Runtime Verification of Prediction Models Based on a Formal Specification of Assumptions

Alberto Bonizzi[1], Davide Calzà[1], Alessandro Flori[1], Andrea Gobbi[1], Konstantinos Kapellos[2], Andrea Micheli[1], Mattia Pujatti[1], and Stefano Tonetta[1]

[1] Fondazione Bruno Kessler, Trento, Italy
[2] NRB - TRASYS International, Belgium

Abstract. The runtime verification of AI-based prediction models is crucial in scenarios where incorrect predictions may lead to safety violations or degraded performance. A key challenge arises when the inputs to the prediction function are not fully observable at runtime, making it difficult to determine whether the predictions are consistent with the actual system behavior.
This paper formalizes the problem of monitoring predictions under partial observability. We define correctness with respect to a belief state derived from an assumption on the system's dynamics and a sequence of partial observations. To solve this problem, we adapt the Assumption-Based Runtime Verification framework to the prediction setting, combining belief-state tracking with a statistically grounded acceptance condition based on confidence intervals and relative standard deviation.
The approach is illustrated through an example and evaluated on a simplified space application scenario involving planetary rover operations. The experimental evaluation shows the potential scalability and feasibility of the approach to detect incorrect predictions under partial observability.

Keywords: Digital twins, AI-based prediction monitoring, trustworthy AI, Assumption-Based Runtime Verification

1 Introduction

Artificial Intelligence (AI) models are increasingly integrated into critical applications, where their predictions provide crucial operational support for decision-making and planning. For instance, in applications like digital twins used for space exploration planning, AI models may predict the resource consumption of the planned activities (see, e.g., [8]), which is necessary for an effective schedule of activities on remote applications such as rovers or satellites. The validation of such prediction models is critical, as failures resulting from incorrect predictions may lead to safety violations or severely degraded operational performance. Therefore, predictions must be robustly checked for consistency against the actual behavior and observations of the physical system during execution.

Formal verification methods are increasingly employed to verify and validate AI systems across critical applications, such as autonomous driving, and aerospace systems. Techniques such as model checking, satisfiability modulo theories (SMT) solving, and abstract interpretation have been adapted to address the verification of AI models, particularly neural networks (see, e.g., [23] for a recent survey). Runtime Verification (RV) [14] offers a complementary approach, providing the advantage of monitoring AI systems or components by treating them as black boxes. This allows for the analysis of system behavior without needing access to the internals of the AI models. However, the formalization of the properties to be monitored is often challenging, which is typically the motivation of using data-driven techniques in the first place.

In this paper, we focus on the runtime verification of prediction models, which are used to take decisions such as planning of actions in autonomous systems and that should be validated against the actual execution of the physical system. Thus, in this case, the specification of correctness can be simply formalized in terms of consistency with the observations. However, a major challenge is that the dataset used for training may be richer than the data that are observable at runtime, when some model's inputs may be not available.

The problem is motivated by an ongoing project, ExploDTwin [8, 21], where digital twins are employed for monitoring and planning the activities of space exploration applications, and use simulation and AI models to predict related resource consumption. While during planning of activities, the inputs to the prediction are provided by the planner to check the feasibility of hypothetical plans, at runtime some inputs cannot be derived from the telemetry.

To address the challenge of runtime verification in partially observable systems, this paper proposes a novel methodology based on Assumption Based Runtime Verification (ABRV), a technique proposed in [6] to enrich runtime verification with prognosis and diagnosis capabilities.

The contributions of this paper are threefold:

1. We formalize the problem of runtime verification of prediction functions under partial observability, defining how predictions can be validated when model inputs are not fully accessible and only partial system observations are available.
2. We adapt the ABRV framework to this setting, introducing a belief-state-based monitoring approach combined with a statistically grounded acceptance condition based on confidence intervals and relative standard deviation.
3. We illustrate and evaluate the methodology through a detailed example and a simplified case study in the context of planetary exploration, using the ROSEX simulation environment.

2 Related works

Several approaches have been proposed to validate the correctness of models at runtime, either focusing on AI components or dynamic systems.

ModelGuard validates predictions from black-box models using statistical sampling, assuming Lipschitz continuity to bound prediction errors and assess correctness probabilistically [24]. ModelPlex synthesizes runtime monitors from formally verified models of cyber-physical systems, checking whether system behavior conforms to verified safety guarantees [18]. Tripuramallu et al. [22] propose a runtime verification framework for cyber-physical systems with neural network controllers, where formal specifications in a variant of timed automata are used to detect deviations in system behavior during execution. However, these works do not deal with partial observability of the system.

Bayesian Verification introduces a probabilistic runtime verification framework that quantifies the confidence that a system satisfies its specification under model uncertainty [11]. This method focuses on prediction confidence but presumes full access to input variables at runtime.

A complementary line of work addresses runtime verification under partial observability. Könighofer et al. survey enforcement techniques for AI systems and discuss how uncertainty in inputs and models complicates runtime assurance [12]. Mannucci and de Oliveira Filho develop runtime monitors for reinforcement learning agents, identifying when learned policies may fail under partial information [16]. In contrast, our work adapts ABRV [6] to monitor prediction functions, combining belief state reasoning and assumption models to validate AI-generated outputs when inputs are not fully observable.

