Predicting Nonfunctional Requirement Violations in Autonomous Systems

XINWEI FANG, SINEM GETIR YAMAN, and RADU CALINESCU, Department of Computer Science, University of York, UK
JULIE WILSON, Department of Mathematics, University of York, UK
COLIN PATERSON, Department of Computer Science, University of York, UK

Autonomous systems are often used in applications where environmental and internal changes may lead to requirement violations. Adapting to these changes proactively, i.e., before the violations occur, is preferable to recovering from the failures that may be caused by such violations. However, proactive adaptation needs methods for predicting requirement violations timely, accurately, and with acceptable overheads. To address this need, we present a method that allows autonomous systems to predict violations of performance, dependability and other nonfunctional requirements, and therefore take preventative measures to avoid or otherwise mitigate them. Our method for predicting these autonomous system disruptions (PRESTO) comprises a design time stage and a run-time stage. At design-time, we use parametric model checking to obtain algebraic expressions that formalise the relationships between the nonfunctional properties of the requirements of interest (e.g., reliability, response time, and energy use) and the parameters of the system and its environment. At run-time, we predict future changes in these parameters by applying piece-wise linear regression to online data obtained through monitoring, and we use the algebraic expressions to predict the impact of these changes on the system requirements. We demonstrate the application of PRESTO through simulation in case studies from two different domains.

CCS Concepts: • **Software and its engineering → Software notations and tools**; • **Computing methodologies → Modeling and simulation**; • **Theory of computation → Formal languages and automata theory;**

Additional Key Words and Phrases: Proactive adaptation, runtime verification, prediction of system behaviour, reliability

ACM Reference format:
Xinwei Fang, Sinem Getir Yaman, Radu Calinescu, Julie Wilson, and Colin Paterson. 2024. Predicting Nonfunctional Requirement Violations in Autonomous Systems. *ACM Trans. Autonom. Adapt. Syst.* 19, 1, Article 6 (February 2024), 25 pages.
https://doi.org/10.1145/3632405

This project is funded by the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'.
Authors' addresses: X. Fang, S. Getir Yaman, R. Calinescu, and C. Paterson, Department of Computer Science, University of York, Deramore Ln, Heslington, York, UK, YO10 5GH; e-mails: {xinwei.fang, sinem.getir.yaman, radu.calinescu, colin.paterson}@york.ac.uk; J. Wilson, Department of Mathematics, University of York, Heslington, York, UK, YO10 5DD; e-mail: julie.wilson@york.ac.uk.

1556-4665/2024/02-ART6 $15.00
https://doi.org/10.1145/3632405

1 INTRODUCTION

Autonomous systems must operate resiliently in environments where change is commonplace. To that end, they need to adapt and evolve in order to mitigate the *disruptions* (i.e., the requirement violations) that may be caused by changes in their internal and environmental parameters. Ideally, this self-adaptation should be *proactive*, that is, the system should respond to potential disruptions before they happen [24]. Proactive adaptation is particularly beneficial in circumstances where its execution is associated with high cost [58] or latency [53]: prevention is better than cure.

Despite its clear benefits, proactive adaptation is currently underrepresented in the repertoire of self-adaptative solutions for autonomous systems. This is largely due to the difficulty of predicting potential disruptions: (i) accurately enough, so that the gains of avoiding requirement violations thanks to true positives outweigh the unnecessary cost of responding to false positives; and (ii) efficiently at run-time, so that these solutions can be used with acceptable overheads [53, 67]. Previous solutions have assumed that disruption occurs when one or more monitored parameters exceed a predetermined threshold. e.g., [1, 47]. However, the impact of such threshold violations on system-level properties is not considered. For autonomous systems, complex nonlinear behaviours mean that large changes in some parameters may be innocuous, while small changes in others may lead to significant disruptions [38]. Therefore, ineffective adaptations may occur when the impact of these changes is disregarded. What the autonomous systems used in many applications need are proactive adaptation mechanisms that can operate effectively and robustly, and that can be configured for deployment in a wide range of adaptation scenarios.

To help address this need for system-level proactive adaptation solutions, we introduce a method for **predicting autonomous system disruptions** (**PRESTO**) associated with the violation of performance, dependability and other nonfunctional requirements. These requirements specify quantitative bounds for nonfunctional system properties. For example, we may require that a robot used in autonomous farming shall "complete the picking of a fruit successfully with the probability of at least 0.8" or, for an assistive-care robot tasked with dressing a frail user, we may require "the expected time for the robot to complete the dressing shall not exceed 38 seconds".

PRESTO is intended for use during the monitoring and analysis steps of the **Monitor-Analyse-Plan-Execute over a shared Knowledge** (**MAPE-K**) feedback control loop [3, 7] of autonomous systems. The method predicts disruptions caused by gradual degradation in the system and/or the environment characteristics[1] through a process comprising a design-time stage and a run-time stage. In the former stage, PRESTO uses parametric model checking [22, 31] to derive algebraic expressions that formalise the relationship between the nonfunctional properties from the system requirements (i.e., the probability of successful fruit picking, and the expected dressing time for the earlier examples) and the relevant parameters of the system and its environment. In the latter stage, PRESTO uses online data obtained from monitoring the relevant system and environmental parameters, first to predict the future evolution of these parameters, and then to establish whether and when the predicted parameter changes will lead to requirement violations.

The main contributions of our article include:

(1) The integration of parametric model checking and time-series analysis to provide an end-to-end framework that allows the accurate prediction of autonomous system disruptions at run-time.

[1]Complementary methods are available for the detection of *sudden* changes that affect autonomous systems, e.g., [28, 70]; predicting such changes remains the subject of future research.

(2) The introduction of a piece-wise linear regression technique that supports the prediction of nonlinear changes in system and environmental parameters.
(3) The evaluation of PRESTO in case studies from two different application domains, which shows that our disruption prediction method can deliver significant reductions in system adaptation costs across a range of scenarios.

These contributions extend our preliminary PRESTO research [32] in several important ways. First, we extend the PRESTO theoretical foundation of our method with a definition of the applicable disruption prediction problem. We provide a formal description of the PRESTO algorithm for the online prediction of nonfunctional requirement violations, and with an analysis of the complexity of this algorithm. Second, we extend the applicability of PRESTO by replacing the simple linear regression employed by the work-in-progress PRESTO variant from [32] with piece-wise linear regression to handle nonlinear parameter changes. Finally, we evaluate PRESTO extensively, with the inclusion of a new case study from the robotic assistive-care domain, and we provide an analysis of threats to the validity for our disruption prediction method.

The remainder of the article is structured as follows. Section 2 introduces key concepts and notation associated with the techniques used within the PRESTO framework: parametric model checking and piece-wise linear regression. Section 3 presents the two stages of the PRESTO method. Section 4 details the case studies, the research questions answered and the experiments used to evaluate PRESTO. This section also discusses threats to validity. We conclude the article with a comparison of PRESTO to related research in Section 5 and a brief summary in Section 6.

2 PRELIMINARIES

2.1 Parametric Model Checking

A **discrete-time Markov chain** (**DTMC**) is a stochastic process comprising a set of states S associated with relevant configurations of the system under analysis, together with probabilities $P(s, s')$ that model the transitions between the system configurations associated with every pair of states $s, s' \in S$, where $\sum_{s' \in S} P(s, s') = 1$ for any $s \in S$. To extend the range of nonfunctional properties that can be analysed using this modelling paradigm, a DTMC can be annotated with **reward structures** $rwd : S \to \mathbb{R}_{>0}$ that specify the values of attributes such as execution time, cost and resource use for the system configurations corresponding to different DTMC states. A **parametric discrete-time Markov chain** (**pDTMC**) is a discrete-time Markov chain in which some or all of the transition probabilities and/or rewards are unknown. These unknowns correspond to parameters of the modelled system and its environment.

Key to the run-time application of PRESTO, **parametric model checking** (**PMC**) [11, 22, 31, 40, 44] is a mathematical technique for the analysis of pDTMCs with properties specified as **probabilistic computation tree logic** (**PCTL**) [6, 20, 41] formulae, potentially extended with rewards [2]. For instance, the probabilistic reachability formula $\mathcal{P}_{\geq 0.8}[\text{F success}]$ and the reward reachability formula $\mathcal{R}^{\text{time}}_{\leq 13}[\text{F done}]$ can be used to formalise the sample requirements from the previous section, i.e., that a robot completes its task successfully with probability of at least 0.8 and that the mission is performed within at most 13s, respectively. For detailed descriptions of the PCTL semantics, see [2, 6, 41].

Probabilistic model checkers, such as PRISM [45] and Storm [26], allow the verification of PCTL-encoded requirements over DTMCs. These tools also support the analysis of quantitative PCTL formulae in which the bounds from the probabilistic operator $\mathcal{P}$ and the reward operator $\mathcal{R}$ are replaced by "=?", e.g., $\mathcal{P}_{=?}[\text{F success}]$ and $\mathcal{R}^{\text{time}}_{=?}[\text{F done}]$. These formulae can be analysed over a DTMC, yielding a numerical value, or over a pDTMC to provide a *PMC expression*,

i.e., a rational function[2] over the pDTMC parameters. In this way, PMC allows the formalisation of the nonfunctional properties of a system as functions that be evaluated at run-time, when the system and environment parameters associated with the pDTMC unknowns are observed or predicted.

2.2 Piece-wise Linear Regression

Linear regression is a modelling technique that fits a linear equation of the form of $value = at+b$, relating a response or dependent variable ($value$) with an independent variable (t) [50]. The linear equation for a set of n data points $\{(value_1, t_1), (value_2, t_2), \ldots, (value_n, t_n)\}$ can be obtained by minimising the sum of squared residuals between the predicted and actual responses:

$$SSR = \sum_{i=1}^{n} (value_i - (a \cdot t_i + b))^2. \tag{1}$$

Given the *slope* a and the *intercept* b, predictions for the response $value$ can be obtained for new values of t with the assumption of continued linearity. **Piece-wise linear modelling** does not assume the same slope and intercept throughout the time series but allows different linear trends to be fitted to different sections. An appropriate linear trend should result in both negative and positive residuals, so the fact that residuals have the same sign over a predetermined time period indicates a poor fit to the data. Thus a threshold on the number of consecutive data points with residuals having the same sign can be used to trigger an update of the model. Predictions are then made from the current linear equation.

3 APPROACH

3.1 Problem Definition

PRESTO aims at predicting the occurrence of nonfunctional requirement violations for an autonomous system in order to trigger early adaptation and avoid the actual violations. To produce its predictions, PRESTO assumes that the following inputs are available:

(1) a pDTMC that models the behaviour of the system and its operating environment, and has $n \geq 1$ parameters $x_1, x_2, \ldots, x_n$;
(2) $K \geq 1$ PCTL-encoded nonfunctional system requirements $\Phi_k \bowtie_k bound_k$, $1 \leq k \leq K$, where, for each k, Φ_k is a quantitative PCTL formula, $\bowtie_k \in \{<, \leq, \geq, >\}$ is a relational operator, and $bound_k \in \mathbb{R}_{\geq 0}$;
(3) time series for each of the n pDTMC parameters, $\langle x_{i1}, x_{i2}, \ldots \rangle_{1 \leq i \leq n}$ comprising run-time observations $x_{ij} = (value_{ij}, t_{ij})$, $j = 1, 2, \ldots$, where $value_{ij}$ represents the value of parameter x_i measured at time t_{ij}, and $t_{ij} > t_{i,j-1}$ for all $j > 0$.

