
Proceedings Paper:
Calinescu, RADU CONSTANTIN, GERASIMOU, SIMOS, Getir Yaman, Sinem et al. (2026) Verification of Multi-Model Stochastic Systems. In: 48th IEEE/ACM International Conference on Software Engineering (ICSE 2026):. IEEE/ACM International Conference on Software Engineering, 15-17 Apr 2026 IEEE/ACM International Conference on Software Engineering (ICSE). IEEE, BRA.

Verification of Multi-Model Stochastic Systems

Radu Calinescu[1], Simos Gerasimou[1,2], Sinem Getir Yaman[1], Gricel Vázquez[1], Micah Bassett[1]
{radu.calinescu,simos.gerasimou,sinem.getir.yaman,gricel.vazquez,micah.bassett}@york.ac.uk
[1]Department of Computer Science, University of York, York, UK
[2]Department of Elect. Engineering, Computer Science and Engineering, Cyprus University of Technology, Cyprus

ABSTRACT

Given its ability to analyse stochastic models ranging from discrete and continuous-time Markov chains to Markov decision processes and stochastic games, probabilistic model checking (PMC) is widely used to verify system dependability and performance properties. However, modelling the behaviour of, and verifying these properties for many software-intensive systems requires the joint analysis of multiple interdependent stochastic models of different types, which existing PMC techniques and tools cannot handle. To address this limitation, we introduce a tool-supported *UniversaL stochasTIc Modelling, verificAtion and synThEsis* (ULTIMATE) framework that supports the representation, verification and synthesis of heterogeneous multi-model stochastic systems with complex model interdependencies. Through its unique integration of multiple PMC paradigms, and underpinned by a novel verification method for handling model interdependencies, ULTIMATE unifies—for the first time—the modelling of probabilistic and nondeterministic uncertainty, discrete and continuous time, partial observability, and the use of both Bayesian and frequentist inference to exploit domain knowledge and data about the modelled system and its context. A comprehensive suite of case studies and experiments confirm the generality and effectiveness of our novel verification framework.

1 INTRODUCTION

Software-intensive systems of all types, from simple computer applications and complex cyber-physical systems to sophisticated AI agents, operate under uncertainty. This uncertainty stems from factors including the *nondeterminism* inherent in their user inputs and the availability of multiple system actions to select from, the *stochasticity* of the execution times and effects of the selected actions, and the *partial observability* resulting from their use of never-perfect machine learning components to perceive the environment. To consider these factors when verifying the dependability, performance and other quality properties of such systems, software engineers are often resorting to *probabilistic model checking* (PMC) [35, 38].

There are many reasons for this frequent use of PMC for the formal modelling and verification of software-intensive systems, e.g. [8, 11, 12, 18, 26, 48]. The models supported by PMC (e.g., discrete- and continuous-time Markov chains, Markov decision processes, and partially observable Markov decision processes) capture key aspects of the uncertainty affecting such systems. The use of expressive probabilistic temporal logics [4, 5, 14, 16, 29] allows the specification of a wide range of properties over these models. The development of efficient algorithms and model checkers such as PRISM [37] and Storm [32] for verifying these properties have greatly eased the adoption of PMC across application domains [36, 38, 39]. The emergence of parametric model checking for Markov chains with transition probabilities and rewards specified as parameters [15, 19, 22, 28] supports the synthesis of probabilistic models [6, 13, 23, 34] corresponding to software system designs [6, 47] and discrete-event software controllers [9] guaranteed to meet complex sets of requirements. Last but not least, the integration of PMC with frequentist [2, 7] and Bayesian [18, 54, 55] inference enables the exploitation of expert knowledge and of data from logs and runtime monitoring to improve the accuracy of probabilistic models, and thus the validity of their verification.

This richness of the PMC landscape [33, 35] enables the verification of key quality properties for numerous software-intensive systems. Nevertheless, analysing all relevant properties of many complex systems requires the *joint use* of several of these PMC modelling, verification and synthesis methods. These systems comprise interacting components that cannot be verified entirely independently, and that exhibit a combination of discrete and continuous stochastic behaviour, nondeterminism, partial observability, etc. Despite notable research on the assume-guarantee verification of interdependent models of the same type with simple model interdependencies [10, 21, 40], the PMC of more general types of multi-model stochastic systems is currently underexplored.

To address this gap, we introduce a *UniversaL stochasTIc Modelling, verificAtion and synThEsis* (ULTIMATE) framework that supports the representation, verification and (when the selection of system actions is required) synthesis of heterogeneous multi-model stochastic systems with complex model interdependencies. As shown in Figure 1, the ULTIMATE verification engine at the core of our framework takes two inputs. The first input, called an *ULTIMATE multi-model*, is a set of $n > 1$ stochastic models of the types encountered in PMC, together with (i) a formal specification of their interdependencies, and (ii) expert knowledge, logs and runtime data to be used for the estimation of their external parameters. The second input is a formally specified property ϕ of one of these models, m_i, that needs to be verified by appropriately resolving all relevant model interdependencies, estimating the required parameters, etc. Given these inputs, the ULTIMATE verification engine produces the required verification result by (i) performing a dependency analysis of the multi-model under verification, (ii) synthesising the sequence of model analysis and parameter computation tasks required to verify the property, and (iii) invoking the combination of probabilistic and parametric model checkers, numeric solvers

ICSE '26, April 12–18, 2026,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

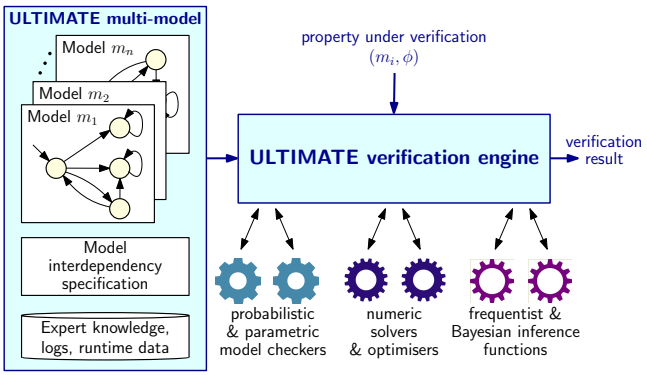

Figure 1: ULTIMATE multi-model verification

and optimisers, and frequentist and Bayesian inference functions needed to execute these tasks. Through its unique integration of multiple PMC paradigms, our ULTIMATE framework unifies—for the first time—the modelling of probabilistic and nondeterministic uncertainty, discrete and continuous time, partial observability, and the use of both Bayesian and frequentist inference to exploit domain knowledge and data about the modelled context.

The main contributions of our paper are:

1) A theoretical foundation comprising a formal definition of multi-model stochastic systems, and a novel algorithm for verifying such systems through analysing their constituent models and subsets of models in an order and through methods that consider their interdependencies and co-dependencies;
2) Tool support that implements our theoretical foundation, automating the verification of multi-model stochastic systems;
3) A suite of case studies that spans multiple application domains and types of software-intensive systems, and demonstrates the applicability and versatility of our framework.

We organised the paper as follows. Section 2 summarises the types of stochastic models that can be included into an ULTIMATE multi-model, the probabilistic temporal logics used to specify their interdependencies and properties, and the high-level modelling language adopted by our framework. Section 3 presents a motivating example, Section 4 covers our theoretical foundation, and Section 5 introduces the verification tool we implemented to automate the use of the framework. We then describe our case studies and experiments in Section 6, compare ULTIMATE to related research in Section 7, and conclude the paper with a brief summary in Section 8.

2 BACKGROUND

Stochastic models. Probabilistic model checking supports the analysis of *stochastic models* comprising *states* that abstract key aspects of the modelled system at different points in time, and *state transitions* that capture its evolution between successive states.[1] Depending on the model type, the state transitions are taken with probabilities (for *discrete-time models*) or rates (for *continuous-time models*) that reflect the stochastic nature of this evolution. Model states are labelled with *atomic propositions* representing basic properties that hold in those states; and non-negative values termed *rewards* can be assigned to states and transitions.

[1]Ensuring that the model states only capture system aspects relevant to the properties of interest—with other system aspects *abstracted out*—is essential to bound the model size so that its PMC is feasible. For example, the request queue length needs to be captured in the model of a client-server system when analysing the server's response time, while aspects like the server storage space should be abstracted out.

Table 1: Characteristics of main PMC stochastic models

Model type	Transitions	Nondeterminism	Observability	#Agents
discrete-time Markov chain (DTMC)	probabilistic	no	full	1
Markov decision process (MDP)	probabilistic	yes	full	1
probabilistic automaton (PA)	probabilistic	yes	full	1
partially observable MDP (POMDP)	probabilistic	yes	partial	1
stochastic game (SG)	probabilistic	yes	full	2+
continuous-time Markov chain (CTMC)	rate-based	no	full	1

When nondeterminism is present, multiple *actions* are possible for at least some states, and each transition is associated with the *action* that enables it. Under *partial observability*, subsets of states are indistinguishable to agents when they select their actions. When modelling multi-agent systems, each action corresponds to a specific agent, or is multi-dimensional and includes a separate action for each agent. Finally, for any state of a discrete-time model, the probabilities of the outgoing transitions associated with the same action (or of all outgoing transitions in the absence of nondeterminism) must add up to 1.