Neural Predictive Monitoring (NPM) under Partial Observability [3] shares our motivation to ensure the reliability of AI-based predictions. Its objective is predicting at runtime whether a future violation of a formally specified requirement (like a reachability property defined on a hybrid automaton) will occur within a given time horizon. To manage the inherent uncertainty of DNN-based predictors, NPM uses Conformal Prediction to accept or reject predictions. While both our work and NPM rely on statistical methods to quantify uncertainty of AI-based prediction, the focus differs. NPM aims to predict future property violations with high confidence. Instead, our approach is two-fold: we first define statistically grounded acceptance conditions to accept or discard predictions, we then evaluate the prediction's consistency against the system's runtime belief state and formal input assumptions.

Another important line of work is the development of confidence monitors and the Confidence Composition (CoCo) framework [20]. The core idea is that design-time guarantees often rely on verification assumptions that may be violated at runtime. The CoCo framework proposes building confidence monitors for each verification assumption and then logically composing their confidences to yield a system-wide assurance confidence in the overall safety guarantee. Our approach differs primarily in the target of monitoring: CoCo monitors the satisfaction of verification assumptions and their logical composition, whereas our method directly monitors the consistency of the prediction outcome itself against the current system state, as captured by the belief state.

3 Prediction with AI and Digital Twins in ExploDTwin

The ExploDTwin project, funded by the European Space Agency (ESA), is an initiative focused on developing a model-based digital twin for space exploration assets. The project aims to provide services such as simulation, monitoring, and automated planning support for remote exploration assets, like planetary rovers. ExploDTwin is an ongoing project, with Trasys serving as the prime contractor and Fondazione Bruno Kessler (FBK) participating with expertise in formal methods, planning, and machine learning.

A key concept in ExploDTwin is the use of a formal model of the system, written in a variant of SysML v2 and used to derive the specification of planning and monitoring problems, as well as to configure the digital twin framework for the specific asset. The formal models specify also "prediction functions", which are opaque entities, meaning their signature is known, but their actual implementation lies outside the model and can be realized by various means, including Machine Learning (ML) models or numerical simulators. These prediction functions are used to specify the parameters of functions that are difficult to capture with traditional modeling approaches, such as resource consumption (e.g., power or duration of an activity).

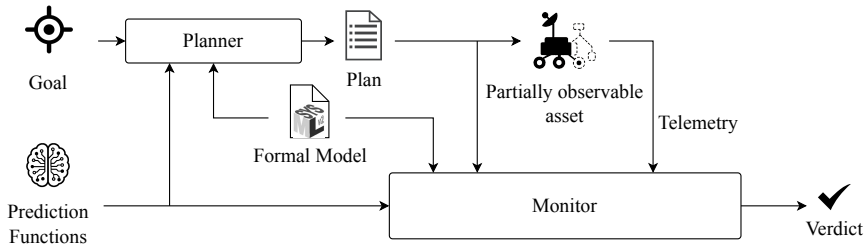

Fig. 1: Prediction function monitoring workflow: The prediction functions, which are used to generate plans, are validated by the monitoring component using telemetry data from a partially observable asset. Both Planner an Monitor are specialized on the use case by the SysML v2 model.

The prediction of these functions are crucial for services built on top of the digital twin. In particular, when planning a sequence of activities, these prediction functions are used for resource prediction (e.g., predicting the duration or power consumption of an activity). The project includes the ExoMars case study as an example, where activities like drilling require resource prediction. Figure 2 shows a simulation snapshot. For instance, an ML model is used to predict the power consumption of the `Drill` activity. This prediction function, for example, *get_ drill_ power_ consumption*, may take inputs such as the current drill depth, soil type, rotation speed, thrust, latitude, and longitude.

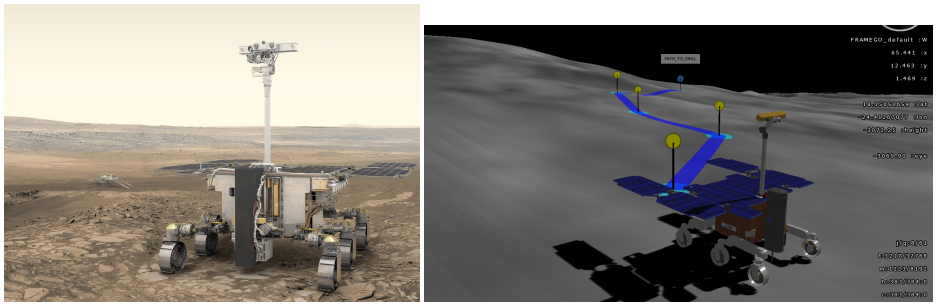

Fig. 2: Picture of the ExoMars rover and the simulator with a drilling plan

However, performing runtime verification of these predictions presents a significant challenge due to partial observability. At runtime, only a subset of system variables is directly observable, typically through telemetry data. This means that not all inputs required by the prediction functions (which were used during planning) are available based solely on the runtime observations.

4 Background

4.1 AI-Based Prediction and Uncertainty Measures

In this work, we use regression ML models for prediction. In such scenario, the task for the ML model is to approximate an unknown function $f : D^n \rightarrow D$ that relates the input vector X, collecting all the features and the settings of the system that is being studied, to the corresponding output y, representing the dependent variable of interest. Given a function error ϕ (for example the RMSE, root-mean-square error), the training phase of an ML model uses the input data X for *creating* the approximation function f_M that minimizes the reconstruction error $\phi(y, f_M(X))$. Among all the possible methods, in the proposed framework we decide to use CatBoost[19] (CB) a **gradient boosting on decision tree** as ML model. The main advantages of using CB with respect to other ML models are:

- native support for categorical data (e.g. integers and operational mode)
- native support for training the model on GPUs
- good performances in terms of prediction error and generalization capabilities (see [13, 9] and the benchmark session in the repository of [19])
- good performance in terms of execution time (see the benchmark session in the repository of [19])

Another important feature of CB is the possibility to train the model as a probabilistic regressor. In such configuration, the learned function f_M associates to each input sample x_i a distribution over the possible outcomes of the target variable, rather than a single point estimate y_i. CB, in particular, models

a gaussian distribution returning, for each point, a pair (μ_i, σ_i^2), representing the mean and the variance of the prediction. This is achieved by minimizing the `Negative Log-Likelihood (NLL)` loss introduced in [7, 15] and defined as $NLL = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{2}\log(2\pi) + \log(\sigma_i) + \frac{(y_i-\mu_i^2)}{2\sigma_i^2}\right)$

With respect to the standard and more interpretable function such as the `MSE`, the NLL provides additional insights about the quality of the estimated uncertainty over the samples.

