Reliable and Configurable Process Simulations via Probabilistic White-Box Models

Francesco Vinci[1], Gyunam Park[2,3], Wil M. P. van der Aalst[2,4], and Massimiliano de Leoni[1]

[1] University of Padua, Padua, Italy
francesco.vinci.1@phd.unipd.it, deleoni@math.unipd.it
[2] Fraunhofer Institute for Applied Information Technology (FIT), Sankt Augustin, Germany
[3] Eindhoven University of Technology, Eindhoven, The Netherlands
g.park@tue.nl
[4] Process and Data Science, RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. Business process simulation (BPS) has emerged as a crucial tool that offers a risk-free virtual environment to analyze, test, and optimize complex service compositions and orchestrations in cloud and edge computing environments. BPS enables the evaluation of alternative scenarios (a.k.a. "what-if" scenarios) by capturing the dynamic behavior of service interactions, including control-flow, task durations, and resource utilization. Several techniques to discover BPS models employ black-box predictors to characterize the run-time simulation aspects. The downside of these approaches is that the rules guiding the predictors cannot be provided in an intelligible form. This means that service architects and process analysts cannot modify or explore the decision-making logic that drives service orchestration and composition in these simulations. Moreover, these models are often deterministic and thus unable to capture uncertainty, which is essential to realistically simulate dynamic environments. This paper presents white-box predictors based on probabilistic decision trees, which are intelligible and easy to configure. The experiments show that white-box predictors can improve the simulation accuracy and variability, while being naturally intelligible.

Keywords: Business Process Simulation · Service Composition · Machine Learning · Explainable AI · Probabilistic Decision Trees

1 Introduction

Service-oriented business processes have become the backbone of modern enterprise architectures, where complex workflows are executed through the composition and orchestration of distributed services [11]. In cloud and edge computing environments, these processes typically coordinate multiple external services, microservices, and APIs to deliver end-to-end business functionality. The dynamic nature of such architectures poses significant challenges in understanding and predicting process behavior [21].

Business process simulation (BPS) has emerged as a key tool in service-oriented computing, offering a risk-free virtual environment to analyze, test, and improve service compositions and orchestrations [4,19]. In the first phase, BPS allows the identification of potential delays (e.g., slow services or bottlenecks) and the validation of process design. In the second phase,

analysts can define *what-if* scenarios by adjusting resource allocations, modifying SLAs, or replacing human tasks with automated services [10].

The starting point in service-oriented BPS is a process model that describes service composition and orchestration logic, extended with an accurate run-time characterization of different service perspectives, primarily data flow, guards, resource allocation, and temporal behavior [20]. However, traditional approaches often fail to reflect real-world behavior. Current state-of-the-art works show that integrating deep learning models into BPS can lead to more realistic run-time characterization [7]. However, deep learning models are black-box: the rules guiding these predictors are not explicitly provided as analytic expressions, thus not being intelligible. As such, it becomes impossible to alter these rules to define *what-if* scenarios and test the consequences on process performance via BPS. Moreover, since deep learning models usually require a large amount of data, deep learning-based BPS models cannot be constructed when the real process data are limited or missing. In addition, these models are often deterministic, which reduces their ability to generalize across different scenarios and limits their applicability in dynamic or uncertain environments.

This paper addresses these limitations by proposing a simulation approach specifically designed for service-oriented business processes, where the run-time characterization is based on probabilistic decision tree models. These white-box models essentially consist of decision rules that service architects and process analysts can inspect and alter to test *what-if* scenarios involving different service configurations, SLA modifications, or infrastructure changes. We introduce **Decision-Aware Business Process Simulation** models, which are basically process models extended with the run-time characterization of BPS based on decision trees (cf. Section 3). Our technique discovers decision trees from service execution data, using machine learning to capture service selection rules and dependencies across various features (e.g., service attributes, timing, workload) (cf. Section 3.3). We also present a method for interacting with these decision trees to define *what-if* scenarios for service-oriented processes, enabling service architects to evaluate the impact of service changes or new service compositions (cf. Section 3.4).

We evaluated our approach in five case studies to assess its accuracy in reproducing the behavior of the real process (cf. Section 4). The results demonstrate that our Decision-Aware Business Process Simulation (DBPS) approach consistently outperforms the state-of-the-art white-box simulation method, namely Simod [9] and achieves competitive accuracy compared to black-box approaches [5,17] while offering superior interpretability for service management decisions. Additionally, our method demonstrates greater realism and variability in simulation outcomes, making it particularly suitable for dynamic service-oriented environments that demand transparent, configurable, and realistic behavioral modeling for effective service orchestration and composition management.

2 Related Work

The work by Rozinat et al. [20] is one of the first combining process mining techniques to discover multiple perspectives of a process (control-flow, attributes, service performance, and resource aspects) and integrating them into complete simulation models. In this work, the authors propose the use of decision tree models as decision guards in the control-flow perspective.

The increasing availability of data and the advancement of new machine and deep learning techniques led to the integration of these into traditional process simulation methods. Camargo

Table 1: Comparison of various simulation approaches across different perspectives (Control-Flow, Temporal, and Resource). A ○ symbol indicates that the perspective is modeled using a white-box approach, while • denotes a black-box approach. $\mathbf{P}$ and $\mathbf{D}$ indicate if the models are probabilistic or deterministic, respectively. A gray box indicates the use of machine/deep learning techniques for the underlying rules. The last column indicates whether the simulation model is centered around the control-flow perspective, particularly relevant for service-oriented processes.

