Mining Specifications for Predictive Safety Monitoring

Mining Specifications for Predictive Safety Monitoring

Eleonora Nesterini
TU Wien and AIT Austrian Institute of Technology
Vienna, Austria
eleonora.nesterini@tuwien.ac.at

Ezio Bartocci
TU Wien
Vienna, Austria
ezio.bartocci@tuwien.ac.at

Alessio Gambi
AIT Austrian Institute of Technology
Vienna, Austria
alessio.gambi@ait.ac.at

Dejan Nickovic
AIT Austrian Institute of Technology
Vienna, Austria
dejan.nickovic@ait.ac.at

Sanjit A. Seshia
University of California Berkeley
Berkeley, California, United States
sseshia@eecs.berkeley.edu

Hazem Torfah
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
hazemto@chalmers.se

ABSTRACT

Safety-critical autonomous systems must reliably predict unsafe behavior to take timely corrective actions. Safety properties are often defined over variables that are not directly observable at runtime, making prediction and detection of violations hard. We present a new approach for learning interpretable monitors characterized by concise Signal Temporal Logic (STL) formulas that can predict safety property violations from the observable sensor data. We train these monitors from synthetic, possibly highly unbalanced data generated in a simulation environment. Our specification mining procedure combines a grammar-based method and two novel ensemble techniques. Our approach outperforms the existing solutions by enhancing accuracy and explainability, as demonstrated in two autonomous driving case studies.

KEYWORDS

Specification Mining, Signal Temporal Logic, Runtime Monitoring.

ACM Reference Format:
Eleonora Nesterini, Ezio Bartocci, Alessio Gambi, Dejan Nickovic, Sanjit A. Seshia, and Hazem Torfah. 2025. Mining Specifications for Predictive Safety Monitoring. In *ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025) (ICCPS '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3716550.3722021

1 INTRODUCTION

The deployment of autonomous Cyber-Physical Systems (CPS) has significantly increased in recent years in safety-critical domains. Applications such as autonomous driving (AD) leverage recent advancements in machine learning (ML) to enable driverless vehicle operation, offering many potential benefits, including reduced traffic accidents, enhanced comfort, and reduced traffic congestion. Given the complex environments in which these safety-critical systems operate and the brittleness of ML components, it is crucial to accurately detect potential safety risks and allow for timely countermeasures to prevent unsafe outcomes like collisions [23].

The detection of potential safety risks in CPS is best managed by runtime monitors that continuously evaluate system safety [3]. An accurate specification that captures the safety requirements is key to effective runtime monitoring. Formal specifications allow developers to precisely define these requirements and offer the crucial advantage of enabling automated monitor generation. However, a significant bottleneck in constructing and integrating such monitors arises because safety specifications are often defined over system-level variables that the autonomous system cannot directly observe during operation. The perfect knowledge of geographical boundaries, roads' type and physical attributes, or the ground-truth distance between the autonomous vehicle and the other traffic participants are features relevant for detecting safety violations in an autonomous driving system that, in practice, are not directly available to the autonomous vehicle.

In order to serve as a practical means to increase the safety of autonomous CPS, the monitors must rely only on variables observable from within the CPS. Variables like sensor data (e.g., images) and internal controller inputs and outputs are generally available; however, using them to design effective safety monitors is challenging because their connection to safety properties is not straightforward, i.e., there is no one-to-one relationship between them.

This paper addresses this challenge by introducing a novel framework for mining safety monitor specifications over observable data. Given a traditional system-level specification represented as a formula in Signal Temporal Logic (STL) [18], our framework learns another STL specification that approximates the validity of the original system-level specification solely using valuations of observable variables. Our framework implements a data-driven approach that collects data from system simulations, evaluates it using the predefined system-level specification, and derives new specifications based on observable variables using a grammar-based specification mining process. This novel specification mining process allows CPS developers to encode prior knowledge and tailor the target specifications to their applications. Recognizing that data-driven approaches may not lead to generating perfectly accurate monitors, our mining process solves an optimization problem to minimize false positive (unwarranted alarms) and false negative (missed violations) rates.

To learn monitors that detect impending safety property violations *before* they occur, thus enabling corrective actions to prevent unsafe scenarios, we extend observable-level validity labels with future system-level predictions following the work by Torfah et al. [24, 25]. However, to address the main shortcomings of previous works, namely the generated monitors' lack of interpretability and (over-)sensitivity to training data, we leverage STL formulas to learn concise specifications and use quantitative STL semantics (i.e., robustness) to combine them into a single monitor capable of dealing with variance in the training data.

We evaluate our approach in two representative case studies in the autonomous driving domain and show that our "robustness-based" ensemble of STL monitors outperforms existing approaches while maintaining a compact, interpretable structure. Although our evaluation focuses on autonomous driving systems, the proposed method is general and can be applied to other autonomous and safety-critical CPS.

To summarize, the main contributions of this paper are:

(1) A novel data-driven approach for mining accurate and explainable predictive STL safety monitors from the observable variables only.
(2) As a central component of the above approach, a grammar-based method for mining STL formulas that discriminate between positive and negative examples, even in the case of highly unbalanced datasets;
(3) Two new ensemble methods that leverage the STL quantitative semantics;
(4) Extensive experimental analysis that compares our approach to the state-of-the-art (in terms of monitors learning and ensemble techniques) and studies its sensitivity to key dataset characteristics (imbalance, size, and prediction horizon).

We release the implementation of our approach along with all the data, scripts, and instructions to reproduce the experiments and analyze the data in the publicly accessible repository *ObSTLearn*[1].

2 RELATED WORK

Recent work [25] introduced the problem of learning observable monitors to predict the safety of system's executions. However, the decision-tree approach employed by the authors produced trees with hundreds of nodes that are hard to interpret. Since explainability is essential in safety-critical domains, we look for monitors in the form of STL specifications, suitable to express temporal properties of CPS in an unambiguous yet human-understandable manner. *Specification mining* [4] is the research field that focuses on automatically inferring properties of a system from its executions. STL mining methods are generally categorized as either *template-based* [2] or *template-free* [6, 19], depending on whether only parameter values or both the formula structure and parameter values must be learned from data, respectively. Recent works [5, 11, 21] mitigated this distinction by enabling the definition of an STL (sub)grammar of admissible formulas and hence allowing for the embedding of a flexible amount of knowledge in the mining process. Existing grammar-based methods aim to learn formulas to (tightly) describe only a single set of positive examples [12]. Conversely, the predictive monitoring problem addressed in this paper requires discriminating between sets of positive and negative examples [6, 19]. In addition, in this context, negative executions result from system faults or safety issues and are much rarer than favorable executions, leading to significantly imbalanced datasets.

[1]https://github.com/eleonoranesterini/ObSTLearn.git

For these reasons, we devised and implemented an extension of a recently-proposed grammar-based approach [5] to learn from both positive and negative examples, even in highly imbalanced settings. Furthermore, we propose two ensemble methods that leverage STL quantitative semantics to improve the accuracy of multiple mined formulas inspired by promising insights in the literature. In particular, authors in [24] proved that classical ensemble methods, such as Majority Voting, reduce the misclassification rate exponentially with the number of monitors. Other works [1, 11] have exploited this ensemble method to enhance the classification performances of STL formulas. However, to the best of our knowledge, ensemble methods that leverage the intrinsic characteristics of STL (such as its quantitative semantics) have not been explored yet.

Other predictive monitoring techniques were proposed in [7, 17] where the authors used a Simulink model [7] or an ML model [17] of the CPS to predict what will happen in the future. However, in both cases, the monitoring properties are assumed to be given, while in our approach they are learned from systems executions.

3 STL BACKGROUND

We define a *trace* of $m \in \mathbb{N}$ real-valued variables and length $n \in \mathbb{N}$ as the finite sequence of the (time,value) pairs $x = (t_1, x_1), \ldots, (t_n, x_n)$, where $t_1 = 0$, $t_i < t_{i+1}$ $\forall i \in \{1, \ldots, n-1\}$, and $x_i \in \mathbb{R}^m$. For a trace x of length n, the *prediction horizon* $h \in \mathbb{N}$ with $h < n$ is the number of time steps ahead of the final step at which we aim to make our prediction. Hence, we will consider the trace x truncated by removing its last h time steps, which we denote by $x^{-h} := (t_1, x_1), (t_2, x_2), \ldots, (t_{n-h}, x_{n-h})$.