PRESTO further assumes that the pDTMC model and K requirements are available prior to the deployment of the autonomous system, and that the run-time observations x_{ij} are available one at a time, as soon as a new parameter measurement is obtained by the autonomous system through monitoring. Under these assumptions, PRESTO does the following after each new observation x_{ij}:

(1) uses the $N > 0$ most recent observations from each data stream to predict the times $violationTime_k$, $1 \leq k \leq K$ when each of the K requirements will be violated, with $violationTime_k = \infty$ used to indicate that the kth requirement will never be violated for the current parameter evolution trends;

[2] A rational function is an algebraic fraction whose numerator and denominator are both polynomials.

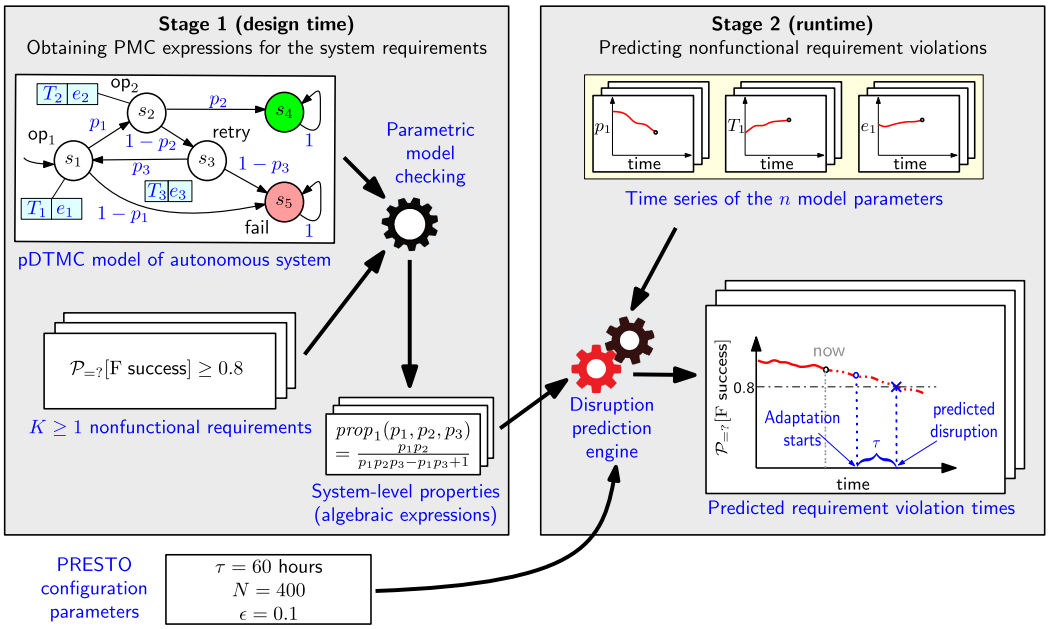


Fig. 1. Diagrammatic summary of the two-stage PRESTO prediction of nonfunctional requirement violations.

(2) triggers an autonomous system adaptation/repair if the predicted $violationTime_k$ for any of the K requirements is within $\tau > 0$ time units from the current time (with a single adaptation/repair triggered if multiple requirements are violated due to the new parameter value).

Any actual adaptations/repairs triggered by PRESTO prediction of requirement violations are outside the scope of this article. For a wide range of such adaptations techniques, see [18, 23, 25]. Furthermore, PRESTO is not dealing with the possibility that interfering triggers may lead to conflicts in adaptation goals, which represents an open challenge according to recent research into the "uncertainty interaction problem" that affects self-adaptive systems [14, 16].

The two stages of PRESTO are depicted in Figure 1 and detailed in the following sections.

3.2 Design-time Stage

In the first stage, PRESTO applies parametric model checking to the pDTMC model of the autonomous system and the PCTL formulae Φ_k, $1 \leq k \leq K$, from the nonfunctional requirements under verification. This parametric model checking is performed using a probabilistic model checker, such as PRISM [45] or Storm [26], and yields a set of rational functions

$$prop_k(x_1, x_2, \ldots, x_n) = \frac{P_k(x_1, x_2, \ldots, x_n)}{Q_k(x_1, x_2, \ldots, x_n)}, \; 1 \leq k \leq K, \tag{2}$$

that formalise the relationship between the system properties from the K requirements and the n pDTMC parameters, where $P_k(\cdot)$ and $Q_k(\cdot)$ are polynomials.

The left-hand side of Figure 1 illustrates this PRESTO stage for a toy example in which the workflow performed by a simple system is modelled by a five-state pDTMC. In the initial pDTMC state (s_1), the system performs an operation op_1 that succeeds with probability p_1, taking the system to state s_2, where a second operation, op_2, is attempted. This operation succeeds with probability p_2,

bringing the system to a success final state s_4. If operation op_1 fails (which happens with probability $1 - p_1$) the workflow is abandoned and the system ends in the final fail state s_5. In contrast, if operation op_2 fails (which happens with probability $1 - p_2$), the system moves to state s_3 from which it can retry the entire workflow with probability p_3 or needs to move to the fail state s_5 with probability $1 - p_3$. Two reward structures are defined over this pDTMC, specifying the mean times T_1, T_2, and T_3 and energy consumptions e_1, e_2, e_3 required to perform operations op_1 and op_2 and to initiate a retry, respectively. The example PCTL formula shown in Figure 1 corresponds to the requirement "The system shall complete its workflow successfully with probability of at least 0.8", and the corresponding rational function $prop(p_1, p_2, p_3)$ was obtained using the model checker Storm. Additional requirements specifying upper bounds for time and energy consumption used to complete the workflow (and referring to the two reward structures, e.g., $\mathcal{R}^{\text{time}}_{=?}[\text{F success} \vee \text{fail}] \leq$ 90s and $\mathcal{R}^{\text{energy}}_{=?}[\text{F success} \vee \text{fail}] \leq 1.2\text{KJ}$) can be defined for the system; for readability, these are not shown in Figure 1.

3.3 Runtime Stage

In this PRESTO stage, the PMC expressions (2) are provided to the **disruption prediction engine** along with the time-series of the model parameters, monitored up to the current time point, as shown on the right-hand side of Figure 1. Three hyperparameters are used to configure the engine: the **prediction window** τ, which specifies the time period within which any disruptions are to be predicted, the **update threshold** N, which triggers an update of the linear regression model and a tailored **tolerance** $\epsilon \geq 0$, used to reduce the number of updates required as described later in this section. The selection of these values and their impact on the results is discussed in Section 4.5.

The PRESTO disruption prediction engine evaluates the system's compliance with its K nonfunctional requirements by executing the function PrestoEval(i, *value*, *now*) given in Algorithm 1 each time a new *value* for parameter x_i is obtained at the current time *now*. This function uses and, when necessary, updates the global variables described below.

(1) The slope a_i and intercept b_i for the current linear estimation of each parameter x_i, $1 \leq i \leq n$. We treat each time series as piece-wise linear and use the current model to predict the parameter values $x_i = a_i t + b_i$. The two variables are initially set to $a_i = 0$ and $b_i = x_i^0$, where x_i^0 represents the nominal (i.e., ideal or expected) value for parameter x_i, as provided in the system specification or by a domain expert.
(2) The counters $cntr_i^-$, $cntr_i^+$ for each parameter x_i, $1 \leq i \leq n$. Initially set to zero, these variables are used to count the current number of consecutive data points above and below the parameter value estimated by $a_i t + b_i$ respectively (subject to a tolerance).
(3) The tailored tolerance ϵ_i for each parameter x_i, $1 \leq i \leq n$. A tolerance $\epsilon_i = \epsilon \cdot (x_{i(\text{max})} - x_{i(\text{min})})$, where $[x_{i(\text{min})}, x_{i(\text{max})}]$ represents the range of possible value for x_i, is used to reduce the number of data points that are counted.
(4) A threshold N that defines the capacity of a circular $buffer_i$. Initially empty, the buffer is used to store the last N observed values of parameter x_i, $1 \leq i \leq n$.
(5) The predicted $violationTime_k$ for each nonfunctional requirement k, $1 \leq k \leq K$, initially set to ∞.

The function PrestoEval (Algorithm 1) uses these global variables as follows. The new observation (*value*, *now*) is added to the circular buffer storing the most recent N samples of parameter x_i (line 2) and, if the new *value* is above or below the estimate $a_i now + b_i$ by more than the tolerance ϵ_i, the counter $cntr_i^-$ or $cntr_i^+$ is incremented respectively (lines 3–12). A counter that is not incremented is reset to zero. Lines 13–22) are only executed if the residuals for the last N consecutive measurements of parameter x_i were all greater than $epsilon_i$. When this is the case, the auxiliary

ALGORITHM 1: PRESTO disruption prediction algorithm

```
1:  function PrestoEval(i, value, now)
2:      buffer_i.Add((value, now))
3:      if value > (a_i now + b_i) + ε_i then
4:          cntr_i^+ ← cntr_i^+ + 1
5:          cntr_i^− ← 0
6:      else if value < (a_i now + b_i) − ε_i then
7:          cntr_i^− ← cntr_i^− + 1
8:          cntr_i^+ ← 0
9:      else
10:         cntr_i^− ← 0
11:         cntr_i^+ ← 0
12:     end if
13:     if cntr_i^+ = N ∨ cntr_i^− = N then
14:         (a_i, b_i) ← UpdateFitting(buffer_i)
15:         cntr_i^− ← 0
16:         cntr_i^+ ← 0
17:         for each k = 1,K do
18:             if ∂prop_k(x_1,x_2,...,x_n)/∂x_i ≠ 0 then
19:                 violationTime_k ← ComputeViolationTime(k, now)
20:             end if
21:         end for
22:     end if
23:     return (violationTime_k)_{k=1,K}
24: end function

25: function ComputeViolationTime(k, now)
26:     pol_k(t) ← (P_k(x_1, x_2, ..., x_n) − bound_k · Q_k(x_1, x_2, ..., x_n))|_{x_1=a_1 t+b_1, x_2=a_2 t+b_2, ..., x_n=a_n t+b_n}
27:     return ComputeNextRoot(pol_k(t), now)
28: end function
```

function UpdateFitting is called to apply linear regression to the N data points from $buffer_i$, and thus obtain a new slope a_i and intercept b_i for parameter x_i (line 14). The two counters are reset to zero (lines 15 and 16) and, in the for loop (lines 17–21), the function ComputeViolationTime recalculates the violation times for any of the K requirements that depend on x_i (cf. the check in line 18). For each requirement k, $1 \leq k \leq K$, this function first assembles a univariate polynomial $pol_k(t)$ by replacing the parameters x_i, $1 \leq i \leq n$, from the rational function (2) corresponding to the kth requirement with the linear estimates $a_i t + b_i$ (line 26). Next, ComputeViolation calls the auxiliary function ComputeNextRoot to obtain the root of $pol_k(t)$ that is closest to and larger than *now* (line 27), as (assuming that the kth requirement is not violated at time *now*) this root represents the time when the requirement will next be violated. The function PrestoEval returns the predicted violation times for each of the K requirements in line 23.