Approach	**Process Perspective**			**Control-Flow Centric**
	Control-Flow	Temporal	Resource	
Rozinat et al. [20]	○$_{\mathbf{D}}$	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	✓
Simod [9]	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	✓
DSim [5]	○$_{\mathbf{P}}$	•$_{\mathbf{D}}$	○$_{\mathbf{P}}$	✓
RIMS [17]	○$_{\mathbf{P}}$	•$_{\mathbf{D}}$	○$_{\mathbf{P}}$	✓
Agent Simulator [12]	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	
Our	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	○$_{\mathbf{P}}$	✓

et al. [7] investigated the benefits of employing deep learning techniques compared to the traditional methods, referred to as "data-driven". The authors found that deep learning techniques produce more accurate simulations, although their technique does not enable *what-if* analyses.

Camargo et al. [5] and Meneghello et al. [17] propose two hybrid approaches, respectively named **DSim** and **RIMS**, where a process model is discovered to model the process' control-flow perspective, which is extended with deep learning models for the run-time characterization of the time perspectives. However, as mentioned in Section 1, the use of black-box models makes it impossible to conduct specific *what-if* analyses. Additionally, deep learning models are usually "data greedy", making these approaches less suitable in case of scarcity of process data for one or more perspectives, or even impossible when no process data is available for different perspectives (e.g. event logs without resource information). Moreover, these models are inherently deterministic, always producing the same output for a given input. This lack of variability introduces a limitation in service-oriented business process simulation, where incorporating noise and randomness is essential to generate realistic and insightful simulation results. As a consequence, their deterministic nature reduces their ability to generalize across different scenarios, limiting their applicability in dynamic or uncertain service environments.

Some body of research exists that focuses on defining white-box models to characterize the run-time BPS aspects. In [14], the authors have proposed to train logistic regression models to define data-dependent probabilistic guards, but they do not consider the perspectives on resource and time. The same limitation is also shared with RIMS as well as with **Simod** [9], another BPS approach that focuses on discovering and modelling data-dependent branching probabilities using rule-aware models. Recently, Kirchdorfer et al. [12] have proposed **Agent Simulator**, an agent-based approach for discovering BPS from an event log, providing high interpretability and adaptability. This approach reverts the center of simulation attention from the control-flow to the resource perspective, using resource models as backbone for BPS models: it follows that can be challenging to perform *what-if* analyses heavily focused on the control-flow perspective. In a context of service-oriented business process, approaches that are not control-flow centric, such as Agent Simulator, focus primarily on individual services. As a

result, it becomes more difficult to directly experiment with alternative ways of orchestrating a set of services to execute business processes. This orchestration is, in fact, the main focus of the control-flow perspective.

Table 1 provides a comparative summary of state-of-the-art process simulation approaches across the key process perspectives (control-flow, temporal, resource), highlighting the nature of the modeling techniques employed and whether or not they are centered on the control-flow, which is sensible for service-oriented business process simulation. The last row refers to the approach proposed in this paper, which - along with Simod - is the only approach that employs probabilistic methods to model the run-time characterization of the process' perspectives on control-flow, time and resource. As mentioned, a probabilistic characterization allows simulations to be more accurate and more generalized, while white-box methods are preferable over black-box because the former makes it extremely easier to configure process simulation models for alternative *what-if* analyses. Among the white-box, probabilistic approaches, Simod uses simple distributions and probabilities to characterize the temporal and resource perspective, which naturally do not provide outcomes that are as accurate as those provided by machine learning approaches. In fact, our approach focuses on stochastic decision trees, highly intuitive and easily configurable. In summary, Table 1 allows one to conclude that our approach is more accurately capture real-world uncertainties, incorporating noise and stochasticity for greater realism, while easily enabling what-if analyses. The experiments presented in Section 4 provide evidence supporting these claims.

Note that every approach discussed so far, including those reported Table 1 and ours, uses methods that capture correlations. These correlations are not used directly for root-cause analysis and process improvements but are merely used to guide the simulation, which is conversely used for root-cause analysis and improvement. There is a large body of research on causal reasoning methods, such as by Alaee et al. [2], which certainly provides more rigorous methods for causal inference but, on the other hand, do not easily enable *what-if* analyses.

3 Decision-Aware Business Process Simulation

In this paper, we introduce Decision-Aware Business Process Simulation models (DBPS), where decision models govern control-flow and temporal simulation parameters based on feature values of simulation instances. These decision models operate within defined feature spaces, allowing users to interact with and modify service rules to perform *what-if* analyses.

We first introduce some preliminary concepts in Section 3.1. In Section 3.2, we formally define DBPS, while Section 3.3 presents a technique to discover them from an input event log $\mathcal{L}$. Finally, Section 3.4 illustrates how users can leverage these simulation models for interactive *what-if* analysis.

3.1 Preliminaries

In this section, we introduce key preliminary concepts essential for discussing our proposal. First, basic process mining definitions are presented, including events, event logs, and Petri net models [1]. Finally, we explore (probabilistic) decision tree models and their application in service-oriented BPS settings.

Process Data. Event logs are collections of traces [1], each recording a process execution (a.k.a. case) as a temporally ordered sequence of events [1]. Each event carries a data payload. Let $\mathcal{I}$, $\mathcal{A}$, and $\mathcal{R}$ denote the sets of all possible case identifiers, activities, and resources, respectively. For an event e, we have $case(e) \in \mathcal{I}$, $act(e) \in \mathcal{A}$, $res(e) \in \mathcal{R}$, and $time(e) \in \mathbb{N}$, corresponding to the case identifier, executed activity, involved resource, and timestamp, respectively. The latter induces the ordering within traces. Moreover, $life(e) \in \{start, complete\}$ indicates whether e marks the start or completion of the activity. Finally, let $\mathcal{V}$ and $\mathcal{U}$ denote the sets of attribute names and values; then $attr(e) : \mathcal{V} \to \mathcal{U}$ maps each $v \in \mathcal{V}$ to its value $attr(e)(v) \in \mathcal{U}$, inherited from the trace to which e belongs.