Table 1 summarises the main types of stochastic models supported by PMC, and a simple example of each model type is shown in Figure 2. Providing formal definitions of these types of stochastic models is beyond the scope of this paper; such definitions are available, for instance, in [33, 35, 38].

Property specification. PMC supports the analysis of stochastic-model properties specified formally in *probabilistic temporal logics*. These expressive logics can be used to encode software-intensive system properties as diverse as 'What is the probability that a web server will successfully handle a user request within 2s?', 'Will a software controller ensure that a mobile robot will complete its mission without crashing into obstacles with at least 0.995 probability?' and 'What software product line variant can minimise the expected execution time of a workflow (and what is this time)?'

Different logics are suited for each type of stochastic model handled by PMC. Probabilistic computation tree logic (PCTL) [29] augmented with rewards [4] is used to express the properties of discrete-time models such as DTMCs, MDPs, PAs and POMDPs (see Table 1 and Figure 2), while continuous stochastic logic (CSL) [5] extends PCTL for continuous-time models such as CTMCs. A range of useful properties of discrete-time models can also be defined in linear temporal logic (LTL) [49], and PCTL* [16], a temporal logic that combines PCTL and LTL. Finally, the properties of stochastic games are expressed in probabilistic alternating-time temporal logic with rewards (rPATL) [14]. Again, we are not providing formal

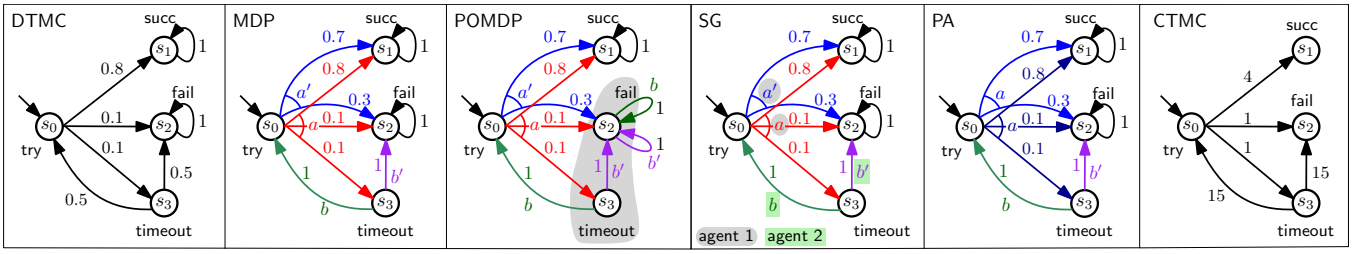

Figure 2: Examples of stochastic models from Table 1: DTMC modelling an agent's execution of a task which, being tried in state s_0, succeeds with probability 0.8 (leading to a DTMC transition to state s_1), fails with probability 0.1 (leading to a transition to state s_2), or times out with probability 0.1 (yielding a transition to state s_3, where the task is re-tried with probability 0.5, or the agent gives up and the task fails with probability 0.5); MDP modelling a variant of the agent in which two actions are available in both state s_0 (a and a') and s_3 (b and b'); POMDP modelling the scenario in which states s_2 and s_3 are indistinguishable to the agent from the MDP; SG modelling the scenario in which two different agents decide the action selected in state s_0 (a or a') and s_3 (b or b'); PA modelling the presence of two transition probability distributions for action a from state s_0; CTMC modelling the rates of transition between the states of the same agent.

definitions of these logics due to space constraints; these definitions are available from the sources cited in this paragraph.

Model representation. Stochastic models are often represented in the high-level modelling language provided by the probabilistic model checker PRISM [37]—a language that we also adopt in our ULTIMATE framework. This language, based on reactive systems [3], models a system as a parallel composition of multiple `modules`. Each such module comprises a state space defined by a set of finite-range local variables, and state transitions specified by commands with the generic form:

$$[action]\ \ guard \rightarrow e_1 : update_1 + e_2 : update_2 + \cdots + e_m : update_m;$$

where:

- *guard* is a boolean expression over the variables of all modules;
- $e_1, e_2, \ldots, e_m$ are arithmetic expressions defined over the same variables, and specifying probabilities, $\sum_{i=1}^{m} e_i = 1$, for discrete-time models, and transition rates for continuous-time models;
- $update_1, update_2, \ldots, update_m$ specify changes to the local variables of the module.

If the *guard* evaluates to true, then $update_i$, $i \in [m]$, is applied with probability e_i for discrete-time models, and with probability e_i/E, where E is the sum of all rates associated with true guards within the model, for continuous-time models.

When an *action* is specified, all modules containing commands with this action must synchronize and execute one of these rules concurrently. Rewards for states and/or transitions can be defined using the `rewards` ... `endrewards` construct.

For partially observable models, the observable subsets of states are explicitly defined using the `observable` construct

`observables` $v_i, v_j, \ldots$ `endobservables`

where $v_i, v_j, \ldots$ are model variables whose values (and associated model states) are observable, with the values of all other model variables being unobservable. State observability affects how the system is verified, particularly when reasoning about the probability of certain outcomes, or synthesising policies of a modelled agent.

Finally, a stochastic game requires the specification of the *players* (i.e., agents) and their control variables using the construct

`player` *pname mname* $[a_1, a_2, \ldots, a_n]$ `endplayer`.

where *pname* is the name of a player that executes the module *mname* and controls the actions $a_1, a_2, \ldots, a_n$.

3 MOTIVATING EXAMPLE

To motivate the need for the probabilistic model checking of multi-model stochastic systems, we consider the PMC of a robot assistive dressing (RAD) system. This cyber-physical system belongs a domain of growing societal importance due to the significant increase in demand for assistive care driven by an ageing population worldwide [46, 53]. The RAD system uses a robotic arm to help a person with restricted mobility to dress with a garment such as a coat, enabling them to live independently at home instead of moving into a care home. We want to use PMC to synthesise a policy that minimises the system's probability of failing to accomplish the dressing procedure. This requires the joint modelling and analysis of several RAD components and processes:

1. the picking of the garment by the robot (from a nearby peg);
2. the monitoring of the user by a deep-learning perception component, to determine whether the user is ok or not;
3. the control component that configures the user monitor;
4. the dressing process, which depends on the successful picking of the garment, and must adapt to the (perceived) user state.

As shown in Figure 3, we need four interdependent stochastic models of several different types to capture these RAD aspects.

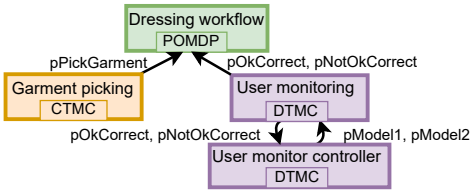

Figure 3: RAD stochastic models and dependencies

```
1  ctmc
2
3  const double rPick;  // garnment picking rate
4  const double psucc;  // successful picking probability
5  const double pRetry; // retry probability
6
7  module pickGarment
8    s : [0..2] init 0;
9
10   [try]  s=0 → psucc*rPick:(s'=1) +
11                (1−psucc)*rPick*pRetry:(s'=0) +
12                (1−psucc)*rPick*(1−pRetry):(s'=2);
13   [succ] s=1 → 1:(s'=1);
14   [fail] s=2 → 1:(s'=2);
15 endmodule
16
17 label "success" = s=1;
```

Listing 1: CTMC modelling the garment picking task

Garment picking. One of the essential tasks of the RAD dressing is the picking of the garment by the robotic arm. To avoid lengthy dressing sessions (which may cause distress to the user), we are interested in the (probability of) successful completion of this task within a 90s time period. A CTMC (which supports the verification of such time-based properties) is therefore used to model the execution of this task. This CTMC (presented in Listing 1) models a robotic arm that tries to pick the garment with a rate rPick, a probability psucc of succeeding in each attempt, and a probability pRetry of retrying the garment picking after an unsuccessful attempt (lines 10–12). If the try is successful, the robotic arm transitions from the initial state s=0 to state s=1 (labelled with the atomic proposition "success" in line 17). Otherwise, it stays in state s=0 for a retry with probability pRetry, or gives up and transition to the fail state s=2 with probability $1 - \text{pRetry}$.

The probabilistic model checking of this CTMC can be used to establish the probability pPickGarment of successful garment picking within 90s by verifying the CSL property

$$\text{pPickGarment} = P_{=?}[F^{[0,90]} \text{ "success"}] \tag{1}$$

We note that the (external) CTMC parameters rPick and psucc (lines 3 and 4) depend on the environment in which the robotic arm is deployed (e.g., on the position of the peg where the garment is located initially, and on the level of lighting from the area). As such, their values need to be derived from experimental data obtained during the preliminary testing of the robotic arm.

User monitoring. This RAD component uses machine learning (ML) perception to classify the user as content with the ongoing dressing (i.e., ok) or not (notok). The component comprises:

- a medium accuracy but fast user-state classifier (ML model 1);
- a high accuracy, but slower and computationally expensive user-state classifier (ML model 2);
- a *verifier* that, given an (input, output) pair from model 1, returns true if the output is very likely correct, and false otherwise (see [9] for examples of such verifiers for DNN classifiers).