The auxiliary functions UpdateFitting and ComputeNextRoot are not presented in Algorithm 1. We assume that UpdateFitting implements standard linear regression [50, 66], and that ComputeNextRoot uses Newton's or Horner's method [55] to approximate the roots of $pol_k(t)$ and returns the root closest to and larger than *now* if such a root exists, or returns ∞ otherwise.

We note that most PrestoEval invocations will be highly efficient, as lines 13–22 will be skipped most of the time. In the worst-case scenario, these lines will only be executed once every N invo-

cations for each of the K requirements, and $N \gg K$ for the envisaged applications of PRESTO. The following theorems show the correctness and provide a formal complexity analysis of the PRESTO algorithm, respectively.

Theorem (Algorithm Correctness). *The function* PrestoEval *terminates and, assuming that the value of the ith system parameter at any future time t is given by $a_i t + b_i$, returns a tuple containing, for each requirement $k \in \overline{1, K}$: (i) the earliest time in the future when the requirement will be violated, if such a violation ever occurs; or (ii) ∞, otherwise.*

Proof. Assuming that the functions UpdateFitting and ComputeNextRoot from lines 14 and 27 of Algorithm 1, respectively, are correctly implemented and therefore terminate, function PrestoEval will also terminate because its only loop (from lines 17 to 21) has a finite number of iterations K.

Assume now that the ith system parameter will have the value $x_i = a_i t + b_i$ at any future time t. To prove the correctness of PrestoEval, we will show that, both prior to the first PrestoEval invocation and after each update from line 19, $violationTime_k$ contains the earliest future time when requirement k will be violated, or ∞ if the requirement will never be violated. For the first case, i.e., before the first invocation of the function, this result holds because we have $x_i = x_i^0$ for every parameters i and, in line with the assumption that the system does not violate any of its requirements when first deployed, $violationTime_k = \infty$. For the second case, we note that, after each update of the linear approximation $x_i = a_i t + b_i$ in line 14, the value of $violationTime_k$ is updated in line 19 unless (see the if statement from line 18) the kth requirement does not depend on parameter x_i (in which scenario the property we want to prove continues to hold for the current $violationTime_k$ value). Finally, if a new $violationTime_k$ value is obtained, this value is computed so that it represents the earliest future time when requirement k will be violated, or ∞ if the requirement will never be violated. As such, the tuple returned by PrestoEval in line 23 satisfies the property specified by the theorem. □

Theorem (Algorithm Complexity). *The function* PrestoEval *requires at most $O(N + Knd)$) steps, where $d = \max_{1 \le k \le K} degree(pol_k(t))$ represents the maximum degree across the K univariate polynomials assembled in line 26 of Algorithm 1.*

Proof. The statements from lines 2–12 and the check from line 13 of PrestoEval require constant time, and the linear regression applied to the N data points from $buffer_i$ in line 14 can be performed in $O(N)$ time using a simple linear fitting function [50]. Each of the at most K invocations of ComputeViolationTime from the for loop in lines 17–21 requires:

- $O(n \cdot \text{degree}(pol_k(t)))$ time to assemble the polynomial $pol_k(t)$ in line 26 by replacing each occurrence of x_i, $1 \le i \le n$, in a polynomial of degree $\text{degree}(pol_k(t))$ with its linear approximation $a_i t + b_i$;
- $O(\text{degree}(pol_k(t)))$ time to compute the roots of $pol_k(t)$, e.g., by using Horner's method, whose complexity is linear in the size of the polynomial's degree [55];
- $O(\text{degree}(pol_k(t)))$ time to find the root closest to and larger than *now*.

Accordingly, the for loop from lines 17–21 has complexity $O(Kn \max_{1 \le k \le K} degree(pol_k(t))) = O(Knd)$, and therefore the overall complexity of PrestoEval is $O(N + Knd)$. □

4 EVALUATION

In this section we evaluate the **effectiveness**, **robustness** and **configurability** of our approach. We introduce the two case studies used to carry out this evaluation (Section 4.1) and illustrate the application of PRESTO to a scenario associated with the first case study (Section 4.2). Next, we

Table 1. Key Characteristics of the Case Study Models used for the PRESTO Evaluation

	Fruit-picking application	**RAD application**
Application domain	Agriculture	Health care
Number of model parameters	8	7
Analysed properties	Reachability Reward × 2	Reachability Reward Unbounded until
Property details	Table 2	Table 3

Table 2. Non-functional Requirements that the Fruit-picking Robot is Expected to Comply with and the PCTL Formula used to Encode Them

ID	**Informal description**	**Requirement (PCTL)**	**Implication of requirement violations**	**Type of requirement**
R1	The robot shall complete the fruit picking successfully with probability of at least 0.8	$\mathcal{P}_{=?}$[F "picking success"] ≥ 0.8	Leaving a large number of fruit unpicked	Reliability
R2	The expected time to complete the picking process shall not exceed 5 seconds.	$\mathcal{R}^{"time"}_{=?}$ [F"done"] $\leq 5s$	The harvesting takes longer to complete	Performance
R3	The expected energy consumption to complete the picking process shall not exceed 5 kilojoules.	$\mathcal{R}^{"energy"}_{=?}$ [F"done"] $\leq$ 5KJ	The robot needs to be charged more frequently resulting a longer time to complete the harvesting	Resource utilisation

describe our experimental setup (Section 4.3) and summarise the research questions underpinning the evaluation (Section 4.4). We conclude with a discussion of the experimental results and threats to validity (Section 4.5).

4.1 Case Studies

The first case study involves a fruit-picking robot, inspired by recent developments in the autonomous farming domain [65, 68], and the second considers a robot-assisted dressing scenario [17] to aid independent living. The models used in both case studies feature multiple transition probabilities and/or reward parameters that are specified as PMC expressions and changes in the model parameters reflect gradual degradation of internal or environmental conditions. The key characteristics of the two case study models are described in Table 1. The nonfunctional requirements for the case studies and the properties they represent are detailed in Tables 2 and 3. Our approach is applicable for any nonfunctional requirements that can be expressed in PCTL, including safety, reliability, performance and maintainability requirements [5].

Case study 1: Fruit-picking robot The robot is expected to harvest fruit on a farm by autonomously performing the following three tasks in a reasonable time:

(1) position itself ready for picking;
(2) attempt to pick fruit;
(3) decide whether to retry if picking was unsuccessful.

The behaviour of the robot is referred to as a picking session. To ensure its reliability and performance, the robot is expected to comply with the three system-level requirements summarised in Table 2.

Case study 2: Robot Assisted Dressing System (RAD) The robot is designed to support a user with restricted mobility to live independently by allowing them to dress without the need for

Table 3. Non-functional Requirements that the RAD Robot is Expected to Comply with and the PCTL Formula used to Encode Them

ID	Informal description	Requirement (PCTL)	Implication of requirement violations	Type of requirement
R1	The robot shall complete the dressing task successfully with probability of at least 0.75	$\mathcal{P}_{=?}$[F "dressing success"] ≥ 0.75	Resetting the system spends more resources including time	Reliability
R2	The robot shall complete the dressing task successfully with probability of at least 0.6 without performing correction task	$\mathcal{P}_{=?}$[¬correction U dressingsuccess] ≥ 0.6	Frequent correction will affect user experience	Robustness
R3	The expected time to complete the dressing process shall not exceed 38 seconds.	$\mathcal{R}^{\text{"time"}}_{=?}$[F"done"] $\leq 38s$	If the dressing task takes longer than expected, it can reduce the system's availability for other users	Performance

additional healthcare support. For each dressing session, the robot needs to perform five tasks in sequence with the user:

(1) Look for the garment;
(2) Orient the garment in the correct position;
(3) Move the garment toward the user;
(4) Determine the pose and arm position of the user;
(5) Perform the dressing task.

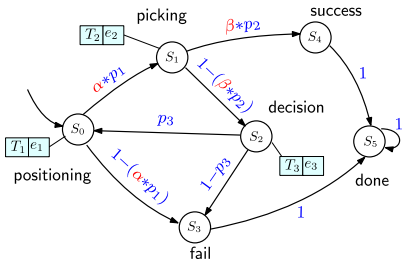
The RAD also needs to comply with the non-functional requirements presented in Table 3.

4.2 Illustration of the PRESTO Application

Before presenting the results obtained for experiments carried out across a wider range of scenarios, we illustrate the application of PRESTO for a single such experiment taken from our first case study. We first describe the pDTMC model for a fruit-picking session and then show the execution of the two stages of our approach. The pDTMC model and other intermediate results for the second case study are available on our project website [59].

Model. Figure 2(a) shows the pDTMC for a fruit picking session, which starts with the robot attempting to correctly position itself next to a piece of fruit for picking (state s_0). This operation may succeed with probability αp_1, $\alpha \in (0, 1]$ is a *degradation coefficient* (discussed in more detail later in this section) and p_1 represents the nominal probability of success for this operation. When this operation succeeds, the robot transitions to state s_1, where picking is attempted, or may fail (with probability $1-\alpha p_1$), in which case the robot transitions to state s_3, where picking is abandoned. Picking (in state s_1) succeeds with probability βp_2 (where $\beta \in (0, 1]$ is a *degradation coefficient* and p_2 the success probability under ideal conditions) and the system moves to state s_4, followed immediately with a transition to s_5 where the picking session ends. If picking is unsuccessful, the process moves to the decision state s_2, where the robot decides whether to move back to state s_0 and retry positioning, or to give up (s_3) and end the session (s_5).

