Noname manuscript No.
(will be inserted by the editor)

Compositional Falsification of Cyber-Physical Systems with Machine Learning Components

Tommaso Dreossi · Alexandre Donzé · Sanjit A. Seshia

the date of receipt and acceptance should be inserted later

Abstract Cyber-physical systems (CPS), such as automotive systems, are starting to include sophisticated machine learning (ML) components. Their correctness, therefore, depends on properties of the inner ML modules. While learning algorithms aim to generalize from examples, they are only as good as the examples provided, and recent efforts have shown that they can produce inconsistent output under small adversarial perturbations. This raises the question: can the output from learning components lead to a failure of the entire CPS? In this work, we address this question by formulating it as a problem of falsifying signal temporal logic (STL) specifications for CPS with ML components. We propose a compositional falsification framework where a temporal logic falsifier and a machine learning analyzer cooperate with the aim of finding falsifying executions of the considered model. The efficacy of the proposed technique is shown on an automatic emergency braking system model with a perception component based on deep neural networks.

Keywords Cyber-physical systems, machine learning, falsification, temporal logic, deep learning, neural networks, autonomous driving

This work is funded in part by the DARPA BRASS program under agreement number FA8750-16-C-0043, NSF grants CNS-1646208, CNS-1545126, and CCF-1139138, and by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. The second author did much of the work while affiliated with UC Berkeley. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

T. Dreossi
University of California, Berkeley
E-mail: dreossi@berkeley.edu

A. Donzé
Decyphir, Inc.
E-mail: alex.r.donze@gmail.com

S. A. Seshia
University of California, Berkeley
E-mail: sseshia@berkeley.edu

1 Introduction

Over the last decade, machine learning (ML) algorithms have achieved impressive results providing solutions to practical large-scale problems (see, e.g., [4, 25, 18, 15]). Not surprisingly, ML is being used in cyber-physical systems (CPS) — systems that are integrations of computation with physical processes. For example, semi-autonomous vehicles employ Adaptive Cruise Controllers (ACC) or Lane Keeping Assist Systems (LKAS) that rely heavily on image classifiers providing input to the software controlling electric and mechanical subsystems (see, e.g., [5]). The safety-critical nature of such systems involving ML raises the need for formal methods [33]. In particular, how do we systematically find bugs in such systems?

We formulate this question as the falsification problem for CPS models with ML components (CPSML): given a formal specification φ (say in a formalism such as signal temporal logic [22]) and a CPSML model M, find an input for which M does *not* satisfy φ. A falsifying input generates a counterexample trace that reveals a bug. To solve this problem, multiple challenges must be tackled. First, the input space to be searched can be intractable. For instance, a simple model of a semi-autonomous car already involves several control signals (e.g., the angle of the acceleration pedal, steering angle) and other rich sensor input (e.g., images captured by a camera, LIDAR, RADAR). Second, CPSML are often designed using languages (such as C, C++, or Simulink), for which clear semantics are not given, and involve third-party components that are opaque or poorly-specified. This obstructs the development of formal methods for the analysis of CPSML models and may force one to treat them as gray-boxes or black-boxes. Third, the formal verification of ML components is a difficult, and somewhat ill-posed problem due to the complexity of the underlying ML algorithms, large feature spaces, and the lack of consensus on a formal definition of correctness (see [33] for a longer discussion). Hence, we need a technique to systematically analyze ML components within the context of a CPS.

In this paper, we propose a new framework for the falsification of CPSML addressing the issues described above. Our technique is compositional (modular) in that it divides the search space for falsification into that of the ML component and of the remainder of the system, while establishing a connection between the two. The obtained subspaces are respectively analyzed by a temporal logic falsifier ("CPS Analyzer") and a machine learning analyzer ("ML analyzer") that cooperate to search for a behavior of the closed-loop system that violates the property φ. This cooperation mainly comprises a sequence of input space projections, passing information about interesting regions in the input space of the full CPSML model to identify a sub-space of the input space of the ML component. The resulting projected input space of the ML component is typically smaller than the full input space. Moreover, misclassifications in this space can be mapped back to smaller subsets of the CPSML input space in which counterexamples are easier to find. Importantly, our approach can handle *any machine learning technique*, including the methods based on

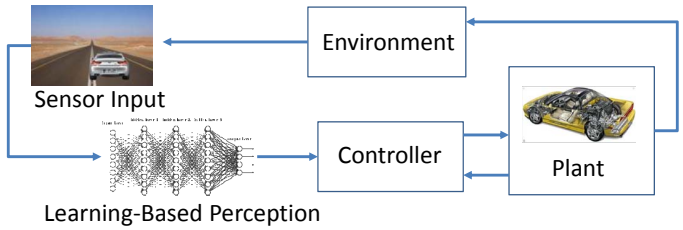


Fig. 1: Automatic Emergency Braking System (AEBS) in closed loop. A machine learning based image classifier is used to perceive objects in the ego vehicle's frame of view.

deep neural networks [15] that have proved effective in many recent applications. The proposed ML Analyzer is a tool that analyzes the input space for the ML classifier and determines a region of the input space that could be relevant for the full cyber-physical system's correctness. More concretely, the analyzer identifies sets of misclassifying features, i.e., inputs that "fool" the ML algorithm. The analysis is performed by considering subsets of parameterized features spaces that are used to approximate the ML components by simpler functions. The information gathered by the temporal logic falsifier and the ML analyzer together reduce the search space, providing an efficient approach to falsification for CPSML models.

Example 1 As an illustrative example, let us consider a simple model of an Automatic Emergency Braking System (AEBS), that attempts to detect objects in front of a vehicle and actuate the brakes when needed to avert a collision. Figure 1 shows the AEBS as a system composed of a controller (automatic braking), a plant (vehicle sub-system under control, including transmission), and an advanced sensor (camera along with an obstacle detector based on deep learning). The AEBS, when combined with the vehicle's environment, forms a closed loop control system. The controller regulates the acceleration and braking of the plant using the velocity of the subject (ego) vehicle and the distance between it and an obstacle. The sensor used to detect the obstacle includes a camera along with an image classifier based on deep neural networks. In general, this sensor can provide noisy measurements due to incorrect image classifications which in turn can affect the correctness of the overall system.

Suppose we want to verify whether the distance between the subject vehicle and a preceding obstacle is always larger than 5 meters. Such a verification requires the exploration of a very large input space comprising the control inputs (e.g., acceleration and braking pedal angles) and the ML component's feature space (e.g., all the possible pictures observable by the camera). The latter space is particularly large — for example, note that the feature space of RGB images of dimension 1000×600px (for an image classifier) contains $256^{1000\times600\times3}$ elements.

At first, the input space of the model described in Example 1 appears intractably large. However, the twin notions of *abstraction* and *compositionality*, central to much of the advances in formal verification, can help address this challenge. As mentioned earlier, we decompose the overall CPSML model input space into two parts: (i) the input space of the ML component, and (ii) the input space for the rest of the system – i.e., the CPSML model with an abstraction of the ML component. A *CPS Analyzer* operates on the latter "pure CPS" input space, while an *ML Analyzer* handles the former. The two analyzers communicate information as follows:

1. The CPS Analyzer initially performs conservative analyses assuming abstractions of the ML component. In particular, consider two extreme abstractions — a "perfect ML classifier" (i.e., all feature vectors are correctly classified), and a "completely-wrong ML classifier" (all feature vectors are misclassified). Abstraction permits the CPS Analyzer to operate on a lower-dimensional input space (the "pure CPS" one) and identify a region in this space that may be affected by the malfunctioning of some ML modules – a so-called "region of interest" or "region of uncertainty." This region is communicated to the ML Analyzer.
2. The ML Analyzer projects the region of uncertainty (ROU) onto its input space, and performs a detailed analysis of that input sub-space. Since this detailed analysis uses only the ML classifier (not the full CPSML model), it is a more tractable problem. In this paper, we present a novel sampling-based approach to explore the input sub-space for the ML component. We can also leverage other advances in analysis of machine learning systems operating on rich sensor inputs and for applications such as autonomous driving (see the related work section that follows).
3. When the ML Analyzer finds interesting test cases (e.g., those that trigger misclassifications of inputs whose labels are easily inferred), it communicates that information back to the CPS Analyzer, which checks whether the ML misclassification can lead to a system-level safety violation (e.g., a collision). If yes, we have found a system-level counterexample. If not, the ROU is updated and the revised ROU passed back to the ML Analyzer.