We adopt Signal Temporal Logic (STL) [18] as our specification language. Its syntax is defined by the following grammar:

$$\varphi := \mathbf{true} \mid f(x) > 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathbf{U}_I \, \varphi_2$$

where $f(x) > 0$ is an atomic proposition ($f : \mathbb{R}^n \to \mathbb{R}$), $\neg$ and $\wedge$ are, respectively, the Boolean negation and conjunction, and $\mathbf{U}_I$ is the *until* temporal operator defined over the dense interval of time I in $\mathbb{R}_{\geq 0}$ - the interval I is generally omitted when $I = [0, \infty)$. The *finally* $\mathbf{F}$ and *globally* $\mathbf{G}$ operators are derived as follows: $\mathbf{F}_I\varphi := \mathbf{true}\,\mathbf{U}_I\varphi$ and $\mathbf{G}_I\varphi := \neg\mathbf{F}_I\neg\varphi$.

STL quantitative semantics was introduced in [8] and is defined in terms of the robustness function ρ that maps an STL specification φ, a trace x, and time t, to the value $\rho(\varphi, x, t)$ representing how far is x from the satisfaction boundary of φ at time t. We write $\rho(\varphi; x)$ to indicate $\rho(\varphi, x, 0)$. Notably, the STL quantitative semantics is *sound*: when $\rho(\varphi; x) > 0$ the trace x satisfies φ (indicated by $x \models \varphi$), whereas when $\rho(\varphi; x) < 0$ the trace x violates φ ($x \not\models \varphi$).

A Parametric STL (PSTL) formula is an STL formula whose temporal and quantitative numerical values are replaced by parameter symbols [2]. As common in the literature, our learning process will be split into (i) exploring the space of PSTL formulas; and (ii) instantiation of the candidate PSTL formula into a concrete STL formula by choosing appropriate values for its parameter symbols.

4 PROBLEM STATEMENT

In Figure 1, we present a general monitoring architecture in a closed-loop CPS. The task of the monitor is to issue a verdict regarding the satisfaction of a specification of interest. The monitor can rely

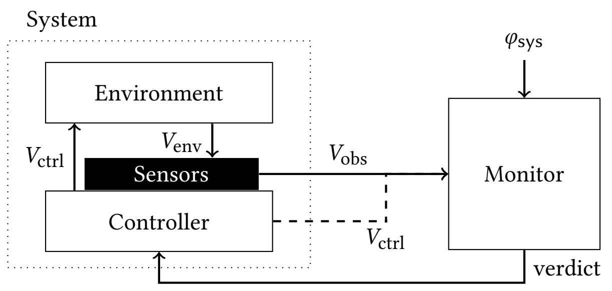


Figure 1: A general monitoring architecture in a CPS. This Figure has been redrawn from [14].

only on measurements received via a designated sensor interface to issue its verdict. This means that the monitor deduces information about the system's state (specifically, the environment) using only valuations of an observable feature space provided by the sensor measurements and may have only partial observability of the system's current state.

Formally, we distinguish between two sets of variables: the set of system-level variables V_{sys}, and the set of observable variables V_{obs}. The sets V_{sys} include variables defining both the actual state of the environment V_{env} and control V_{ctrl}. The set V_{obs} includes variables that capture the sensors' measurements.

In the case of autonomous car driving on a city road, the set V_{sys} might include variables such as the absolute coordinates of the *ego car* (namely, the self-driving vehicle under study), the distance between the ego car and other agents, the characteristics of objects on the road. In contrast, V_{obs} might include information collected by sensors mounted on the car, such as images collected by a front-facing camera, radar data, etc., and sometimes also part of V_{ctrl} defining the state of the internal control, e.g., velocity, steering, etc.

Let Σ_{sys} and Σ_{obs} define the sets of valuations of the set of variables V_{sys} and V_{obs}, respectively. The system behavior corresponds to a set $S_{\text{sys}} \subseteq \Sigma^*_{\text{sys}}$ of finite traces over Σ_{sys}, which we denote as system-level traces, while its executions observable to the monitor are defined as a set $S_{\text{obs}} \subseteq \Sigma^*_{\text{obs}}$ of traces over Σ_{obs}, denoted as observable traces. Considering the partiality on the monitor side, different system-level traces may induce the same observable trace. We assume, however, that a system-level trace always induces the same observable trace[2]. Consequently, the (unknown) relation between the system-level behaviors and the corresponding observations can be expressed by a mapping $\Omega\colon S_{\text{sys}} \longrightarrow S_{\text{obs}}$.

The system-level specification φ_{sys} is known and defined as a set of traces over valuations of the variables in V_{sys}. It is used to determine whether an execution of S is *safe* or *unsafe*: a system behavior $\sigma \in S_{\text{sys}}$ is considered *safe* if $\sigma \models \varphi_{\text{sys}}$, and *unsafe* otherwise (for consistency with the notation of STL, we use the symbol $\models$ for generic specifications to indicate membership in the set, i.e. $\in$). Since not every variable in φ_{sys} may be directly observable by our monitor, our goal is to translate φ_{sys} to a specification φ_{obs} defined over variables in V_{obs} that captures the validity of φ_{sys} on the observation level. Ideally, we want to find a specification $\varphi_{\text{obs}} \subseteq \Sigma^*_{\text{obs}}$ such that, for every observable trace $x \in S_{\text{obs}}$, $x \models \varphi_{\text{obs}}$ if and only if, $\forall \sigma \in \Omega^{-1}(x), \sigma \models \varphi_{sys}$. Such a formulation of the problem is however idealized and may not be feasible in practice [25]. First, this formulation is too conservative. An observation trace is excluded if it can be mapped back to any system-level trace that violates the system-level specification. In practice, this will mostly lead to a specification that rejects most observation traces. A more practical definition of the problem takes into consideration the occurrence rate of system-level traces and optimizes a specification for traces that are more frequent. Furthermore, depending on the (syntactic) class of specifications over the observation space from which φ_{obs} is to be chosen, we may not always find an exact match to φ_{sys}. Yet, we might want to search for the optimal specification within this class. To address these challenges, we change our formulation to define a quantitative optimization problem that mines optimal solutions.

[2]We assume a setting with deterministic sensors. We consider the setting of noisy sensors to be an extension of our approach and leave it for future work.

Before introducing our quantitative formulation of the mining problem, we point to two key points:

- in practice, one may want to bias the search toward higher false positive rates to reduce the rate of false negatives. In our formulation, we encode this using specific cost measures for the false positives and false negatives.
- monitors need to be predictive, i.e., they should determine the violation/satisfaction of a specification many steps in advance. To this end, we define our problem in terms of a prediction horizon h, which refers to the violation/satisfaction of a specification h time steps before the end of the execution.

With that, we can now define the problem of mining temporal properties for predictive monitoring as follows.

Problem 1. *Given a specification $\varphi_{sys} \subseteq \Sigma^*_{sys}$, a prediction horizon h, and a grammar G with variables in V_{obs}, find a specification $\varphi_{obs} \in L(G)$, such that:*

$$\begin{aligned} &\varphi_{obs} \in \operatorname*{argmin}_{\psi \in L(G)} \\ &\mu_1(\{\Omega(\sigma) \mid \sigma \in S_{sys} \wedge \sigma \models \varphi_{sys} \wedge (\Omega(\sigma))^{-h} \not\models \psi\}) + \\ &\mu_2(\{\Omega(\sigma) \mid \sigma \in S_{sys} \wedge \sigma \not\models \varphi_{sys} \wedge (\Omega(\sigma))^{-h} \models \psi\}), \end{aligned} \tag{1}$$

where $\mu_1, \mu_2 \colon \mathcal{P}(S_{obs}) \longrightarrow \mathbb{R}^+$ are appropriate measures of cost.