We use the DTMC in Listing 2 to model the operation of this user monitoring component, and to establish its probability of predicting each user state correctly. The external parameters of this DTMC reflect the accuracy of the two ML models (lines 3–6), e.g., p1_ok_correct represents the probability that model M1 makes a correct prediction when the true user state is ok. The dependency

```
1  dtmc
2
3  // parameters of ML model 1/model 2/model 1 with verification
4  const double p1_ok_correct;
5  const double p1_notok_correct;
6  ...
7  // Probabilities of using ML model 1 or 2
8  const double pModel1;
9  const double pModel2;
10
11 module UserPerception
12   s : [0..4] init 0;       // step
13   m : [1..3];              // operation mode
14   ok : bool;               // true user state
15   predOk : bool;           // predicted user state
16   verified : bool;         // verifier result
17
18   [] s=0 → 0.5:(s'=1)&(ok'=true) + 0.5:(s'=1)&(ok'=false);
19   [] s=1 → pModel1:(s'=2)&(m'=1) + pModel2:(s'=2)&(m'=2) +
20             (1−pModel1−pModel2):(s'=2)&(m'=3);
21
22   // Operation mode 1: use only ML model 1
23   [] s=2 & ok & m=1 → p1_ok_correct:(s'=4)&(predOk'=true)
24                     + (1−p1_ok_correct):(s'=4)&(predOk'=false);
25   [] s=2 & !ok & m=1 → p1_notok_correct:(s'=4)&(predOk'=false)
26                     + (1−p1_notok_correct):(s'=4)&(predOk'=true);
27
28   // Operation mode 2: use only ML model 2
29   ...
30   // Operation mode 3: use both ML models
31   ...
32
33   [done] s=4 → true;
34 endmodule
35
36 label "done" = s=4;
```

Listing 2: DTMC model of user monitoring component

parameters pModel1 and pModel2 specify the probabilities of using different ML models to monitor the user state. The ML model 1 is used with probability pModel1 (by setting m=1 in line 19, and therefore moving to the commands from lines 22–26), ML model 2 with probability pModel2 (by setting m=2 in line 19, and therefore moving to the commands starting in line 28), and both ML models are used as follows by m=3 with probability $1 - \text{pModel1} - \text{pModel2}$ in line 20: first model 1 generates a prediction, which serves as the output of the monitor if its verification by the verifier yields true, or otherwise model 2 is additionally used and its prediction becomes the output of the monitor.

PMC applied to this DTMC and the PCTL-encoded properties below can be used to establish the probabilities that the user state is correctly predicted when the user's true state is ok and notok:

$$\begin{aligned} \text{pOkCorrect} &= P_{=?}[F \text{ "done"} \wedge \text{ok} \wedge \text{predictedOk}] \\ \text{pNotOkCorrect} &= P_{=?}[F \text{ "done"} \wedge \neg\text{ok} \wedge \neg\text{predictedOk}] \end{aligned} \tag{2}$$

User monitor controller. As shown by its DTMC model from Listing 3, this RAD component uses the probabilities (2) with which the user monitor outputs a true positive/negative, first to calculate the monitor's F1 score metric. and then to determine the probabilities with which the monitor should use its ML models 1 and 2:

$$\text{pModel1} = P_{=?}[F\ s = 1],\ \text{pModel2} = P_{=?}[F\ s = 2] \tag{3}$$

Dressing workflow. The execution of a dressing session is modeled as a POMDP (Listing 4) because the true status of the user (ok=true or ok=false) is not observable. As such, oK does not appear in the list of observable variables from line 10. The dressing starts

```
1  dtmc
2
3  const double pOkCorrect;
4  const double pNotOkCorrect;
5  const double precision =
6                 pNotOkCorrect/(0.5+pNotOkCorrect-pOkCorrect);
7  const double recall = 2*pNotOkCorrect;
8  const double F1 = 2*precision*recall/(precision+recall);
9
10 module ModelSelector
11   s : [0..5] init 0;
12   [] s=0 -> 0.47*F1:(s'=1)+(1-0.82*F1):(s'=2)+0.35*F1:(s'=3);
13   [] s>0 -> true;
14 endmodule
```

Listing 3: DTMC model of user monitor controller

with the garment picking by the robot (s=1 in line 17). This succeeds with probability pPickGarment given by (1), and is followed by observing the user state, with the (unobservable) true user state ok and the predicted user state predOk appropriately set depending on the probability pOk that the user is in an ok state, and of the probabilities (2) of correctly predicting this user state (lines 18–21). Next, the system must decide between two actions (lines 23–31):

- dressSlow, which performs the user dressing slowly, taking longer but having a better chance of success if the user is notok;
- dressFast, which performs the dressing fast, but with a higher failure probability if the user is not ok.

Depending on the selected action (i.e., on the POMDP *policy*) and the user state, the dressing may succeed—in which case the success state in line 34 is reached, or not—in which case a retry is possible if allowed by the user (line 33). The workflow fails (line 35) if either the robot cannot pick the garment (line 17) or the user disagrees to dressing being retried after an unsuccessful attempt (line 33).

We note that the parameters pPickGarment, pOkCorrect and pNotOkCorrect of the POMDP take values obtained through the PMC of two other stochastic models presented in this section, reflecting the dependency of the POMDP on these models. All other POMDP parameters are external parameters. Finally, the choice between a dressSlow/dressFast action (i.e., the synthesis of a POMDP policy) is made by optimising a PCTL-encoded objective such as

$$\text{pFailMin} = Pmin_{=?}[F\ \text{step=6}], \tag{4}$$

which requests the minimisation of the workflow failure probability.

4 THEORETICAL FOUNDATION

4.1 Problem definition

Our ULTIMATE framework supports the joint verification of sets of heterogeneous, interdependent stochastic models, such as the RAD models from Section 3. We refer to these as *multi-model stochastic systems*, and provide a definition below.

Definition 1. A multi-model stochastic system is a tuple

$$U = (M, D, E), \tag{5}$$

where:

- $M = \{m_1, m_2, \ldots, m_n\}$ is a set of $n > 1$ stochastic models such that the transition probabilities/rates and rewards of each model $m_i \in M$ are defined by rational functions over a set $D_i = \{d_1^i, d_2^i, \ldots, d_{k_i}^i\}$ of $k_i \geq 0$ *dependency parameters* and a set $E_i = \{e_1^i, e_2^i, \ldots, e_{l_i}^i\}$ of $l_i \geq 0$ *external parameters*;
- $D = \{(m_i, d, m_j, \phi) \mid m_i, m_j \in M \wedge d \in D_i \wedge d = pmc(m_j, \phi)\}$ represents the set of *dependency relationships* between the n models, with each element $(m_i, d, m_j, \phi) \in D$ specifying that the value of the dependency parameter $d \in D_i$ of m_i is to be obtained by applying PMC to the property ϕ of model m_j;
- $E = \{(m_i, e, f, O) \mid m_i \in M \wedge e \in E_i \wedge e = f(O)\}$ represents the set of *external-parameter inferences*, with each element $(m_i, e, f, O) \in E$ specifying that the value of the external parameter $e \in E_i$ of m_i is to be obtained by applying the (frequentist or Bayesian) inference function f to the *observations* (i.e., log entries or runtime measurements) O.

```
1  pomdp
2
3  const double pPickGarnment; // dependency parameter eq. (1)
4  const double pOkCorrect;    // dependency parameter eq. (2)
5  const double pNotOkCorrect; // dependency parameter eq. (2)
6  const double pOk;           // probability that user is ok
7  const double pDressOkSlow;  // slow-dress success prob/user ok
8  const double pDressOkFast;  // fast-dress success prob/user ok
9  ...
10 observables s, predOk endobservables
11
12 module Workflow
13   s : [1..6] init 1;        // workflow step
14   ok : bool init true;      // true user state (hidden)
15   predOk : bool init true; // predicted user state
16
17   [pick] s=1 -> pPickGarnment:(s'=2) + (1-pPickGarnment):(s'=6);
18   [obsv] s=2 -> pOk*2*pOkCorrect:(s'=3) +
19      pOk*(1-2*pOkCorrect):(s'=3)&(predOk'=false)+
20      (1-pOk)*2*pNotOkCorrect:(s'=3)&(ok'=false)&(predOk'=false)+
21      (1-pOk)*(1-2*pNotOkCorrect):(s'=3)&(ok'=false);
22
23   // dress slowly (higher success prob., longer time) or fast
24   [dressSlow] s=3 & ok -> pDressOkSlow:(s'=5) +
25                           (1-pDressOkSlow):(s'=4);
26   [dressFast] s=3 & ok -> pDressOkFast:(s'=5) +
27                           (1-pDressOkFast):(s'=4);
28   [dressSlow] s=3 & !ok -> pDressNotOkSlow:(s'=5) +
29                            (1-pDressNotOkSlow):(s'=4);
30   [dressFast] s=3 & !ok -> pDressNotOkFast:(s'=5) +
31                            (1-pDressNotOkFast):(s'=4);
32
33   [retry] s=4 -> pRetryAllowed:(s'=3)+(1-pRetryAllowed):(s'=6);
34   [succ] s=5 -> true;
35   [fail] s=6 -> true;
36 endmodule
```

Listing 4: Dressing workflow modelled as a POMDP

The dependency relationships D of a multi-model stochastic system (5) can express practical inter-model dependencies in which the behaviour—and therefore the properties—of one model depend on any of the following classes of properties of another model:

- Performance properties: response time, throughput, etc.
- Dependability properties: reliability, availability, etc.
- Cost properties: monetary cost, energy use, risk incurred, etc.
- Utility properties: tasks completed, profit, etc.