Working with point-wise normal probability distribution enables the possibility to use the confidence interval (see [17] for more details) for determining the degree of the dispersion of the prediction and formalize better when the real and predicted value are close. The confidence interval is then defined as $[-z_{\alpha/2} \cdot \sigma, z_{\alpha/2} \cdot \sigma]$, where $z_{\alpha/2} \approx 1.96$ for a 95% confidence interval ($\alpha = 0.05$, see [17] for more details). The prediction is typically considered valid if it falls within such confidence interval, i.e., if $\left|y - \mu\right| < z_{\alpha/2} \cdot \sigma$.

4.2 Partially Observable Symbolic Transition Systems

To model system dynamics under partial observability, we adopt the framework of *partially observable symbolic transition systems*. This abstraction allows reasoning over hidden or uncertain variables during runtime verification.

Definition 1 (Symbolic Transition System). *A* symbolic transition system *is a tuple $\mathcal{S} = (V, D, I, T)$ where V is a finite set of state variables, D is the domain of the variables in V, I is a formula over V representing the set of initial states, T is a formula over $V \cup V'$ representing the transition relation, where V' denotes the next-state versions of the variables in V. A* state *s is an assignment of values from D to all variables in V. A* trace *is a sequence $s_0, s_1, \ldots, s_n$ of states such that $s_0 \models I$ and $(s_i, s_{i+1}) \models T$ for all $i < n$.*

Definition 2 (Partial Observability). *Let $O \subseteq V$ be the set of observable variables. A trace $s_0, s_1, \ldots, s_n$ yields an observation sequence $o_0, o_1, \ldots, o_n$ where $o_i = s_i|_O$ is the projection of state s_i onto the observable variables O.*

We say that a trace is compatible *with an observation sequence $o_0, \ldots, o_n$ if for each i, $s_i|_O = o_i$.*

In this framework, we reason not over a single state but over a *belief state*, which is a set of all states compatible with a given sequence of observations and the transition model. Belief state exploration enables us to consider all possible hidden assignments consistent with the observed telemetry and system dynamics.

4.3 Assumption Based Runtime Verification

Assumption-Based Runtime Verification (ABRV) extends classical runtime verification to systems with partial observability by reasoning over the set of traces that are compatible with both the observations and a formal assumption about the system behavior. The assumption is represented as a symbolic transition

system, and observations are expressed as sequences of valuations over a subset of the system's observable variables.

In ABRV, properties are expressed in Linear-time Temporal Logic (LTL). As we are not using LTL in this paper, we refer the reader to [6] for a full definition of ABRV with LTL. In ABRV, the interpretation of a formula φ over a finite sequence σ of observations is defined over four truth values.

Definition 3 (ABRV Monitor Semantics). *Let φ be an LTL formula over V, and let σ be a finite sequence of observations. Let $B(\sigma)$ denote the set of traces compatible with σ and the assumption $\mathcal{S}$. The ABRV monitor returns a verdict in the set $\mathbb{B}_4 = \{\mathsf{true}, \mathsf{false}, \mathsf{unknown}, \mathsf{outofmodel}\}$, defined as follows:*

- true *if all traces in $B(\sigma)$ satisfy φ;*
- false *if all traces in $B(\sigma)$ violate φ;*
- $\mathsf{unknown}$ *if $B(\sigma)$ contains both satisfying and violating traces;*
- $\mathsf{outofmodel}$ *if $B(\sigma) = \emptyset$, i.e., no trace is compatible with the observation sequence and the assumption.*

To compute this verdict online, the ABRV algorithm incrementally maintains a belief state—a set of possible states consistent with the assumption and the observation sequence. Starting from the initial set of states defined by the assumption, each new observation restricts the belief state by filtering out transitions that are not compatible with the observable behavior. The belief state thus represents the reachable executions of the system that remain consistent with both the observed trace and the assumption model.

5 Runtime Verification of Prediction

5.1 Definition of the Problem

In this section, we formalize the problem of monitoring prediction functions. A model of the system represents the assumption on how the system behaves, and it contains an external function associated to a prediction model. For simplicity, the model is a symbolic transition system as defined in Section 4 with one external function. In ExploDTwin, the model is specified in SysML v2 and is much more elaborated, with multiple components, state machines, types of variables, and prediction functions. We focus here on the foundational concepts of the monitoring problem, and thus we assume that the SysML v2 model has been translated into a flattened transition system. We also assume to have just one external function and that and all variables have value in a domain D, although in practice we handle multiple functions and types.

Acceptance condition Before defining the monitoring problem, we need to define the criterion to accept a prediction. As in standard model theory, we distinguish between a function symbol f of arity n and its interpretation $f_M : D^n \to D$ given by a model M. We distinguish between the ground truth interpretation

f_{GT} of f which is considered the real correct interpretation and the interpretation f_M given by a prediction model M that tries to approximate f_{GT}. Moreover, based on the discussion given in Section 4.1, we assume to have also the standard deviation $\sigma_M : D^n \to \mathbb{R}$ associated to f_M. Thus, we define a prediction model in this context as follows.