We define the argument of μ_1 as our set of *false positives*: safe executions that the learned specification predicts to be unsafe. Conversely, the argument of μ_2 will be our set of *false negatives*: unsafe executions misclassified as safe.

In the next section, we introduce a framework for solving the problem for specifications given in terms of STL formulas.

5 METHODOLOGY

In this section, we describe our approach for mining φ_{obs} from Eq. (1) as an STL formula. Figure 2 summarizes our three main steps: the dataset generation (sec 5.1), the grammar-based STL learning (sec 5.2), and the building of the robustness-based ensembles (sec 5.3).

5.1 Dataset Generation

We structure the set of system executions as a labeled dataset in the following way. For each execution, we collect both the system-level

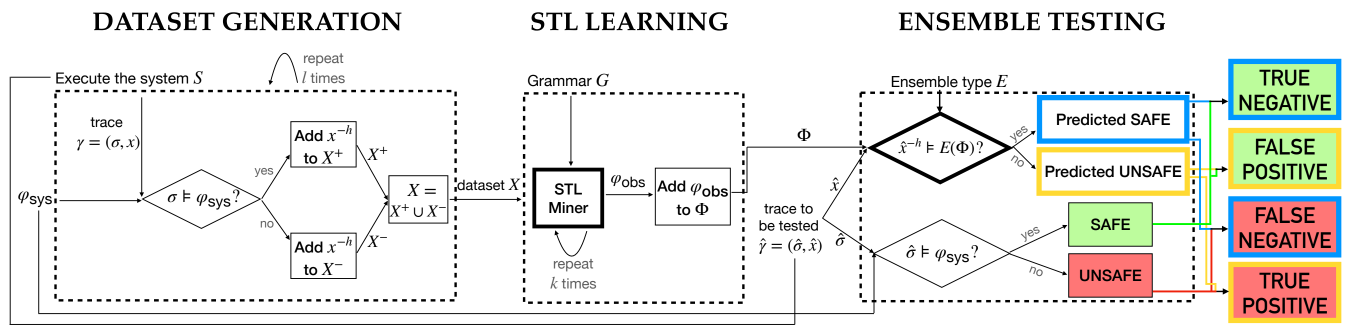


Figure 2: Diagram representing the three main steps of our approach

behavior $\sigma \in S_{sys}$ and the corresponding observations $\Omega(\sigma) = x \in S_{obs}$ to create the trace $\gamma = (\sigma, x)$. Then, we monitor σ against the system-level specification φ_{sys}: if $\sigma \models \varphi_{sys}$, we add the trimmed observation x^{-h} to the set of safe examples X^+; whereas, if $\sigma \not\models \varphi_{sys}$, we add x^{-h} to the set of unsafe examples X^-. The input to our mining method will then be the dataset of size l (defined by the user) containing the trimmed labeled observations of our system: $X = X^+ \cup X^-$, where, for each $z \in X$, we keep the information whether z is a positive ($z \in X^+$) or a negative ($z \in X^-$) example[3]. With this notation, given a candidate STL specification φ, the corresponding set of *false positives* correspond to $F_p(\varphi) := \{z \in X^+ \mid z \not\models \varphi\}$, while the set of *false negatives* to $F_n(\varphi) = \{z \in X^- \mid z \models \varphi\}$. Analogously, the *true positives* are $T_p(\varphi) := \{z \in X^+ \mid z \models \varphi\}$, and the *true negatives* $T_n(\varphi) := \{z \in X^- \mid z \not\models \varphi\}$.

5.2 Grammar-Based Mining of STL Monitors

We adopt a grammar-based approach for mining STL properties, inspired by [5]. In that previous work, the user chooses a (sub)-grammar that defines the set of admissible (e.g., STL) specifications, which are then learned from a set of positive examples. This grammar-based customization allows to encode prior knowledge, making the inferred specifications more application-specific. Our work advances this approach by addressing the need for binary classification, which is essential for distinguishing safe executions from unsafe ones. We therefore introduce a procedure that mines STL specification classifiers from both positive and negative examples. Our method integrates [5] with a Simulated Annealing [15] approach with the goal of solving the optimization problem defined in Eq.(1) and finding an approximation φ^* for φ_{obs} (we will detail our choice of the cost measures in section 5.2.1).

Algorithm 1 outlines the main steps in our procedure. We start by uniformly randomly sampling a PSTL formula template among all the possible ones admitted by the grammar G and having the user-defined l_{max} maximum length. We then instantiate its parameters by computing an approximation of the cost function minimum with the off-the-shelf Powell method [22] implemented in the SciPy [13] Python library. We store the resulting formula as our initial optimal solution φ^* and enter a loop to refine the candidate formula. This loop ends either when a satisfying STL formula is found (i.e., with cost below a given threshold ε) or after a maximum number of iterations. As standard in Simulated Annealing algorithms, iterations in this loop are characterized by the variable temperature (*temp*), which regulates the typical trade-off between exploration and exploitation throughout the execution of the program. In particular, higher values of *temp* are employed in the initial phases to favor exploration by allowing for bigger changes in the candidate formula and higher probabilities of accepting mutations that worsen its cost. During the execution of the program, the number of iterations with the same value of *temp* increases, while *temp* decreases to gradually shift the goal toward exploitation (characterized by smaller changes that have lower probability to be accepted when increasing the cost).

We apply changes to the candidate formula φ_{par} using mutations [5]: the PSTL formula is represented as a syntax tree and one of its subtrees is randomly sampled and replaced by a new subtree generated randomly and such that the resulting formula is still admitted by the grammar G and has maximum length l_{max}. The function *ApplyChanges* we propose here differs from its previous version in two aspects: (i) The value of the temperature variable is affecting the length of the subtree to be removed: for smaller values of *temp* (i.e., toward the final sets of iterations), only smaller subtrees are allowed to be removed; (ii) The new subtree does not have to be of the same length as the older one.

We instantiate the parameters values for the mutated PSTL formula ψ_{par} and update the candidate formula φ to ψ with probability $p = \exp(-\frac{\text{Cost}(\psi,X) - \text{Cost}(\varphi,X)}{temp})$ if ψ has higher cost than φ, and with probability 1 otherwise. Note that p decreases when increasing the cost of ψ or when decreasing *temp*. Such a choice is adopted to favor exploration in the initial phases of the algorithm (when *temp* is high), while, at the same time, discouraging very poor formulas ψ. Finally, at every iteration, the best formula φ^* is updated if and only if a new formula with lower cost is found.

5.2.1 *Objective Function.*

For a given labeled dataset $X = X^+ \cup X^-$ and an STL specification φ, we define the cost $Cost(\varphi, X)$ according to Eq. (1) as the sum of a cost on the false positives and a cost on the false negatives, namely $Cost(\varphi, X) := \mu_1(F_p(\varphi)) + \mu_2(F_n(\varphi))$. We want to define the cost measures μ_1 and μ_2 to account for not only the ratio of misclassified traces, but also the *degrees* of such errors in terms of their robustness values.

[3] We recall that the set X may be unbalanced, with $|X^+| \gg |X^-|$.