As an example, the *reliability* of the robot-assisted dressing workflow from our motivating example (modelled by the POMDP in Listing 4) depends on the garment being *available* in the first workflow step (line 17), having been picked successfully by the robot arm (modelled by the CTMC in Listing 1). As another example (taken

from the DPM-FX case study we present later in Section 6), the *response time* of a service-based system (whose behaviour is modelled by a DTMC) depends on the *throughput* of a database residing on a dynamically power-managed hard disk (whose behaviour is modelled by a CTMC). All classes of properties listed above can be specified in the rewards-augmented probabilistic temporal logics used by probabilistic model checkers [35, 38, 39], and are therefore supported by our framework.

Example 1. The multi-model stochastic system for the RAD cyber-physical system from Section 3 is $U_{RAD} = (M, D, E)$, where:

- $M = \{m_{gp}, m_{um}, m_{umc}, m_{dp}\}$ is the set of stochastic models comprising the garment picking CMTC in Listing 1 (m_{gp}), the user monitoring DTMC in Listing 2 (m_{um}), the user monitor controller DTMC in Listing 3 (m_{umc}), and the dressing workflow POMDP model from Listing 4 (m_{dp});
- D contains elements that define the 7 dependency relationships from Figure 3; for example, (m_{dp}, pPickGarment, m_{gp}, $P_{=?}[F^{[0,90]}$ "success"]) $\in D$ specifies that the dependency parameter pPickGarment of m_{dp} needs to be calculated by applying PMC to property (1) of m_{gp};
- E contains elements that specify the inference functions and observations to be used for estimating the external parameters of the RAD models, e.g., (m_{gp}, rPick, *meanRate*, $\{47, 92, 61, \ldots\}$) to specify that the external parameter rPick of m_{gp} should be approximated as the mean rate of a process with the observed durations from the set $\{47, 92, 61, \ldots\}$. Other options for determining the value of an external parameter include using a Bayesian estimator to derive a posterior value from a prior value and a set of observations, or simply using a predefined value.

Given a multi-model stochastic system $U = (M, D, E)$ and a formally specified property ϕ_v of a model $m_v \in M$, the verification problem tackled by our ULTIMATE framework is to establish the value of property ϕ_v, i.e., to compute $pmc(m_v, \phi_v)$. Solving this verification problem is challenging as it may require: (i) the PMC of additional models from M to establish the value of the dependency parameters of model m_v, which (as detailed in Section 4.2) is particularly complex in the presence of circular model interdependencies, (ii) the synthesis of suitable policies for any verified models (e.g., MPDs and POMDPs) that contain nondeterminism, and (iii) the estimation of the external parameters of all models involved (directly or through model interdependencies) in the verification.

4.2 ULTIMATE verification algorithm

Before presenting our algorithm for the verification of ULTIMATE multi-model stochastic systems, we need to define two concepts.

Definition 2. The *dependency graph* induced by an ULTIMATE multi-model stochastic system $U = (M, D, E)$ is the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{1, 2, \ldots, n\}$ comprising a vertex for each model $m \in M$, and edge set $\mathcal{E} = \{(i, j) \mid \exists (m_i, d, m_j, \phi) \in D\}$.

Definition 3. Given an ULTIMATE multi-model stochastic system $U = (M, D, E)$ and its induced dependency graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *strongly connected component* associated with model $m_i \in M$ is the subset of models $SCC(m_i) = \{m_j \mid m_j \in M \wedge j \in \mathcal{V}_i\}$, where $\mathcal{V}_i \subseteq \mathcal{V}$ is the set of vertices from the strongly connected component of G that includes vertex i.

Example 2. The vertices and edges of the dependency graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ induced by the RAD multi-model stochastic system U_{RAD} from Example 1 are $\mathcal{V} = \{1, 2, 3, 4\}$ (corresponding to the four U_{RAD} models) and $\mathcal{E} = \{(1, 4), (2, 3), (2, 4), (3, 2)\}$ (corresponding to the arrows from Figure 3). Accordingly, the strongly connected components associated with the four models are $SCC(m_{gp}) = \{m_{gp}\}$, $SCC(m_{um}) = SCC(m_{umc}) = \{m_{um}, m_{umc}\}$ and $SCC(m_{dp}) = \{m_{dp}\}$.

We note that the set of strongly connected components

$$SCC = \{SCC(m_1), SCC(m_2), \ldots, SCC(m_n)\} \tag{6}$$

associated with all models of a multi-model stochastic system $U = (M, D, E)$ can be computed efficiently in linear time. This computation involves first assembling the dependency graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of U (in linear, $O(\#M + \#D)$ time, by examining each model from M and each dependency from D once), and then applying Tarjan's strongly connected component detection algorithm [51] (which takes linear, $O(\#V + \#E)$ time) to this graph.

Having defined these concepts, we can now present our ULTIMATE verification process, which is carried out by function VERIFY from Algorithm 1. This function takes four arguments—the multi-model stochastic system $U = (M, D, E)$, model $m_v \in M$ and property ϕ_v to be verified, and U's set SCC of strongly connected components (6)—and performs the verification of property ϕ_v of model m_v in the three stages described below.

Stage 1: In this stage, the for loop in lines 2–11 iterates through each model m from the strongly connect component $SCC(m_v)$ associated with the model under verification, using:

- the inner for loop in lines 3–7 to extract each dependency parameter d of m (line 3) and, if the origin of the dependency is a model m' external to the strongly connected component (line 4), to obtain the value of that parameter by invoking VERIFY recursively, and to set the parameter d to that value (line 5);
- the inner for loop in lines 8–10 to extract each external parameter e of m (line 8), and to obtain and set the value of that parameter according to its specification from Definition 1.

Stage 2: In this stage, the if statement in lines 12–14 resolves any co-dependencies among the models of the strongly connected component $SCC(m_v)$ associated with the model under verification. To that end, the function VERIFYSCC is invoked in line 13 if $SCC(m_v)$ is not a "degenerate" strongly connected component containing only model m_v. Given a strongly connected component M' comprising models whose dependency parameters depend on models from outside M' and external parameters are already computed, lines 19–32 of this function assemble a system of equations whose solution in line 33 provides the values for setting the remaining dependency parameters of the models from M' in lines 34–36. The system of *Equations* solved in line 33 is assembled by iterating through every model from M' (line 20) and dependency parameter of that model (line 21), and adding a new *equation* to *Equations* if the origin of the dependency is a model from the strongly connected component M' (line 22). This *equation* can take one of two forms. If *all* dependency

Algorithm 1 Verification of multi-model stochastic systems

```
1: function Verify((M, D, E), m_v, φ_v, SCC)
2:     for m ∈ SCC(m_v) do
3:         for (m, d, m', φ) ∈ D do
4:             if m' ∉ SCC(m_v) then
5:                 SetParam(m, d, Verify((M, D, R), m', φ, SCC)
6:             end if
7:         end for
8:         for (m, e, f, O) ∈ E do
9:             SetParam(m, e, f(O))
10:        end for
11:    end for
12:    if SCC(m_v) ≠ {m_v} then
13:        VerifySCC((M, D, E), SCC(m_v))
14:    end if
15:    return PMC(m_v, φ_v)
16: end function
17:
18: function VerifySCC((M, D, E), M')
19:    Equations ← {}, ParamList ← {}
20:    for m ∈ M' do
21:        for (m, d, m', φ) ∈ D do
22:            if m' ∈ M' then
23:                if ParametricVerifFeasible(M', D) then
24:                    equation ← 'd = ParametricMC(m', φ)'
25:                else
26:                    equation ← 'd = pmc(m', φ)'
27:                end if
28:                Equations ← Equations ∪ {equation}
29:                ParamList ← ParamList ∪ {(m, d)}
30:            end if
31:        end for
32:    end for
33:    Solution ← Solve(Equations)
34:    for (m, d) ∈ ParamList do
35:        SetParam(m, d, Solution(d))
36:    end for
37: end function
```

parameters between pairs of models from M' (i.e., all parameters for which an equation needs to be assembled) require the verification of a property that can be analysed using parametric model checking, then the auxiliary function ParametricVerifFeasible returns true,[2] and parametric model checking is used to obtain an algebraic *equation* in line 24. Otherwise, line 26 creates an *equation* whose right-hand side (i.e., $pmc(m', \phi)$) can only be evaluated by applying probabilistic model checking to instances of m' whose unknown dependency parameters are set to specific combinations of values.

Depending on whether the *Equations* are of the first or the second form, the Solve function from line 33 employs numeric solvers and optimisers as appropriate (see Figure 1). In particular, when these *Equations* contain algebraic formulae, Solve resorts to solving a system of nonlinear equations either analytically (e.g., via algebraic manipulation) or numerically (e.g., using the Newton-Raphson optimisation method [45]). When ParametricVerifFeasible returns false and probabilistic model checking is used, the Solve function leverages derivative-free optimization (e.g., Powell's optimization method [50]) to estimate the dependency parameter values by minimising the sum of squared residuals between those values and the outcome of $pmc(m', \phi)$.