Figure 2(b) shows the encoding of this pDTMC in the high-level modelling language of the model checker PRISM [45]. Lines 3–8 define the model parameters associated with (a) the operational profile, (b) the degradation coefficients, and (c) the mean execution times and the mean energy consumption of the three tasks. Here the operational profile parameters, $p_{j\in\{1,2,3\}}$, are the probabilities when the robot is in perfect working order and we assume that they are fixed domain-specific values (e.g., provided by the farmer). As mentioned earlier, the degradation coefficients, α and β, reflect changes in these probabilities as the system gradually degrades, with the assumption that their values can be obtained by either monitoring [46] or self-testing [56]. For example,



(a) The pDTMC model of the fruit picking robot behaviour in a visual diagram where blue boxes represent state rewards

```
1   dtmc
2   // Operational profile parameters
3   const double p1=0.9; const double p2=0.95; const double p3=0.6;
4   // Degradation coefficients
5   const double α; const double β;
6   // Reward parameters
7   const double T1; const double T2; const double T3
8   const double e1; const double e2; const double e3
9   // Modelling the preocess
10  module FruitPicking
11    s : [0..5] init 0;
12    [positioning] s=0 → (α*p1):(s'=1) + (1-α*p1):(s'=3);
13    [picking] s=1 → (1-β*p2):(s'=2) + (β*p2):(s'=4);
14    [decision] s=2 → p3:(s'=0) + (1-p3):(s'=3);
15    [fail] s=3 → 1:(s'=5);
16    [success] s=4 → 1:(s'=5);
17    [done] s=5 → 1:(s'=5);
18  endmodule
19  // Rewards
20  rewards "time"
21    (s=0):T1; (s=1):T2; (s=2):T3;
22  endrewards
23  rewards "energy"
24    (s=0):e1; (s=1):e2; (s=2):e3;
25  endrewards
```

(b) The pDTMC model of the fruit picking robot behaviour in PRISM syntax.

Fig. 2. pDTMC modelling for the fruit-picking robot.

a decrease of α and β could be associated with mud slowly building up on the wheels of the robot (leading to reduced success of positioning) and with lighting conditions (resulting in picking being more difficult), respectively. Finally, $T_{j\in\{1,2,3\}}$ and $e_{j\in\{1,2,3\}}$ represent the mean operation execution times and mean energy consumption for the three tasks. The picking session is modelled by the module FruitPicking (Lines 10–18 in Figure 2(b)), and the mean execution times and mean energy consumption are used to define reward structures over the pDTMC (lines 20–25). We note that this pDTMC has $n = 8$ parameters: α, β, and T_i and e_i, $1 \leq i \leq 3$.

Stage 1. In its design-time stage, PRESTO applies parametric model checking to the pDTMC model from Figure 2(b) in order to obtain PMC expressions for the system properties associated with the three requirements from Table 2. Using the model checker Storm to perform this PMC analysis yields the following property expressions (2) for these requirements:

$$\begin{aligned} prop_1(\alpha,\beta) &= \frac{855\beta\alpha}{513\alpha\beta - 540\alpha + 1000} \\ prop_2(\alpha,\beta,t1,t2,t3) &= \frac{-342\alpha\beta t3 - 540\alpha t3 + 1000(t1+t3) + 900\alpha t2}{513\alpha\beta - 540\alpha + 1000} \\ prop_3(\alpha,\beta,e1,e2,e3) &= \frac{-342\alpha\beta e3 - 540\alpha e3 + 1000(e1+e3) + 900\alpha e2}{513\alpha\beta - 540\alpha + 1000} \end{aligned} . \tag{3}$$

Stage 2. In the run-time stage, the property expressions (3) are used to predict requirement violation times during invocations of the function PrestoEval from Algorithm 1. Prior to the first invocation, the circular buffers for the eight pDTMC parameters are empty, and the linear estimates of these parameters are determined by their nominal values α^0, β^0, and T_i^0, e_i^0, $1 \leq i \leq 3$. For this illustration, we consider one of the experiments from our evaluation, in which the PRESTO hyperparameters were $N = 400$, $\tau = 400$ and $\epsilon = 0$ determined based on domain knowledge.[3] Points at which line 13 of Algorithm 1 is satisfied are highlighted in blue in Figure 3(a) to (h). As

[3]A wide range of parameter values is expected to be effective, and their impacts on the results will be discussed in RQ3.

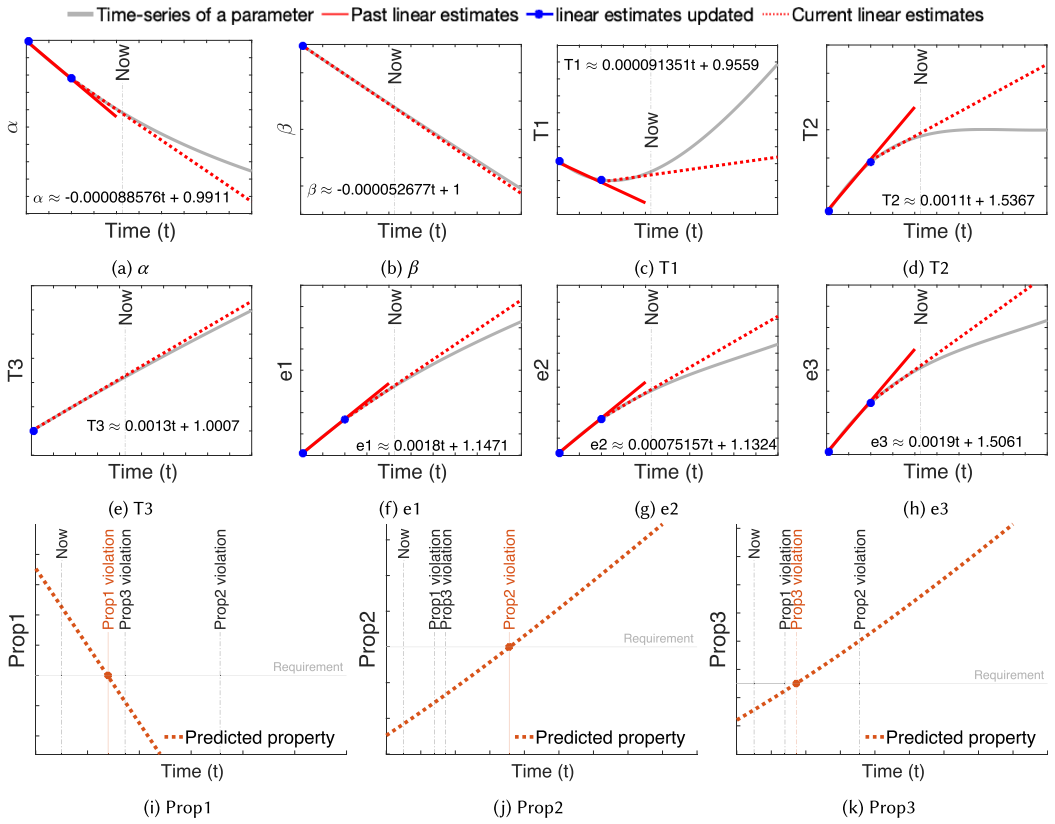


Fig. 3. PRESTOEVAL predictions of: (a)–(h) parameter values; and (i)–(k) requirement violations times for the fruit-picking robot at a point in time (where the distance between ticks on the time axis is 200 points).

.

α appears in all three properties (3), any updating of its linear approximation triggers the recalculation of new violation-time predictions for all three requirements as depicted in Figure 3(i), (j) and (k). However, as the earliest predicted violation time, *Prop1*, is more than τ hours into the future, no adaptation is triggered.

4.3 Experimental Setup

The PRESTO algorithm and the simulations were implemented using Python 3.7 with Stormpy 1.6.4 [26] to produce algebraic expressions from the Storm parametric model checker. All experiments were carried out using a MacBook Pro with a 2.3 GHz Quad-Core Intel Core i7 CPU and 32 GB RAM. The evaluation process, and simulator behaviour, are described below.

For each case study, we compared PRESTO to a baseline approach in which regular maintenance was carried out at fixed intervals to replicate the effects of a routine maintenance schedule. The cost and frequency of maintenance was varied throughout the experiments to evaluate the value of PRESTO against a range of possible maintenance plans. We simulated data for PRESTO and the baseline system for both case studies over a period long enough to ensure multiple violations, this being 6-months for the fruit-picking application and a year for the RAD application.

The current version of PRESTO predicts violations which arise due to disruptions caused by the slow degradation of internal, or environmental conditions. These changes are observed through the monitoring of system parameters which can be susceptible to noise. To simulate such data

pattern, we **generated time-series** for the n parameters of each system from our case studies using the five steps detailed below and summarised in Algorithm 2:

(1) The length of a simulated time-series is determined by randomly generating an integer, J, between 7,200 and 10,080 (line 2), corresponding to 30–42 days (i.e., *timestep* = 10 minutes) and 60–84 days (i.e., *timestep* = 20 minutes) worth of data for the fruit-picking and RAD applications, respectively, resulting in observations $X_{ij} = (value_{ij}, t_{ij})$, $1 \leq i \leq n$, $j = 1, 2, \ldots, J$ of the n pDTMC parameters being returned in line 24. Randomly generating J allows the time-series to have different trends after the re-scaling in Step (4).
(2) To generate a time-series with a non-linear pattern, an integer variable, *scale*, is randomly selected between 0.5·J and 2·J (line 4).
(3) An exponential function with the base being Euler's number is used to provide $value_{ij} = e^{t_{ij}/scale}$, $j = 1, 2, \ldots, J$ for each parameter x_i. Increasing or decreasing trends are generated according to the nature of the parameter with decreasing trends achieved using $-e^{t_{ij}/scale}$, $j = 1, 2, \ldots, J$. For example, cost and time might have an increasing trend and parameters associated with degradation will be represented by a decreasing function (lines 7 to 11).
(4) The time-series derived in Step (3) is then re-scaled to a range determined by domain knowledge that ensures the occurrence of system-level violation (line 12).
(5) In order to simulate a non-monotonic data pattern, a random number sampled from a Gaussian distribution with zero mean and variance $\sigma^2 = \gamma \cdot (\max_{j=1,J} X_{ij} - \min_{j=1,J} X_{ij})$ for some $\gamma \in (0, 1)$ is added (lines 14 to 21) before smoothing with a high order polynomial to produce the final time-series (line 22).

These five steps ensure that the generated time series have a non-monotonic trend that reflects the gradual degradation of internal and environmental conditions and that will lead to requirement violations. In addition, two different levels of noise are added to simulate different environmental interference [29, 30] as: *low-noise* where $\gamma = 0.15$ and *high-noise* where $\gamma = 0.3$.

Process simulator: We simulated both case studies over a period of time, during which multiple disruptions are expected. For each case study, we considered two service strategies as follows:

— "baseline": when the preventative maintenance is carried out at regular intervals;[4]
— "PRESTO": when the preventative maintenance is triggered by predicted violations or regular intervals, whichever happens first.

In the **baseline** scenario, the parameters are re-set to pre-degradation values after each maintenance operation or violation and new time series are generated. This simulates the effect of a successful maintenance activity or repair and the process continues until the predefined simulation period ends.

For the **PRESTO** scenario, we not only use the regular maintenance as in the baseline scenario but also predict potential disruptions. If the predicted disruption is τ away from "*now*", a preventative service will be called, which re-sets all parameters to pre-degradation level. This simulates the effect of proactive adaptation.

We evaluate the performance of PRESTO through a cost formula (detailed in Subsection 4.5) that considers **true positives** (**TP**), **false positives** (**FP**), and **false negatives** (**FN**). A TP occurs when a disruption is predicted to occur less than τ before the actual disruption. A disruption that occurs before it is predicted is classed as a false negative and a disruption predicted to occur more than τ before the actual disruption is classed as a false positive case.