Algorithm 1: Grammar-based Simulated Annealing for mining an STL classifier

```
Input: Set of labeled traces $X$, STL (sub)grammar $G$,
       maximum formula length $l_{\max}$, initial temperature
       $temp$, cost threshold $\varepsilon$, maximum number of
       iterations $N_{\max}$
Output: STL formula $\varphi^*$ generated from $G$ classifying
        traces in $X$
1  $\varphi_{\text{par}} \leftarrow$ SampleInitialFormula$(G, l_{\max})$
2  $\varphi \leftarrow$InstantiateParameters$(\varphi_{\text{par}}, X)$
3  $\varphi^* \leftarrow \varphi$
4  $i \leftarrow 0$
5  while True do
6      for $n_{temp} = 1, \ldots, ComputeNumbIter(temp)$ do
7          if $(Cost(\varphi^*, X) \leq \varepsilon) \vee (i \geq N_{\max})$ then return $\varphi^*$
8          $\psi_{\text{par}} \leftarrow$ ApplyChanges$(\varphi_{\text{par}}, G, temp, l_{\max})$
9          $\psi \leftarrow$InstantiateParameters$(\psi_{\text{par}}, X)$
10         if $UpdateCandidate(\varphi, \psi, X, temp)$ then $\varphi \leftarrow \psi$
11         if $Cost(\psi, X) < Cost(\varphi^*, X)$ then $\varphi^* \leftarrow \varphi$
12         $i \leftarrow i + 1$
13     $temp \leftarrow$ DecreaseTemp$(temp)$
```

Formally, we define $\mu_1, \mu_2 \colon \mathcal{P}(S_{\text{obs}}) \to \mathbb{R}^+$ as, for any $Y \in \mathcal{P}(S_{\text{obs}})$, $\mu_1(Y) := \mu(Y, |X^+|)$ and $\mu_2(Y) := \mu(Y, |X^-|)$ where the auxiliary function $\mu \colon \mathcal{P}(S_{\text{obs}}) \times \mathbb{N} \to \mathbb{R}^+$ is defined as

$$\mu(Y, n) := \frac{|Y|}{n} + \frac{1}{2} \cdot \frac{\sum_{y \in Y} |\rho(\varphi; y)|}{|Y|} + 2 \cdot \mathbb{1}_{\left\{\frac{|Y|}{n} > 0.7\right\}}. \tag{2}$$

The first term in Eq. (2) is standard in binary classification as it is used to evaluate the ratio of misclassified traces. Indeed, when computing $\mu_1(F_p(\varphi)) = \mu(F_p(\varphi), |X^+|)$, this term corresponds to the *ratio of false positives* $\frac{|F_p(\varphi)|}{|X^+|}$, namely the number of traces incorrectly classified as safe over the total number of safe traces. Analogously, the first addend in $\mu_2(F_n(\varphi))$ computes the *ratio of false negatives.*

The second addend corresponds to the half average absolute robustness value of the traces in the input set Y. We interpret the robustness as a confidence measure of the classifier: hence, we would like such a confidence to be low whenever a misclassification occurs (i.e., when $Y = F_p(\varphi)$ or $Y = F_n(\varphi)$). In order to give every variable in φ the same weight, we assume the dataset X had been previously normalized such that each variable V_{obs} varies between 0 and 1. Consequently, it is possible to prove recursively on the definition of ρ that the fraction takes values in $[0, 1]$. We halve this value to place less emphasis on this term relative to others.

Finally, in the third addend, the indicator function, which has value 1 if $\frac{|Y|}{n} > 0.7$ and 0 otherwise, is used to penalize high ratios of misclassified traces. The value of 2 is chosen to prevail over the other two terms (whose sum can be maximum 1.5) in case the condition $\frac{|Y|}{n} \leq 0.7$ is violated. This term introduces a significant jump discontinuity in the objective function, but is fundamental for dealing with highly imbalanced datasets because it avoids that all traces are predicted in the most numerous class.

5.3 Robustness-based STL Ensembles

To improve the classification performance, we run Algorithm 1 several times and learn a set Φ of k STL formulas (in general, the randomness in our procedure will produce different formulas even for the same training dataset). To predict the safety of the (possibly unseen) execution $\hat{y} = (\hat{\sigma}, \hat{x})$, we evaluate its truncated observation $\hat{x}^{-h}$ with respect to each monitor and combine their verdicts into a single one. To do so, we propose two new voting criteria that leverage the STL quantitative semantics: *Total Robustness Voting* and *Largest Robustness Voting.*

For the *Total Robustness Voting* (TRV), we compute the sum of the robustness values of the formulas in Φ: $R := \sum_{\varphi \in \Phi} \rho(\varphi; \hat{x}^{-h})$. If R is positive, the ensemble TRV will predict that the trace $\hat{y}$ is safe; otherwise, it predicts it to be unsafe. This ensemble method can be seen as an extension of *majority voting* that, instead, assigns each monitor a binary verdict (either 1 or −1). Conversely, we weight each verdict by the confidence level associated with that prediction, as quantified by the robustness function.

The *Largest Robustness Voting* (LRV) selects the STL specification $\bar{\varphi}$ in the set Φ with the largest absolute robustness value: $\bar{\varphi} = \text{argmax}_{\varphi \in \Phi} |\rho(\varphi; \hat{x}^{-h})|$. Then, if $\hat{x}^{-h} \models \bar{\varphi}$, the trace is predicted to be safe, and negative otherwise.

6 EVALUATION

To assess whether the ensembles of STL formulas that we mine from the observable variables are effective predictive safety monitors and study how sensitive they are to dataset characteristics, we investigate the following research questions:

RQ1. [Predictive Performance] *Can our method learn accurate, efficient, and explainable safety predictor monitors?* We use our monitors in safety-critical applications, hence it is essential to evaluate both the accuracy of their predictions and the ease of interpreting their meaning. In addition, it is important to assess the cost-effectiveness of our approach.

Metrics: We use standard metrics for assessing the quality of binary classification systems. Given a binary classifier and a labeled dataset, we compute the corresponding number of true positives tp (correctly predicted safety violations), true negatives tn (correctly predicted safe behaviors), false positives fp (unwarranted alarms), and false negatives fn (missed violations). Consequently, we define:

$$\textbf{Accuracy} := \frac{tp + tn}{tp + tn + fp + fn}; \quad \textbf{Precision} := \frac{tp}{tp + fp};$$

$$\textbf{Recall} := \frac{tp}{tp + fn}; \quad \textbf{F1-Score} := 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\text{Precision} + \text{Recall})}$$

Accuracy is the proportion of correctly classified samples; however, since this measure is unreliable for imbalanced datasets, we also consider Precision and Recall. Precision corresponds to the classifier's ability to raise an alert *only* for unsafe executions, whereas Recall is the ability to raise an alert for *all* unsafe executions. The F1-Score, instead, combines them using harmonic mean. All the metrics defined above vary between 0 and 1, with higher values corresponding to better predictors. We assess the explainability of the mined monitors through a manual qualitative analysis and by evaluating their size, as monitors with fewer formulas, conditions, and predicates are intuitively easier to read and analyze. As detailed in section 6.1.2, we contextualize the results by comparing them against those achieved by existing approaches.

RQ2. [Voting Criterion] *Is robustness a suitable voting criterion?* We propose two new voting criteria (TRV and LRV) for STL specifications that leverage robustness values. Since robustness-based voting criteria were never studied before, investigating whether they lead to better predictions than existing criteria is important.

Metrics: We use the metrics defined in RQ1 (i.e., Accuracy, Recall, Precision, and F1-Score) to evaluate whether using robustness as a voting criterion improves predictors' performance.

RQ3. [Sensitivity Analysis] *How sensitive is our method to dataset characteristics?* We use passive learning to infer predictive STL safety monitors from labeled scenario executions. It follows that the training dataset is fixed for the mining task, and we cannot add new training examples. Therefore, the training dataset can have an important influence on the learning process, and we must study its impact on the accuracy of our predictions. More specifically, we study the robustness of the mining process with respect to the size of the dataset, the choice of features, and the degree of imbalance between the safe and unsafe labels in the data. Studying the sensitivity of our predictors concerning the training dataset is important to understand their practical applicability to safety, where nominal cases are much more common than critical ones.

Metrics: We use F1-Score to study how the classification performance varies with different characteristics of the training dataset. We use computational time to study the scalability of our method and ensure it remains effective under different conditions.

6.1 Experimental Setting

In this section, we report on the two case studies used in the evaluation and the state-of-art approaches selected as baselines to contextualize our results. Next, we report on the dataset, hyperparameter settings, and other setups.

6.1.1 Case Studies. Our evaluation considers the Lead-Follower (LF) and the Traffic Cones (TC) case studies involving an autonomous vehicle under test (the *ego* car) that must solve a critical task.