Event timestamps define key elements to model in a process simulator. The **inter-arrival times** are the times between the arrival of two consecutive cases; the **waiting time** of an event is the time between the start of that event and the time is enabled to start; the service **execution time** of an event is the time needed for executing an activity.

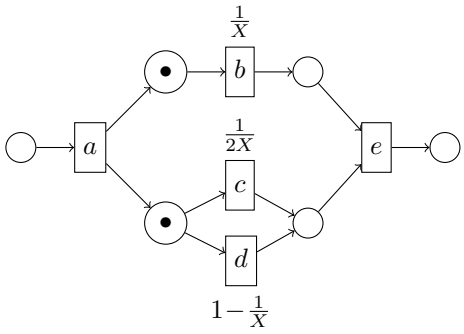

Fig. 1: Example of a Stochastic Labelled Petri net. The black dots (•) represent the current marking. Transitions of the enabled transitions are annotated with weights depending on the variable X.

Process Models. In this paper, we use Petri nets to model and represent the activity control-flow of service processes with a set of recorded traces $\mathcal{L}$ [1]. We opted for Petri nets because they have a simple yet formal syntax and semantics for defining the control-flow stochasticity. However, the discussion remains valid even using other process model notations, such as process trees or BPMN models. A **Petri net** is a tuple $(P,T,F,\mathcal{A},\lambda)$, where P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, and $\lambda : T \to \mathcal{A} \cup \{\tau\}$ is a labelling function where τ denotes the label of invisible transitions. A Petri net marking is a function $m : P \to \mathbb{N}$, that assigns to each place the number of tokens it contains, representing the current state of the net.

A transition $t \in T$ is said to be enabled in a marking m if each of its input places contains at least one token. When an enabled transition fires, it consumes one token from each of its input places and produces one token in each of its output places, resulting in a new marking that reflects the updated state of the net.

For modeling the inherent stochasticity of the process simulations, we used the concept of stochastic Petri nets following the procedure described in [14], where each transition $t \in T$ is associated with a weight $\theta_t(X) \geq 0$ dependent on some values X. Specifically, the probability of firing a transition $t \in T$ at a certain marking m is defined as

$$P(t,m) = \begin{cases} \frac{\theta_t(X)}{\sum_{t' \in E(m)} \theta_{t'}(X)} & \text{if } t \in E(m) \\ 0 & \text{otherwise} \end{cases}$$

where $E(m) \subseteq T$ is the set of transitions enabled at m.

Example 1 (Stochastic Labelled Petri Net). Let N be a Petri net as shown in Figure 1, with the current marking m highlighted with black dots. The set of enabled transition is then $E(m) = \{b,c,d\}$. The probability of firing the transition b is computed as $P(b,m) = \frac{\frac{1}{X}}{\frac{1}{X}+\frac{1}{2X}+1-\frac{1}{X}} = \frac{2}{1+2X}$.

Probabilistic Decision Trees. In this paper, we leverage **probabilistic decision tree** models for their interpretability, explainability, and ease of user interaction. Their white-box, interpretable nature makes them very suitable to enable *what if* analyses, where deep learning models fail (cf. Section 2). Other white-box models, such as logistic regression, rely on counterintuitive coefficient, making them less practical for domain analysts who aim to define *what-if* analyses. Probabilistic decision trees make predictions by hierarchically splitting the feature space into distinct regions based on specific decision rules. In these models, internal nodes and arcs define the decision criteria based on features in $\mathcal{F}$, while leaf nodes represent probability distributions. The use of probabilistic distributions make them be different from traditional decision tree, where leafs are associated to single static values.

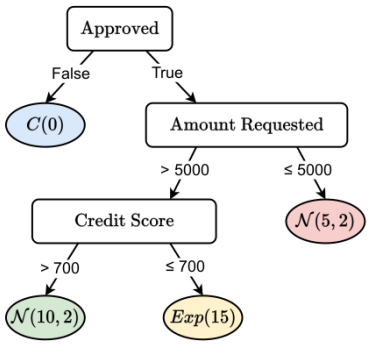

Fig. 2: Example of a stochastic decision tree that models the execution time of the activity *Close Application* in a credit loan process. $C(t)$ indicates a constant distribution with value t, $Exp(t)$ the exponential distribution with mean t, $\mathcal{N}(\mu,\sigma)$ the normal distribution with mean μ and standard deviation σ. Numbers at leaf values are in minutes.

In the BPS domain, stochasticity plays a crucial role. Traditional deterministic decision tree models may be inadequate, as they always produce the same output without accounting for noises or variability. Through stochastic decision tree, the final output is then generated by sampling from the distribution associated to the interested leaf.

Example 2. In a loan application process, service times of certain activities may depend on rules based on previous activities and/or on process attributes. Figure 2 illustrates an example of a decision tree modeling the execution time of the closure application activity. If the loan has not been approved the application closes immediately. Otherwise, for requested amounts below 5000, the duration follows a normal distribution with mean 5 and standard deviation 2. For higher amounts, service time increases and depends on the applicant's credit score. For example, for higher requested amount from customers with a low score the closure time follows an Exponential distribution with average 15.