[4]When the number of services is zero, it is equivalent to reactive adaptation, as maintenance is triggered by requirement violations.

ALGORITHM 2: Time Series Generation

function GenerateTimeSeries(n, *timestep*, *high_noise*, *smoothing_degree*)
  $J \leftarrow$ RandomInt(7200, 10080) ▷ Select time-series length J randomly
  for $i = 1$ to n **do**
    $scale \leftarrow$ RandomInt($0.5 \cdot J, 2 \cdot J$) ▷ Select *scale* randomly
    for $j = 1$ to J **do**
      $t_{ij} \leftarrow j \cdot timestep$
      if parameter i is "increasing" **then**
        $value_{ij} \leftarrow e^{t_{ij}/scale}$
      else
        $value_{ij} \leftarrow -e^{t_{ij}/scale}$
      end if
      $X_{ij} \leftarrow$ (Rescale($i, value_{ij}$), t_{ij})
    end for
    $range \leftarrow \max_{j=1,J} X_{ij} - \min_{j=1,J} X_{ij}$
    for $j = 1$ to J **do**
      if *high_noise* **then**
        $X_{ij}.value_{ij} \leftarrow X_{ij}.value_{ij} +$ Gaussian$(0, 0.3 \cdot range)$
      else
        $X_{ij}.value_{ij} \leftarrow X_{ij}.value_{ij} +$ Gaussian$(0, 0.15 \cdot range)$
      end if
    end for
    $X_i \leftarrow$ PolynomialFit(X_i, *smoothing_degree*)
  end for
  return $(X_i)_{i=\overline{1,n}}$
end function

4.4 Research Questions

The purpose of our approach is to enable autonomous systems to predict requirement violations and avoid them through adaptations triggered before the violations occur. As such, the evaluation presented in this section focuses on comparing the number of requirement violations and the cost of dealing with such violations, preventatively or reactively. In both cases, the adaptation may consist of the system initiating a self-correction or repair process or, for autonomous systems with *human in the loop* [8, 21, 39], calling human operator support. To ensure that the evaluation is fair, we considered scenarios in which such self-correction/self-calibration is also applied regularly, as part of a maintenance process that the autonomous system undertakes automatically or that involves maintenance by a support engineer. In line with the fact that prevention is typically better than cure, we further assumed that the preventative adaptation cost was lower than that of reactive adaptation.

With these considerations in mind, we carried out experiments aimed at answering the following research questions:

RQ1 (Effectiveness): *Does PRESTO reduce adaptation cost?* We assessed whether PRESTO could reduce the overall adaptation costs associated with both regular maintenance and undetected violations, by predicting, and responding to, disruptions without the need for frequent, fixed-interval servicing.

RQ2 (Robustness): *Is the performance of PRESTO sensitive to noise in the measurements of the system parameters?* We assessed whether the *effectiveness* of PRESTO is affected by noise in the

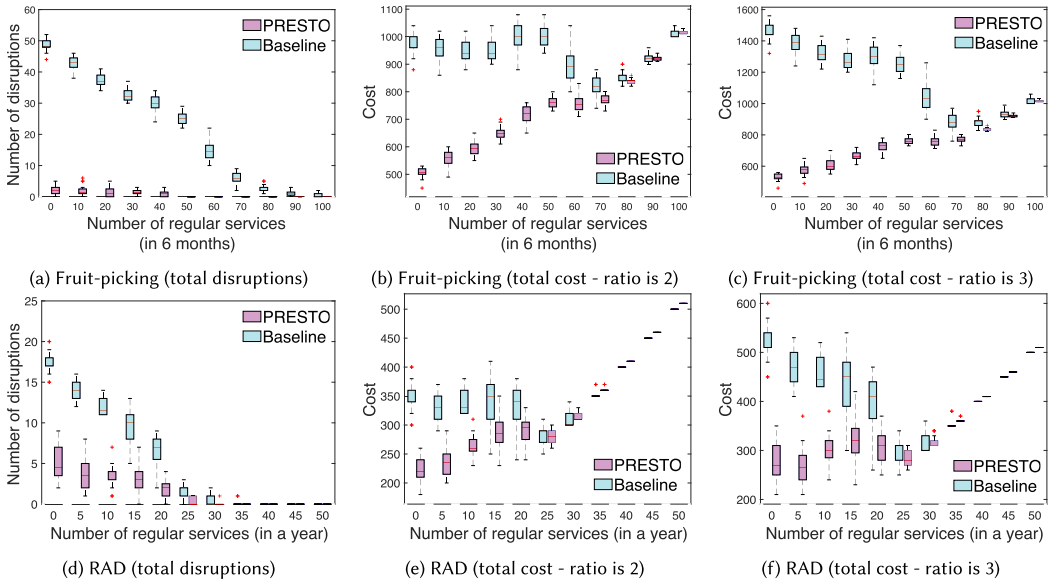


Fig. 4. The relationship between the number of fixed-interval maintenance and disruptions and costs. Each boxplot shows the result from 20 trials.

measurements, commonly observed in real data. We tested PRESTO's ability to predict violations in the presence of measurement noise at varying levels.

RQ3 (Configurability): *Can the PRESTO hyperparameters be tuned to obtain different tradeoffs between the overheads and effectiveness of the approach?* We evaluated the impact of different combinations of hyperparameter values on the computational overheads and adaptation-cost savings delivered by our approach.

4.5 Results and Discussion

We used the simulator detailed in Section 4.3 to obtain results for both case studies using the baseline approach and PRESTO, and recorded the number of disruptions that occurred and those which were successfully predicted as well as those for which a repair was required (i.e., the disruption was missed). To address RQ1 and RQ2, we set the following parameters based on our best estimation: N=400, ϵ=0, and τ=10 hours for fruit-picking, and τ=40 hours for RAD application. To facilitate a more informed selection of these parameters, we will investigate the impact of different parameter values in RQ3.

A total service cost was then recorded for each approach as follows. Let m be the number of undetected disruptions which repairs are required and c_1 be the cost of repair. Let n be the number of maintenance operations and c_0 be the cost of a service. For the "baseline" approach, the overall cost is calculated as

$$cost_{baseline} = m \cdot c_1 + n \cdot c_0, \tag{4}$$

and, for PRESTO, the overall cost is

$$cost_{PRESTO} = (TP + FP) \cdot c_0 + FN \cdot c_1, \tag{5}$$

assuming both TP and FP will trigger a preventative maintenance and FN will require a repair.

RQ1 (Effectiveness): Figure 4(a) and (d) show how the number of disruptions is reduced as the frequency of regular services increases for both applications. The figures indicate that the total number of disruptions occurring with PRESTO is lower than the baseline approach unless preventative maintenance is carried out very frequently. For example, when there are no fixed interval services, the baseline approach reports on average 48 and 17 disruptions for the fruit picking and RAD applications respectively. In contrast, PRESTO suffers only 3 unpredicted disruptions for fruit-picking and 6 for RAD in the same period, giving a reduction in disruptions of 93.8% and 64.7% respectively.

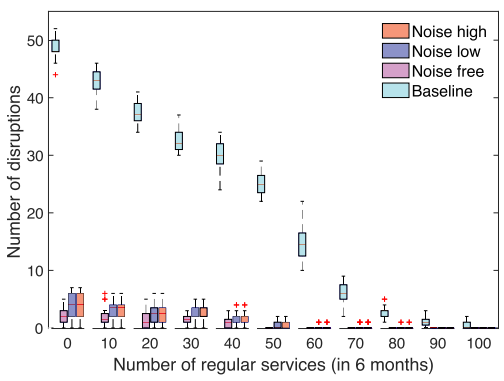
Although the number of disruptions decreases for both approaches as the number of regular services increases, the baseline approach may report disruptions after 100 regular services in the fruit-picking application in comparison to PRESTO which requires just 50 regular services to avoid all disruptions in the time period. Similarly, while 35 regular services are sufficient for PRESTO in the RAD application, the baseline approach requires an additional 5 services to void the disruption completely. This result suggests that PRESTO is able to reduce the frequency of services required.

Figure 4(b) to (f) shows the total service costs for each application with the ratio of repair to preventative maintenance ($\frac{c_1}{c_0}$) set to 2 and 3. It can be seen that the cost of the baseline approach was at least twice as high as PRESTO with no regular service. As the frequency of regular service increases, the gap in costs reduces as expected since disruptions are less likely with frequent routine maintenance. However, we note that such a maintenance strategy is highly wasteful.

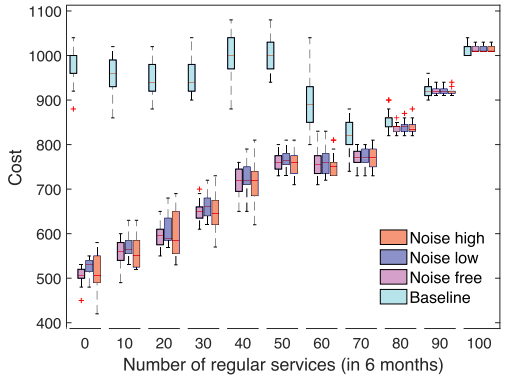
RQ2 (Robustness): Measurement noise is commonly observed in sensory data and has a negative impact on system performance. Using data with different noise levels as discussed in Section 4.3, we found that PRESTO is largely insensitive to noise with the high noise level having a marginal effect on the number of disruptions and the cost of services (Figure 5). Indeed, in all cases, PRESTO continues to outperform the noise-free baseline approach. The ability of PRESTO to reject normally distributed noise with zero mean is due to the use of the linear regression algorithm which approximates an expected noise value of zero when enough samples are obtained.

RQ3 (Configurability): We evaluated the impact of varying two hyperparameters (i.e., N and τ) on (1) the number of disruptions that PRESTO is able to predict and mitigate, (2) the total service cost incurred during the operation period and (3) computation overheads. The impact of ϵ seen during these experiments is minor when compared to N, which is unsurprising since the experiments presented consider disruptions caused by the gradual degradation of model parameters. The hyperparameter ϵ plays a less important role in such settings. For alternate modes of change, which will be considered in future development, ϵ is believed to be more important for PRESTO. Given the limited impact caused by varying ϵ we do not include the results in this article, instead, they are available on our supporting website [59]. We carefully chose a wide range of parameter values and tested the impact of their combinations. For example, the update threshold (N) varied from 50 to 1,200, and τ varied from 5 hours to the maximal of 120 hours for the fruit-picking application and 10 hours to 680 hours for the RAD application. The results of these experiments are presented in Figures 6 and 7 for the fruit-picking application and RAD application respectively. In addition, we examined computation overheads for operating PRESTO in terms of mean time to calculate violation time (i.e., ComputeViolationTime, line 14 in Algorithm 1), new linear estimates (i.e., UpdateFitting, line 10 in Algorithm 1) and the average frequency that new linear estimates are required before a disruption is predicted.