Stage 3: Finally, with all dependency and external parameters of model m_v resolved, the final stage of Verify invokes a probabilistic model checking engine to obtain and return the required verification result in line 15.

4.3 Correctness of the verification algorithm

To demonstrate the correctness of Algorithm 1, we need the following definition.

Definition 4. Consider a multi-model stochastic system $U = (M, D, E)$, and the directed acyclic graph (i.e., the dag) comprising a vertex for each strongly connected component of U and an edge between each pair of vertices (v, v') for which the strongly connected component associated with v' contains a model with a dependency parameter defined in terms of a property of a model from the strongly connected component associated with v. The *dependency level* of a model $m \in M$ is defined as the length of the longest path from a vertex of this dag to the vertex associated with $SCC(m)$, the strongly connected component of m.

Example 3. Consider the multi-model stochastic system U_{RAD} from Example 1. As discussed in Example 2, U_{RAD} has three strongly connected components: $\{m_{gp}\}$, $\{m_{um}, m_{umc}\}$ and $\{m_{dp}\}$. As such, its dag from Definition 4 will have three vertices, v_1, v_2 and v_3, associated (in order) with these strongly connected components, and (given the model dependencies depicted in Figure 3) the edges (v_1, v_3) and (v_2, v_3). Models m_{gp}, m_{um} and m_{umc} belong to strongly connected components associated with vertices v_1 and v_2, for which the maximum path length from another vertex in the dag is 0. Therefore, their dependency level is 0. In contrast, model m_{dp} belongs to the strongly connected component associated with vertex v_3, for which the maximum path length from another vertex in the dag is 1, and thus the dependency level of this model is 1.

Theorem 1. *Function Verify from Algorithm 1 terminates and returns the correct value of property ϕ_v of stochastic model m_v.*

Proof. We prove this result by induction over the dependency level of m_v. For the **base step**, which corresponds to m_v having dependency level 0, we need to consider two cases.

The **first base case** is when m_v belongs to a 'degenerate" strongly connected component, i.e., $SCC(m_v) = \{m_v\}$. The garment picking CTMC from Figure 3 is an example of such a model. In this case, the for loop in lines 2–11 is only executed once, for $m = m_v$. This execution skips the inner for loop from lines 3–7 (since m_v has no dependency parameter), and sets all the external parameters of m_v in the for loop from lines 8–10. The function then moves directly to computing the property ϕ_v of model m_v in line 15, since the if statement condition from line 12 does not hold. As m_v has no dependency parameters and all its external parameters were already

[2] At the time of writing, parametric model checking can handle DTMC properties without inner probabilistic operators, and non-transient CTMC properties, so this is what the auxiliary function ParametricVerifFeasible (which we do not include due to space constraints) needs to check.

set, this computation is feasible, and its result—the correct value of ϕ_v—is obtained and returned, ending the execution of VERIFY.

The **second base case** is when m_v belongs to a strongly connected component that also contains additional models, i.e., $SCC(m_v) \setminus \{m_v\} \neq \emptyset$. The user monitoring DTMC from Figure 3 is an example of such a model. In this case, the outer for loop in lines 2–11 iterates through every model $m \in SCC(m_v)$. Each such iteration: (i) skips the recursive invocation of VERIFY from line 5 because the origin model m' of every dependency parameter is in $SCC(m_v)$, so the condition of the if statement from line 4 is false; and (ii) calculates and sets all the external parameters of m in the for loop from lines 8–10. With all these parameters resolved, VERIFYSCC is called in line 13, as $SCC(m_v) \neq \{m_v\}$. As a result, each dependency parameter of these models is visited precisely once by the two nested for loops from lines 20–32, so that a system of equations comprising as many equations as there are dependency parameters is assembled. The relevant solution to these equations (obtained in line 33) is used to set the values of all dependency parameters of all the models from $SCC(m_v)$ in lines 34–36, after which VERIFYSCC exits, enabling the VERIFY function to then obtain and return the value of property m_v in line 15, and to terminate.

For the **inductive step**, we assume that the theorem holds for all verified models with dependency level up to $N \geq 0$, and we consider the verification of a model m_v with dependency level $N + 1$. In this case, every dependency parameter d of m_v and of any other model from $SCC(m_v)$ whose origin is a model m' from outside $SCC(m_v)$ is visited once by the two nested for loops starting in lines 2 and 3, and is computed through a recursive VERIFY invocation in line 5. Since the dependency level of model m' is smaller than that of m_v, and thus between 0 and N, our assumption implies that this recursive invocation of VERIFY terminates and returns the correct verification result required to set the dependency parameter d in line 5. Hence, all dependency parameters of models from $SCC(m_v)$ that depend on models from outside $SCC(m_v)$ are correctly set by the for loop in lines 3–7, all external parameters of these models are calculated and set by the for loop from lines 8–10, and (for the same reasons as in the second base case) the dependency parameters that depend on models from within $SCC(m_v)$ are computed and set, if needed because $SCC(m_v) \neq \{m_v\}$, by the invocation of VERIFYSCC from line 13. As such, the verification of property ϕ_v of m_v can be performed successfully in line 15, and the algorithm terminates—which completes the proof of the inductive step. □

We end this section by noting that a formal complexity analysis of our verification algorithm is not possible because of its use of multiple reasoning engines whose configuration can influence their execution time significantly. These include (Figure 1) probabilistic and parametric model checkers (lines 15 and 24), numeric solvers and optimisers (line 33), and frequentist and Bayesian inference functions (line 9). To compensate for this, we provide an empirical assessment of the algorithm performance in Section 6.

5 IMPLEMENTATION

To support the use and adoption of our verification framework, we developed an open-source ULTIMATE verification tool [52] with the architecture from Figure 1. At the core of this tool is a Java implementation of the verification functions from Algorithm 1, and

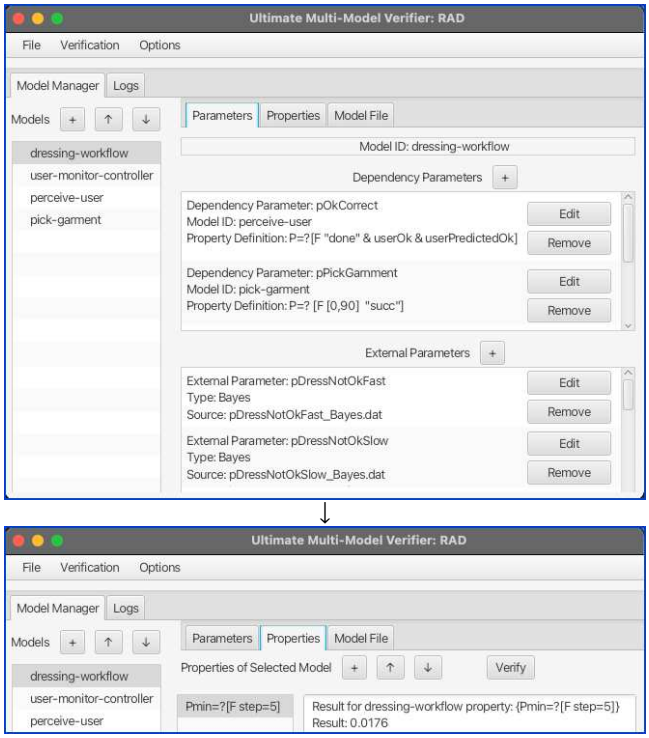

Figure 4: RAD system verification using the ULTIMATE tool

a graphical user interface (Figure 4) that enables users to: (i) define ULTIMATE multi-model stochastic systems (with their component stochastic models, dependency parameters, and external parameters); (ii) specify properties of these systems that require verification; (iii) run verification sessions; and (iv) examine both end and intermediate verification results.

Our verification tool can handle multi-model stochastic systems comprising combinations of all the model types from Table 1, with each dependency parameter defined in the rewards-extended probabilistic temporal logic(s) allowed by the type of the model whose analysis determines the parameter value (i.e., by the type of the model m_j from Definition 1). To that end, the tool invokes as needed: the probabilistic and parametric model checkers PRISM [37] and Storm [32], and the stochastic games model checker PRISM-games [42]; the numeric solvers and optimisers from the Scipy open-source optimisation library; and our Java implementations of the KAMI Bayesian estimator from [18, 27] and of simple (frequentist) mean functions.

6 CASE STUDIES AND EXPERIMENTS

We carried out experiments to answer three research questions:

RQ1: To what extent does the ULTIMATE framework support the verification of diverse software-intensive systems that cannot be modelled (or cannot be modelled conveniently) using a unified modelling formalism?

RQ2: To what extent does the framework enable the specification of a diverse range of classes of practical inter-model dependencies associated with the components of software-intensive systems?

RQ3: How does the ULTIMATE verification scale when the (i) number of models, (ii) number of dependency levels, (iii) number of model interdependencies, and/or (iv) model size increase?

To answer research questions RQ1 and RQ2, we used the ULTIMATE framework and tool in four case studies covering software-intensive systems from a broad range of application domains. To answer research question RQ3, we carried out separate scalability experiments. These case studies are described below (and summarised in Figure 5 and Table 2), followed by a presentation of the scalability experiments. Further details and supporting material (models, verified properties, external parameter datasets, descriptions) are available in our public online repository [52].