The communication between the CPS Analyzer and ML Analyzer continues until either we find a system-level counterexample, or we run out of resources. For the class of CPSML models we consider, including those with highly non-linear dynamics and even black-box components, one cannot expect to prove system correctness. We focus on specifications in Signal Temporal Logic (STL), and for this reason use a temporal logic falsifier, Breach [8], as our CPS Analyzer. Even though temporal logic falsification is a mature technology with initial industrial adoption (e.g., [42]), several technical challenges remain. First, we need to construct the validity domain of an STL specification — the input sub-space where the property is satisfied — for a CPSML model with abstracted (correct/incorrect) ML components, and identify the region of uncertainty (ROU). Second, we need a method to relate the ROU to the feature space of the ML modules. Third, we need to systematically analyze the feature space of the ML component with the goal of finding feature vectors leading

to misclassifications. We describe in detail in Sections 3 and 4 how we tackle these challenges.

In summary, the main contributions of this paper are:

- A compositional framework for the falsification of temporal logic properties of arbitrary CPSML models that works for any machine learning classifier.
- A machine learning analyzer that identifies misclassifications leading to system-level property violations, based on two main ideas:
  - An input space parameterization used to abstract the feature space of the ML component and relate it to the CPSML input space, and
  - A classifier approximation method used to abstract the ML component and identify misclassifications that can lead to executions of the CPSML that violate the temporal logic specification.
- An experimental demonstration of the effectiveness of our approach on two instantiations of the Automatic Emergency Braking System (AEBS) example with multiple deep neural networks trained for object detection and classification, including some developed by experts in the machine learning and computer vision communities.

In Sec. 5, we give detailed experimental results on an Automatic Emergency Braking System (AEBS) involving an image classifier for obstacle detection based on deep neural networks developed and trained using leading software packages — AlexNet developed with Caffe [18] and Inception-v3 developed with TensorFlow [23]. In this journal version of our original conference paper [10], we also present a new case study, an AEBS deployed within the Udacity self-driving car simulator [2] trained on images generated from the simulator.

Related Work

The verification of both CPS and ML algorithms have attracted several research efforts, and we focus here on the most closely related work. Techniques for the falsification of temporal logic specifications against CPS models have been implemented based on nonlinear optimization methods and stochastic search strategies (e.g., Breach [8], S-TaLiRo [3], RRT-REX [9], C2E2 [12]). While the verification of ML programs is less well-defined [33], recent efforts [36] show how even well trained neural networks can be sensitive to small adversarial perturbations, i.e., small intentional modifications that lead the network to misclassify the altered input with large confidence. Other efforts have tried to characterize the correctness of neural networks in terms of risk [39] (i.e., probability of misclassifying a given input) or robustness [13, 7] (i.e., a minimal perturbation leading to a misclassification), while others proposed methods to generate pictures [28,11] or perturbations [26,28,16] including methods based on satisfiability modulo theories (SMT) [20] in such a way to "fool" neural networks. These methods, while very promising, are mostly limited to analyzing the ML components in isolation, and not in the

context of a complex, closed-loop cyber-physical system. To the best of our knowledge, our work is the first to address the verification of temporal logic properties of CPSML—the combination of CPS and ML systems. The work that is closest in spirit to ours is that on DeepXplore [31], where the authors present a whitebox software testing approach for deep learning systems. However, there are some important differences: their work performs a detailed analysis of the learning software, whereas ours analyzes the entire closed-loop CPS while delegating the software analysis to the machine learning analyzer. Further, we consider temporal logic falsification whereas their work uses software and neural network coverage metrics. It may be interesting to see how these approaches can be combined.

2 Background

2.1 CPSML Models

In this work, we consider models of cyber-physical systems with machine learning components (CPSML). We assume that a system model is given as a gray-box simulator defined as a tuple $M = (S, U, sim)$, where S is a set of system states, U is a set of input values, and $sim : S \times U \times T \to S$ is a simulator that maps a state $\mathbf{x}(t_k) \in S$ and input value $\mathbf{u}(t_k) \in U$ at time $t_k \in T$ to a new state $\mathbf{x}(t_{k+1}) = sim(\mathbf{x}(t_k), \mathbf{u}(t_k), t_k)$, where $t_{k+1} = t_k + \Delta_k$ for a time-step $\Delta_k \in \mathbb{Q}_{>0}$.

Given an initial time $t_0 \in T$, an initial state $\mathbf{x}(t_0) \in S$, a sequence of time-steps $\Delta_0, \ldots, \Delta_n \in \mathbb{Q}_{>0}$, and a sequence of input values $\mathbf{u}(t_0), \ldots, \mathbf{u}(t_n) \in U$, a simulation trace of the model $M = (S, U, sim)$ is a sequence:

$$(t_0, \mathbf{x}(t_0), \mathbf{u}(t_0)), (t_1, \mathbf{x}(t_1), \mathbf{u}(t_1)), \ldots, (t_n, \mathbf{x}(t_n), \mathbf{u}(t_n))$$

where $\mathbf{x}(t_{k+1}) = sim(\mathbf{x}(t_k), \mathbf{u}(t_k), \Delta_k)$ and $t_{k+1} = t_k + \Delta_k$ for $k = 0, \ldots, n$.

The gray-box aspect of the CPSML model is that we assume some knowledge of the internal ML components. Specifically, these components, termed *classifiers*, are functions $f : X \to Y$ that assign to their input *feature vector* $\mathbf{x} \in X$ a *label* $y \in Y$, where X and Y are a feature and label space, respectively. Without loss of generality, we focus on binary classifiers whose label space is $Y = \{0, 1\}$. A ML algorithm selects a classifier using a training set $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$ where the $(\mathbf{x}^{(i)}, y^{(i)})$ are labeled examples with $\mathbf{x}^{(i)} \in X$ and $y^{(i)} \in Y$, for $i = 1, \ldots, m$. The quality of a classifier can be estimated on a test set of examples comparing the classifier predictions against the labels of the examples. Precisely, for a given test set $T = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(l)}, y^{(l)})\}$, the number of false positives $fp_f(T)$ and false negatives $fn_f(T)$ of a classifier f on T are defined as:

$$\begin{aligned} fp_f(T) &= \mid \{\mathbf{x}^{(i)} \in T \mid f(\mathbf{x}^{(i)}) = 1 \text{ and } y^{(i)} = 0\} \mid \\ fn_f(T) &= \mid \{\mathbf{x}^{(i)} \in T \mid f(\mathbf{x}^{(i)}) = 0 \text{ and } y^{(i)} = 1\} \mid \end{aligned} \tag{1}$$

The error rate of f on T is given by:

$$err_f(T) = (fp_f(T) + fn_f(T))/l \tag{2}$$

A low error rate implies good predictions of the classifier f on the test set T.

2.2 Signal Temporal Logic

We consider Signal Temporal Logic [22] (STL) as the language to specify properties to be verified against a CPSML model. STL is an extension of linear temporal logic (LTL) suitable for the specification of properties of CPS.

A *signal* is a function $s : D \to S$, with $D \subseteq \mathbb{R}_{\geq 0}$ an interval and either $S \subseteq \mathbb{B}$ or $S \subseteq \mathbb{R}$, where $\mathbb{B} = \{\top, \bot\}$ and $\mathbb{R}$ is the set of reals. Signals defined on $\mathbb{B}$ are called *booleans*, while those on $\mathbb{R}$ are said *real-valued*. A *trace* $w = \{s_1, \ldots, s_n\}$ is a finite set of real-valued signals defined over the same interval D.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a finite set of predicates $\sigma_i : \mathbb{R}^n \to \mathbb{B}$, with $\sigma_i \equiv p_i(x_1, \ldots, x_n) \lhd 0$, $\lhd \in \{<, \leq\}$, and $p_i : \mathbb{R}^n \to \mathbb{R}$ a function in the variables $x_1, \ldots, x_n$.

An STL formula is defined by the following grammar:

$$\varphi := \sigma \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \varphi U_I \varphi \tag{3}$$

where $\sigma \in \Sigma$ is a predicate and $I \subset \mathbb{R}_{\geq 0}$ is a closed non-singular interval. Other common temporal operators can be defined as syntactic abbreviations in the usual way, like for instance $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \varphi_2)$, $F_I\varphi := \top U_I \varphi$, or $G_I\varphi := \neg F_I \neg\varphi$. Given a $t \in \mathbb{R}_{\geq 0}$, a shifted interval I is defined as $t + I = \{t + t' \mid t' \in I\}$.