Definition 4 (Prediction model). *Given a function symbol f of arity n, a prediction model M is an interpretation of f that defines a function $f_M : D^n \to D$ that returns the predicted value and a function $\sigma_M : D^n \to \mathbb{R}$ that returns the associated standard deviation.*

Definition 5 (Acceptance condition). *Given a function symbol f and a prediction model M, an acceptance condition for f is a Boolean function $\mathsf{PredOK}_M : D^{n+1} \to \{\bot, \top\}$ that, given in input $\langle \overline{x}, y \rangle \in D^{n+1}$ such that $y = f_{GT}(\overline{x})$, decides if $f_M(\overline{x})$ is a good approximation of $f_{GT}(\overline{x})$.*

The most natural formulation of PredOK_M is reported in Equation 1.

$$\mathsf{PredOK}_M^{CI}(\overline{x}, y) \iff \left| y - f_M(\overline{x}) \right| < z_{\alpha/2} \cdot \sigma_M(\overline{x}) \tag{1}$$

In other words, we accept a prediction if it falls within the confidence interval of the predicted distribution.

However, a high variability associated to a lower confidence of M for a particular input $\overline{x}$ causes a large confidence interval and, consequently, a higher acceptance rate. For this, a second term is added to PredOK_M^{CI}, discarding prediction with a relative standard deviation too large, as can be read in Eq.2:

$$\mathsf{PredOK}_M(\overline{x}, y) \iff \left(\left| y - f_M(\overline{x}) \right| < z_{\alpha/2} \cdot \sigma_M(\overline{x}) \right) \wedge \left(\frac{\sigma_M(\overline{x})}{|f_M(\overline{x})| + \epsilon} < th \right) \tag{2}$$

for a given threshold th and a small value of $\epsilon > 0$ for avoiding division by 0.

Remark 1. Note that a similar control, to avoid excessively large confidence intervals, could have been achieved by imposing constraints on the absolute standard deviation of the predicted distribution. However, this requires previous knowledge about target-specific domain information that is not always available. Furthermore, relative standard deviation is meaningful for measurement with a natural zero such as resource consumption as in our target application. For different measurement, such as a temperature, a different condition must be defined.

Monitoring problem In the context of this paper, an assumption is a partially observable transition system with an external function and a constraint. The constraint defines the input/output variables of the function and a condition specifying when it holds.

Definition 6 (Assumption model). *An assumption model is a tuple $\langle V, S, O, f, \psi \rangle$ where V is a set of variables with domain D, S is a transition system over V,*

$O \subseteq V$ is the set of observable variables, $f : D^n \to D$ is a function of arity n on the domain D, and ψ is a constraint of the form $\alpha \to v = f(x_1, \ldots, x_n)$ where $v, x_1, \ldots, x_n$ are variables in V and α is a condition over the observable variables O.

Definition 7 (Monitoring of function under partial observability). *Given an assumption model $\langle V, S, O, f, \psi \rangle$ with constraint ψ defined as $\alpha \to v = f(x_1, \ldots, x_n)$, a prediction model M of f, and a sequence $o_1, \ldots, o_n$ of assignments to O, check if there exists a path $s_1, \ldots, s_n$ of S compatible with $o_1, \ldots, o_n$ such that, for all $i \in [1, n]$, if $s_i(O) \models \alpha$, then $\mathsf{PredOK}_M(s_i(x_1), \ldots, s_i(x_n), s_i(v))$.*

Remark 2. Note that in the above definitions the constraint $v = f(x_1, \ldots, x_n)$ relates input and output of the function in the same state. However, in practice it is often the case that the function must relate variables at different points in time. For example, the function may require as input the duration of a durative action. We assume that the assumption has sufficient history variables to be able to evaluate the condition in the current state. For example, a variable stores the timestamp of the state in which a durative action starts so that when the action ends, we can easily compute the duration.

Definition 8 (Verdict for monitor of function under partial observability). *Let $\langle V, S, O, f, \psi \rangle$ be an assumption model, with constraint ψ defined as $\alpha \to v = f(x_1, \ldots, x_n)$, and M a prediction model of f. Let $\sigma = o_1, \ldots, o_n$ be a sequence of assignments to O and $s_0, \ldots, s_n$ a trace compatible with σ. We define $\mathcal{B}_i$ as the set of all states s, such that s is the i^{th} is state of a trace compatible with the current observation sequence. Then $Verdict(\mathcal{B}_i)$ is one of the following values.*

- consistent *if there exists $s \in \mathcal{B}_i$ s.t.* $\mathsf{PredOK}_M(s(x_1), \ldots, s(x_n), s(v)) = \top$.
- valid *if for all $s \in \mathcal{B}_i$ s.t.* $\mathsf{PredOK}_M(s(x_1), \ldots, s(x_n), s(v)) = \top$.
- error *otherwise.*

Remark 3. An error verdict always requires a corrective action (for example, retraining the prediction model with updated ground truth), while a valid verdict requires none. The interpretation of a consistent verdict depends on the use case. A consistent verdict may be acceptable in a system with high uncertainty, but it may raise an alarm when the belief state has low variability.