Robot assistive dressing (RAD). This case study involved the verification of the RAD system from our motivating example introduced in Section 3. We verified the property (4) of the dressing-process POMDP from Listing 4 for different values of:

- the *pOk* probability that the user state is "ok" in this POMDP,
- the garment-picking CTMC's *pRetry* probability (Listing 1),

synthesising RAD controller policies that minimise the failure probability of the dressing process for these combinations of parameter values—see Figure 6a.

Smart motion detection (SMD). Our second case study verifies a cyber-physical system comprising two co-dependent components: a night-time motion sensor, and a smart lighting component. The motion sensor is activated every few seconds, and triggers an alarm if it detects a moving object. This detection happens with a probability that depends on two factors. The first factor is the actual scenario encountered when the sensor is activated: (i) large, relevant moving object (intruder, cat, dog, etc.) present in the monitored area, (ii) small, irrelevant object (e.g., leaves moving due to wind) present, or (iii) no moving object present. The second factor is the level of lighting in the monitored area. This can be low (with probability pLow), medium (with probability pMed), or high (with probability pHigh = 1 − pLow − pMed), where these three probabilities are determined by the operation of the smart lighting component. This component decides a (potentially new) level of lighting every half minute, based on (i) the current level of lighting, and (ii) the output of the motion sensor (i.e., moving object detected or not) when the decision is made. As such, the lighting level depends on the probability `pDetect` that a moving object is detected by the motion sensor. With the behaviour of each component modelled as a DTMC (m_{ms} for the motion sensor, and m_{sl} for the smart lighting), we assembled the ULTIMATE multi-model stochastic system shown in Table 2, and used it to establish (Figure 6b) (i) the probability that a large object is detected by the SMD system, and (ii) the power consumption for the smart lighting component, for a range of probabilities that a large object is present in the monitored area.

Mobile robot fleet (RoboFleet). In this case study, we consider a human-supervised fleet of N mobile robots. Each robot performs an independent mission within a grid world in which it can move up, down, left or right between locations, subject to remaining within the bounds of the grid world, and to not entering locations that contain obstacles. The mission involves navigating from an initial location to a goal location where the robot needs to perform a task. At each location, the robot may become stuck with a small probability that depends of the distance between that location and any nearby obstacles. We use separate MDPs ($m_{r_1}, m_{r_2}, \ldots, m_{r_N}$) to model the robots and their missions, and to derive the maximum mission success probabilities pR1, pR2, …, pRN, and the associated optimal MDP policies, which correspond to the sequences of movements that the robots should use to maximise their mission success. A human supervisor whose operation depends on these probabilities is monitoring the robot fleet, trying to unstuck blocked robots, first by attempting to remotely manoeuvre them (for a maximum of nAttempts attempts), and then by performing a hard reset. The first approach has a low cost but only a medium probability of success, while the hard reset has a high cost and higher probability of success. We model the supervisor's operation using a DTMC m_{sup} augmented with reward structures enabling the verification of the expected number of failed robots and supervisor-intervention cost across all missions performed by the robot fleet. Figure 6c shows how these RoboFleet properties vary when the external parameter nAttempts is varied between 1 and 4 for a fleet with $N = 2$ robots.

Dynamic power management (DPM) for foreign exchange (FX) system. The DPM-FX case study considers the FX service-based system from [19, 25], which performs automated financial transactions in the foreign exchange market, and the database used by the FX services tasked with the sophisticated market analyses behind this system's decision making. To reduce power usage, the hard disk storing this large database uses a DPM component with the characteristics from [43]. The role of this component is to switch the hard disk to a low-power "sleep" mode of operation when the disk is idle (with a configurable probability `pIdle2Sleep`). We model the joint behaviour of FX and DPM using two co-dependent stochastic models: a DTMC m_{fx} (taken from [19]) and a CTMC m_{dpm} (adapted from [43]), respectively. In this co-dependency, the average length of DPM's queus of disk operations avrQueueDiskOps influences the number of transactions that can be performed by FX, and therefore the number of disk operations diskOps that FX sends to the database managed by DPM. We use the multi-model stochastic system formed by these two models and their interdependencies (Figure 5) to verify how the expected execution time of the FX workflow varies for different predefined values of the external DPM parameter pIdle2Sleep (Figure 6d).

Scalability experiments. To evaluate the scalability of the ULTIMATE framework, we performed experiments involving the verification of the reachability property $P_{=?}[F \text{ success}]$ (representing the probability of successful workflow execution) for:

- variants of the FX system model from our DPM-FX case study with: 283 states (small); 41873 states (large); and 529239 states (very_large);
- stochastic multi-models comprising: 3 FX models, 1 dependency level, 3 dependencies (experiment_1); 11 FX models, 2 dependency levels, 10 dependencies (experiment_2); and 75 FX models, 3 dependency levels, 74 dependencies (experiment_3).

These verification times required to complete each of these experiments (on the MacBook with the specification provided at the bottom of Table 2) are reported in Table 3.

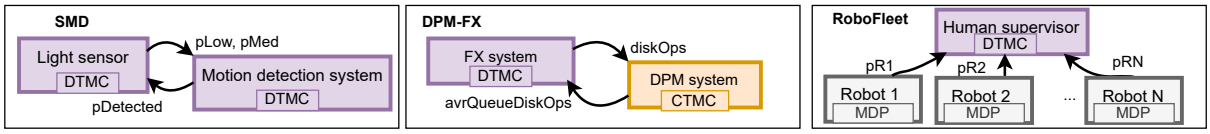

Figure 5: Stochastic models and dependencies for the SMD, DPM-FX and RoboFleet case studies

Table 2: Case study summary (see [52] for further details and supporting material)

ID	Multi-model stochastic system	Sample verified model : property†	Time‡ (s)
RAD	U_{RAD} multi-model from Example 1	$m_{dp} : Pmin_{=?}[F \text{ step=6}]$	3.10s
SMD	$U_{SMD} = (M, D, E)$, where: $-\ M = \{m_{ms}, m_{sl}\}$ $-\ D = \{(m_{ms}, \text{pLow}, m_{sl}, R^{\text{low}}_{=?}[C^{\leq 3600}]/3600), (m_{ms}, \text{pMed}, m_{sl}, R^{\text{med}}_{=?}[C^{\leq 3600}]/3600),$ $(m_{sl}, \text{pDetect}, m_{ms}, P_{=?}[F \text{ detected}])\}$ $-\ E = \{(m_{ms}, \text{pLargeObject}, \textit{predefined}, 0.05)\}$	$m_{ms} : P_{=?}[F \text{ largeObject} \wedge \text{detected}]/$ $P_{=?}[F \text{ largeObject}])$ $m_{sl} : R^{\text{power}}_{=?}[F \text{ done}]$	2.30s 3.0s
Robo-Fleet	$U_{RoboFleet} = (M, D, E)$, where: $-\ M = \{m_{r_1}, m_{r_2}, \ldots, m_{r_N}, m_{sup}\}$ $-\ D = \{(m_{sup}, \text{pR1}, m_{r_1}, Pmax_{=?}[F \text{ done}]), \ldots, (m_{sup}, \text{pRN}, m_{r_N}, Pmax_{=?}[F \text{ done}])\}$ $-\ E = \{(m_{sup}, \text{nAttempts}, \textit{predefined}, 3)\}$	$m_{sup} : R^{\text{failures}}_{=?}[F \text{ done}]$ $m_{sup} : R^{\text{cost}}_{=?}[F \text{ done}]$	0.03s 0.03s
DPM-FX	$U_{DPM} = (M, D, E)$, where: $-\ M = \{m_{dpm}, m_{fx}\}$ $-\ D = \{(m_{dpm}, \text{diskOps}, m_{fx}, R^{\text{ops}}_{=?}[F \text{ done}]), (m_{fx}, \text{avrQueueDiskOps}, m_{dpm}, R^{\text{size}}_{=?}[S])\}$ $-\ E = \{(m_{dpm}, \text{pIdle2Sleep}, \textit{predefined}, 0.65), \ldots\}$	$m_{dpm} : R^{\text{time}}_{=?}[F \text{ done}]$	1.50s

†The reward formula $R^{\text{rwd}}_{=?}[F\ \phi]$ denotes the expected reward rwd cumulated to reach a future state of the verified model in which the formula ϕ holds
‡Total time to verify the sample property on a MacBook Pro with M1 Pro chip and 32GB RAM, averaged over 10 executions of Algorithm 1

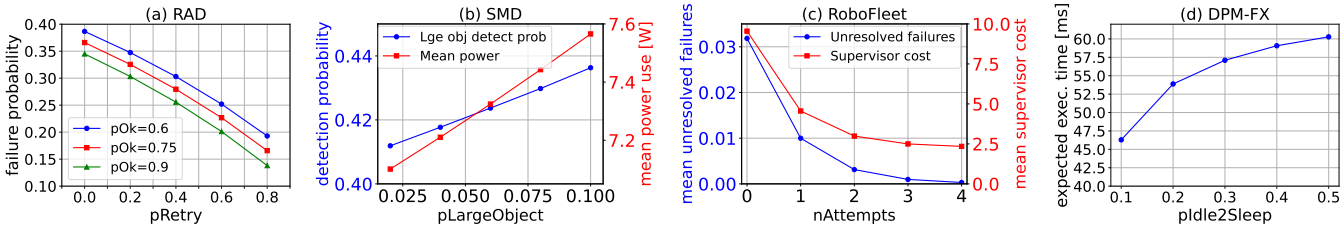

Figure 6: Experimental results

Table 3: Verification times for scalability experiments

Model size	experiment_1	experiment_2	experiment_3
small	0.79s	2.06s	12.26s
large	1.22s	3.65s	22.74s
very_large	9.57s	35.14s	233.36s

Discussion. The case studies presented in this section show that our ULTIMATE approach and verification tool are applicable to multi-model stochastic systems with a diversity of heterogeneous model types, verified properties and dependency patterns.