The **Lead-Follower** (Figure 3) case study was originally presented by Yalcinkaya et al. [27] in the context of compositional analysis of autonomous systems. This case study involves a (black) leading car programmed to drive across intersections and the ego car that must follow it. The ego car autonomously steers by coupling a vision-based component with a controller. The vision-based component analyses the images captured by a front-facing camera mounted on the ego car to estimate the heading difference between the lead and the ego car. The controller acts on the steering wheel to minimize such a difference. As illustrated in Figure 3, the ego vehicle is challenged by dark objects (e.g., a black cube) on the road that might cause the vision-based component to wrongly estimate the position of the lead car and eventually crash. To disambiguate critical situations like the one depicted in Figure 3 or situations in which the ego car loses sight of the lead vehicle from correct ones, we rely on the following system-level specification: $\varphi_{\mathsf{sys}} := \mathbf{G}((d > 5) \wedge (d < 15))$, where d represents the Euclidean distance between the lead and the ego car. The specification φ_{sys} is violated either when the distance between two vehicles becomes too small ($d \leq 5$) or too large ($d \geq 15$). The first case represents a clear safety issue. The second case indicates that the ego is not able to follow anymore the lead vehicle. While this type of violation does not necessarily represent a safety risk, it arises in situations where the ego vehicle hits an obstacle.

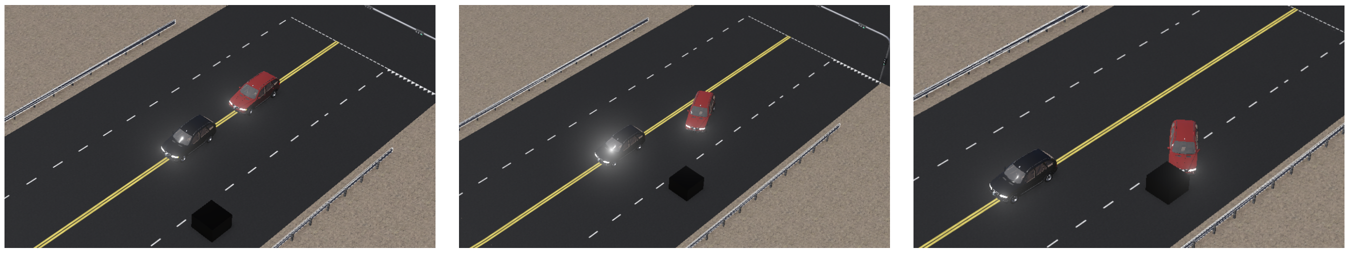
The **Traffic Cone** (Figure 4) case study was originally proposed to showcase VerifAI [9], a platform for the formal design and analysis of learning-enabled cyber-physical systems. According to the original setup, a broken car is on the left side of a three-lane road, and three traffic cones in front of it signal the issue to oncoming traffic. The ego car is approaching in the middle lane and must move to the rightmost lane to avoid the obstacle. The ego car implements vision-based lane-keeping and collision avoidance. To keep the lane, it estimates the distance to the lane center by analyzing images collected from a front-facing camera and adjusting the steering angle accordingly. Simultaneously, it performs object detection using a pre-trained convolutional neural network (CNN) to detect the presence of traffic cones and initiate the lane-changing maneuver if the estimated distance from the traffic cones falls below 15 meters. We consider safe executions those cases in which the ego car detects the traffic cones and changes lanes before touching them. Formally, we rely on the following system-level specification to identify safe executions: $\varphi_{\mathsf{sys}} := \mathbf{G}(d > 0.75)$, where d indicates the Euclidean distance between the ego car and the traffic cones' center, and 0.75 is the radius of standard traffic cones.

6.1.2 Baselines for the Evaluation. To contextualize the results achieved by our STL predictors (TRV and LRV), we compare their results against those of existing works from the literature. Specifically, we consider *MV-dt* by Torfah et al. [24], who defined a majority voting ensemble over Decision Trees, and two template-free STL miners for binary classification: *STL-dt* by Bombara et al. [6], based on decision trees, and *STL-enum* by Mohammadinejad et al. [19], which learns STL formulas by enumerating PSTL formulas.

6.1.3 Dataset. To collect the training dataset, we instantiated the LF and TC scenarios randomly. For LF, we randomly placed a colored box on the side of the road; the box's color and position were sampled randomly. Likewise, we randomly sampled the initial coordinates and orientations of the two cars. For TC, we randomly configured the color and rotation of the broken car, the initial velocity, position, and heading of the ego car; additionally, we fuzzed the initial position of the traffic cones.

For each case study, we collected 2000 executions to train the safety predictors and 400 to test them. Unsafe simulations are rare in both scenarios, so the resulting datasets are highly imbalanced: for LF approximately 15% of the traces are unsafe, whereas for TC, unsafe traces account for only 5% of the traces.

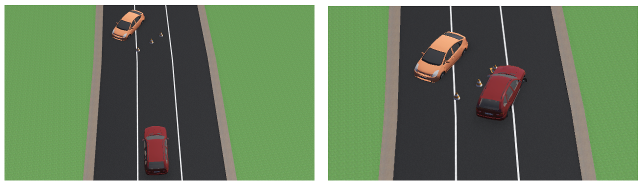
Since we did not constrain the scenario execution, traces do not have a fixed length. For instance, the average trace length is 1748 simulation steps in the Lead-Follower case study and 377 simulation steps in the Traffic-Cone case study. *MV-dt*, *STL-dt*, and *STL-enum* require the traces to be all of the same length. Therefore, we preprocessed their training datasets by applying a sliding window of length j, thus generating $|x| - j + 1$ new segments for each trace x used for training. We labeled each segment independently depending on whether it satisfied the safety specification φ_{sys}.


(a) 150 time steps before the end of the execution: the ego car correctly follows the lead.

(b) 100 time steps before the end of the execution: the ego car deviates its trajectory toward the box.

(c) End of the execution: the ego car crashes against the box.

Figure 3: Snapshots of three key moments in an unsafe execution of the Lead-Follower case study


(a) Ego car performing *lane keeping*.

(b) Ego car performing *lane changing* too late and hitting one traffic cone.

Figure 4: Snapshots from an unsafe execution in TC

6.1.4 Hyperparameters. We mined 10 STL formulas for every ensemble Φ , so each specification was learned from a different *batch* of $\frac{2000}{10} = 200$ training traces. Given the randomness intrinsic to Algorithm 1, we repeated every experiment 10 times. We fixed the maximum formula length $l_{\max}$ to 7 (measured as the sum of Boolean/temporal operators and predicates) , the grammar G to the until-free fragment of STL whose predicates are defined over variables V_{obs}, the cost threshold ε to 0.05, and the maximum number of iterations $N_{\max}$ to 50. Finally, we set the prediction horizon h to 100 simulation steps to balance the trade-off between capturing deviations from the desired behavior (which requires more observations) and allowing enough time for corrective actions before a potential violation (which requires prompt intervention).

6.1.5 Software and Hardware Setup. We used Scenic [10] to define the distributions over the initial conditions, VerifAI [9] to randomly sample among them and launch the corresponding execution, Webots [26] to simulate the system, and RTAMT [20] to monitor STL formulas and compute their robustness. We ran the experiments on a non-dedicated cluster with three Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and Ubuntu 20.04 nodes.