Definition 1 (Qualitative semantics) Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The *qualitative semantics* of φ is inductively defined as follows:

$$\begin{aligned}
w, t &\models \sigma \text{ iff } \sigma(w(t)) \text{ is true} \\
w, t &\models \neg\varphi \text{ iff } w, t \not\models \varphi \\
w, t &\models \varphi_1 \wedge \varphi_2 \text{ iff } w, t \models \varphi_1 \text{ and } w, t \models \varphi_2 \\
w, t &\models \varphi_1 U_I \varphi_2 \text{ iff } \exists t' \in t + I \text{ s.t. } w, t' \models \varphi_2 \text{ and } \forall t'' \in [t, t'], w, t'' \models \varphi_1
\end{aligned} \tag{4}$$

A trace w satisfies a formula φ if and only if $w, 0 \models \varphi$, in short $w \models \varphi$. For given signal w, time instant $t \in \mathbb{R}_{\geq 0}$, and STL formula φ, the *satisfaction signal* $\mathcal{X}(w, t, \varphi)$ is $\top$ if $w, t \models \varphi$, $\bot$ otherwise.

Given a CPSML model $M = (S, U, sim)$, $M \models \varphi$ if every simulation trace of M satisfies φ.

Definition 2 (Quantitative semantics) Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The *quantitative semantics* of φ is defined as follows:

$$\begin{aligned} \rho(p(x_1, \ldots, x_n) \lhd 0, w, t) &= p(w(t)) \text{ with } \lhd \in \{<, \leq\} \\ \rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\ \rho(\varphi_1 \wedge \varphi_2, w, t) &= \min(\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)) \\ \rho(\varphi_1 U_I \varphi_2, w, t) &= \sup_{t' \in t+I} \min(\rho(\varphi_2, w, t'), \inf_{t''[t,t']} \rho(\varphi_1, w, t'')) \end{aligned} \tag{5}$$

The *robustness* of a formula φ with respect to a trace w is the signal $\rho(\varphi, w, \cdot)$.

Given a CPSML model $M = (S, U, sim)$, and a temporal logic formula φ, the *validity domain* of φ for model M is the subset of U for which traces of M satisfies φ. We denote the validity domain by U_φ; the remaining set of inputs $U \setminus U_\varphi$ is denoted by $U_{\neg\varphi}$. Simulation-based verification tools (such as [8]) can approximately compute validity domains via sampling-based methods.

3 Compositional Falsification Framework

In this section, we formalize the falsification problem for STL specifications against CPSML models, define our compositional falsification framework, and show its functionality on the AEBS system of Example 1.

Definition 3 (Falsification of CPSML) Given a model $M = (S, U, sim)$ and an STL specification φ, find an initial state $\mathbf{x}(t_0) \in S$ and a sequence of input values $\mathbf{u} = \mathbf{u}(t_0), \ldots, \mathbf{u}(t_n) \in U$ such that the trace of states $w = \mathbf{x}(t_0), \ldots, \mathbf{x}(t_n)$ generated by the simulation of M from $\mathbf{x}(t_0) \in S$ under $\mathbf{u}$ does not satisfy φ, i.e., $w \not\models \varphi$. We refer to such $(\mathbf{x}(t_0), \mathbf{u})$ as *counterexamples* for φ. The problem of finding a counterexample is often called *falsification problem.*

We now present the compositional framework for the falsification of STL formulas against CPSML models. Intuitively, the proposed method decomposes a given model into two abstractions: a version of the CPSML model under the assumption of perfectly correct ML modules and its actual ML components. The two abstractions are separately analyzed, the first by a temporal logic falsifier that builds the validity domain with respect to the given specification, the second by an ML analyzer that identifies sets of feature vectors that are misclassified by the ML components. Finally, the results of the two analysis are composed and projected back to a targeted input subspace of the original CPSML model where counterexamples can be found by invoking a temporal logic falsifier. Let us formalize this procedure.

Let $M = (S, U, sim)$ be a CPSML model and φ be an STL specification. Consider creating an "optimistic" abstraction M^+ of M: in other words, M^+ is a version of M with perfect ML components, that is, every feature vector of the ML feature space is correctly classified. Let us denote by ml the isolated ML components of the model M.

Under the assumption of correct ML components, the lower-dimensional input space of M^+ can be analyzed by constructing the validity domain of φ,

that is the partition of the input space into the sets U_φ and $U_{\neg\varphi}$ that do and do not satisfy φ, respectively. Note that considering the original model M, a possible misclassification of the ML components ml might affect the elements of U_φ and $U_{\neg\varphi}$. In particular, we are interested in the elements of U_φ that, due to misclassifications of ml, do not satisfy φ anymore. This corresponds to analyze the behavior of the ML components ml on the input set U_φ. We refer to this step as the ML analysis, that can be seen as the procedure of finding a subset $U^{ml} \subseteq U_\varphi$ of input values that are misclassified by the ML components ml. It is important to note that the input space of the CPS model M^+ and the feature spaces of the ML modules ml are different, thus the ML analyzer must adapt and relate the two different spaces. This important step will be clarified in Section 4.

Finally, the intersection $U_\varphi \cap U^{ml}$ of the subsets identified by the decomposed analysis of the CPS model and its ML components targets a small set of input values that are misclassified by the ML modules and are likely to falsify φ. Thus, counterexamples in $U_\varphi \cap U^{ml} \subseteq U$ can be determined by invoking a temporal logic falsifier on φ against M.

As explained below, we can pair the "optimistic" abstraction explained above with a "pessimistic" abstraction as well, so as to obtain a further restriction of the input space.

Algorithm 1 CPSML falsification scheme (one iteration between CPS Analyzer and ML Analyzer)

1: **function** CompFalsfy(M, φ) ▷ M CPSML, φ STL specification
2: $[M^+, ml] \leftarrow$Decompose(M) ▷ M^+ – perfect ML, ml – ML component
3: $[U^+_\varphi, U^+_{\neg\varphi}] \leftarrow$ValidityDomain($M^+, U, \varphi$) ▷ Validity domain of φ w.r.t. M^+
4: $[M^-, ml] \leftarrow$Decompose(M) ▷ M^- – wrong ML, ml – ML component
5: $[U^-_\varphi, U^-_{\neg\varphi}] \leftarrow$ValidityDomain($M^-, U, \varphi$) ▷ Validity domain of φ w.r.t. M^-
6: $U_{rou} \leftarrow U^+_\varphi \setminus U^-_\varphi$ ▷ Compute ROU
7: $U^{ml} \leftarrow$ MLAnalysis(ml, U_{rou}) ▷ Find misclassified feature vectors in ROU
8: $U^{ml}_{\neg\varphi} \leftarrow$Falsify($M, U_\varphi \cap U^{ml}, \varphi$) ▷ Falsify on targeted input
9: **return** $U_{\neg\varphi} \cup U^{ml}_{\neg\varphi}$
10: **end function**

The compositional falsification procedure is formalized in Algorithm 1. CompFalsfy receives as input a CPSML model M and an STL specification φ, and returns a set of falsifying counterexamples. At first, the algorithm decomposes M into M^+ and ml, where M^+ is an abstract version of M with ML components ml that return perfect answers (classifications) (Line 2). Then, the validity domain of φ with respect to the abstraction M^+ is computed by ValidityDomain (Line 3). Next, the algorithm computes from M, M^-, and ml, where M^- is an abstract version of M with ML components ml that always return wrong answers (misclassifications) (Line 4). Note that this step can be combined with Line 2, but we leave it separate for clarity in the abstract algorithm specification. Then, the validity domain of φ with respect to the abstraction M^- is computed by ValidityDomain (Line 5). The region of

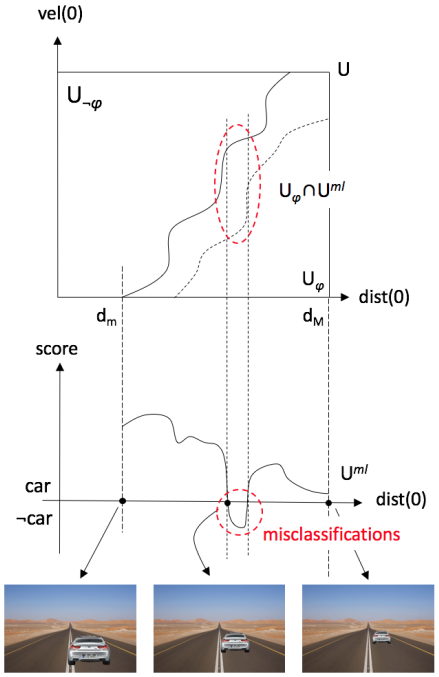


Fig. 2: Compositional falsification scheme on AEBS model. The "score" indicates the confidence level with which the classifier determines whether the image contains a car or not.

uncertainty (ROU), where misclassifications of the ML components can lead to violations of φ, is then computed as U_{rou} (Line 6). From this, the subset of inputs that are misclassified by ml is identified by MLAnalysis (Line 7). Finally, the targeted input set $U_\varphi \cap U^{ml}$, consisting in the intersection of the sets identified by the decomposed analysis, is searched by a temporal logic falsifier on the original model M (Line 8) and the set of inputs that falsify the temporal logic formula are returned.