Example We illustrate our monitoring problem with a simple example focused on power consumption prediction during a drilling activity. Let $f(h)$ represent the estimation of such power consumption as a function of the hidden soil hardness h. For the sake of this example, we set $h \in \{1, 2, 3\}$, low, medium and high hardness, respectively. The prediction model M returns both the predicted mean and an associated standard deviation $\sigma(h)$. Assume: $f_M(1) = 100, \sigma_M(1) = 3, f_M(2) = 120, \sigma_M(2) = 8, f_M(3) = 140, \sigma_M(3) = 25$.

The assumption has as initial condition $h_0 = 1$ and as transition condition the formula: $T(h, h') = (h' = h - 1 \wedge h > 1) \vee (h' = h) \vee (h' = h + 1 \wedge h < 3)$,

Algorithm 1 Prediction monitoring

```
1: procedure MonitorPrediction
2:     while True do
3:         obs ← GetObservation
4:         belief_state ← Monitor(obs)
5:         if α(obs) then
6:             results ← []
7:             for all estimated_state in belief_state do
8:                 results ← results+ PredOK_M(estimated_state)
9:             end for
10:            verdict ← Verdict(results)
11:            NotifyVerdict(verdict)
12:        end if
13:    end while
14: end procedure
15: function Verdict(results)
16:    if ⊥ ∉ results then
17:        return valid
18:    else if ⊤ ∈ results then
19:        return consistent
20:    else
21:        return error
22:    end if
23: end function
```

specifying that the hardness can change by at most one unit per step. The assumption's constraint on f is $\top \rightarrow y = f(h)$.

Now consider the following observed sequence of power consumption values: at time 0, the observed power y is 99, while at time 1 y is 143.

At $t = 0$, the belief state is: $\mathcal{B}_0 = \{h_0 = 1\}$. We evaluate: $f_M(1) = 100$, $\sigma_M(1) = 3$, CI $= [94.12,\ 105.88]$, RSD $= \frac{3}{101} \approx 0.0297$. Since $99 \in$ CI and RSD < 0.15, the prediction is accepted: $\mathsf{PredOK}(1, 99) = \top$.

At $t = 1$, based on the transition model, the belief state becomes: $\mathcal{B}_1 = \{h_1 = 1,\ h_1 = 2\}$. Evaluate each:

- $h = 1$: $f = 100$, $\sigma = 3$, CI $= [94.12,\ 105.88] \Rightarrow 143 \notin$ CI $\Rightarrow \mathsf{PredOK}(1, 143) = \bot$
- $h = 2$: $f = 120$, $\sigma = 8$, CI $= [104.32,\ 135.68]$
  RSD $= 8/121 \approx 0.066$
  $143 \notin$ CI $\Leftrightarrow \mathsf{PredOK}(2, 143) = \bot$

No value in the belief state satisfies the predicate. Therefore, the monitor returns: $\mathsf{Verdict}(\mathcal{B}_1) = \mathsf{error}$.

5.2 Solution based on ABRV

Algorithm 1 presents a possible loop responsible for monitoring and evaluating predictions based on incoming observations. Upon receiving a new observation,

the algorithm updates its internal belief state using the MONITOR function. This function is an instance of an ABRV monitor, which leverages a set of predefined assumptions to infer the current belief state, i.e., the set of states reachable with path compatible with the observations processed so far.

If the current observation satisfies the condition guard α, the algorithm proceeds to evaluate the prediction. This is done by applying the acceptance condition for the prediction model to each estimated state in the belief state.

The results of these evaluations are aggregated and passed to the VERDICT function, which classifies the overall outcome into one of three categories, as per Definition 8. The verdict can then be communicated through a notification mechanism, enabling further actions or logging.

We implemented the algorithm on top of the NuRV tool [5], a framework for ABRV that supports monitoring under partial observability. NuRV provides the core infrastructure for belief state tracking with respect to a symbolic transition system representing the system assumption, specified in the modelling language SMV [4]. The monitoring loop from Algorithm 1 was implemented in Python and interacts with NuRV via a Python wrapper of the NuRV library and calls directly the prediction models.

6 Experimental Evaluation

In order to demonstrate the applicability and effectiveness of the proposed approach, we evaluated the implementation on the ExoMars rover module described in Section 3. The code, models, and data to reproduce the results are available at https://es-static.fbk.eu/people/tonetta/papers/easi25.tar.gz.

6.1 ExoMars Case Study

Activities and prediction functions The prediction functions are used to plan a sequence of activities, which are the operations to be executed by the rover. In the context of our use case, we focus on predicting some characteristics of two rover activities: `RVWakeup` and `Drill`. The `RVWakeup` activity is responsible for powering up the rover, a process whose duration is dependent on environmental factors such as the ambient temperature. The `Drill` activity, on the other hand, commands the rover drill to penetrate the Martian soil to collect samples for subsequent analysis. This operation is energy-intensive, and its power consumption is significantly influenced by environmental variables, with soil type being the most critical. In this work, we focus on monitoring different prediction functions: the duration of the `RVWakeup` activity, the energy consumption and duration of the `Drill` activity.

We use two simulators to mimic the actual system and generate the traces we monitor: the ROSEX simulator, and an SMV model. The ROSEX simulator provides a more realistic and complex simulation environment, while the SMV model can be used to generate traces of arbitrary length with the nuXmv model checker [4].

Assumptions The assumption used for ABRV is another SMV model that describes the current location of the rover and the hardness of the terrain. More in detail, the rover is located in a 10km^2 grid, where each cell is associated with a terrain hardness level (or soil type). We observe the movements of the rover, and the assumption infers a set of possible discrete locations, and the corresponding soil types.