Given a set of features $\mathcal{F}$, a deterministic decision tree is defined as a function $\theta \in (\mathcal{F} \to \mathbb{R}) \to \mathbb{R}$, where, for a given input $x \in (\mathcal{F} \to \mathbb{R})$, that assigns a numerical value to each feature, $\theta(x)$ represents the decision tree prediction. A probabilistic decision tree is a function $\theta_{\mathcal{P}} \in (\mathcal{F} \to \mathbb{R}) \to \mathcal{P}$, where $\mathcal{P}$ denotes a set of possible probability distributions. For a given input $x \in (\mathcal{F} \to \mathbb{R})$, the output $\theta_{\mathcal{P}}(x)$ represents a probability distribution over possible

predictions. Hereafter, $\Theta^{\mathcal{F}} = (\mathcal{F} \to \mathbb{R}) \to \mathbb{R}$ and $\Theta^{\mathcal{F}}_{\mathcal{P}} = (\mathcal{F} \to \mathbb{R}) \to \mathcal{P}$ indicate the set of all possible deterministic and stochastic decision trees built using features $\mathcal{F}$, respectively.

3.2 Definition

Business process simulation models are typically defined by combining a process model with a set of simulation parameters. Indeed, process models offer a clear and understandable structure for representing the activity control-flow of a process.

We define a **Decision-Aware Business Process Simulation** (DBPS) model as a tuple (N,D,P) where:

- $N = (P,T,F,\mathcal{A},\lambda)$ is a Petri net describing the activity control-flow.
- D represents descriptive parameters, including resource calendars, the set of possible resources $\mathcal{R}$, and distributions of trace service attributes.
- $P = (p^{CF},p^{ET},p^{WT},\theta^{AT})$ is the tuple of predictive parameters:
  - $p^{CF} : T \to \Theta^{\mathcal{F}_{CF}}$ assigns to each transition a decision tree that determines the **transition weights** (cf. Section 3.1) based on the set of control-flow-related features $\mathcal{F}_{CF}$. These features include trace history (i.e., how many times an activity occurred in past executions of the same trace) and trace attributes, following approaches such as those in [14], where past executions and trace-level context influence branching decisions.
  - $p^{ET} : \mathcal{A} \to \Theta^{\mathcal{F}_{ET}}_{\mathcal{P}}$ maps each activity to a probabilistic decision tree that predicts its service **execution time**. The set of features $\mathcal{F}_{ET}$ includes the resource involved in the event, the trace history, and the trace attributes. This is consistent with state-of-the-art works where execution duration is shown to depend on who performs the task [16];
  - $p^{WT} : \mathcal{R} \to \Theta^{\mathcal{F}_{WT}}_{\mathcal{P}}$ returns, for each resource $r \in \mathcal{R}$, a probabilistic decision tree that predicts **waiting times** of events performed by r. The features in $\mathcal{F}_{WT}$ incorporate the resource workload (i.e. the current amount of events performed by the resource r), trace history and attributes. Waiting time is thus defined as resource-dependent, coherently with classical process simulator approaches, where waiting times are driven by resource availabilities and queues;
  - $\theta^{AT} \in \Theta^{\mathcal{F}_{AT}}_{\mathcal{P}}$ is a probabilistic decision tree that predicts **inter-arrival times** of new process instances. The feature set $\mathcal{F}_{AT}$ includes global context features - specifically weekday and hour of the day - since arrival times are externally driven and must be modeled independently of the internal process execution.

Once a DBPS model is defined, it can be used to simulate the process. The predictive models are then integrated at run-time, following the approach of [17]. This enables the generation of event logs that contain a specified number of traces, starting from a given initial timestamp.

3.3 Discovery

In this section, we present a methodology for discovering Decision-Aware Business Process Simulation models from an input event log $\mathcal{L}$. The objective is to identify service rule patterns within the event log data for building decision tree models.

The Petri net model $N = (P,T,F,\mathcal{A},\lambda)$ can be obtained using process discovery algorithms such as Inductive Miner [13], while the set of descriptive parameters D can be obtained using several techniques proposed in the literature, e.g. [15].

The event log $\mathcal{L}$ is then aligned with the control-flow model N, resulting in an event log $\mathcal{L}_N$ compliant with N. This alignment procedure serves as a basis for the construction of the training datasets required by predictive models, namely, the decision trees, used in the DBPS. The predictive parameters are then discovered as follows.

Transition Weights. For each transition $t \in T$ a training dataset is constructed following a procedure similar to that proposed in [14], but using decision trees instead of logistic regression models. Specifically, the training dataset is generated by collecting feature values of $\mathcal{F}_{CF}$ (i.e., trace history and trace attributes), whenever t is enabled in the aligned event log $\mathcal{L}_N$. The goal is to predict if t will be fired or not. A decision tree is then trained as a binary classifier. Finally the leaf values are computed as the proportion of correctly classified samples in each leaf, representing the probabilities of firing the transition t when enabled. Each decision tree defines the transition weight function $p^{CF}(t) \in \Theta^{\mathcal{F}_{CF}}$ for each $t \in T$.

Execution Times. For each activity $a \in \mathcal{A}$, we observe the events $e \in \mathcal{L}$ such that $act(e) = a$, and build a training set of feature values of $\mathcal{F}_{ET}$, which includes resource $res(e)$, trace history and trace attributes $attr(e)$ with their values, aiming to predict the execution time of e. A decision tree is then built for each activity $a \in \mathcal{A}$. Finally, for each leaf, a best fitting distribution is determined from a set of predefined ones by filtering the data based on the corresponding decision rules. The distributions are discovered following the procedure in [6], which aims to minimize the Wasserstein distance. The set of predefined distributions $\mathcal{P}$ include: *Normal*, *Exponential*, *LogNormal*, *Uniform*, and *Constant*. This results in probabilistic decision trees (cf. Section 3.1) defining $p^{ET}(a) \in \Theta_{\mathcal{P}}^{\mathcal{F}_{ET}}$ for each activity $a \in \mathcal{A}$.

