10.1109/TSMC.2015.2433892

[12] Erik M. Fredericks. 2016. Automatically Hardening a Self-Adaptive System against Uncertainty. In *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 16–27. https://doi.org/10.1109/SEAMS.2016.010

[13] Ziwei Guan, Tengyu Xu, and Yingbin Liang. 2021. When will generative adversarial imitation learning algorithms attain global convergence. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1117–1125.
[14] Shivani Gupta and Atul Gupta. 2019. Dealing with Noise Problem in Machine Learning Data-sets: A Systematic Review. *Procedia Computer Science* 161 (2019), 466–474. https://doi.org/10.1016/j.procs.2019.11.146 The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.
[15] Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 4572–4580.
[16] Anthony Hu, Gianluca Corrado, Nicolas Griffiths, Zachary Murez, Corina Gurau, Hudson Yeo, Alex Kendall, Roberto Cipolla, and Jamie Shotton. 2022. Model-based imitation learning for urban driving. *Advances in Neural Information Processing Systems* 35 (2022), 20703–20716.
[17] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.* 50, 2, Article 21 (apr 2017), 35 pages. https://doi.org/10.1145/3054912
[18] ISO 11270:2014 2014. *Intelligent transport systems — Lane keeping assistance systems (LKAS) — Performance requirements and test procedures.* Standard. International Organization for Standardization.
[19] ISO 15622:2018 2018. *Intelligent transport systems — Adaptive cruise control systems — Performance requirements and test procedures.* Standard. International Organization for Standardization.
[20] Radoslav Ivanov, Taylor J Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2020. Verifying the safety of autonomous systems with neural network controllers. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 1 (2020), 1–26.
[21] Rohit Jena, Changliu Liu, and Katia Sycara. 2020. Augmenting GAIL with BC for sample efficient imitation learning. *arXiv* (2020). arXiv:2001.07798 [cs.LG]
[22] Mladan Jovanovic and Mark Campbell. 2022. Generative artificial intelligence: Trends and prospects. *Computer* 55, 10 (2022), 107–112.
[23] James M Joyce. 2011. Kullback-leibler divergence. In *International encyclopedia of statistical science*. Springer, 720–722.
[24] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. *arXiv* (2017). arXiv:1412.6980 [cs.LG]
[25] Axel Legay, Benoît Delahaye, and Saddek Bensalem. 2010. Statistical model checking: An overview. In *International conference on runtime verification*. Springer, 122–135.
[26] Bo Hu Li, Lin Zhang, Tan Li, Ting Yu Lin, and Jin Cui. 2017. Simulation-Based Cyber-Physical Systems and Internet-of-Things. In *Guide to Simulation-Based Disciplines*. Springer, 103–126.
[27] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2018. Flexible and Efficient Decision-Making for Proactive Latency-Aware Self-Adaptation. *ACM Trans. Auton. Adapt. Syst.* 13, 1, Article 3 (apr 2018), 36 pages. https://doi.org/10.1145/3149180
[28] Thuy Nguyen. 2017. A modelling & simulation based engineering approach for socio-cyber-physical systems. In *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*. IEEE, 702–707.
[29] Eva Ostertagová. 2012. Modelling using polynomial regression. *Procedia Engineering* 48 (2012), 500–506.
[30] Menghai Pan, Weixiao Huang, Yanhua Li, Xun Zhou, and Jun Luo. 2020. xgail: Explainable generative adversarial imitation learning for explainable human decision analysis. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1334–1343.
[31] Angela Pappagallo, Annalisa Massini, and Enrico Tronci. 2020. Monte carlo based statistical model checking of cyber-physical systems: A review. *Information* 11, 12 (2020), 588.
[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA, 8026–8037.
[33] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[34] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Trans. Graph.* 37, 4, Article 143 (jul 2018), 14 pages. https://doi.org/10.1145/3197517.3201311
[35] Georg Püschel, Christian Piechnick, Sebastian Götz, Christoph Seidl, Sebastian Richly, Thomas Schlegel, and Uwe Aßmann. 2014. A combined simulation and test case generation strategy for self-adaptive systems. *Journal On Advances in Software* 7, 3&4 (2014), 686–696.
[36] Yi Qin, Chang Xu, Ping Yu, and Jian Lu. 2016. SIT: Sampling-based interactive testing for self-adaptive apps. *Journal of Systems and Software* 120 (2016), 70–88. https://doi.org/10.1016/j.jss.2016.07.002

[37] M.Y Rafiq, G Bugmann, and D.J Easterbrook. 2001. Neural network design for engineering applications. *Computers & Structures* 79, 17 (2001), 1541–1552. https://doi.org/10.1016/S0045-7949(01)00039-6

[38] André Reichstaller and Alexander Knapp. 2018. Risk-Based Testing of Self-Adaptive Systems Using Run-Time Predictions. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. 80–89. https://doi.org/10.1109/SASO.2018.00019

[39] Stefan Schaal. 1996. Learning from Demonstration. In *Advances in Neural Information Processing Systems*. 1040–1046. http://papers.nips.cc/paper/1224-learning-from-demonstration

[40] Achim Schilling, Claus Metzner, Jonas Rietsch, Richard Gerum, Holger Schulze, and Patrick Krauss. 2019. How deep is deep enough? – Quantifying class separability in the hidden layers of deep neural networks. *arXiv* (2019). arXiv:1811.01753 [cs.LG]

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv* (2017). arXiv:1707.06347 [cs.LG]

[42] Mark R Segal. 2004. Machine learning benchmarks and random forest regression. (2004).