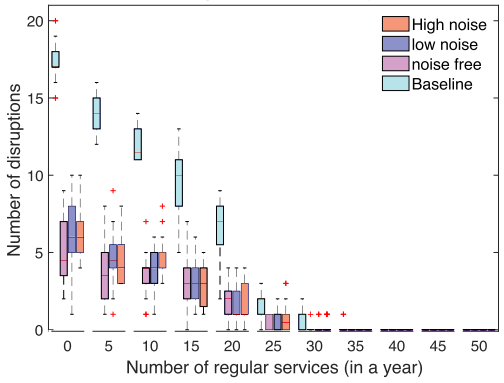
Figure 6 shows the evaluation results for the fruit-picking application. For a small τ ($\tau \leq 40$h), the results are more sensitive to varying N in comparison to a larger τ ($\tau > 40$h). However, the impact of varying N on the results is less significant than the impact of varying τ. Small τ (5h) leads to the greatest number of unpredicted disruptions, which then results in a high service cost.



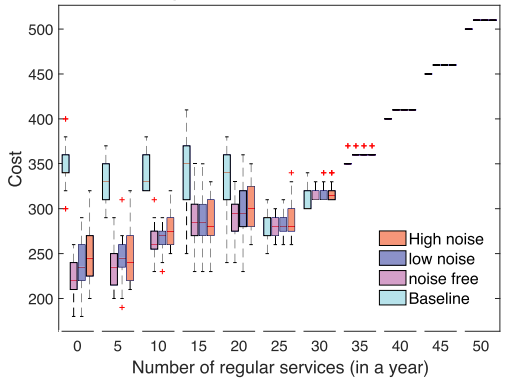
(a) Fruit Picking robot - Total Disruptions



(b) Fruit Picking robot - Total costs (cost ratio is 2)

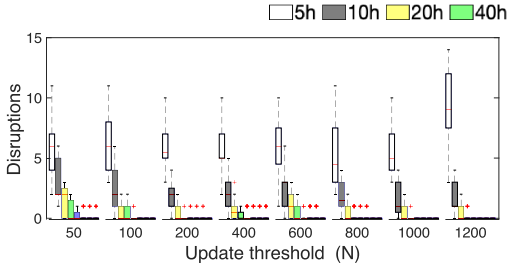


(c) RAD application - Total Disruptions

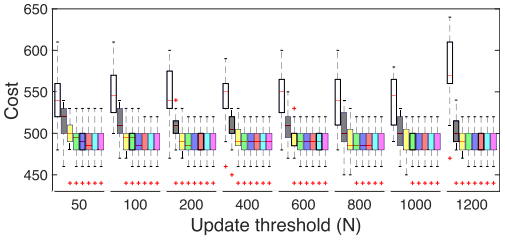


(d) RAD application - Total costs (cost ratio is 2)

Fig. 5. The impact of noise on the number of disruptions detected by PRESTO and the associated cost.



(a) Total disruptions $\epsilon = 0$

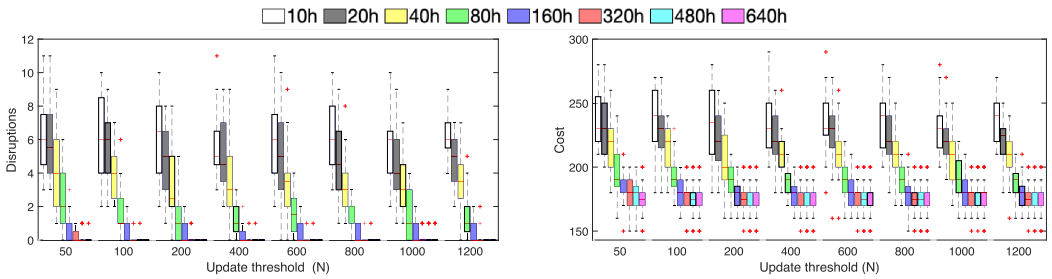


(b) Total costs $\epsilon = 0$

Fig. 6. The total disruptions and cost with different user-specified parameters (Fruit Picking robot)

However, as τ is increased, more disruptions can be predicted and mitigated by PRESTO until the point where all disruptions are mitigated.

The evaluation results from the RAD application are shown in Figure 7. In comparison to the results for the fruit-picking application, we found that a larger τ is needed to completely avoid all disruptions in the RAD application. The results suggest that, despite a similar pattern, the actual value of this hyperparameter will depend on the application and should be carefully chosen according to requirements.



(a) Total disruptions $\epsilon = 0$ (b) Total costs $\epsilon = 0$

Fig. 7. The total disruptions and cost with different user-specified parameters (RAD robot)

Table 4. PRESTO Computational Overhead

	Mean time for computing the polynomial roots	**Mean time for linear regression**†	**Number of linear fitting required (mean)**†
Fruit-picking	34ms	0.29ms - 0.41ms	8-1
RAD	15ms	0.30ms - 0.34ms	4-1

†The range was determined using $\tau = 50$,N =50, and $\tau = 1200$,N =1200, ($\epsilon = 0$)

PRESTO is expected to work with a wide range of values for the hyperparameters N and τ. The selection of these values by exploiting domain knowledge is expected to be effective, as is the case for obtaining prior knowledge in Bayesian learning methods. The results presented in Figures 6 and 7 highlight the impact of these hyperparameters, and provide useful insight for their determination. In general, the results are less sensitive to different values of N in contrast to τ. We observe that a smaller τ, appropriate to its application context (such as 5 and 10 hours in fruit-picking and RAD applications, respectively), leads to a larger number of unpredicted disruptions. More disruptions can be avoided by gradually increasing τ up to a point where no further improvement can be observed. Identifying the optimal values for N and τ in PRESTO resembles determining optimal values for the hyperparameters of other machine learning techniques.

The impact of N and τ is also reflected in the computation time as shown in Table 4. Although the linear equations are updated more frequently when both parameters are small ($\tau = 50$ and N =50), less time is required for computing linear estimates as fewer data points (N) are used in the regression. When both parameters are large ($\tau = 1,200$ and N =1,200), the opposite trend is observed as the equations are updated less frequently but more time is required for computation. Most computation overheads are due to calculation of the roots of polynomials derived from PMC expressions, but this only takes milliseconds to complete on the computer detailed in Section 4.3.

Discussion: The results presented above show that PRESTO is able to effectively predict and mitigate disruptions before they occur. Our results suggest that the smaller the number of regular services, the more significant the performance gain is from using PRESTO over the baseline approach. In fact, when the PRESTO is running in between regular services, the cost comparison between PRESTO and the baseline approach only relies on the ratio of true positives to false positives in relation to the cost difference between preventative maintenance and urgent repairs. To see this, consider a period P between scheduled maintenance arrangements and assume first that at least one violation occurs. Then equation 4 becomes

$$cost_{baseline} = (TP + FN) \cdot c_1, \tag{6}$$

as $n = 0$ and $m = TP + FN$; and PRESTO therefore reduces costs in the period P if:

$$(TP + FP) \cdot c_0 < TP \cdot c_1. \tag{7}$$

Let the difference in cost in term of ratio be $(\frac{c_1}{c_0})$, then PRESTO reduces costs in the period P if:

$$\frac{c_1}{c_0} > 1 + \frac{FP}{TP}. \tag{8}$$

where $TP \neq 0$.

PRESTO predicts disruptions by continually assessing the gradual change observed in system/environmental parameters, a common phenomenon in real-world applications [63, 64] that is underexplored by current research into self-adaptive systems. The first stage is design-time and we use the fact that computational resources are less of an issue by calculating PMC expressions that can be updated at run-time. This stage could be exploited further as prior knowledge of the order of importance of the parameters in the system equations and how they interact could reduce the level of monitoring required. In the second stage, at run-time, system disruptions are predicted by linear equations that are used to provide future parameter values for the PMC algebraic expressions. We use piece-wise linear modelling to simplify prediction whilst allowing the trend to change over time. This also allows information about changes in slope to be stored so that any underlying periodicity or seasonal component could potentially be recognised and accounted for. We summarise these and other opportunities for enhancing PRESTO when we propose future work directions for our project in Section 6.

Threats to Validity: Construct validity threats may arise from over-simplifications and unrealistic assumptions made when establishing experiments for evaluation. To mitigate these threats, we used two case studies inspired by real autonomous systems described in the recent research literature, i.e., a fruit-picking robot [65, 68] and a robotic assistive dressing system [17]. We also avoided the use of explicit costings and instead used cost ratios where we assumed that the cost of repair exceeded the cost of maintenance. In addition, the evaluation of PRESTO relies on simulated patterns of parameter change, which mimic a slow degradation in the system. Additional experiments are necessary to assess the performance of PRESTO for data patterns derived from real systems, and presenting more complex patterns of change.

Internal validity threats can originate from bias in interpretation of the experimental results. To mitigate this threat we evaluated PRESTO by answering three independent research questions. To further reduce the likelihood of having a biased result, the evaluations were compared against a baseline approach that is widely used in deployed systems, and we considered a broad range of parameter values for costs and service frequency. Finally, to enable the independent verification of our results, the source code of our simulator and the data from all the experiments to produce the tables and figures in the article are available online at [59].

External threats to validity may arise if the systems under consideration or the assumptions made in the experimental framework are not indicative of the characteristics of other systems to which PRESTO may be applied. To mitigate these threats, we evaluated PRESTO using two case studies from different application domains. Each of the models arising from these case studies makes use of a pDTMC with a different model structures. External validity threats may also arise due to restrictions in the types of non-functional requirements supported by PRESTO. To mitigate these threats, we chose a set of properties that covered a broad range of requirements (reliability, performance, resource utilisation, etc.). Additional experiments are still required to confirm that PRESTO can predict a wide range of disruption types in other application domains.

5 RELATED WORK

Adaptive systems may be classified as either reactive or proactive with the former investigating adaptations in response to observed changes whilst the latter anticipates change in order to mitigate unwanted system behaviours before they occur [52]. Although both approaches are capable of handling a range of change types in the presence of uncertainty [4, 42], proactive adaptation is particularly important when property violation, or non-compliance with requirements, is associated with high costs or safety critical outcomes [58]. Proactive adaptation approaches for autonomous systems are analogous to predictive maintenance from control theory [61]. They predict the future values of operational and environmental parameters, allow potential violations to be identified early, and enable adaptations to be applied to avoid or mitigate undesirable outcomes. Despite the potential for such methods to improve system performance and reduce operational costs, the existing solutions that implement proactive adaptation exhibit the three important limitations summarised below.