To answer research question **RQ1**, we note that using a single modelling formalism is not possible for the verified systems from three of our four case studies:

- the multi-model stochastic systems required for the RAD and DPM-FX case studies (Figure 3 and centre of Figure 5, respectively) include a continuous-time model (CTMC) and at least one discrete-time model (two DTMCs and a POMDP for RAD, and a DTMC for DPM-FX)—two distinct formalisms that cannot be unified into a single model;
- the multi-model stochastic system required for the RoboFleet case study (Figure 5, right) uses N Markov decision processes to capture the N-fold nondeterminism that needs to be resolved independently for each robot.

For the fourth case study (SMD), while the two DTMCs used to model the system components (Figure 5, left) could be combined into a single DTMC, this would compromise the modularity and flexibility of easily replacing one of the models with an alternative (corresponding to a different variant of that component). Based on these findings, we conclude that ULTIMATE supports the verification of software-intensive systems ranging from cyber-physical to service-based systems that cannot be modelled (or cannot be modelled conveniently) using a unified modelling formalism.

To answer research question **RQ2**, we first refer back to our summary of the practical inter-model dependencies supported by ULTIMATE from Section 4.1, which explains how the RAD and DPM-FX case studies illustrate the specification of dependability (i.e., reliability/availability) and performance (i.e., response time/throughput) inter-model dependencies, respectively. Additionally, ULTIMATE inter-model dependencies are used to express how a "cost" property (i.e., power consumption) of one system component is influenced by the dependability (i.e., the reliable detection of a moving object) of another component in the SMD case study, and how the dependability of individual robots impacts the mission cost and success probability in the RoboFleet case study. These results evidence that ULTIMATE supports the specification of a diverse range of classes of practical inter-model dependencies.

For research question **RQ3**, we first note that the ULTIMATE verification times depend on those of the model checkers and solvers invoked by Algorithm 1. Given the efficiency of modern probabilistic model checkers, all properties from Table 2 could be verified within seconds on a standard laptop. The results of the separate scalability experiments (Table 3) show:

- approximately linear scalability with respect to the numbers of models, dependency levels and model interdependencies (for a fixed model size in Table 3, experiment_2 and experiment_3 require solving a problem approximately 3 and 25 times larger than experiment_1, respectively).
- thanks to the good scalability of the established probabilistic model checkers used by ULTIMATE (PRISM, Storm), sub-linear scalability with respect to the model size (for a fixed experiment in Table 3, large and very_large models are approximately 147 and 1870 times larger than small models, respectively).

In three of the case studies (RAD—see Figure 3, and SMD and DPM-FX—see Figure 5), these dependency patterns included co-dependencies (i.e., circular dependencies) that are very challenging to resolve. ULTIMATE handled these co-dependencies successfully using one of the two methods from the description of Algorithm 1. For the case studies with co-dependencies within strongly connected components comprising only DTMCs (i.e., RAD and SMD), the co-dependencies were handled automatically by using parametric model checking to assemble a system of polynomial equations that were then solved numerically to obtain the values of the co-dependent dependency parameters. For the DPM-FX case study, whose circular dependency is between a DTMC and a CTMC, the co-dependent parameters were estimated (also automatically) using Powell's derivative-free numerical optimisation method [50]. Our ULTIMATE tool has the in-built capability to select the appropriate method for handling each type of co-dependency.

The ULTIMATE tool also provides an intuitive, user-friendly graphical user interface for assembling a multi-model stochastic system from its component models, and defining their dependency and external parameters. A demonstration video illustrating this functionality is available in our online repository [52].

Threats to validity. We limit **construct validity threats** that could arise due to simplifications and assumptions from the adopted experimental methodology by using four case studies of software-intensive systems from a diverse set of application domains, and, where appropriate, obtaining models and their properties from the software engineering research literature [25, 43].

We reduce **internal validity threats** that could introduce bias when determining cause-effect relationships in the experimental study by cross-validating the developed models independently using multiple probabilistic model checkers. Moreover, the multi-model stochastic system specifications were manually reviewed and corroborated by at least two researchers independently. Finally, we enable replication and verification of our findings by making all case studies and experimental results publicly available online.

We mitigate **external validity threats** that could affect the generalisability of our findings by demonstrating the applicability of ULTIMATE using a diverse combination of probabilistic models (see Figures 3 and 5) that represent software-intensive systems from four different application domains. Also, ULTIMATE uses the high-level modelling language provided by the probabilistic model checker PRISM [37] for model representation, and the probabilistic model checkers Storm [32] and PRISM [37] within its verification engine, thus enhancing further its applicability due to the familiarity of the research community with these tools. However, additional experiments are needed to confirm that ULTIMATE can analyse multi-model stochastic systems for software systems and processes other than those employed in our evaluation.

7 RELATED WORK

The probabilistic model checking of interdependent stochastic models is an underexplored area of research. The only other approach proposed so far for the PMC of combinations of interdependent stochastic models has been the compositional assume-guarantee verification of probabilistic models [10, 20, 41, 44], which aims to improve PMC scalability. However, unlike our ULTIMATE framework, this compositional verification paradigm supports only one type of stochastic models (probabilistic automata), can only deal with very simple interdependencies among the models it verifies compositionally, and does not provide external parameter estimation capabilities. As such, this compositional assume-guarantee verification approach cannot be used to verify any of the multi-model stochastic systems from our case studies in Section 6.

Our ULTIMATE verification algorithm (Section 4.2) operates with the strongly connected components (SCCs) of the dependency graph induced by the multi-model stochastic system under verification. The use of SCC decomposition is well established in probabilistic and parametric model checking [1, 30, 31], but at a different stage of the PMC verification process, and for a completely different purpose to ours. Thus, leading probabilistic model checkers including PRISM [37] and Storm [32] decompose the single stochastic models they analyse (e.g., DTMCs and MDPs) into SCCs in order to speed up their verification. In contrast, our ULTIMATE verification framework applies SCC decomposition to the dependency graph induced by a multi-model stochastic system, for the purpose of ordering the verification of its component models.

To the best of our knowledge, ULTIMATE is the first tool-supported approach capable of jointly verifying sets of heterogeneous stochastic models with the complex types of model interdependencies illustrated by the case studies from Section 6.

8 CONCLUSION

We introduced ULTIMATE, a novel framework for the verification of heterogeneous multi-model stochastic systems with complex interdependencies. ULTIMATE provides a unified approach to stochastic modelling, verification, and synthesis, accommodating probabilistic and non-deterministic uncertainty, discrete and continuous time, and partial observability. Furthermore, ULTIMATE is underpinned by a novel verification method for managing model interdependencies, automating dependency analysis, synthesis of analysis and parameter computation tasks, and the invocation of diverse verification engines, including probabilistic and parametric model checkers, numeric solvers, optimisers, and inference functions based on frequentist and Bayesian principles. Through a comprehensive suite of case studies, we demonstrated ULTIMATE's capabilities to support the rigorous verification of complex software-intensive systems.

Our future work will focus on (1) extending the applicability of ULTIMATE to an even wider range of domains; (2) further refining its capacity to handle increasingly intricate model interactions and incorporate diverse forms of domain knowledge and data; and (3) supporting automatic parameter synthesis [17, 24].

ACKNOWLEDGEMENTS

This work has been supported by the UK Advanced Research and Invention Agency's Safeguarded AI programme under grant 'ULTIMATE: Universal Stochastic Modelling, Verification & Synthesis Framework', and the European Union's Horizon projects GuardAI and AI4Work (grant agreements No 101168067 and 101135990, respectively).

REFERENCES

[1] Erika Abrahám, Nils Jansen, Ralf Wimmer, Joost-Pieter Katoen, and Bernd Becker. 2010. DTMC model checking by SCC reduction. In *2010 Seventh International Conference on the Quantitative Evaluation of Systems*. IEEE, 37–46.

[2] Naif Alasmari, Radu Calinescu, Colin Paterson, and Raffaela Mirandola. 2022. Quantitative verification with adaptive uncertainty reduction. *J. Syst. Softw.* 188 (2022), 111275. https://doi.org/10.1016/J.JSS.2022.111275

[3] R. Alur and T.A. Henzinger. 1996. Reactive modules. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*. 207–218. https://doi.org/10.1109/LICS.1996.561320

[4] Suzana Andova, Holder Hermanns, and Joost-Pieter Katoen. 2004. Discrete-Time Rewards Model-Checked. In *FORMATS 2003*, K. G. Larsen and P. Niebert (Eds.). Lecture Notes in Computer Science, Vol. 2791. Springer Verlag, 88–104.