Machine Learning Models In order to provide the data necessary for the training of the ML models, the ROSEX simulator was used. In particular, extensive data generation was performed related to the `Drill` and the `RVWakeup` activities. The process of execution of a specific activity by the ROSEX simulator depends on two main sets of parameters: the initial conditions, which represent the set of initial states of the rover, and the activity parameters, if they are required. Therefore, a total of 756000 experiments were produced by spacing some combinations of initial conditions and parameters for the `Drill` activity, and 1155 combinations of only initial conditions for `RVWakeup`, which does not include activity parameters. The initial conditions describe the coordinates (`SYS_ROVER_LATITUDE`, `SYS_ROVER_LONGITUDE`), the timestamp (`SYS_SIMSTART_UTC`) and the initial position of the drill (`DRILL_ROD_Q`). The `Drill` activity parameters include the final position of the drill (`final_pos`), the `thrust`, the rotation speed (`rot_speed`) and the expected soil type category (`soil_type`). From the logs produced by the simulator, the two target labels of interest are extracted. The power consumption is computed by performing the cumulative sum of the instantaneous power of the `DrillMamissSPDS` subsystem for the `Drill` activity, and of all the subsystem for the `RVWakeup` activity; the duration label is extracted in both activities by looking at the elapsed time reported in the logs of the `Controller`.

The aforementioned parameters are employed as inputs to the ML models. A preprocessing stage is included, which consists in a pipeline that:

1. transforms timestamp and coordinates into *cyclic* features by computing their sine/cosine;
2. extracts *mars-related* features from coordinates and timestamp, including local true solar time, heliocentric distance, solar radiation, time to sunrise and sunset, and solar elevation [1, 2];
3. performs a min-max scaling of the continuous inputs and outputs.

For assessing the capability of the PredOK function (in Eq.2) of detecting good predictions, three models for the power consumption of the `Drill` activity have been trained using different levels of data knowledge:

- standard set (SS): all the data are randomly split into train (80%) and test (20%) sets; the training set is further divided into train (80%) and validation (20%) splits;
- unknown soil type (S5): the test set contains all the samples with `soil_type` = 5, the remaining data is randomly split into train (80%) and validation (20%) sets;

– unknown final position range (FP): the test set contains all the samples in the range $0 \leq$ `final_pos` ≤ 0.2; the remaining data is randomly split into train (80%) and validation (20%) sets.

The first set (SS) reproduces a standard situation in which a representative dataset have been collected. This serves as a benchmark, in order to verify if the uncertainty measures on the test set are reasonably low. The remaining sets (S5 and FP) represent two critical situations in which the models are employed to perform the inference on a space of parameters which was not explored during the training phase.

For each dataset, a `CatBoostRegressor` has been trained in a probabilistic regression setup to minimize the `RMSEWithUncertainty` (see Sec. 4) loss function. The object was initialized with the default parameters except for the depth of the trees, for which the value 8 represented a good trade off between computational performance and final accuracy. The training loop, lasting at most 10000 steps, implemented an early stopping rule, ending the model updates after 800 iterations without improves in the validation loss.

6.2 Scalability evaluation

In this section we evaluate the scalability of the approach by increasing the size of the belief states. We prepared four test configurations, using traces generated via the SMV model of the rover. We focused on predicting the duration and power consumption of the `Drill` activity. The first two configurations use one prediction function that is run sequentially. We estimate the duration of the `Drill` activity, and both duration and power consumption for the remaining configurations, via either an algorithmic procedure or the `Drill` ML model. Predictions of power and duration of the same `Drill` activity are computed using the same belief state.

Each configuration is run against the same traces generated from the SMV model. The traces portray the following scenario: the rover moves a certain number of steps, drills, moves again the same number of steps, drills again. Traces differ in the parameters of the activities, and in the number of steps the rover moves before performing them. Moving before starting an activity allow us to generate more belief states, and of increasing size.

Figure 3 and 4 plot the size of the belief state against the time taken to compute a prediction for each state, and compares the approach that uses algorithmic procedures against ML models. The plots are stacked to show the decomposition of the monitoring algorithm, taking into account only telemetries that require to perform a prediction. The top-plot shows the overall time the algorithm takes to compute a prediction for each of the enumerated states. The middle one plots only the time needed to enumerate the states. The third row shows the time required to compute the prediction, for each state collected during the step above. The left column of both figures displays data generated by the algorithmic procedure, while the right column shows data from the ML model.

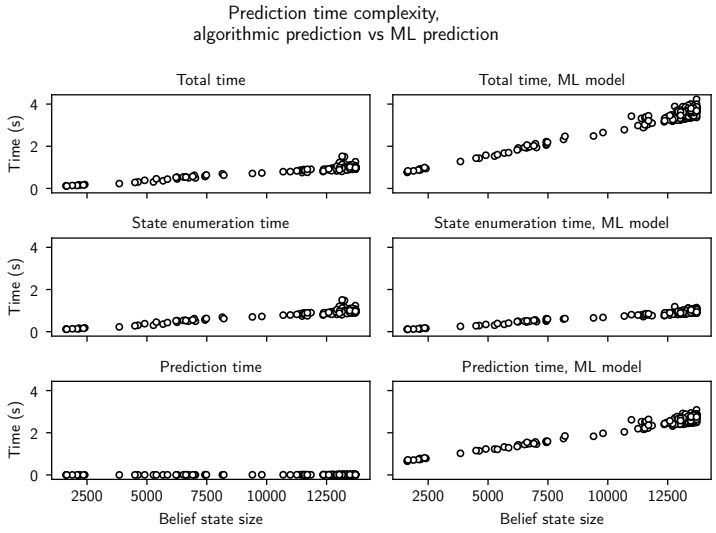

Fig. 3: Runtime prediction monitoring algorithm execution time against size of belief state, comparing an algorithmic procedure against an ML model as prediction.