Note that the above approach can be implemented even without computing M^- (Lines 4-6), in which case the entire validity domain of φ is considered as the ROU. For simplicity, we will take this truncated approach in the example described below. In Section 5, we will describe results on the AEBS case study with the full approach.

Example 2 Let us consider the model described in Example 1 and let us assume that the input space U of the model M consists of the initial velocity of the subject vehicle $vel(0)$, the initial distance between the vehicle and the proceeding obstacle $dist(0)$, and the set of pictures that can be captured by the camera. Let $\varphi := G_{[0,T]}(dist(t) \geq \tau)$ be a specification that requires the vehicle to be always farther than τ from the preceding obstacle. Instead of

analyzing the whole input space U (including a vast number of pictures), we can adopt our compositional framework to target a specific subset of U. Let M^+ be the AEBS model with a perfectly working image classifier and ml be the actual classifier. We begin by computing the validity subsets U_φ and $U_{\neg\varphi}$ of φ against M^+, considering only $vel(0)$ and $dist(0)$ and assuming exact distance measurements during the simulation. Next, we analyze only the image classifier ml on pictures of obstacles whose distances fall in U_φ, say in $[d_m, d_M]$ (see Figure 2). Our ML analyzer generates only pictures of obstacles whose distances are in $[d_m, d_M]$, finds possible sets of images that are misclassified, and returns the corresponding distances that, when projected back to U, yield the subset $U_\varphi \cap U^{ml}$. Finally, a temporal logic falsifier can be invoked over $U_\varphi \cap U^{ml}$ and a set of counterexamples is returned.

Algorithm 1 and the above example show how our compositional approach relies on three key steps: (i) computing the validity domain for an STL formula for a given simulation model; (ii) falsifying an STL formula on a simulation model, and (iii) a ML analyzer that computes a sub-space of its input feature space that lead to misclassifications. The first two steps have been well-studied in the literature on simulation-based verification of CPS, and implemented in tools such as Breach [8]. We discuss our approach to Step (iii) in the next section — our ML analyzer that identifies misclassifications of the ML component relevant to the overall CPSML input space.

4 Machine Learning Analyzer

A central idea in our approach to analyzing CPSML models is to use *abstractions* of the ML components. For instance, in the preceding section, we used the notions of *perfect* ML classifiers and *always-wrong* classifiers in computing the region of uncertainty (ROU). In this section, we extend this abstraction-based approach to the ML classifier and its input (feature) space.

One motivation for our approach comes from the application domain of autonomous driving where machine learning is used for object detection and perception. Instead of exploring the high-dimensional input space for the ML classifier involving all combinations of pixels, we instead perform the key simplification of *exploring realistic and meaningful modifications* to a given image dataset that corresponds to the ROU. Autonomous driving groups spend copius amounts of time collecting images and video to train their learning-based perception systems with. We focus on analyzing the space of images that is "close" to this data set but with semantically significant modifications that can identify problematic cases for the overall system.

The space of modifications to input feature vectors (say, images) induces an abstract space over the concrete feature (image) space. Let us denote the abstract input domain by A. Given a classifier $f : X \to Y$, our ML analyzer computes a simpler function $\tilde{f} : A \to Y$ that approximates f on the abstract domain A. The abstract domain of the function $\tilde{f}$ is analyzed and clusters of misclassifying abstract elements are identified. The concretizations of such

elements are subsets of features that are misclassified by the original classifier f. We describe further details of this approach in the remainder of this section.

4.1 Feature Space Abstraction

Let $\tilde{X} \subseteq X$ be a subset of the feature space of $f : X \to Y$. Let $\leq$ be a total order on a set A called the abstract set. An abstraction function is an injective function $\alpha : \tilde{X} \to A$ that maps every feature vector $\mathbf{x} \in \tilde{X}$ to an abstract element $\alpha(\mathbf{x}) \in A$. Conversely, the concretization function $\gamma : A \to \tilde{X}$ maps every abstraction $\mathbf{a} \in A$ to a feature $\gamma(\mathbf{a}) \in \tilde{X}$.

The abstraction and concretization functions play a fundamental role in our falsification framework. First, they allow us to map the input space of the CPS model to the feature space of its classifiers. Second, the abstract space can be used to analyze the classifiers on a compact domain as opposite to intractable feature spaces. These concepts are clarified in the following example, where a feature space of pictures is abstracted into a three-dimensional unit hyper-box.

Example 3 Let X be the set of RGB pictures of size 1000×600, i.e., $X = \{0, \ldots, 255\}^{1000 \times 600 \times 3}$. Suppose we are interested in analyzing a ML image classifier in the context of our AEBS system. In this case, we are interested in images of road scenarios rather than on arbitrary images in X. Further, assume that we start with a reference data set of images of a car on a two-lane highway with a desert road background, as shown in Figure 3. Suppose that we are interested only in the constrained feature space $\tilde{X} \subseteq X$ comprising this desert road scenario with a single car on the highway and three dimensions along which the scene can be varied: (i) the x-dimension (lateral) position of the car; (ii) the z-dimension (distance from the sensor) position of the car, and (iii) the brightness of the image. The x and z positions of the car and the brightness level of the picture can be seen as the dimensions of an abstract set A. In this setting, we can define the abstraction and concretization functions α and γ that relate the abstract set $A = [0, 1]^3$ and $\tilde{X}$. For instance, the picture $\gamma(0, 0, 0)$ sees the car on the left, close to the observer, and low brightness; the picture $\gamma(1, 0, 0)$ places the car shifted to the right; on the other extreme, $\gamma(1, 1, 1)$ has the car on the right, far away from the observer, and with a high brightness level. Figure 3 depicts some car pictures of $\tilde{S}$ disposed accordingly to their position in the abstract domain A (the surrounding box).

4.2 Approximation of Learning Components

We now describe how the feature space abstraction can be used to construct an approximation that helps the identification of misclassified feature vectors.

Given a classifier $f : X \to Y$ and a constrained feature space $\tilde{X} \subseteq X$, we want to determine an approximated classifier $\tilde{f} : A \to Y$, such that $err_{\tilde{f}}(T) \leq \epsilon$, for some $0 \leq \epsilon \leq 1$ and test set $T = \{(\mathbf{a}^{(1)}, y^{(1)}), \ldots, (\mathbf{a}^{(l)}, y^{(l)})\}$, with $y^{(i)} = f(\gamma(\mathbf{a}^{(i)}))$, for $i = 1, \ldots, l$.

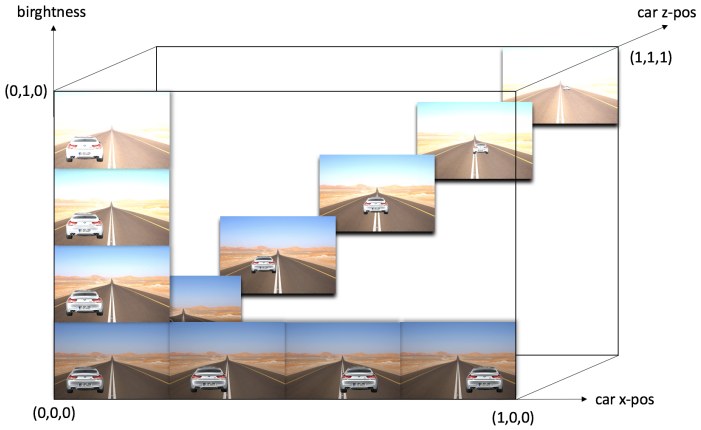


Fig. 3: Feature Space Abstraction. The cube represents the abstract space A with the three dimensions corresponding to three different image modifications. The displayed road images correspond to concretized elements of the concrete feature space $\tilde{X}$.