Waiting Times. For each resource $r \in \mathcal{R}$ we observed the events $e \in \mathcal{L}_N$ such that $res(e) = r$ and build a training set of feature values in $\mathcal{F}_{WT}$ (i.e., resource workload, trace history and trace attributes) with the goal of predicting the waiting time. Specifically, the waiting time can be computed from the aligned event log by looking at the time between the start timestamp of the event e and the time the correspondent transition, in the aligned event log, is enabled. This results in a probabilistic decision tree $p^{WT}(r) \in \Theta_{\mathcal{P}}^{\mathcal{F}_{WT}}$ for each resource $r \in \mathcal{R}$ that models the waiting time distribution for each leaf.

Inter-Arrival Times. A training dataset is built as follows: for each trace $\sigma \in \mathcal{L}$, the feature values of $\mathcal{F}_{AT}$ (i.e., the current weekday and hour) are retrieved in order to predict the arrival time of σ. A decision tree is then trained to estimate the arrival time from these feature values. The same procedure as for execution and waiting time is then followed to build the probabilistic decision tree $\theta^{AT} \in \Theta_{\mathcal{P}}^{\mathcal{F}_{AT}}$.

3.4 User Interaction for What-If Analysis

In service-oriented business process simulation, explainability is the key. The user aims to interact with the simulator to investigate several *what-if* scenarios, enabling them to evaluate the potential impacts of changes without putting them in production.

Since DBPSs are defined via (probabilistic) decision trees, such analyses are based on their modifications, which are of three types:

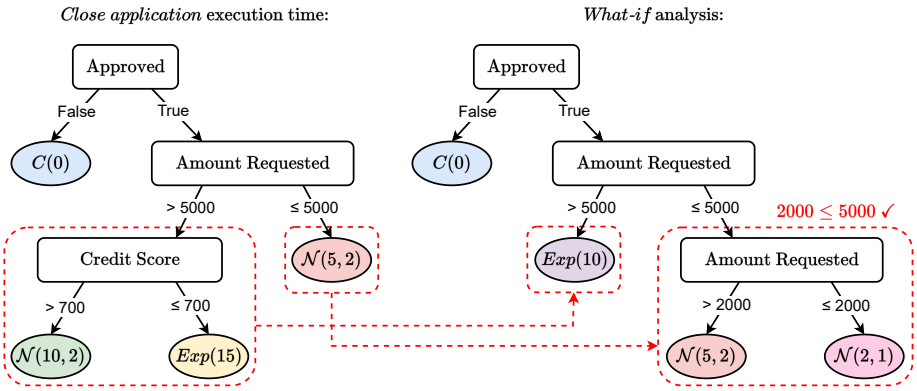

Fig. 3: User interaction example on the decision tree illustrated in Figure 2. $C(t)$ indicates a constant distribution with value t, $Exp(t)$ the exponential distribution with mean t, $\mathcal{N}(\mu,\sigma)$ the normal distribution with mean μ and standard deviation σ. Rules are modified by removing a decision tree and adding a new one in a different branch. The new rule satisfies the constraint of the parent node ($2000 < 5000$).

1. **Change leaves values**, namely distribution parameters in a certain leaf are altered.
2. **Remove subtrees**, namely a subtree is replaced by a single leaf.
3. **Add subtrees**, namely a composite tree replaces a single leaf.

Note that these interactions can also be combined, and the user can also define new decision tree models (e.g. when including more resources or new activities in the process models). When including new resources the user must also model the waiting time for them, which may depend on some features in $\mathcal{F}_{WT}$. Alternatively, users can also reuse existing decision trees from other resources to mimic the same behaviors. Also, when including a new activity, i.e., new transitions in the process model with a new activity label, the user can similarly introduce new decision trees for each new transition defining the probability of firing that transition when enabled.

Example 3. Figure 3 illustrates an example of interacting with the decision tree from Example 2 to perform a *what-if* analysis. In this new scenario, the service execution time of *Closure* is no longer dependent on the *Credit Score* for requested amounts exceeding 5000. Additionally, for requested amounts below 2000, the service time is reduced. This adjustment allows evaluating how the process performance is affected by these changes.

4 Experiments

This section presents the experiments conducted to evaluate the performance of our approach compared with the state-of-the-art works. The goal is to show the ability of DBPS in producing accurate and realistic simulations from various perspectives.

Accuracy evaluation focuses on computing distances between the distributions observed in the simulated event logs and the real ones [8]. We compared our results with those obtained

by state-of-the-art methods (cf. Section 2), showing that our approach outperforms the white-box simulation model Simod [9], and often achieves competitive or superior performance compared to the black-box methods [5,17].

Beyond the overall simulation accuracy, our method also demonstrates high precision in reproducing decision behavior, thanks to explicit modeling of decision rules. Furthermore, our simulations exhibit greater generalization and variability, key indicators of realistic behavior, as evidenced by the entropy analysis.

4.1 Experimental Setup

The simulator has been implemented as a Python package: the user can manually define each element of the DBPS or can discover one by giving an input event log and/or a Petri net model.[5]

We temporally split each event log into train and test sets with respectively 80% and 20% of the total traces. The train set is used to discover the simulation parameters as described in Section 3.3, while the test set is employed for the final evaluation.