Let us focus on Figure 3, where we predict sequentially the duration of `Drill`. The plot shows that time complexity grows approximately in a linear fashion, and it depends on the cardinality of the belief state.

In Figure 4, we predict duration and power consumption of `Drill`. The state enumeration is performed only once, and then we compute both predictions.

It takes roughly double the time to compute the same number of states, while obviously it takes the same time to compute the state enumeration.

The results confirm that the performance of the algorithm depends on the size of the belief state, whose time-complexity grows linearly along with the number of states. Then, the prediction function adds to runtime, and it may become the dominant factor.

6.3 Error detection effectiveness evaluation

In this section, we evaluate the effectiveness of the approach in detecting a wrong prediction. We use the ROSEX simulator to create a more realistic setting, and we show the importance of choosing a good correctness function to monitor the prediction function.

We run a simulation using the ROSEX and different prediction models. The simulation closely mirrors the ExoMars rover module's typical Martian day: it wakes up, via the `RVWakeup` activity, charges its batteries via solar panels, travels to a scientifically relevant location, takes terrain measurements with its sensors, then drills, via the `Drill` activity, and collects a sample. An ML model predicts the duration of `RVWakeup`. We use models SS, S5, FP to monitor the power prediction of `Drill`. The models are described in greater details in Section 6.1.

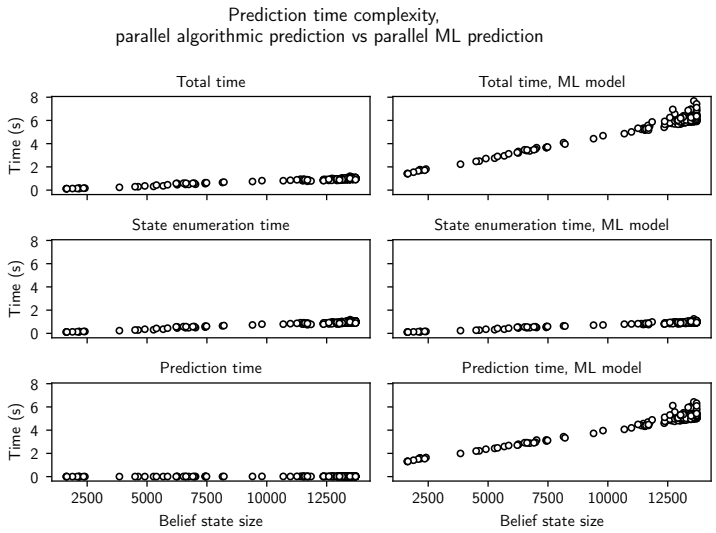

Fig. 4: Time of runtime prediction monitoring algorithm against size of belief state, running predictions in parallel, comparing an algorithmic procedure against an ML model as prediction.

Finally, we compare the outputs of different correctness implementations:

- $\mathsf{PredOK}^{rel}_M(\overline{x}, y) = \|f(x) - y\| \leq 0.1$;
- PredOK^{CI}_M is defined in Equation 1;
- PredOK^{T}_M is defined as Equation 2 without the term from PredOK^{CI}_M;
- PredOK_M is defined in Equation 2.

Table 1 shows the results obtained: each row gathers the result of a prediction function, and for each of them we show how many states are correct. Note the difference in belief state size between `RVWakeup` and `Drill` related predictions. The `RVWakeup` duration prediction is performed at the start of the simulation, when our assumption identifies fewer states. `Drill` predictions use the same belief state, computed later in the simulation, when due to uncertainty in the movement of the rover, we have more states.

Prediction	Feature	PredOK^{rel}_M	PredOK^{CI}_M	PredOK^{T}_M	PredOK_M	# states
RVWakeup	time	2	0	2	0	2
Drill SS	time	10	0	266	0	266
Drill SS	power	10	10	132	10	266
Drill S5	power	0	0	266	0	266
Drill FP	power	10	10	52	0	266

Table 1: Summary of the prediction correctness of different prediction functions, using different acceptance conditions.

According to correctness function PredOK_M^{rel}, every model but S5 outputs a consistent verdict. Similar characteristics are exhibited by 10 out of the 266 states in the belief state. Model SS performs poorly when evaluated using PredOK_M^T and PredOK_M: each of the 10 predictions previously considered correct have high variance.

Regarding models S5 and FP, the predictions exhibit low variance. However, these predictions are subsequently discarded by the confidence interval validation step. These results show the definition of PredOK_M provides expected verdicts.

Predictions that are considered correct by PredOK_M^{Rel} are instead discarded due to the high variance. This leads us to question the quality of the model. High variance may be attributed to high variability in the training data or to other underlying factors. The user should perform further analysis of the model outputs to determine the cause and implement appropriate corrective measures.

7 Conclusions and Future Works

This paper formalizes the problem of runtime verification of prediction functions under partial observability. We focus on the setting in which a prediction model is deployed in a system whose relevant internal state is not fully observable. The core challenge lies in determining whether the observed outcomes are consistent with the predictions, despite uncertainty over the model's inputs. To address this, we adapt the ABRV framework to the monitoring of prediction functions. The approach uses an assumption, encoded as a symbolic transition system, to track a belief state representing all possible input states consistent with observations. At runtime, the prediction is evaluated over the belief state using an acceptance criterion based on confidence intervals and relative standard deviation.

A known limitation of the proposed approach is its reliance on a probabilistic model capable of providing a confidence estimate for each prediction. Nevertheless, uncertainty quantification remains an active and rapidly evolving research area, with numerous new methods and techniques emerging in recent years [10]. We demonstrate the method in the context of digital twins for space exploration, using the ROSEX simulator and abstract examples.