Limited adaptation approach flexibility. Proactive adaptation requires an adaptation approach initiated before an anticipated violation of non-functional requirements. Existing solutions like **Proactive Latency-aware Adaptation** (**PLA**) and **Control-based Requirements-oriented Adaptation** (**CobRA**) [51–53] incorporate such mechanisms and support a broad range of non-functional requirements—including requirements specified in **Probabilistic Reward Computation Tree Logic** (**PRCTL**) for the probabilistic model checking version of PLA (PLA-PMC) [52, 53]. However, PLA and CobRA focus on adapting the system proactively to avoid a requirement violation, or to maximise some utility function that can be composed of different quality measures. As such, these approaches are all about the adaptation decision, but cannot be used to trigger other adaptation approaches. In contrast, PRESTO focuses on predicting the future point in time when a requirement violation will occur, without restricting the choice of adaptation approach, and thus leading to greater flexibility in choosing the most suitable adaptation approach for the system.

High run-time computational overheads. Online testing techniques provide a useful means for the prediction of changes and deviations before they happen [43], but such techniques are often computationally intensive. Similarly, online reinforcement learning has also been proposed [48] and does not require predefined threshold values in order to trigger the adaptation. However, the slow convergence of the reinforcement may hinder its use at run-time. A range of research activities have been conducted on the impacts of limited computational resources [60, 72] using machine learning and time series. In [49], deep learning ensembles were employed for prediction and to trigger adaptation. However, triggering adaptation based on the prediction of data rather than the violation of system-level requirements is likely to result in over or under adaptations. Unlike these data-driven techniques, PRESTO provides *formal guarantees* using parametric model checking for the the analysis of system-level requirements. Parametric model checking considers the parameter uncertainty on system verification [10, 13, 27, 57, 62, 71] differently from the conventional probabilistic model checking techniques that have been widely used in the verification of system properties. Probabilistic model checking has been used to identify optimal, latency-aware adaptation decisions [53] and the impact of the adaptation latency on system performance are studied using stochastic multi-player games and model checking [15]. The approach from [15] considers a prediction horizon with stochastic behaviours in the environment but, while the method can be applied at run-time, the entire model needs to be updated when changes in the probabilities are observed. In contrast, PRESTO utilises parametric model checking to derive the algebraic expressions of model properties at design-time, with only re-evaluation for new parameter predictions being required at runtime, and therefore can operate with low run-time computational overheads [9, 12, 34, 36, 37]. Moreno et al. [54] improve the solution from [53] by analysing the MDPs at design time to avoid

the re-evaluation at runtime using a stochastic dynamic programming. However, this approach requires additional handwritten input describing the adaptation tactics encoded in Alloy language, whereas PRESTO does not require any effort to prepare the input, and needs only the parametric version of a probabilistic model of the relevant behaviour of the autonomous system.

Focus on degradation of system (components). In control theory, approaches such as the one proposed by You et al. [69] define a cost-effective sequential update policy to determine a real-time **preventive maintenance** (**PM**) schedule for continuously monitored degrading systems. The expected maintenance cost of a system for the remaining time includes replacement cost, imperfect PM cost and minimal corrective maintenance cost (i.e., recovery cost). Similarly, Chouikhi et al. [19] and Feng et al. [33] propose condition-based maintenance for thermodynamic and production systems affected by degradation. Nevertheless, different types of change in the parameters of autonomous systems can appear due to the high uncertainty of their environments. These changes can be linear, cyclic or seasonal—as well as changes that correspond to degrading system components, which these approaches have been designed for. As PRESTO does not make any assumption about the types of parameter changes that may lead to future violations of nonfunctional requirements, our approach is well positioned to deal with all these types of changes.

6 CONCLUSION

We presented PRESTO, a method that allows autonomous systems to predict violations of nonfunctional requirements, and therefore to take preventative measures to avoid or otherwise mitigate them. Intended for use in the monitoring and analysis steps of the MAPE-K feedback control loop, PRESTO returns the time when a disruption is predicted, and can automatically trigger an adaptation in a predefined time window.

We implemented a simulator to evaluate the performance of PRESTO against one of the most widely used approaches that is currently used to tackle disruptions before they happen - regular maintenance/self-calibration. Two case studies and six different non-functional requirements from different application domains were used in the evaluation. The results from our experiments show that PRESTO is able to predict disruptions caused by the gradual degradation of internal or environmental parameters, and thus to reduce the overall self-adaptation/repair cost in comparison to the baseline model. Furthermore, the computational overheads determined from the evaluation suggest that PRESTO can efficiently operate at runtime. PRESTO benefits from the efficient parametric model checking at runtime providing formal boundaries for non-functional requirement violation. We note that the efficiency of the PRESTO prediction (owing to the fact that calculations are simple and can be completed with low overheads) is of modest benefit for the domains considered in our evaluation, as new parameter observations for the fruit-picking and assistive-care robots are only collected and processed every few minutes. Nevertheless, this efficiency may still reduce the use of key resources (e.g., energy for a battery-powered robot), and will play a major role for systems with much higher frequency of parameter observations.

In future work, we plan to explore several PRESTO extensions. First, we will investigate a technique to further reduce the runtime overheads of our method by determining the dependence, importance and variability of the pDTMC parameters with respect to the system-level properties. For example, suppose there can be no requirement violation unless one particular parameter exceeds a certain threshold, at which point the other parameters need to be considered. Then some parameters may need to be monitored less frequently and only when the most influential parameter is predicted to reach this threshold would predictions be required for the other parameters. This could significantly reduce the resources needed for parameter monitoring, and for storing and processing the observations generated by this monitoring. As all parameters' time series are

treated as piece-wise linear, this could be achieved using their current linear approximation. Furthermore, information on changes in the slope over the piece-wise linear history could be used to account for long-term trends. There is certainly a tradeoff between the cost of monitoring and the availability of sufficient reliable data when it is required. However, prior knowledge of parameter behaviour from historical data, as well as information on parameter interactions in the non-functional requirement equations, can be gained at design-time and continuously updated at run-time.

Second, we will explore possibility to learn and exploit certain types of parameter changes, such as periodic changes, or changes due to seasonal patterns. Third, information on the frequency of slope changes can allow a suitable prediction window to be determined, whilst data on the magnitude of residuals could provide a confidence interval for predictions in future work. Last but not least, we will explore the possibility to integrate into PRESTO additional time-series prediction methods, alongside methods that detect and predict other types of changes in self-adaptive systems, for example, step changes in system and/or environment parameter values due to sudden component failures [35, 70].

REFERENCES

[1] Ayman Amin, Lars Grunske, and Alan Colman. 2012. An automated approach to forecasting QoS attributes based on linear and non-linear time series modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 130–139.

[2] Suzana Andova, Holger Hermanns, and Joost-Pieter Katoen. 2003. Discrete-time rewards model-checked. In *Proceedings of the International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 88–104.

[3] Paolo Arcaini, Elvinia Riccobene, and Patrizia Scandurra. 2015. Modeling and analyzing MAPE-K feedback loops for self-adaptation. In *Proceedings of the 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 13–23.

[4] Davide Arcelli. 2020. Exploiting queuing networks to model and assess the performance of self-adaptive software systems: A survey. *Procedia Computer Science* 170 (2020), 498–505.

[5] Marco Autili, Lars Grunske, Markus Lumpe, Patrizio Pelliccione, and Antony Tang. 2015. Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar. *IEEE Transactions on Software Engineering* 41, 7 (2015), 620–638. DOI: https://doi.org/10.1109/TSE.2015.2398877

[6] Andrea Bianco and Luca de Alfaro. 1995. Model checking of probabilistic and nondeterministic systems. In *Proceedings of the Foundations of Software Technology and Theoretical Computer Science*. P. S. Thiagarajan (Ed.), LNCS, Vol. *1026*, Springer Berlin, 499–513. DOI: https://doi.org/10.1007/3-540-60692-0_70

[7] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. 2009. Engineering self-adaptive systems through feedback loops. In *Proceedings of the Software Engineering for Self-adaptive Systems*. Springer, 48–70.

[8] Radu Calinescu, Javier Cámara, and Colin Paterson. 2019. Socio-cyber-physical systems: Models, opportunities, open challenges. In *Proceedings of the IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems*. IEEE, 2–6.

[9] Radu Calinescu, Carlo Ghezzi, Kenneth Johnson, Mauro Pezzé, Yasmin Rafiq, and Giordano Tamburrelli. 2016. Formal verification with confidence intervals to establish quality of service properties of software systems. *IEEE Transactions on Reliability* 65, 1 (2016), 107–125.

[10] Radu Calinescu, Kenneth Johnson, and Colin Paterson. 2016. FACT: A probabilistic model checker for formal verification with confidence intervals. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 540–546.

[11] R. Calinescu, C. A. Paterson, and K. Johnson. 2019. Efficient parametric model checking using domain knowledge. *IEEE Transactions on Software Engineering* 47, 6 (2019), 1114–1133.

[12] Radu Calinescu, Yasmin Rafiq, Kenneth Johnson, and Mehmet Emin Bakır. 2014. Adaptive model learning for continual verification of non-functional properties. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*. 87–98.

[13] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2017. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1039–1069.

[14] Javier Cámara, Radu Calinescu, Betty H. C. Cheng, David Garlan, Bradley Schmerl, Javier Troya, and Antonio Vallecillo. 2022. Addressing the uncertainty interaction problem in software-intensive systems: Challenges and desiderata. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. 24–30.

[15] Javier Cámara, Gabriel A. Moreno, and David Garlan. 2014. Stochastic game analysis and latency awareness for proactive self-adaptation. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 155–164.

[16] Javier Cámara, Javier Troya, Antonio Vallecillo, Nelly Bencomo, Radu Calinescu, Betty H. C. Cheng, David Garlan, and Bradley Schmerl. 2022. The uncertainty interaction problem in self-adaptive systems. *Software and Systems Modeling* 21, 4 (2022), 1277–1294.

[17] A. Camilleri, S. Dogramadzi, and P. Caleb-Solly. 2022. A study on the effects of cognitive overloading and distractions on human movement during robot-assisted dressing. *Frontiers in Robotics and AI* 9 (2022), 1–17. Retrieved from https://eprints.whiterose.ac.uk/187214/

[18] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee (Eds.). 2009. *Software Engineering for Self-adaptive Systems [Outcome of a Dagstuhl Seminar]*. Lecture Notes in Computer Science, Vol. 5525. Springer. DOI: https://doi.org/10.1007/978-3-642-02161-9

[19] Houssam Chouikhi, Abdelhakim Khatab, and Nidhal Rezg. 2012. Condition-based maintenance for availability optimization of production system under environment constraints. In *Proceedings of the 9th International Conference on Modeling, Optimization and Simulation*.

[20] Frank Ciesinski and Marcus Größer. 2004. On probabilistic computation tree logic. In *Proceedings of the Validation of Stochastic Systems - A Guide to Current Research.*, Christel Baier et al. (Eds.), LNCS, Vol. *2925*, Springer, 147–188.

[21] Javier Cámara, Gabriel Moreno, and David Garlan. 2015. Reasoning about human participation in self-adaptive systems. In *Proceedings of the 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 146–156. DOI: https://doi.org/10.1109/SEAMS.2015.14

[22] Conrado Daws. 2005. Symbolic and parametric model checking of discrete-time Markov chains. In *Proceedings of the 1st International Conference on Theoretical Aspects of Computing*. 280–294.