To ensure a fair evaluation of the different methods, we used the same control-flow model as the backbone of the simulation models of the different methods. Specifically, we employed the models used in [5] and [17].

Decision trees have been discovered using the CART algorithm. To maximize accuracy, we performed a 3-fold cross-validation with hyperparameter optimization. Specifically, we defined the hyperparameter search space for the maximum tree depth by varying its values in the range 1 to 5. In fact, the more terminal nodes and the deeper the tree, the more difficult it becomes to understand the decision rules of a tree [18].

4.2 Processes & Event Logs

To assess the accuracy of our approach, we discovered the DBPS models (cf. Section 3.3) using available event logs from five different business processes:

- Purchase to Pay (P2P), which registers executions of a purchasing process.[6]
- CVS retail pharmacy (CVS), which focuses on a retail pharmacy process.[7]
- Academic Credentials Recognition (ACR), which records the executions process at the University of Los Andes in Colombia.[7]
- BPIC12W, which captures the execution of the subprocess for the workflow-relevant activities, namely those starting with W, of a loan application process run by a Dutch financial institution in 2011.[8]
- BPIC17, which refers to the same sub-process as BPC12W, but related to the executions in 2016, when the process underwent some significant changes.[9]

The use cases include two synthetic and three real-life event logs. The BPIC12W and BPIC17W event logs also contain trace attributes that are used in the simulations.

[5] https://github.com/franvinci/prosit
[6] https://fluxicon.com/academic/material
[7] https://zenodo.org/records/4699983
[8] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[9] https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

Table 2: Simulation distances results. For each case study, correspondent metric results are reported for each method: Simod [9], our approach DBPS, DSim [5], and RIMS [17]. The best results are highlighted in bold. Cells highlighted in green indicate best results among the white-box methods (i.e. Simod and DBPS). Last rows, denoted with *Avg.*, indicate average results for each metric and method across the case studies.

Case Study	**Metric**	**White-Box**		**Black-Box**	
		Simod [9]	DBPS	DSim [5]	RIMS [17]
P2P	NGD	0.58	**0.35**	0.60	0.59
	CTD	453.62	**432.12**	584.30	578.33
	CAR	731.43	**669.17**	770.54	773.02
ACR	NGD	0.53	**0.23**	0.25	**0.23**
	CTD	716.05	190.08	70.19	**46.28**
	CAR	250.33	**180.24**	236.81	233.66
CVS	NGD	0.58	**0.12**	0.34	0.38
	CTD	269.93	59.90	**52.43**	99.25
	CAR	**5.39**	17.78	20.37	20.26
BPIC12W	NGD	0.79	**0.35**	0.66	0.63
	CTD	191.92	**32.08**	153.37	92.28
	CAR	**28.74**	79.71	97.49	92.74
BPIC17W	NGD	0.81	**0.32**	0.58	0.58
	CTD	148.79	54.97	114.31	**36.53**
	CAR	145.96	53.72	**24.33**	82.29
Avg.	NGD	0.66	**0.28**	0.49	0.48
	CTD	356.06	**153.83**	194.92	170.53
	CAR	232.37	**200.12**	229.91	240.39

4.3 Accuracy Evaluation

The goal of the accuracy evaluation is to assess how realistic the simulation conducted with our method is and to compare it with Simod [9], which also uses white-box models, as well as with RIMS [17] and DeepSimulator (DSim) [5], which are hybrid BPS methods that leverage deep learning techniques.

Since Agent Simulator [12] does not have an explicit control-flow model, this method cannot be compared against the others, which conversely have the control-flow model as backbone. After all, Agent Simulator has a distinct goal, focusing on resources as the main process perspective, whereas the others highlight the control-flow perspective.

To assess the accuracy of our proposed method, we conducted simulations and computed distances between the event logs obtained by the different simulation methods and the original ones, which were used to build the BPS models. We used three key distance metrics introduced by Chapela-Campa et al. [8], each targeting a specific perspective of process behavior that aligns with the focus of our approach: **N-Gram Distance** (NGD), with $N=3$ for measuring the control-flow quality; **Cycle Time Distribution distance** (CTD) to measure the ability of the model for capturing the times of process instances; **Case Arrival Rate distance** (CAR) to computes the difference in the temporal distribution of case arrivals throughout the process timeline.

Table 3: Rule-based distance results. Each row reports a case study. Results are grouped by metric (Rule-CFD, Rule-ATD, Rule-ETD, Rule-WTD), with Simod [9] and our approach (DBPS). The best results are highlighted in bold. Averages are reported in the last row.

Case Study	Rule-CFD		Rule-ATD		Rule-ETD		Rule-WTD	
	Simod [9]	DBPS	Simod [9]	DBPS	Simod [9]	DBPS	Simod [9]	DBPS
P2P	0.16	**0.05**	**768.31**	786.06	128.82	**24.76**	**1525.20**	1706.52
ACR	0.28	**0.13**	136.77	**81.43**	**6.22**	10.54	**364.92**	742.67
CVS	0.24	**0.01**	**0.37**	0.60	3.23	**0.70**	1222.63	**515.57**
BPIC12W	0.26	**0.08**	5.26	**2.26**	10.35	**3.57**	471.85	**224.02**
BPIC17W	0.23	**0.08**	2.35	**1.91**	11.71	**8.95**	**827.22**	873.44
Avg.	0.23	**0.07**	182.61	**174.45**	32.07	**9.70**	882.37	**812.44**

For each case study, we ran five simulations to obtain event logs containing the same number of traces as the real ones, and computed the average distances between the obtained event logs. Table 2 presents the results of simulation distances across five case studies, comparing our approach (DBPS) with both white-box (Simod [9]) and black-box (DSim [5], RIMS [17]) methods. Across nearly all metrics and datasets, DBPS consistently outperforms Simod, except for two metrics, establishing it as the most effective white-box method. Notably, our method even matches or comes close to the performance of state-of-the-art black-box methods RIMS and DSim.