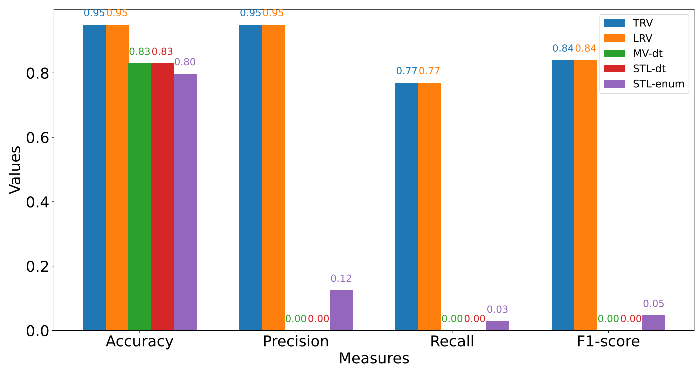
6.2 RQ1 - Performance Evaluation

To answer RQ1, we learned the monitors using the selected training data, predicted the label of the test data, and computed the accuracy, precision, recall, and F1-Score achieved by the two versions of our approach, TRV and LRV, and the three baselines, *MV-dt*, *STL-dt*, and *STL-enum*. We repeated the experiments 10 times to account for the intrinsic randomness of TRV and LRV in training. We report the average values across the runs in Figure 5. Notably, since the three baselines work on trace segmented, we computed their performance metrics as follows: we predicted the label of a trace x by evaluating the monitor against all segments generated from x and classifying x as unsafe if and only if at least one of its segments is predicted unsafe. To select the appropriate value for the sliding window length (j), we conducted a preliminary evaluation varying the value of j among 15, 150, 300 (for TC) and 1500 (for LF). We selected the largest value such that predictors were able to complete training and produced results. In summary, for *MV-dt* we set $j = 15$ for both LF and TC, whereas for *STL-dt* and *STL-enum* we set $j = 1500$ for LF and $j = 300$ for TC [4].

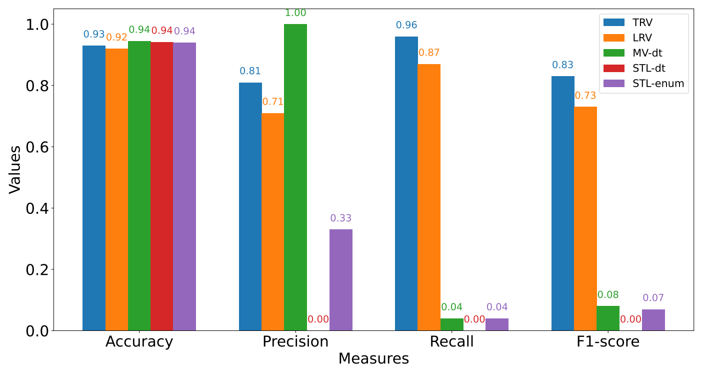
In the LF case study (Figure 5a), both TRV and LRV outperformed the existing works on all four metrics. In particular, we notice that although the accuracy of all the methods is comparable, there is a significant difference between the Precision, Recall, and F1-Score of TRV and LRV compared to the three baselines. We explain this remarkable difference in terms of the imbalance of the dataset. Since the majority of the training data are safe, *MV-dt* and *STL-dt* classify all traces as safe; this, in turn, inflates the accuracy but reduces the other metrics. *STL-enum* is slightly more robust with 2 detected true positives (out of 69); consequently, its precision and recall are greater than zero. However, these values remain too low to be satisfactory. Conversely, TRV and LRV detected 53 (out of 69) true positives on average, so they achieved higher precision, recall, and F1-Score. We make similar observations for the Traffic Cones use case (Fig. 5b): although all five methods achieved similar accuracy, they have significantly different distributions of misclassified traces and F1-Score values, with TRV and LRV outperforming the others.

A possible explanation for the poor performance of the three baselines is that they are driven by the misclassification rate, which does not effectively account for the possible imbalances of the dataset. In particular, in our experiments, we observed that *STL-dt* and *STL-enum* returned the first candidate formula they considered: **true** for the former, and a predicate ($p \geq l_p$) stating that a variable p is greater than its lower bound l_p on the first time step for the latter. These predictors accepted the first candidate formulas because they were satisfied by (almost) all traces in the training dataset, thus achieving a low misclassification rate. On the contrary, our proposed objective function did not lead to cluster all traces into a single class because it strongly penalizes high ratios of false negatives and false positives. However, by doing so, we

[4]For *STL-enum* we had to reduce the size of the training batches from 200 to 50.



(a) Lead-Follower



(b) Traffic Cones

Figure 5: Average values for accuracy, precision, recall, and F1-Score for the two variants of our approach (TRV and LRV), and three methods from the literature: *MV-dt* [24], *STL-dt* [6], and *STL-enum* [19].

Table 1: Comparison of computational time required to mine an ensemble of 10 monitors (in hours).

	TRV	LRV	MV-dt	STL-dt	STL-enum
LF	16.0 ± 2.2	16.0 ± 2.2	0.97	$1 \cdot 10^{-4}$	1.8
TC	1.3 ± 0.2	1.3 ± 0.2	0.14	$9 \cdot 10^{-5}$	0.5

introduced non-continuity in the objective function, thus making it more challenging to optimize. This drawback is reflected in the high computational time required by our methods, as summarized in Table 1. Notably, the significant discrepancy in computational times between the two case studies arises from the longer executions of LF scenarios compared to TC ones (approximately four times longer). We remark that the times reported in Table 1 refer to the training phase of the STL monitors. In particular, these learning costs do not impact the time required for making predictions at runtime, which depends on the chosen STL monitor and the number of monitors in the ensemble. Therefore, in practice, this high computational cost does not limit the applicability of our approach in real-time scenarios.

In terms of explainability, we note that the formulas returned by *STL-dt* (**true**) and *STL-enum* ($p \geq l_p$) are certainly straightforward to interpret, but useless in practice. The trees learned by *MV-dt*, instead, contain between one and two hundred nodes for the Lead-Follower case study and 30 to 120 nodes for the Traffic-Cone case study, making any interpretation of these monitors challenging.

In contrast, our mined STL formulas have a bounded size by construction, making them significantly easier to parse. We remark that short specifications are not necessarily interpretable if they are too abstract. However, our grammar-based approach helps mitigate this risk, as the user can filter the admissible basic predicates in the grammar according to their explainable relevance to the specific application. As a result, the mined formulas can be easily understood, thus providing precious insights into the characterization of safety. To illustrate this point, we report below an example of mined formulas for each case study.

$$\mathbf{F}_{[321:+\infty]}\mathbf{G}(n\_black\_pix < 37.6) \tag{3}$$

$$\mathbf{G}\neg\mathbf{G}(\mathbf{F}(n\_orange\_pix < 0) \vee size\_sec\_bb < 373.9) \tag{4}$$

Formula 3 states that after at least 321 simulation steps, the number of black pixels in the image (n_black_pix) must remain below 37.6 (effectively, below 38); this formula matches our intuition that, in safe executions, the follower should not get too close to any particular black object. Although Formula 4 does not look intuitive, we note that it can be simplified using trivial equivalence rules. For instance, using the equivalence $\mathbf{F}\varphi = \neg\mathbf{G}\neg\varphi$, it becomes:

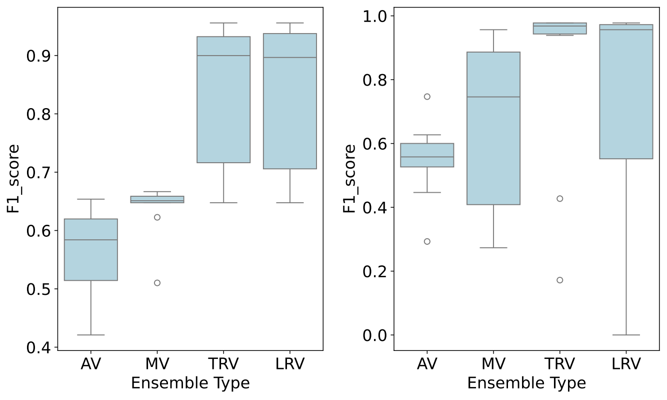
$$\mathbf{GF}(\mathbf{G}(n\_orange\_pix \geq 0) \wedge size\_sec\_bb \geq 373.9).$$

Since the number of orange pixels (n_orange_pix) in an image cannot be negative, we can further simplify the formula to obtain $\mathbf{GF}(size\_sec\_bb \geq 373.9)$, which succinctly captures the lower bound on the size of the second biggest bounding box ($size\_sec\_bb$) returned by the CNN in the final part of the execution. In practice, Formula 4 is violated in unsafe executions likely because, toward the end of the execution, the ego car has not yet correctly identified all three traffic cones. As a result, the second biggest bounding box - if even existing - is too small.