A number of directions will be explored in the future. First, we aim to support richer forms of specification for prediction correctness, such as prediction over temporally extended inputs. Second, belief state computation will be extended beyond finite domains, using abstraction or probabilistic reasoning. Finally, we intend to integrate the monitor within the digital twin framework of ExploDTwin and integrate the retraining of the models based on the monitoring feedback.

References

1. Allison, M.: Accurate analytic representations of solar time and seasons on mars with applications to the pathfinder/surveyor missions. Geophysical Research Letters **24**(16), 1967–1970 (1997)

2. Allison, M., McEwen, M.: A post-pathfinder evaluation of areocentric solar coordinates with improved timing recipes for mars seasonal/diurnal climate studies. Planetary and Space Science **48**(2-3), 215–235 (2000)
3. Cairoli, F., Bortolussi, L., Paoletti, N.: Neural Predictive Monitoring Under Partial Observability. In: RV. Lecture Notes in Computer Science, vol. 12974, pp. 121–141. Springer (2021)
4. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: CAV. pp. 334–342 (2014)
5. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based runtime verification with partial observability and resets. In: Finkbeiner, B., Mariani, L. (eds.) Runtime Verification. pp. 165–184. Springer International Publishing, Cham (2019)
6. Cimatti, A., Tian, C., Tonetta, S.: Assumption-based Runtime Verification. Formal Methods Syst. Des. **60**(2), 277–324 (2022)
7. Duan, T., Avati, A., Ding, D.Y., Thai, K.K., Basu, S., Ng, A.Y., Schuler, A.: Ngboost: Natural gradient boosting for probabilistic prediction (2019). https://doi.org/10.48550/ARXIV.1910.03225, https://arxiv.org/abs/1910.03225
8. Flori, A., Fonda, T., Gobbi, A., Gobbi, S., Kapellos, K., Micheli, A., Tonetta, S., Valentini, A., Ntagiou, E.: Planning and Scheduling with External Functions in the ExploDTwin project. In: International Workshop on Planning & Scheduling for Space (IWPSS) (2025)
9. Hancock, J., Khoshgoftaar, T.M.: Performance of catboost and xgboost in medicare fraud detection. In: 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA). p. 572–579. IEEE (Dec 2020). https://doi.org/10.1109/icmla51294.2020.00095, http://dx.doi.org/10.1109/ICMLA51294.2020.00095
10. He, W., Jiang, Z., Xiao, T., Xu, Z., Li, Y.: A survey on uncertainty quantification methods for deep learning (2025), https://arxiv.org/abs/2302.13425
11. Jha, S., Seshia, S.A., Sadraddini, S., Murray, R.M., Agha-Mohammadi, A.A.: A bayesian approach to model uncertainty for runtime verification. Formal Methods in System Design **54**(2), 240–264 (2019)
12. Könighofer, B., Bloem, R., Ehlers, R., Pek, C.: Correct-by-Construction Runtime Enforcement in AI - A Survey. In: Principles of Systems Design. Lecture Notes in Computer Science, vol. 13660, pp. 650–663. Springer (2022)
13. Kumar, G.S., Dhanalakshmi, R.: Performance analysis of catboost algorithm and xgboost algorithm for prediction of co2emission rating. In: 2023 6th International Conference on Contemporary Computing and Informatics (IC3I). p. 1497–1501. IEEE (Sep 2023). https://doi.org/10.1109/ic3i59117.2023.10398160, http://dx.doi.org/10.1109/IC3I59117.2023.10398160
14. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Methods Program. **78**(5), 293–303 (2009)
15. Malinin, A., Prokhorenkova, L., Ustimenko, A.: Uncertainty in gradient boosting via ensembles (2020). https://doi.org/10.48550/ARXIV.2006.10562, https://arxiv.org/abs/2006.10562
16. Mannucci, T., de Oliveira Filho, J.: Runtime Verification of Learning Properties for Reinforcement Learning Algorithms. In: FMAS@iFM. EPTCS, vol. 395, pp. 205–219 (2023)
17. Moore, D.S., Notz, W.I., Fligner, M.A.: The basic practice of statistics. W.H. Freeman, New York, NY, 6 edn. (2012)
18. Platzer, A.: Modelplex: Verified runtime validation of verified cyber-physical system models. Formal Methods in System Design **49**(1-2), 33–74 (2016)

19. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: Catboost: unbiased boosting with categorical features. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. p. 6639–6649. NIPS'18, Curran Associates Inc., Red Hook, NY, USA (2018)
20. Ruchkin, I., Cleaveland, M., Ivanov, R., Lu, P., Carpenter, T.J., Sokolsky, O., Lee, I.: Confidence Composition for Monitors of Verification Assumptions. In: ICCPS. pp. 1–12. IEEE (2022)
21. Stefano Tonetta: A formal model for space exploration digital twins. URL: https://sites.google.com/view/dtw25/invited-speakers/stefano-tonetta (2025)
22. Tripuramallu, D., Anand, A., Pinisetty, S., Pearce, H., Roop, P.S.: Runtime Verified Neural Networks for Cyber-Physical Systems. In: VORTEX@ISSTA. pp. 44–51. ACM (2024)
23. Urban, C., Miné, A.: A Review of Formal Methods applied to Machine Learning. CoRR **abs/2104.02466** (2021)
24. Zhao, Y., Nikolaidis, S.: Modelguard: Probabilistic verification of predictive models in robot planning. In: RSS (2021)