Although both Simod [9] and RIMS [17] incorporate decision models within the control-flow, our approach demonstrates superior effectiveness. This can be attributed to two key factors: (1) unlike Simod and RIMS, which rely solely on event attributes, our method also leverages process history as a predictive feature; and (2) we model transition behavior using weighted probabilities on transitions, allowing us to capture priority in concurrent executions (see Example 1), while Simod and RIMS apply decision trees only at XOR gateways.

To further evaluate the accuracy of our method in modeling decision rules, we analyzed the behavior of simulated and real event data at the level of decision tree leaves. For time-related decision trees (i.e., arrival time, execution time, and waiting time; see Section 3), we used the Wasserstein distance to compare distributions (respectively indicated with Rule-ATD, Rule-ETD, Rule-WTD). For decision trees modeling transition weights, we computed the absolute difference in transition firing frequencies (Rule-CFD). We compared our approach with Simod [9], as DSim [5] and RIMS [17] either do not produce resource information - used as features in some decision trees - or lack the necessary trace attribute (cf. Section 3.2). Table 3 summarizes the results across the four rule-based metrics. Our approach (DBPS) outperforms Simod [9] in the vast majority of cases, achieving the lowest average distances in all four metrics. In particular, DBPS shows substantial improvements in Rule-CFD and Rule-ETD, indicating more accurate modeling of control-flow decisions and execution time distributions. These results confirm the enhanced precision of our simulations in replicating the decision logic within the process.

We attribute the superior and competitive performance of our method compared to black-box ones to its inherently probabilistic nature. While RIMS [17] and DSim [5] rely on deterministic deep learning models to generate execution times — often resulting in rigid, overfitted behavior with limited temporal variability — our method leverages probabilistic

Table 4: Average entropy of execution time distributions. For each case study, correspondent results are reported for each method: Simod [9], our approach DBPS, DSim [5], RIMS [17], and our deterministic approach DBPS-det. Best results are highlighted in bold. Red background is used to indicate the worst results. Last row, denoted with *Avg.*, indicate average results for each metric and method across the case studies.

Case Study	**Probabilistic**		**(Partly) Deterministic**		
	Simod [9]	DBPS	DSim [5]	RIMS [17]	DBPS-det
P2P	1.33	**1.38**	1.16	1.20	0.56
ACR	1.01	**1.15**	0.74	0.59	0.85
CVS	**0.64**	0.36	0.57	0.57	0.08
BPIC12W	1.65	**2.25**	0.56	0.56	0.60
BPIC17W	1.97	**2.21**	0.61	0.61	0.76
Avg.	1.32	**1.47**	0.73	0.71	0.57

decision trees to effectively capture uncertainty and reflect the natural variability observed in real-world processes. To further support this claim, we computed the entropy of execution time distributions across all activities on the simulated event logs. Entropy has been widely used in the literature as an indicator of variability and generalization capacity of process data [3]. Higher entropy reflects a greater ability to reproduce the variability found in real processes, which is desirable for realistic simulation. Specifically, since execution times are continuous variables, we approximated their entropy by discretizing the observed simulated values into bins b_1,...,b_k. The optimal number of bins is based on the Sturges' rule, which computes the number of bins as $k = \lceil \log_2(n) + 1 \rceil$, where n is the number of observed samples. Then the entropy of a probability distribution p^a of execution time of activity a is computed as $H(p^a) = -\sum_{i=1}^{k} p_i^a \log(p_i^a)$, where p_i^a expresses the probability of a value being in the bin b_i. The average execution time entropy is then obtained as $\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} H(p^a)$, with $\mathcal{A}$ the set of possible activities.

Table 4 reports the average execution time entropy values, with methods grouped into *Probabilistic* and *Deterministic* categories. We also include DBPS-det, a deterministic variant of our method that replaces probabilistic decision trees with deterministic ones. As expected, all deterministic approaches exhibit substantially lower entropy, confirming their limited ability to capture temporal variability. This statement is particularly confirmed if we compare the entropy values of our approach DBPS with DBPS-det: the simple replacement in the latter of probabilistic decision trees with deterministic ones causes a significant drop in the entropy values, further pointing out that the improvement is certainly related to being probabilistic. Our approach (DBPS) achieves higher entropy on average and across all case studies, except for CVS. The CVS dataset warrants special attention. As noted by its authors, it is synthetically generated, likely without emphasizing behavioral variability. Consequently, it exhibits limited behavior variability, which explains the relatively lower entropy observed in this case.

These findings underscore the effectiveness of our method in generating behaviorally accurate, realistic, and diverse simulations while maintaining full interpretability. We consistently outperform the white-box Simod [9], not only in overall simulation accuracy but also in faithfully modeling decision rules. Moreover, our method achieves competitive accuracy

when compared to black-box state-of-the-art approaches, while offering greater realism by capturing a higher degree of behavioral variability, as demonstrated by the entropy analysis.

5 Conclusion

The simulation of business processes that involve human resources and/or services is based on a model that describes how activities are executed, along with a characterization of how the perspectives on data, resource, and time are simulated. This characterization refers to new process-instance arrival times, activity durations, resource assignment, data-dependent probabilistic guards, etc.