The safety monitors generated by our method from imbalanced data outperform the monitors learned from the same data by the existing techniques. Our monitors are also easier to interpret than the decision tree-based methods, although they have a significantly higher computational price.

6.3 RQ2 - Voting Criterion

To address RQ2, we compare the performance of our robustness-based ensembles, TRV and LRV, against the traditional voting mechanism of majority voting (MV). For a given set Φ of STL formulas and the trace for which we want to predict safety, MV evaluates the binary outcome of each monitor (safe vs unsafe) and chooses the prediction voted by the majority of the formulas as the final outcome. For completeness, we also compare the performances



(a) Lead-Follower **(b) Traffic Cones**

Figure 6: Boxplots of F1-Score values for four kinds of ensemble: AV, MV and the robustness-based TRV and LRV.

of TRV and LRF against the average prediction of each individual monitor (AV): in this case, we evaluate each prediction separately without ensembling them and then we take the average value for each performance metric. Thus, AV does not represent a proper voting criterion, but it is still relevant as baseline. For a fair comparison, we use the same set Φ of 10 STL formulas learned via Algorithm 1 for all voting mechanisms.

Figure 6 plots the distribution of F1-Score over 10 runs of the four ensemble techniques on the two case studies. In the Lead-Follower scenario (Figure 6a), we observe that TRV and LRV performed equally well and outperformed MV, while MV outperformed AV. The Traffic Cone case study (Fig. 6b) confirms the superior performance of the TRV ensemble. While LRV shows a similar median, its distribution has much greater variability, indicating that LRV's predictions are less reliable. We, therefore, performed a more quantitative analysis to get a better understanding of these results. More specifically, we assessed the statistical significance of the results using the Mann-Whitney U test and measured the strength of the significance using Vargha and Delaney's effect size, expressed with labels according to Kitchenham et al.'s classification [16]. This analysis confirmed that LRV achieved significantly better performance than MV and AV with *medium* effect size.

The TRV and LRV ensemble methods, which leverage the robust semantics of STL, significantly increase the prediction accuracy of the safety monitors compared to classical ensemble and averaging methods.

6.4 RQ3 - Dataset Characteristics

To evaluate how variations in key characteristics of the training dataset affect the performance of our predictors, we conducted three additional sets of experiments modifying:

- *Imbalance between positive and negative samples.* In the LF case study, we reduced the number of negative samples from 15% to 10% and 5%, whereas in the TC case study, we incremented it from 5% to 10% and then reduced it to 2.5%.
- *Dataset size.* Large training sets usually result in better prediction performance at the price of longer training times and higher costs. We were interested in understanding how much our predictors' performance drops when using smaller training datasets that entail shorter training procedures. Therefore, we reduced the batch size from 200 to 150, 100, and 50 resulting in training sets of 1500, 1000, and 500 traces.
- *Prediction horizon (h).* We recall that our safety monitors are predictive, i.e., they are trained to detect potential safety violations h steps in advance. The higher the predictive horizon, the more time the system has to take corrective action and avoid safety violations. However, the more difficult it is for the monitor to make an accurate prediction. We studied the effect of the prediction horizon size on the accuracy of our safety monitors. The prediction horizon directly impacts the dataset since we trim h steps from every trace. We experimented with h values set to 50, 100, and 150.

We focused on the best-performing ensemble, i.e., TRV, and investigated how the above dataset characteristics impact its performance. We report in Figure 7 the results of our experiments in terms of F1-Score (performance, top-row) and computational time (training effort, bottom row) across 10 runs. The solid lines in the plots mark average values, while the shaded regions identify the confidence intervals around them. In both the Lead-Follower (Figure 7a) and Traffic Cones (Figure 7b) case studies, we observe that TVR's performance and training time were stable against variations in the imbalance, suggesting the ensemble is not sensitive to them. As expected, training time increased with the size of the training dataset (bottom row, middle plots). However, although the performance increased with larger training datasets in the Lead-Follower case study, this was different in the Traffic Cones. This observation is due to the restrained parameter space and, consequently, the minor variability of concrete scenarios in the Traffic Cones case study. Even a small dataset of simulations generated from this case study sufficiently represents the system.

Finally, when extending the prediction horizon to 150 simulation steps, our method maintains a comparably high F1-Score despite the increased difficulty of earlier predictions. As expected, the F1-score remains high also for the shorter 50-step horizon since predicting criticality is easier when closer to a potential crash. It is particularly interesting to notice that our method achieves perfect prediction performance in the Traffic Cones case study, suggesting that this time interval may be too short to avoid a crash.

Our mining approach is generally robust to changes in the level of imbalance in the training data and its size, which makes it suitable for practical applications. However, the choice of prediction horizon is application-dependent and may affect the quality of the predictions. Nevertheless, our method has shown stability under significant variations in h.

6.5 Threats to Validity

We identified the following main threats that might affect the validity of our conclusions.

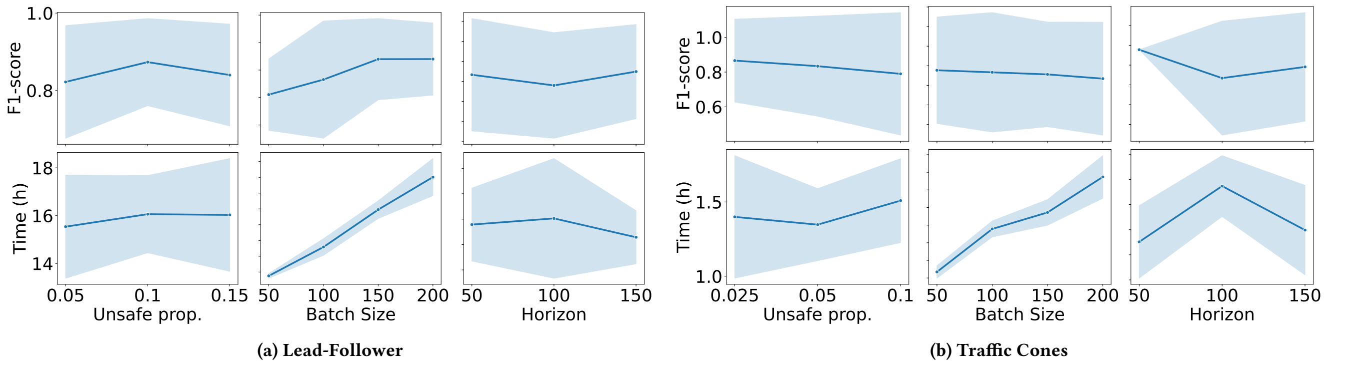


Figure 7: F1-Score and computational time (in hours) achieved by TRV when varying the proportion of unsafe traces, batch size, or prediction horizon. The line indicates the average values over 10 runs, while the shaded regions their confidence intervals.

Internal Validity. Our method can be sensitive to two sources of uncertainty: (i) the stochastic nature of our mining algorithm and (ii) the variability of the training data. To address this, we repeated each experiment 10 times and analyzed the robustness of our algorithm with respect to dataset characteristics. Since our method relies on passive learning, it may be necessary to collect more data to obtain accurate monitors. We mitigated this risk by considering sufficiently large datasets (2000 randomly generated traces) and by studying the accuracy in function of the dataset size.

External Validity. Our experiments encounter a limitation in the generalizability of the results, given that we have applied our approach to one domain with synthetically generated datasets. To some extent, we mitigated this threat by selecting two different tasks for the case studies. We plan the application of our method to other cyber-physical domains in future work. We believe that, even though real-world environments may contain more noise and uncertainty, the classification performance of STL monitors will remain satisfactory, as they capture declarative and abstract system properties without overfitting the data. This claim is supported by the experimental results shown in Figure 7, where the F1-scores remain highly stable across different case studies and dataset characteristics. Conversely, we expect scalability of computational time to more complex scenarios to be influenced by both the number and length of traces in the dataset, as demonstrated by the variability in the results of the two case studies. Nevertheless, we aim to optimize the current implementation and introduce syntactic rules to infer the satisfaction of STL formulas, thereby skipping unnecessary monitoring steps and accelerating the overall procedure.