[5] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. 1996. Verifying continuous time Markov chains. In *Computer Aided Verification*. Springer, 269–276.

[6] Radu Calinescu, Milan Ceska, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. 2018. Efficient synthesis of robust models for stochastic systems. *J. Syst. Softw.* 143 (2018), 140–158. https://doi.org/10.1016/J.JSS.2018.05.013

[7] Radu Calinescu, Carlo Ghezzi, Kenneth Johnson, Mauro Pezzè, et al. 2016. Formal Verification With Confidence Intervals to Establish Quality of Service Properties of Software Systems. *IEEE Trans. Reliab.* 65, 1 (2016), 107–125. https://doi.org/10.1109/TR.2015.2452931

[8] Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2011. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Software Eng.* 37, 3 (2011), 387–409. https://doi.org/10.1109/TSE.2010.92

[9] Radu Calinescu, Calum Imrie, Ravi Mangal, et al. 2024. Controller Synthesis for Autonomous Systems with Deep-Learning Perception Components. *IEEE Trans. Software Eng.* (2024), 1–22. https://doi.org/10.1109/TSE.2024.3385378

[10] Radu Calinescu, Shinji Kikuchi, and Kenneth Johnson. 2012. Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In *Monterey Workshop*. Springer, 303–329.

[11] Radu Calinescu and Marta Z. Kwiatkowska. 2009. Using quantitative analysis to implement autonomic IT systems. In *31st International Conference on Software Engineering (ICSE)*. IEEE, 100–110. https://doi.org/10.1109/ICSE.2009.5070512

[12] Radu Calinescu, Danny Weyns, Simos Gerasimou, et al. 2018. Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases. *IEEE Trans. Software Eng.* 44, 11 (2018), 1039–1069. https://doi.org/10.1109/TSE.2017.2738640

[13] Milan Češka, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. 2019. Shepherding hordes of Markov chains. In *25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 172–190.

[14] Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. 2013. Automatic verification of competitive stochastic systems. *Formal Methods in System Design* 43 (2013), 61–92.

[15] Conrado Daws. 2005. Symbolic and Parametric Model Checking of Discrete-time Markov Chains. In *First International Conference on Theoretical Aspects of Computing (ICTAC)*. 280–294.

[16] Luca De Alfaro. 1998. *Formal verification of probabilistic systems*. Ph. D. Dissertation. Stanford University.

[17] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. 2015. Prophesy: A probabilistic parameter synthesis tool. In *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I 27*. Springer, 214–231.

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

[19] Xinwei Fang, Radu Calinescu, Simos Gerasimou, and Faisal Alhwikem. 2023. Fast Parametric Model Checking With Applications to Software Performability Analysis. *IEEE Trans. Software Eng.* 49, 10 (2023), 4707–4730. https://doi.org/10.1109/TSE.2023.3313645

[20] Lu Feng. 2014. *On learning assumptions for compositional verification of probabilistic systems*. Ph. D. Dissertation. University of Oxford.

[21] Lu Feng, Tingting Han, Marta Kwiatkowska, and David Parker. 2011. Learning-based compositional verification for synchronous probabilistic systems. In *9th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Springer, 511–521.

[22] Paul Gainer, Ernst Moritz Hahn, and Sven Schewe. 2018. Accelerated model checking of parametric Markov chains. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Springer, 300–316.

[23] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. 2018. Synthesis of probabilistic models for quality-of-service software engineering. *Autom. Softw. Eng.* 25, 4 (2018), 785–831. https://doi.org/10.1007/S10515-018-0235-8

[24] Simos Gerasimou, Javier Cámara, Radu Calinescu, Naif Alasmari, Faisal Alhwikem, and Xinwei Fang. 2021. Evolutionary-guided synthesis of verified pareto-optimal MDP policies. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 842–853.

[25] Simos Gerasimou, Giordano Tamburrelli, and Radu Calinescu. 2015. Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering (T). In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 319–330. https://doi.org/10.1109/ASE.2015.22

[26] Carlo Ghezzi and Amir Molzam Sharifloo. 2013. Model-based verification of quantitative non-functional properties for software product lines. *Information and Software Technology* 55, 3 (2013), 508–524.

[27] Carlo Ghezzi and Giordano Tamburrelli. 2009. Predicting Performance Properties for Open Systems with KAMI. In *Architectures for Adaptive Software Systems*, Raffaela Mirandola, Ian Gorton, and Christine Hofmeister (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 70–85.

[28] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. 2010. PARAM: A Model Checker for Parametric Markov Models. In *International Conference on Computer Aided Verification (CAV)*. 660–664.

[29] Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5 (1994), 512–535.

[30] Arnd Hartmanns, Sebastian Junges, Tim Quatmann, and Maximilian Weininger. 2023. A practitioner's guide to MDP model checking algorithms. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 469–488.

[31] Arnd Hartmanns, Bram Kohlen, and Peter Lammich. 2023. Fast verified SCCs for probabilistic model checking. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 181–202.

[32] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2022. The probabilistic model checker Storm. *International Journal on Software Tools for Technology Transfer* (2022), 1–22.

[33] Christian Hensel, Sebastian Junges, Tim Quatmann, and Matthias Volk. 2025. *Riding the Storm in a Probabilistic Model Checking Landscape*. Springer Nature Switzerland, 98–114. https://doi.org/10.1007/978-3-031-75775-4_5

[34] Sebastian Junges, Erika Ábrahám, Christian Hensel, Nils Jansen, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2024. Parameter synthesis for Markov models: covering the parameter space. *Formal Methods in System Design* (2024), 1–79.

[35] Joost-Pieter Katoen. 2016. The probabilistic model checking landscape. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science*. 31–45.
[36] Joost-Pieter Katoen. 2016. The Probabilistic Model Checking Landscape. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science*. 31–45. https://doi.org/10.1145/2933575.2934574
[37] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *23rd International Conference on Computer Aided Verification (CAV) (LNCS, Vol. 6806)*, G. Gopalakrishnan and S. Qadeer (Eds.). Springer, 585–591.
[38] M. Kwiatkowska, G. Norman, and D. Parker. 2017. Probabilistic Model Checking: Advances and Applications. In *Formal System Verification*. Springer, 73–121.
[39] Marta Kwiatkowska, Gethin Norman, and David Parker. 2022. Probabilistic Model Checking and Autonomy. *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), 385–410.
[40] Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2010. Assume-guarantee verification for probabilistic systems. In *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 23–37.
[41] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. 2010. Assume-Guarantee Verification for Probabilistic Systems. In *Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10) (LNCS, Vol. 6105)*, J. Esparza and R. Majumdar (Eds.). Springer, 23–37.
[42] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2018. PRISM-games: Verification and strategy synthesis for stochastic multi-player games with multiple objectives. *Int. J. Softw. Tools Technol. Transf.* 20, 2 (2018), 195–210. https://doi.org/10.1007/s10009-017-0476-z
[43] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2007. Stochastic Model Checking. In *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM (LNCS, Vol. 4486)*. Springer, 220–270.
[44] Yang Liu and Rui Li. 2020. Compositional stochastic model checking probabilistic automata via assume-guarantee reasoning. *International Journal of Networked and Distributed Computing* 8, 2 (2020), 94–107.
[45] Jorge Nocedal and Stephen J Wright. 1999. *Numerical optimization*. Springer.
[46] World Health Organization. 2021. Ageing and Health. https://www.who.int/news-room/fact-sheets/detail/ageing-and-health Accessed: 2024-12-05.
[47] Corina S. Pasareanu, Ravi Mangal, Divya Gopinath, Sinem Getir Yaman, Calum Imrie, Radu Calinescu, and Huafeng Yu. 2023. Closed-Loop Analysis of Vision-Based Autonomous Systems: A Case Study. In *35th International Conference on Computer Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 13964)*, Constantin Enea and Akash Lal (Eds.). Springer, 289–303. https://doi.org/10.1007/978-3-031-37706-8_15
[48] Colin Paterson and Radu Calinescu. 2020. Observation-Enhanced QoS Analysis of Component-Based Systems. *IEEE Trans. Software Eng.* 46, 5 (2020), 526–548. https://doi.org/10.1109/TSE.2018.2864159
[49] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*. 46–57.
[50] Michael JD Powell. 2007. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications* 43, 5 (2007), 170–174.
[51] Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1 (1972), 146–160. https://api.semanticscholar.org/CorpusID:16467262
[52] ULTIMATE project. 2025. GitHub repository. https://github.com/ULTIMATE-YORK/ULTIMATE.
[53] United Nations Population Fund (UNFPA). 2022. The State of World Population 2022: Seeing the Invisible. https://www.unfpa.org/ageing Accessed: 2024-12-05.
[54] Xingyu Zhao, Radu Calinescu, Simos Gerasimou, et al. 2020. Interval Change-Point Detection for Runtime Probabilistic Model Checking. In *35th IEEE/ACM International Conference on Automated Software Engineering(ASE)*. IEEE, 163–174. https://doi.org/10.1145/3324884.3416565
[55] Xingyu Zhao, Simos Gerasimou, Radu Calinescu, et al. 2024. Bayesian learning for the robust verification of autonomous robots. *Nature Comms. Eng.* 3, 1 (2024), 18.