Intuitively, the proposed approximation scheme samples elements from the abstract set, computes the labels of the concretized elements using the analyzed learning algorithm, and finally, interpolates the abstract elements and the corresponding labels in order to obtain an approximation function. The obtained approximation can be used to reason on the considered feature space and identify clusters of potentially misclassified feature vectors.

Algorithm 2 Approximation construction of classifier $f : X \to Y$

1: **function** APPROXIMATION(A, γ, ϵ) ▷ A abstract set ($\gamma : A \to \tilde{X}$), $0 \leq \epsilon \leq 1$
2: $T_I \leftarrow \emptyset$
3: **repeat**
4: $T_I \leftarrow T_I \cup$ SAMPLE(A, f)
5: $\tilde{f} \leftarrow$ INTERPOLATE(T_I)
6: $T_E \leftarrow$ SAMPLE(A, f)
7: **until** $err_{\tilde{f}}(T_E) \leq \epsilon$
8: **return** $\tilde{f}$
9: **end function**

The APPROXIMATION algorithm (Algorithm 2) formalizes the proposed approximation construction technique. It receives in input an abstract domain A for the concretization function $\gamma : A \to \tilde{X}$, with $\tilde{X} \subseteq X$, the error threshold $0 \leq \epsilon \leq 1$, and returns a function $\tilde{f} : A \to Y$ that approximates f on the constrained feature space $\tilde{X}$. The algorithm consists in a loop that iteratively improves the approximation $\tilde{f}$. At every iteration, the algorithm populates the interpolation test set T_I by sampling abstract features from A and computing

the concretized labels accordingly to f (Line 4), i.e., SAMPLE(A, f)= $\{(\mathbf{a}, y) \mid \mathbf{a} \in \tilde{A}, y = f(\gamma(\mathbf{a}))\}$, where $\tilde{A} \subseteq A$ is a finite subset of samples determined with some sampling method. Next, the algorithm interpolates the points of T_I (Line 5). The result is a function $\tilde{f} : A \to Y$ that simplifies the original classifier f on the concretized constrained feature space $\tilde{X}$. The approximation is evaluated on the test set T_E. Note that at each iteration, T_E changes while T_I incrementally grows. The algorithm iterates until the error rate $err_{\tilde{f}}(T_E)$ is smaller than the desired threshold ϵ (Line 7).

The technique with which the samples in T_E and T_I are selected strongly influences the accuracy of the approximation. In order to have a good coverage of the abstract set A, we propose the usage of low-discrepancy sampling methods that, differently from uniform random sampling, cover sets quickly and evenly. In this work, we use the Halton and lattice sequences, two common and easy-to-implement sampling methods, which we explain next.

4.3 Sampling Methods

Discrepancy is a notion from equidistribution theory [41,32] that finds application in quasi-Monte Carlo techniques for error estimation and approximating the mean, standard deviation, integral, global maxima and minima of complicated functions, such as, e.g., our classification functions.

Definition 4 (Discrepancy [27]) Let $X = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ be a finite set of points in n-dimensional unit space, i.e., $X \subset [0,1]^n$. The *discrepancy* of X is given by:

$$D(X) = \sup_{B \in J} \mid \frac{\#(X, B)}{m} - vol(B) \mid \tag{6}$$

where $\#(X, B) = |\{\mathbf{x} \in X \mid \mathbf{x} \in B\}|$, i.e., the number of points in X that fall in B, $vol(B)$ is the n-dimensional volume of B, and J is the set of boxes of the form $\{\mathbf{x} \in \mathbb{R}^n | a_i \leq \mathbf{x}_i \leq b_i\}$, where $i = 1, \ldots, n$ and $0 \leq a_i < b_i < 1$.

Definition 5 (Low-discrepancy sequence [27]) A *low-discrepancy sequence*, also called *quasi-random sequence*, is a sequence with the property that for all $m \in \mathbb{N}$, its subsequence $X = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ has low discrepancy.

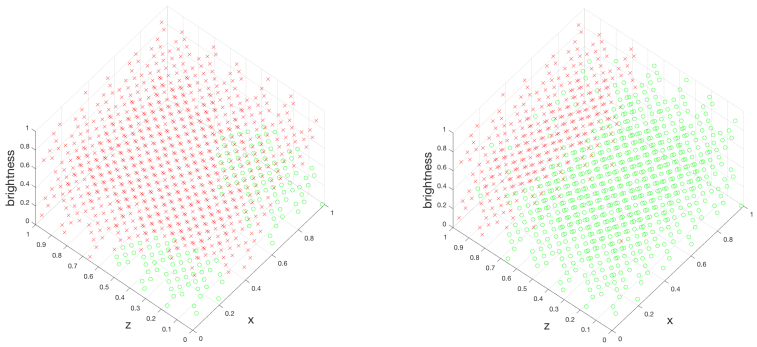
Low-discrepancy sequences fill spaces more uniformly than uncorrelated random points. This property makes low-discrepancy sequences suitable for problems where grids are involved, but it is unknown in advance how fine the grid must be to attain precise results. A low-discrepancy sequence can be stopped at any point where convergence is observed, whereas the usual uniform random sampling technique requires a large number of computations between stopping points [38]. Low-discrepancy sampling methods have improved computational techniques in many areas, including robotics [6], image processing [14], computer graphics [34], numerical integration [35], and optimization [30].

We now introduce two low-discrepancy sequences that will be used in this work. For more sequences and details see, e.g., [29].

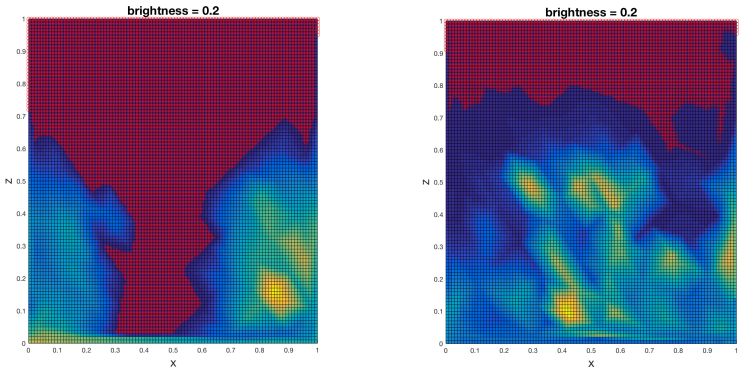
1. *Halton sequence* [27]. Based on the choice of an arbitrary prime number p, the i-th sample is obtained by representing i in base p, reversing its digits, and moving the decimal point by one position. The resulting number is the i-th sample in base p. For the multi-dimensional case, it is sufficient to choose a different prime number for each dimension. In practice, this procedure corresponds to choosing a prime base p, dividing the $[0, 1]$ interval in p segments, then p^2 segments, and so on.
2. *Lattice sequence* [24]. A lattice can be seen as the generalization of a multi-dimensional grid with possibly nonorthogonal axes. Let $\alpha_1, \ldots, \alpha_{n-1} \in \mathbb{R}_{>0}$ be irrational numbers and $m \in \mathbb{N}$. The i-th sample of a lattice sequence is $(i/m, \{i\alpha_1\}, \ldots, \{i\alpha_{n-1}\})$, where the curly braces $\{\cdot\}$ denote the fractional part of the real value (modulo-one arithmetic).

Example 4 We now analyze two Convolutional Neural Networks (CNNs): the Caffe [18] version of AlexNet [21] and the Inception-v3 model of Tensorflow [23], both trained on the ImageNet database [1]. We sample 1000 points from the abstract domain defined in Example 3 using the lattice sampling techniques. These points encode the x and z displacements of a car in a picture and its brightness level (see Figure 3). Figure 4 (a) depicts the sampled points with their concretized labels. The green circles indicate correct classifications, i.e., the classifier identified a car, the red circles denote misclassifications, i.e., no car detected. The linear interpolation of the obtained points leads to an approximation function. The error rates $err_{\tilde{f}}(T_E)$ of the obtained approximations (i.e., the discrepancies between the predictions of the original image classifiers and their approximations) computed on 300 randomly picked test cases are 0.0867 and 0.1733 for AlexNet and Inception-v3, respectively. Figure 4 (b) shows the projections of the approximation functions for the brightness value 0.2. The more red a region, the larger the sets of pictures for which the neural networks do not detect a car. For illustrative purposes, we superimpose the projections of Figure 4 (b) over the background used for the picture generation. These illustrations show the regions of the concrete feature vectors in which a vehicle is misclassified.