Conclusion Validity. The accuracy of our conclusions may be affected by the selection of basic predicates and observable variables. We argue that minimal expertise and knowledge are needed for a meaningful mining process. To mitigate this risk, we had the same setup with the other related methods, and, besides selecting the basic predicates, we left the search for STL formulas unrestricted.

7 CONCLUSION AND FUTURE WORK

We introduced the first approach to learning predictive and explainable monitors for safety in the form of STL specifications from observable variables. Our approach's uniqueness lies in (i) a novel grammar-based method for distinguishing between highly imbalanced sets of positive and negative traces, and (ii) two robustness-based ensemble techniques. Experimental results demonstrated the advantages of our method over the state-of-the-art in two automotive case studies.

In future work, we plan to extend the mining method from passive to active learning by generating new traces that will allow us to refine our inferred specifications and increase their prediction accuracy. With the foreseen active mining approach, we can also provide statistical guarantees about the inferred monitors.

We noted that this paper focuses on autonomous driving applications and safety requirements. However, we foresee applying our framework to other classes of cyber-physical systems and properties defined over system-level variables that are not directly observable by the monitors. We plan to explore case studies from other domains, broadening our framework's applicability.

Finally, we have seen that our mining approach does not optimize the syntax of the specifications. We intend to define syntactic translation rules that will allow us to further simplify the mined specifications before presenting them to the user, making them even more accessible to interpret. We also plan to conduct a more systematic evaluation of explainability, in which users not involved in the grammar definition will examine the mined formulas and asses their understandability.

ACKNOWLEDGMENTS

This project has received funding from the Austrian FFG ICT of the Future program under grant agreement No 894789 (FATE project), and from the TU Wien-funded Doctoral College for SecInt: Secure and Intelligent Human-Centric Digital Technologies. This work was also partly supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by the UC Berkeley iCyPhy center. The Austrian Marshall Plan Foundation funded the Marshall Plan Scholarship, which supported Nesterini's research visit to the University of California Berkeley. Finally, the authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

REFERENCES

[1] Erfan Aasi, Cristian Ioan Vasile, Mahroo Bahreinian, and Calin Belta. 2022. Classification of Time-Series Data Using Boosted Decision Trees. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1263–1268. https://doi.org/10.1109/IROS47612.2022.9982105

[2] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. 2011. Parametric Identification of Temporal Properties. In *Proc. of RV 2011 (LNCS, Vol. 7186)*. Springer, 147–160. https://doi.org/10.1007/978-3-642-29860-8_12

[3] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. 2018. Introduction to Runtime Verification. In *Lectures on Runtime Verification - Introductory and Advanced Topics*. LNCS, Vol. 10457. Springer, 1–33. https://doi.org/10.1007/978-3-319-75632-5_1

[4] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. 2022. Survey on mining signal temporal logic specifications. *Inf. Comput.* 289, Part (2022), 104957. https://doi.org/10.1016/j.ic.2022.104957

[5] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Ničković. 2023. Mining Hyperproperties using Temporal Logics. *ACM Trans. Embed. Comput. Syst.* 22, 5s, Article 156 (2023), 26 pages. https://doi.org/10.1145/3609394

[6] Giuseppe Bombara, Cristian Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. 2016. A Decision Tree Approach to Data Classification using Signal Temporal Logic. In *Proc. of HSCC 2016*. ACM, 1–10. https://doi.org/10.1145/2883817.2883843

[7] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. 2014. On-Line Monitoring for Temporal Logic Robustness. In *Proc. of RV 2014: the 5th International Conference on Runtime Verification (LNCS, Vol. 8734)*. Springer, 231–246. https://doi.org/10.1007/978-3-319-11164-3_19

[8] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Proc. of FORMATS 2010 (LNCS, Vol. 6246)*. Springer, 92–106. https://doi.org/10.1007/978-3-642-15297-9_9

[9] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2019. VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In *Proc of CAV 2019 (LNCS, Vol. 11561)*. Springer, 432–442. https://doi.org/10.1007/978-3-030-25540-4_25

[10] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2022. Scenic: A Language for Scenario Specification and Data Generation. *Machine Learning Journal* 112 (2022), 3805—-3849. https://doi.org/10.1007/s10994-021-06120-5

[11] Patrick Indri, Alberto Bartoli, Eric Medvet, and Laura Nenzi. 2022. One-Shot Learning of Ensembles of Temporal Logic Formulas for Anomaly Detection in Cyber-Physical Systems. In *Genetic Programming (LNCS, Vol. 13223)*. Springer, 34–50. https://doi.org/10.1007/978-3-031-02056-8_3

[12] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. 2019. TeLEx: Learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design* 54, 3 (2019), 364–387. https://doi.org/10.1007/s10703-019-00332-1

[13] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001. SciPy: Open source scientific tools for Python. http://www.scipy.org/

[14] Sebastian Junges, Sanjit A. Seshia, and Hazem Torfah. 2025. Active Learning of Runtime Monitors Under Uncertainty. In *Integrated Formal Methods*, Nikolai Kosmatov and Laura Kovács (Eds.). Springer Nature Switzerland, Cham, 297–306.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671

[16] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Softw. Engg.* 22, 2 (apr 2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[17] Meiyi Ma, John A. Stankovic, Ezio Bartocci, and Lu Feng. 2021. Predictive Monitoring with Logic-Calibrated Uncertainty for Cyber-Physical Systems. *ACM Trans. Embed. Comput. Syst.* 20, 5s, Article 101 (2021), 25 pages. https://doi.org/10.1145/3477032

[18] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS and FTRTFT 2024 (LNCS, Vol. 3253)*. Springer, 152–166. https://doi.org/10.1007/978-3-540-30206-3_12

[19] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh Gopinath Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. 2020. Interpretable classification of time-series data using efficient enumerative techniques. In *Proc. of HSCC 2020*. ACM, 9:1–9:10. https://doi.org/10.1145/3365365.3382218

[20] Dejan Nickovic and Tomoya Yamaguchi. 2020. RTAMT: Online Robustness Monitors from STL. In *Proc. of ATVA 2020 (LNCS, Vol. 12302)*. Springer, 564–571. https://doi.org/10.1007/978-3-030-59152-6_34

[21] Federico Pigozzi, Eric Medvet, and Laura Nenzi. 2021. Mining Road Traffic Rules with Signal Temporal Logic and Grammar-Based Genetic Programming. *Applied Sciences* 11, 22 (2021). https://doi.org/10.3390/app112210573

[22] Michael J.D. Powell. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. In *The Computer Journal*. Vol. 7. 155–162. Issue 2. https://doi.org/10.1093/comjnl/7.2.155

[23] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. 2022. Toward Verified Artificial Intelligence. *Commun. ACM* 65, 7 (2022), 46–55. https://doi.org/10.1145/3503914

[24] Hazem Torfah, Aniruddha Joshi, Shetal Shah, S. Akshay, Supratik Chakraborty, and Sanjit A. Seshia. 2023. Learning Monitor Ensembles for Operational Design Domains. In *Proc of RV 2023 (LNCS, Vol. 14245)*. Springer, 271–290. https://doi.org/10.1007/978-3-031-44267-4_14

[25] Hazem Torfah, Carol Xie, Sebastian Junges, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2022. Learning Monitorable Operational Design Domains for Assured Autonomy. In *Proc of ATVA 2022 (LNCS, Vol. 13505)*. Springer, 3–22. https://doi.org/10.1007/978-3-031-19992-9_1

[26] Webots. 2023. http://www.cyberbotics.com Open-source Mobile Robot Simulation Software.

[27] Beyazit Yalcinkaya, Hazem Torfah, Daniel J. Fremont, and Sanjit A. Seshia. 2023. Compositional Simulation-Based Analysis of AI-Based Autonomous Systems for Markovian Specifications. In *Proc of RV 2023 (LNCS, Vol. 14245)*. Springer, 191–212. https://doi.org/10.1007/978-3-031-44267-4_10