While several approaches exist for discovering business process simulation models and characterizing their run-time behavior, most rely on deep learning and/or deterministic models. Deep learning models are typically black-boxes and do not expose their decision logic in an interpretable, analytical form. This limits their configurability and hinders the ability to perform meaningful what-if analyses. Deterministic models, in turn, fail to capture the inherent stochasticity and uncertainty of real-world processes.

This paper proposes a novel approach that uses probabilistic decision trees for run-time characterization. These decision models are both white-box, allowing for easy interpretation and configurability, and stochastic, enabling the simulation of natural process variability.

Our technique has been compared with those existing related techniques that are based on black-box and/or deterministic models as well as with the only technique from the state of the art that is based on stochastic, white-box models. Five different processes were used in the evaluation, and the results outline that our method enables running business process simulations that are more realistic than and generalize more than these existing techniques.

We acknowledge that the current evaluation is largely quantitative on the ability of decision trees to adequately characterize the run-time characterization of business simulation. As future work, we aim to assess the human usability and interpretability of stochastic decision trees to conduct *what-if* analyses. So we plan to investigate the trade-off between usability and interpretability, on the one hand, and decision tree depth, on the other hand: we hypothesize that deeper trees, while potentially more accurate, may hinder users' ability to understand and effectively utilize them for *what-if* analyses. We plan to conduct user studies with process experts using consolidated techniques for human-computer interaction and cognitive psychology.

Acknowledgments. F. Vinci is financially supported by MUR (PNRR) and University of Padua. M. de Leoni is supported by the European Union – Next Generation EU under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.1 - Call PRIN 2022 PNRR No. 1409 of September 14, 2022 of Italian Ministry of University and Research; Project P20222XM58 (subject area: SH) *Emergency medicine 4.0: an integrated data-driven approach to improve emergency department performances*.

References

1. van der Aalst, W.: Process Modeling and Analysis, pp. 55–88. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

2. Alaee, A.J., Weidlich, M., Senderovich, A.: Data-driven decision support for business processes: Causal reasoning and discovery. In: Business Process Management Forum. pp. 90–106. Springer Nature Switzerland, Cham (2024)
3. Back, C.O., Debois, S., Slaats, T.: Entropy as a measure of log variability. Journal of Data Semantics **8**, 129–156 (2019)
4. Bartsch, C., Mevius, M., Oberweis, A.: Simulation of it service processes with petri-nets. In: Service-Oriented Computing – ICSOC 2008 Workshops. pp. 53–65 (2009)
5. Camargo, M., Báron, D., Dumas, M., González-Rojas, O.: Learning business process simulation models: A hybrid process mining and deep learning approach. Information Systems **117**, 102248 (2023)
6. Camargo, M., Dumas, M., González-Rojas, O.: Automated discovery of business process simulation models from event logs. Decision Support Systems **134**, 113284 (2020)
7. Camargo, M., Dumas, M., Rojas, O.G.: Discovering generative models from event logs: Data-driven simulation vs deep learning. PeerJ Computer Science **7** (2021)
8. Chapela-Campa, D., Benchekroun, I., Baron, O., Dumas, M., Krass, D., Senderovich, A.: A framework for measuring the quality of business process simulation models. Information Systems **127**, 102447 (2025)
9. Chapela-Campa, D., López-Pintado, O., Suvorau, I., Dumas, M.: Simod: Automated discovery of business process simulation models. SoftwareX **30**, 102157 (2025)
10. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Publishing Company, Incorporated, 2nd edn. (2018)
11. Henkel, M., Zdravkovic, J., Johannesson, P.: Service-based processes: design for business and technology. In: Proceedings of the 2nd International Conference on Service Oriented Computing. p. 21–29. ICSOC '04 (2004)
12. Kirchdorfer, L., Blümel, R., Kampik, T., Van der Aa, H., Stuckenschmidt, H.: Agentsimulator: An agent-based approach for data-driven business process simulation. In: 2024 6th International Conference on Process Mining (ICPM). pp. 97–104 (2024)
13. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Application and Theory of Petri Nets and Concurrency. pp. 311–329. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
14. de Leoni, M., Vinci, F., Leemans, S.J.J., Mannhardt, F.: Investigating the influence of data-aware process states on activity probabilities in simulation models: Does accuracy improve? In: Business Process Management. pp. 129–145. Springer Nature Switzerland, Cham (2023)
15. López-Pintado, O., Dumas, M.: Discovery and simulation of business processes with probabilistic resource availability calendars. In: 2023 5th International Conference on Process Mining (ICPM). pp. 1–8 (2023)
16. López-Pintado, O., Dumas, M., Berx, J.: Discovery, simulation, and optimization of business processes with differentiated resources. Information Systems **120**, 102289 (2024)
17. Meneghello, F., Di Francescomarino, C., Ghidini, C., Ronzani, M.: Runtime integration of machine learning and simulation for business processes: Time and decision mining predictions. Information Systems **128**, 102472 (2025)
18. Molnar, C.: Interpretable Machine Learning. 3 edn. (2025)
19. Pourbafrani, M., van der Aalst, W.M.P.: Interactive process improvement using simulation of enriched process trees. In: Service-Oriented Computing – ICSOC 2021 Workshops (2022)
20. Rozinat, A., Mans, R., Song, M., van der Aalst, W.: Discovering simulation models. Information Systems **34**(3), 305–327 (2009)
21. Wang, J., Lu, C., Yu, Y., Cao, B., Fang, K., Fan, J.: Higpp: A history-informed graph-based process predictor for next activity. In: Service-Oriented Computing. pp. 337–353 (2025)