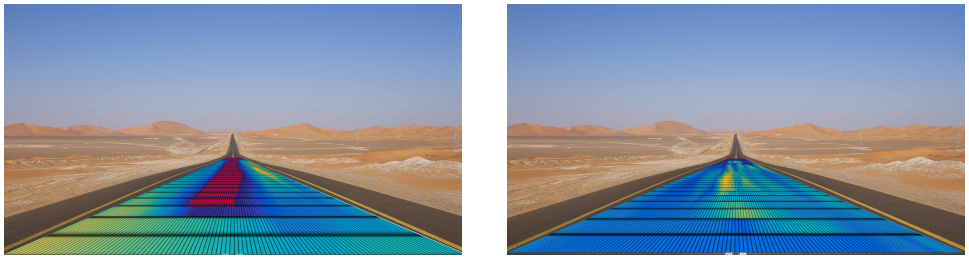
The analysis of Example 4 on AlexNet and Inception-v3 provides useful insights. First, we observe that Inception-v3 outperforms AlexNet on the considered road pictures since it correctly classifies more pictures than AlexNet. Second, we notice that AlexNet tends to correctly classify pictures in which the x abstract component is either close to 0 or 1, i.e., pictures in which the car is not in the middle of the street, but on one of the two lanes. This suggests that the model might not have been trained enough with pictures of cars in the center of the road. Third, using the lattice method on Inception-v3, we were able to identify a corner case misclassification in a cluster of correct predictions (note the isolated red cross with coordinates $(0.1933, 0.0244, 0.4589)$). All this information provides insights on the classifiers that can be useful in the hunt for counterexamples.



(a) Sampling.



(b) Interpolation projection.



(c) Feature space analysis.

Fig. 4: ML analysis of AlexNet network developed with Caffe (top) and Inception-v3 network developed with Tensorflow (bottom) on a road scenario.

5 Experimental Results

In this section we present two case studies, both involving an Automatic Emergency Braking System (AEBS), but differing in the details of the underlying simulator and controller. The first is a Simulink-based AEBS, the second is a Unity-Udacity simulator-based AEBS.

The falsification framework for the first case study has been implemented in a Matlab toolbox.[1] The framework for the second case study has been written in Python and C#.[2] Our tools deal with models of CPSML and STL specifications. They mainly consist of a temporal logic falsifier and an ML analyzer that interact to falsify the given STL specification against the decomposed models. As an STL falsifier, we chose the existing tool Breach [8], while the ML analyzer has been implemented from scratch. The ML analyzer implementation has two components: the feature space abstractor and the ML approximation algorithm (see Section 4). The feature space abstractor implements a scene generator that concretizes the abstracted feature vectors. The algorithm that computes an approximation of the analyzed ML component gives the user the option of selecting the sampling method and interpolation technique, as well as setting the desired error rate. Our tools are interfaced with the deep learning frameworks Caffe [18] and Tensorflow [23]. We ran our tests on a desktop computer Dell XPS 8900, Intel (R) Core(TM) i7-6700 CPU 3.40GHz, DIMM RAM 16 GB 2132 MHz, GPUs NVIDIA GeForce GTX Titan X and Titan Xp, with Ubuntu 14.04.5 LTS and Matlab R2016b.

5.1 Case Study 1: Simulink-based AEBS

Our first case study is a closed-loop Simulink model of a semi-autonomous vehicle with an Advanced Emergency Braking System (AEBS) [37] connected to a deep neural network-based image classifier. The model mainly consists of a four-speed automatic transmission controller linked to an AEBS that automatically prevents collisions with preceding obstacles and alleviate the harshness of a crash when a collision is likely to happen (see Figure 5). The AEBS determines a braking mode depending on the speed of the vehicle v_s, the possible presence of a preceding obstacle, its velocity v_p, and the longitudinal distance *dist* between the two. The distance *dist* is provided by radars having 30m of range. For obstacles farther than 30m, the camera, connected to an image classifier, alerts the AEBS that, in the case of detected obstacle, goes into warning mode.

Depending on $v_s, v_p, dist$, and the presence of obstacles detected by the image classifier, the AEBS computes the time to collision and longitudinal safety indices, whose values determine a controlled transition between safe, warning, braking, and collision mitigation modes. In safe mode, the car does

[1] `https://github.com/dreossi/analyzeNN`

[2] `https://bitbucket.org/sseshia/uufalsifier`

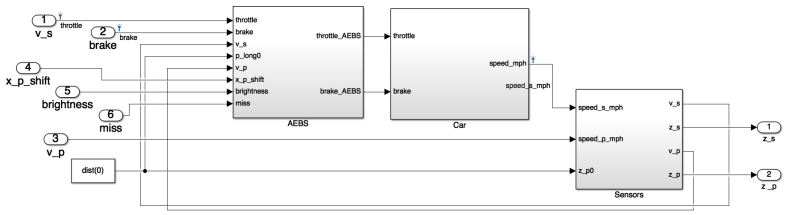


Fig. 5: Simulink model of a semi-autonomous vehicle with AEBS.

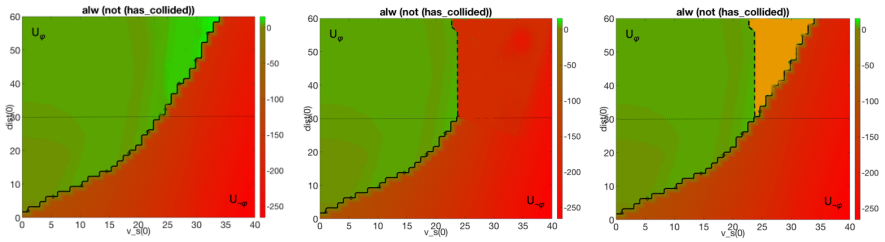


Fig. 6: Validity domain for $G(\neg(dist(t)) \leq 0)$ and AEBS model with different abstractions of ML component. The initial velocity and distance are on the x and y axes respectively. The dotted (horizontal) line is the image classifier activation threshold. Green indicates combinations of initial velocity and distance for which the property is satisfied and red indicates combinations for which the property is falsified. Our ML analyzer performs both optimistic (left) and pessimistic (middle) abstractions of the neural network classifier. On the right-most image, the yellow region denotes the region of uncertainty (ROU).

not need to brake. In warning mode, the driver should brake to avoid a collision. If this does not happen, the system goes into braking mode, where the automatic brake slows down the vehicle. Finally, in collision mitigation mode, the system, determining that a crash is unavoidable, triggers a full braking action aimed to minimize the damage.

To establish the correctness of the system and in particular of its AEBS controller, we formalize the STL specification $G(\neg(dist(t)) \leq 0)$, that requires $dist(t)$ to always be positive, i.e., no collision happens. The input space is $v_s(0) \in [0, 40]$ (mph), $dist(0) \in [0, 60]$ (m), and the set of all RGB pictures of size 1000×600. The preceding vehicle is not moving, i.e., $v_p(t) = 0$ (mph).