[43] Yong-Jun Shin, Joon-Young Bae, and Doo-Hwan Bae. 2021. Concepts and Models of Environment of Self-Adaptive Systems: A Systematic Literature Review. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. 296–305. https://doi.org/10.1109/APSEC53868.2021.00037

[44] Yong-Jun Shin, Eunho Cho, and Doo-Hwan Bae. 2021. Pasta: An efficient proactive adaptation approach based on statistical model checking for self-adaptive systems. In *International Conference on Fundamental Approaches to Software Engineering*. Springer International Publishing Cham, 292–312.

[45] Yong-Jun Shin, Lingjun Liu, Sangwon Hyun, and Doo-Hwan Bae. 2021. Platooning LEGOs: An Open Physical Exemplar for Engineering Self-Adaptive Cyber-Physical Systems-of-Systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 231–237. https://doi.org/10.1109/SEAMS51251.2021.00038

[46] Avi Singh, Eric Jang, Alexander Irpan, Daniel Kappler, Murtaza Dalal, Sergey Levinev, Mohi Khansari, and Chelsea Finn. 2020. Scalable Multi-Task Imitation Learning with Autonomous Improvement. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2167–2173. https://doi.org/10.1109/ICRA40945.2020.9197020

[47] Surinder Sood, Avinash Malik, and Partha Roop. 2019. Robust Design and Validation of Cyber-physical Systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 6 (2019), 1–21.

[48] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[49] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12 (1999).

[50] Daniel Sykes, Domenico Corapi, Jeff Magee, Jeff Kramer, Alessandra Russo, and Katsumi Inoue. 2013. Learning revised models for planning in adaptive systems. In *2013 35th International Conference on Software Engineering (ICSE)*. 63–71. https://doi.org/10.1109/ICSE.2013.6606552

[51] Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T Johnson, and Xenofon Koutsoukos. 2019. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–22.

[52] Yoshihisa Tsurumine, Yunduan Cui, Kimitoshi Yamazaki, and Takamitsu Matsubara. 2019. Generative adversarial imitation learning with deep p-network for robotic cloth manipulation. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE, 274–280.

[53] Yu Wang, Nima Roohi, Matthew West, Mahesh Viswanathan, and Geir E Dullerud. 2021. Verifying Stochastic Hybrid Systems with Temporal Logic Specifications via Model Reduction. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 6 (2021), 1–27.

[54] Yu Wang, Mojtaba Zarei, Borzoo Bonakdarpour, and Miroslav Pajic. 2019. Statistical verification of hyperproperties for cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–23.

[55] Li Da Xu and Lian Duan. 2019. Big data for cyber physical systems in industry 4.0: a survey. *Enterprise Information Systems* 13, 2 (2019), 148–169. https://doi.org/10.1080/17517575.2018.1442934

[56] Wenhua Yang, Chang Xu, Minxue Pan, Chun Cao, Xiaoxing Ma, and Jian Lu. 2018. *Journal of Systems and Software* 138 (2018), 82–99. https://doi.org/10.1016/j.jss.2017.12.009

[57] Zhi Zeng, Yuan Liu, Weijun Tang, and Fangjiong Chen. 2021. Noise Is Useful: Exploiting Data Diversity for Edge Intelligence. *IEEE Wireless Communications Letters* 10, 5 (2021), 957–961. https://doi.org/10.1109/LWC.2021.3051688

[58] Dandan Zhang, Qiang Li, Yu Zheng, Lei Wei, Dongsheng Zhang, and Zhengyou Zhang. 2021. Explainable hierarchical imitation learning for robotic drink pouring. *IEEE Transactions on Automation Science and Engineering* 19, 4 (2021), 3871–3887.

[59] Xin Zhang, Yanhua Li, Ziming Zhang, and Zhi-Li Zhang. 2020. f-GAIL: Learning f-Divergence for Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 12805–12815. https://proceedings.neurips.cc/paper/2020/file/967990de5b3eac7b87d49a13c6834978-Paper.pdf

[60] Xin Zhang, Yanhua Li, Xun Zhou, and Jun Luo. 2020. CGAIL: Conditional generative adversarial imitation learning—An application in taxi Drivers' strategy learning. *IEEE Transactions on Big Data* (2020).

Received 25 August 2022; revised 2 October 2023; accepted 6 November 2023