[23] Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese (Eds.). 2017. *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*. Lecture Notes in Computer Science, Vol. 9640, Springer. DOI: https://doi.org/10.1007/978-3-319-74183-3

[24] Rogério De Lemos, David Garlan, Carlo Ghezzi, Holger Giese, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Danny Weyns, Luciano Baresi, Nelly Bencomo, Yuriy Brun, Javier Camara, Radu Calinescu, Myra B. Cohen, Alessandra Gorla, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, Raffaela Mirandola, Marco Mori, Hausi A. Müller, Romain Rouvoy, Cecília M. F. Rubira, Eric Rutten, Mary Shaw, Giordano Tamburrelli, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, and Franco Zambonelli. 2017. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In *Proceedings of the Software Engineering for Self-Adaptive Systems III. Assurances*. Springer, 3–30.

[25] Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). 2013. *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Lecture Notes in Computer Science, Vol. 7475, Springer. DOI: https://doi.org/10.1007/978-3-642-35813-5

[26] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. 2017. A storm is coming: A modern probabilistic model checker. In *Proceedings of the Computer Aided Verification*. Rupak Majumdar and Viktor Kunčak (Eds.),. Springer International Publishing, Cham, 592–600.

[27] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. 2009. Model evolution by run-time parameter adaptation. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. IEEE, 111–121.

[28] Ilenia Epifani, Carlo Ghezzi, and Giordano Tamburrelli. 2010. Change-point detection for black-box services. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 227–236.

[29] Xinwei Fang. 2018. *Improving Data Quality for Low-cost Environmental Sensors*. Ph. D. Dissertation. University of York.

[30] Xinwei Fang and Iain Bate. 2020. An improved sensor calibration with anomaly detection and removal. *Sensors and Actuators B: Chemical* 307 (2020), 127428.

[31] Xinwei Fang, Radu Calinescu, Simos Gerasimou, and Faisal Alhwikem. 2021. Fast parametric model checking through model fragmentation. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering* . IEEE, 835–846.

[32] X. Fang, R. Calinescu, C. Paterson, and J. Wilson. 2022. PRESTO: Predicting system-level disruptions through parametric model checking. In *Proceedings of the 2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Computer Society, Los Alamitos, CA, 91–97.

[33] Qianmei Feng, Lei Jiang, and David Coit. 2016. Reliability analysis and condition-based maintenance of systems with dependent degrading components based on thermodynamic physics-of-failure. *The International Journal of Advanced Manufacturing Technology* 86, 9 (2016), 913–923. DOI: https://doi.org/10.1007/s00170-015-8220-x

[34] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. 2011. Run-time efficient probabilistic model checking. In *Proceedings of the 2011 33rd International Conference on Software Engineering*. IEEE, 341–350.
[35] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. 2012. A formal approach to adaptive software: Continuous assurance of non-functional requirements. *Formal Aspects of Computing* 24, 2 (2012), 163–186.
[36] Antonio Filieri, Lars Grunske, and Alberto Leva. 2015. Lightweight adaptive filtering for efficient learning and updating of probabilistic models. In *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 200–211.
[37] Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. 2015. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering* 42, 1 (2015), 75–99.
[38] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. 2005. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*. 196–205.
[39] Miriam Gil, Vicente Pelechano, Joan Fons, and Manoli Albert. 2016. Designing the human in the loop of self-adaptive systems. In *Proceedings of the Ubiquitous Computing and Ambient Intelligence*. Carmelo R. García, Pino Caballero-Gil, Mike Burmester, and Alexis Quesada-Arencibia (Eds.), Springer International Publishing, Cham, 437–449.
[40] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. 2011. Probabilistic reachability for parametric Markov models. *International Journal on Software Tools for Technology Transfer* 13, 1 (2011), 3–19.
[41] Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5 (1994), 512–535. DOI: https://doi.org/10.1007/BF01211866
[42] Sara M. Hezavehi, Danny Weyns, Paris Avgeriou, Radu Calinescu, Raffaela Mirandola, and Diego Perez-Palacin. 2021. Uncertainty in self-adaptive systems: A research community perspective. *ACM Transactions on Autonomous and Adaptive Systems* 15, 4 (2021), 1–36.
[43] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. 2008. A framework for proactive self-adaptation of service-based applications based on online testing. In *Proceedings of the European Conference on a Service-Based Internet*. Springer, 122–133.
[44] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. 2014. Accelerating parametric probabilistic verification. In *Proceedings of the 11th International Conference on Quantitative Evaluation of Systems*. 404–420.
[45] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification*. 585–591.
[46] Guozhi Li, Fuhai Zhang, Yili Fu, and Shuguo Wang. 2019. Joint stiffness identification and deformation compensation of serial robots based on dual quaternion algebra. *Applied Sciences* 9, 1 (2019), 65.
[47] Andreas Metzger, Rod Franklin, and Yagil Engel. 2012. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *Proceedings of the 2012 Annual SRII Global Conference*. IEEE, 313–322.
[48] Andreas Metzger, Tristan Kley, and Alexander Palm. 2020. Triggering proactive business process adaptations via online reinforcement learning. In *Proceedings of the International Conference on Business Process Management*. Springer, 273–290.
[49] Andreas Metzger, Adrian Neubauer, Philipp Bohn, and Klaus Pohl. 2019. Proactive process adaptation using deep learning ensembles. In *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, 547–562.
[50] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. 2006. *Introduction to Linear Regression Analysis (4th ed.)*. Wiley & Sons.
[51] Gabriel Moreno, Alessandro Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley Schmerl. 2017. Comparing model-based predictive approaches to self-adaptation: CobRA and PLA. In *Proceedings of the 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 42–53. DOI: https://doi.org/10.1109/SEAMS.2017.2
[52] Gabriel A. Moreno. 2017. *Adaptation Timing in Self-adaptive Systems*. Ph. D. Dissertation. Carnegie Mellon University.
[53] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 1–12.
[54] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2018. Flexible and efficient decision-making for proactive latency-aware self-adaptation. *ACM Transactions on Autonomous and Adaptive Systems* 13, 1 (2018), 1–36.
[55] M. Pakdemirli, H. Boyacı, and H. A. Yurtsever. 2007. Perturbative derivation and comparisons of root-finding algorithms with fourth order derivatives. *Mathematical and Computational Applications* 12, 2 (2007), 117–124. Retrieved from https://www.mdpi.com/2297-8747/12/2/117
[56] Antonis Paschalis and Dimitris Gizopoulos. 2004. Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-aided design of integrated circuits and systems* 24,

1 (2004), 88–99.
[57] Colin Paterson and Radu Calinescu. 2018. Observation-enhanced QoS analysis of component-based systems. *IEEE Transactions on Software Engineering* 46, 5 (2018), 526–548.
[58] Vahe Poladian, David Garlan, Mary Shaw, Mahadev Satyanarayanan, Bradley Schmerl, and Joao Sousa. 2007. Leveraging resource prediction for anticipatory dynamic configuration. In *Proceedings of the 1st International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 214–223.
[59] PRESTO. 2022. Predicting Nonfunctional Requirement Violations in Autonomous System. Retrieved 27 November 2023 from https://github.com/xinwei2124/TAAS
[60] Federico Quin, Danny Weyns, Thomas Bamelis, Sarpreet Singh Buttar, and Sam Michiels. 2019. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In *Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 1–12.
[61] Yongyi Ran, Xiaoxia Zhou, Pengfeng Lin, Yonggang Wen, and Ruilong Deng. 2019. A survey of predictive maintenance: Systems, purposes and approaches. 1–36. Retrieved from https://arxiv.org/abs/1912.07383
[62] Arthur Rodrigues, Ricardo Diniz Caldas, Genaína Nunes Rodrigues, Thomas Vogel, and Patrizio Pelliccione. 2018. A learning approach to enhance assurances for real-time self-adaptive systems. In *Proceedings of the 2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 206–216.
[63] Matthew B. Russell, Evan M. King, Chadwick A. Parrish, and Peng Wang. 2021. Stochastic modeling for tracking and prediction of gradual and transient battery performance degradation. *Journal of Manufacturing Systems* 59 (2021), 663–674.
[64] Chaitanya Sankavaram, Bharath Pattipati, Anuradha Kodali, Krishna Pattipati, Mohammad Azam, Sachin Kumar, and Michael Pecht. 2009. Model-based and data-driven prognosis of automotive and electronic systems. In *Proceedings of the 2009 IEEE International Conference on Automation Science and Engineering*. IEEE, 96–101.
[65] Nikolaus Wagner, Raymond Kirk, Marc Hanheide, Grzegorz Cielniak, et al. 2021. Efficient and robust orientation estimation of strawberries for fruit picking applications. In *Proceedings of the 2021 IEEE International Conference on Robotics and Automation*. IEEE, 13857–13863.
[66] Sanford Weisberg. 2014. *Applied Linear Regression* (4th. ed.). Wiley, Hoboken NJ. Retrieved from http://z.umn.edu/alr4ed
[67] Danny Weyns, Nelly Bencomo, Radu Calinescu, Javier Camara, Carlo Ghezzi, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, Raffaela Mirandola, Marco Mori, and Giordano Tamburrelli. 2017. Perpetual assurances for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III. Assurances*, Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese (Eds.). Springer International Publishing, Cham, 31–63.
[68] Ya Xiong, Yuanyue Ge, and Pål Johan From. 2021. An improved obstacle separation method using deep learning for object detection and tracking in a hybrid visual control loop for fruit picking in clusters. *Computers and Electronics in Agriculture* 191 (2021), 106508.
[69] Ming-Yi You, Lin Li, Guang Meng, and Jun Ni. 2010. Cost-effective updated sequential predictive maintenance policy for continuously monitored degrading systems. *IEEE T. Automation Science and Engineering* 7, 04 (2010), 257–265. DOI: https://doi.org/10.1109/TASE.2009.2019964
[70] Xingyu Zhao, Radu Calinescu, Simos Gerasimou, Valentin Robu, and David Flynn. 2020. Interval change-point detection for runtime probabilistic model checking. In *Proceedings of the 2020 35th IEEE/ACM International Conference on Automated Software Engineering*. 163–174.
[71] Tao Zheng, C. Murray Woodside, and Marin Litoiu. 2008. Performance model estimation and tracking using optimal filters. *IEEE Transactions on Software Engineering* 34, 3 (2008), 391–406.
[72] Yuansheng Zhu, Weishi Shi, Deep Shankar Pandey, Yang Liu, Xiaofan Que, Daniel E. Krutz, and Qi Yu. 2021. Uncertainty-aware multiple instance learning from large-scale long time series data. In *Proceedings of the 2021 IEEE International Conference on Big Data*. IEEE, 1772–1778.

Received 8 November 2022; revised 29 September 2023; accepted 2 November 2023