At first, we compute the validity domain of φ assuming that the radars are able to provide exact measurements for any distance $dist(t)$ and the image classifier correctly detects the presence of a preceding vehicle. The computed validity domain is depicted in Figure 6 (left-most image): green for U_{φ} and red for $U_{\neg\varphi}$. Next, we try to identify candidate counterexamples that belong to the satisfactory set (i.e., the inputs that satisfy the specification) but might be influenced by a misclassification of the image classifier. Since the AEBS

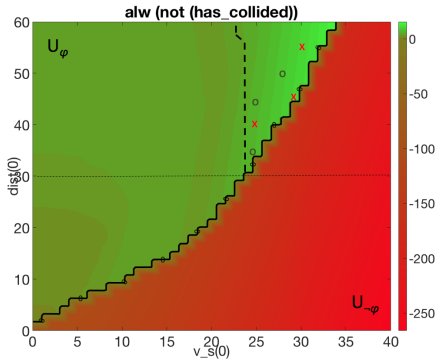


Fig. 7: Analysis of Region of Uncertainty (ROU) for AEBS and property $G(\neg(dist(t)) \leq 0)$. Red crosses in the ROU denote misclassifications generated by the ML analyzer that leads to a system-level counterexample. A circle denotes a "benign" misclassification.

relies on the classifier only for distances larger than 30m, we can focus on the subset of the input space with $dist(0) \geq 30$. Specifically, we identify potential counterexamples by analyzing a pessimistic version of the model where the ML component always misclassifies the input pictures (see Figure 6, middle image). From these results, we can compute the region of uncertainty, shown in Figure 6 on the right. We can then focus our attention on the ROU, as shown in Fig. 7. In particular, we can identify candidate counterexamples, such as, for instance, $(25, 40)$ (i.e., $v_s(0) = 25$ and $dist(0) = 40$).

Next, let us consider the AlexNet image classifier and the ML analyzer presented in Section 4 that generates pictures from the abstract feature space $A = [0, 1]^3$, where the dimensions of A determine the x and z displacements of a car and the brightness of a generated picture, respectively. The goal now is to determine an abstract feature $\mathbf{a}_c \in A$ related to the candidate counterexample $(25, 40)$, that generates a picture that is misclassified by the ML component and might lead to a violation of the specification φ. The $dist(0)$ component of $\mathbf{u}_c = (25, 40)$ determines a precise z displacement $\mathbf{a}_2 = 0.2$ in the abstract picture. Now, we need to determine the values of the abstract x displacement and brightness. Looking at the interpolation projection of Figure 4 (b), we notice that the approximation function misclassifies pictures with abstract component $\mathbf{a}_1 \in [0.4, 0.5]$ and $\mathbf{a}_3 = 0.2$. Thus, it is reasonable to try to falsify the original model on the input element $v_s(0) = 25, dist(0) = 40$, and concretized picture $\gamma(0.5, 0.2, 0.2)$. For this targeted input, the temporal logic falsifier computed a robustness value for φ of -24.60, meaning that a falsifying counterexample has been found. Other counterexamples found with the same technique are, e.g., $(27, 45)$ or $(31, 56)$ that, associated with the correspondent concretized pictures with $\mathbf{a}_1 = 0.5$ and $\mathbf{a}_3 = 0.2$, lead to the robustness values -23.86 and -24.38, respectively (see Figure 7, red crosses). Conversely, we also disproved some candidate counterexamples, such as $(28, 50)$, $(24, 35)$, or

$(25, 45)$, whose robustness values are $9.93, 7.40$, and 7.67 (see Figure 7, green circles).

For experimental purposes, we try to falsify a counterexample in which we change the x position of the abstract feature so that the approximation function correctly classifies the picture. For instance, by altering the counterexample $(27, 45)$ with $\gamma(0.5, 0.225, 0.2)$ to $(27, 45)$ with $\gamma(1.0, 0.225, 0.2)$, we obtain a robusteness value of 9.09, that means that the AEBS is able to avoid the car for the same combination of velocity and distance of the counterexample, but different x position of the preceding vehicle. Another example, is the robustness value -24.38 of the falsifying input $(31, 56)$ with $\gamma(0.5, 0.28, 0.2)$, that altered to $\gamma(0.0, 0.28, 0.2)$, changes to 12.41.

Finally, we test Inception-v3 on the corner case misclassification identified in Section 4.2 (i.e., the picture $\gamma(0.1933, 0.0244, 0.4589)$). The distance $dist(0) = 4.88$ related to this abstract feature is below the activation threshold of the image classifier. Thus, the falsification points are exactly the same as those of the computed validity domain (i.e., $dist(0) = 4.88$ and $v_s(0) \in [4, 40]$). This study shows how a misclassification of the ML component might not affect the correctness of the CPSML model.

5.2 Case Study 2: Unity-Udacity Simulator-based AEBS

We now analyze an AEBS deployed within Udacity's self-driving car simulator.[3] The simulator, built with the Unity game engine[4], can be used to teach cars how to navigate roads using deep learning. We modified the simulator in order to focus exclusively on the braking system. In our settings, the car steers by following some predefined waypoints, while acceleration and braking are controlled by an AEBS connected to a CNN. An onboard camera sends images to the CNN whose task is to detect cows on the road. Whenever an obstacle is detected, the AEBS triggers a brake that slows the vehicle down and prevents the collision against the obstacle.

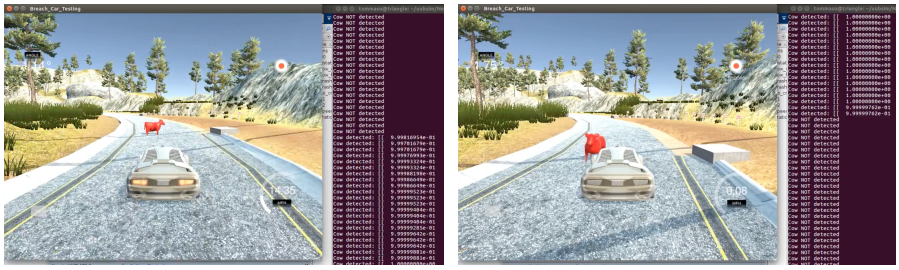
We implemented a CNN that classifies the pictures captured by the onboard camera in two categories "cow" and "not cow". The CNN has been implemented and trained using Tensorflow. We connected the CNN to the Unity C# class that controls the car. The communication between the neural network and the braking controller happens via Socket.IO protocol.[5] A screenshot of the car braking in presence of a cow is shown in Figure 8a. A video of the AEBS in action can be seen at `https://www.youtube.com/watch?v=Sa4oLGcHAhY`.

The CNN architecture is depicted in Figure 9. The network consists of eight layers: the first six are alternations of convolutions and max-pools with ReLU activations, the last two are a fully connected layer and a softmax that outputs the network prediction. The dimensions and hyperparameters of our

[3] Udacity's Self-Driving Car Simulator: `https://github.com/udacity/self-driving-car-sim`

[4] Unity: `https://unity3d.com/`

[5] Socket.IO protocol: `https://github.com/socketio`



(a) Correct detection and braking. (b) Misclassification and collision.

Fig. 8: Unity-Udacity simulator AEBS. The onboard camera sends images to the CNN. When a cow is detected a braking action is triggered until the car comes to a complete stop. Full videos available at `https://www.youtube.com/watch?v=Sa4oLGcHAhY` and `https://www.youtube.com/watch?v=MaRoU5OgimE`.

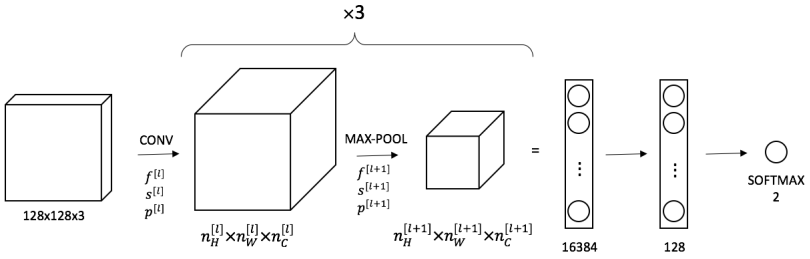


Fig. 9: CNN architecture.

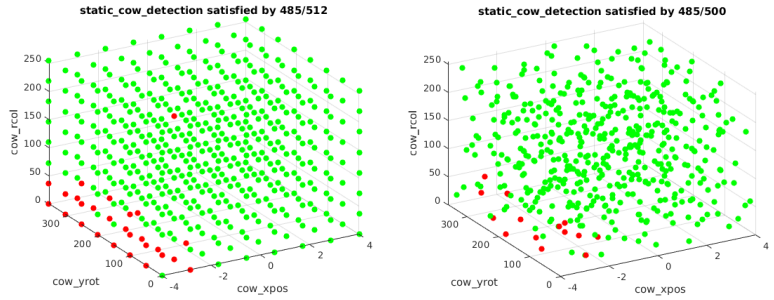
	0	1	2	3	4	5	6	7	8
$n_H^{[l]} \times n_W^{[l]}$	128×128	128×128	64×64	64×64	32×32	32×32	16×16	128×1	2×1
$n_C^{[l]}$	3	32	32	32	32	64	64	1	1
$f^{[l]}$	-	3	2	3	2	3	2	-	-
$p^{[l]}$	-	1	0	1	0	1	0	-	-
$s^{[l]}$	-	1	2	1	2	1	2	-	-

Table 1: CNN dimensions and hyperparameters.

neural network are shown in Table 1, where l is a layer, $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$ is the dimension of the volume computed by the layer l, $f^{[l]}$ is the filter size, $p^{[l]}$ is the padding, and $s^{[l]}$ is the stride.

Our dataset, composed by 1k road images, was split into 80% train data and 20% validation. We trained our model using cross-entropy cost function and Adam algorithm optimizer with learning rate 10^{-4}. Our model reached 0.95 accuracy on the validation set.

In our experimental evaluation, we are interested in finding a case where our AEBS fails, i.e., the car collides against a cow. This requirement can be



(a) Grid-based sampling. (b) Halton sequence sampling.

Fig. 10: CNN analysis.

formalized as the STL specification $G(\|\mathbf{x}_{car} - \mathbf{x}_{cow}\| > 0)$ that imposes the Euclidean distance of the car and cow positions ($\mathbf{x}_{car}$ and $\mathbf{x}_{cow}$, respectively) to be always positive.

We analyzed the CNN feature space by considering the abstract space $A = [0, 1]^3$, where the dimensions of A determine the displacement of the cow of $\pm$4m along the x-axis, its rotation along the y-axis, and the intensity of the red color channel. We sampled the elements from the abstract space using both Halton sequence and a grid-based approach. The obtained results are shown in Figure 10. In both figures, green points are those that lead to images that are correctly classified by the CNN; conversely, red points denote images that are misclassified by the CNN and can potentially lead to a system falsification. Note how we were able to identify a cluster of misclassifying images (lower-left corners of both Figures 10a and 10b) as well as an isolated corner case (upper-center, Figure 10a).

Finally, we ran some simulations with the misclassifying images identified by our analysis. Most of the them brought the car to collide against the cow. A screenshot of a collision is shown in Figure 8b. The full video is available at `https://www.youtube.com/watch?v=MaRoU5OgimE`.

6 Conclusion

We presented a compositional falsification framework for STL specifications against CPSML models based on a decomposition between the analysis of machine learning components and the system containined them. We introduced an ML analyzer able to abstract feature spaces, approximate ML classifiers, and provide sets of misclassified feature vectors that can be used to drive the falsification process. We implemented our framework and showed its effectiveness for an autonomous driving controller using perception based on deep neural networks.

This work lays the basis for future advancements. There are several directions for future work, both theoretical and applied. In the remainder of this section, we describe this landscape for future work. See [33] for a broader discussion of these points in the context of the goal of verified intelligent systems.
Improvements in the ML Analyzer: We intend to improve our ML Analyzer exploring the automatic generation of feature space abstractions from given training sets. One direction is to exploit the structure of ML components, e.g., the custom architectures that have been developed for deep neural networks in applications such as autonomous driving [17]. For instance, one could perform a sensitivity analysis that indicates along which axis in the abstract space we should move in order to change the output label or reduce the confidence of the classifier on its output. Another direction is to improve the sampling techniques that we have explored so far, ideally devising one that captures the probability of detecting a corner-case scenario leading to a property violation. Of particular interest are adaptive sampling methods involving further cooperation between the ML Analyzer and the CPS Analyzer. We are also interested in integrating other techniques for generating misclassifications of ML components (e.g., [26,16,7]) into our approach.
Impacting the ML component design: Our falsification approach produces input sequences that result in the violation of a desired property. While this is useful, it is arguably even more useful to obtain higher-level interpretable insight into where the training data falls short, what new scenarios must be added to the training set, and how the learning algorithms' parameters must be adjusted to improve accuracy. For example, one could use techniques for mining specifications or requirements (e.g., [19,40]) to aggregate interesting test images or video into a cluster that can be represented in a high-level fashion. One could also apply our ML Analyzer outside the falsification context, such as for controller synthesis.
Further Applications: Although our approach has shown initial promise for reasoning about autonomous driving systems, much more remains to be done to make this practical. Real sensor systems for autonomous driving involve multiple sensors (cameras, LIDAR, RADAR, etc.) whose raw outputs are often fused and combined with deep learning or other ML techniques to extract higher level information (such as the location and type of objects around the vehicle). This sensor space has very high dimensionality and high complexity, not to mention streams of sensor input (e.g., video), that one must be able to analyze efficiently. To handle industrial-scale production systems, our overall analysis must be scaled up substantially, potentially via use of cloud computing infrastructure. Finally, our compositional methodology could be extended to other, non-cyber-physical, systems that contain ML components.

References

1. Imagenet. `http://image-net.org/`.
2. Udacity self-driving car simulator built with unity. `https://github.com/udacity/self-driving-car-sim`.

3. Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 254–257, 2011.
4. A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.
5. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
6. M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1481–1487. IEEE, 2001.
7. N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57, 2017.
8. A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification, CAV*, pages 167–170, 2010.
9. T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods, NFM*, pages 127–142, 2015.
10. T. Dreossi, A. Donzé, and S. A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods Conference (NFM)*, May 2017.
11. T. Dreossi, S. Ghosh, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. Systematic testing of convolutional neural networks for autonomous driving. In *ICML Workshop on Reliable Machine Learning in the Wild (RMLW)*, 2017. Published on Arxiv: abs/1708.03309.
12. P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2E2: a verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 68–82. Springer, 2015.
13. A. Fawzi, O. Fawzi, and P. Frossard. Analysis of classifiers' robustness to adversarial perturbations. *arXiv preprint arXiv:1502.02590*, 2015.
14. B. Hannaford. Resolution-first scanning of multidimensional spaces. *CVGIP: Graphical Models and Image Processing*, 55(5):359–369, 1993.
15. G. Hinton et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
16. X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. *CoRR*, abs/1610.06940, 2016.
17. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
18. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia Conference, ACMMM*, pages 675–678, 2014.
19. X. Jin, A. Donzé, J. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 34(11):1704–1717, 2015.
20. G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *29th International Conference on Computer Aided Verification (CAV)*, pages 97–117, 2017.
21. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
22. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
23. Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
24. J. Matousek. *Geometric discrepancy: An illustrated guide*, volume 18. Springer Science & Business Media, 2009.

25. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.
26. S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.
27. W. J. Morokoff and R. E. Caflisch. Quasi-random sequences and their discrepancies. *SIAM Journal on Scientific Computing*, 15(6):1251–1279, 1994.
28. A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition, CVPR*, pages 427–436. IEEE, 2015.
29. H. Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of number theory*, 30(1):51–70, 1988.
30. H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.
31. K. Pei, Y. Cao, J. Yang, and S. Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, pages 1–18, 2017.
32. J. Rosenblatt and M. Wierdl. Pointwise ergodic theorems via harmonic analysis. In *Conference on Ergodic Theory*, number 205, pages 3–151, 1995.
33. S. A. Seshia, D. Sadigh, and S. S. Sastry. Towards verified artificial intelligence. *CoRR*, abs/1606.08514, 2016.
34. P. Shirley et al. Discrepancy as a quality measure for sample distributions. In *Proc. Eurographics*, volume 91, pages 183–194, 1991.
35. I. H. Sloan and S. Joe. *Lattice methods for multiple integration*. Oxford University Press, 1994.
36. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.
37. L. Taeyoung, Y. Kyongsu, K. Jangseop, and L. Jaewan. Development and evaluations of advanced emergency braking system algorithm for the commercial vehicle. In *Enhanced Safety of Vehicles Conference, ESV*, pages 11–0290, 2011.
38. Trandafir, Aurel and Weisstein, Eric W. Quasirandom sequence. From MathWorld–A Wolfram Web Resource.
39. V. Vapnik. Principles of risk minimization for learning theory. In *NIPS*, pages 831–838, 1991.
40. M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia. Logical clustering and learning for time-series data. In *Computer Aided Verification - 29th International Conference (CAV)*, pages 305–325, 2017.
41. H. Weyl. Über die gleichverteilung von zahlen mod. eins. *Mathematische Annalen*, 77(3):313–352, 1916.
42. T. Yamaguchi, T. Kaga, A. Donzé, and S. A. Seshia. Combining requirement mining, software model checking, and simulation-based verification for industrial automotive systems. In *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, October 2016.