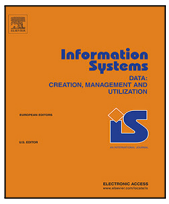
Information Systems 128 (2025) 102472

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Runtime integration of machine learning and simulation for business processes: Time and decision mining predictions☆

Francesca Meneghello [a,b,*], Chiara Di Francescomarino [c], Chiara Ghidini [d], Massimiliano Ronzani [b]

[a] Sapienza University of Rome, Rome, Italy
[b] Fondazione Bruno Kessler, Trento, Italy
[c] University of Trento, Trento, Italy
[d] Free University of Bolzano, Bolzano, Italy

ARTICLE INFO

Keywords:
Business process simulation
Deep learning
Hybrid simulation
Decision mining

ABSTRACT

Recent research in Computer Science has investigated the use of Deep Learning (DL) techniques to complement outcomes or decisions within a Discrete Event Simulation (DES) model. The main idea of this combination is to maintain a white box simulation model complement it with information provided by DL models to overcome the unrealistic or oversimplified assumptions of traditional DESs. State-of-the-art techniques in BPM combine Deep Learning and Discrete Event Simulation in a post-integration fashion: first an entire simulation is performed, and then a DL model is used to add waiting times and processing times to the events produced by the simulation model.

In this paper, we aim at taking a step further by introducing Rims (Runtime Integration of Machine Learning and Simulation). Instead of complementing the outcome of a complete simulation with the results of predictions a posteriori, Rims provides a tight integration of the predictions of the DL model *at runtime* during the simulation. This runtime-integration enables us to fully exploit the specific predictions while respecting simulation execution, thus enhancing the performance of the overall system both w.r.t. the single techniques (Business Process Simulation and DL) separately and the post-integration approach. In particular, the runtime integration ensures the accuracy of intercase features for time prediction, such as the number of ongoing traces at a given time, by calculating them during directly the simulation, where all traces are executed in parallel. Additionally, it allows for the incorporation of online queue information in the DL model and enables the integration of other predictive models into the simulator to enhance decision point management within the process model. These enhancements improve the performance of Rims in accurately simulating the real process in terms of control flow, as well as in terms of time and congestion dimensions. Especially in process scenarios with significant congestion – when a limited availability of resources leads to significant event queues for their allocation – the ability of Rims to use queue features to predict waiting times allows it to surpass the state-of-the-art. We evaluated our approach with real-world and synthetic event logs, using various metrics to assess the simulation model's quality in terms of control-flow, time, and congestion dimensions.

1. Introduction

Business process simulation (BPS) [1] provides a widely used and flexible approach to analyze and improve business processes. It exploits a process simulation model, that is, a process model extended with additional information for a probabilistic characterisation of the different run-time aspects such as case arrival rate, task durations, routing probabilities, resource utilization, and so on, to produce a significantly large number of process runs. Statistics over these runs are then collected to gain insight into the process, and to determine the possible issues such as bottlenecks, wastes, or costs. Through simulation experiments, also, various 'what if' questions can be answered, and different process redesign alternatives can be compared with respect to the key performance indicators of interest.

Building simulation models is a costly task that often requires great expertise and knowledge of the domain at hand. To overcome this

☆ We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.
* Corresponding author at: Sapienza University of Rome, Rome, Italy.
E-mail addresses: fmeneghello@fbk.eu (F. Meneghello), c.difrancescomarino@unitn.it (C.D. Francescomarino), Chiara.Ghidini@unibz.it (C. Ghidini), mronzani@fbk.eu (M. Ronzani).

https://doi.org/10.1016/j.is.2024.102472
Received 4 April 2024; Received in revised form 26 September 2024; Accepted 1 October 2024
Available online 9 October 2024

problem Data-Driven Process Simulation (DDPS) approaches have been introduced in literature [2–7]. The main idea of DDPS is to build a simulation model by using the knowledge contained in process execution data (i.e., event logs). For this purpose, event logs can provide process analysts with the knowledge needed to manually construct a simulation model by defining the process model and tuning the simulation parameters. Alternatively, event logs can be exploited to directly discover the simulation model in a fully automated manner [2].

However, when building simulation-ready models from an event log, many unrealistic or oversimplifying assumptions are usually made [8], such as the absence of resource constraints [4] or waiting times that are generated only by resource contention. Moreover, the direct discovery of simulation models often does not take into account behaviors such as multitasking, batching, fatigue effects, and inter-process resource sharing, among others [2,9].[1]

Another significant limitation usually affecting existing DDPS models concerns the implementation of process decision points in the simulation model. Indeed, decision points are usually determined using branching probabilities computed on the event log [2,5,10]. This approach can result in infeasible and unrealistic process behaviors, as the sequence of activities executed in the traces is determined independently of any trace-specific properties, such as previously performed activities, assigned resources, current timestamp, and/or other event attributes.

The recent development of Deep Learning (DL) techniques applied to Predictive Process Monitoring [11] offers an alternative way to "create" hypothetical process runs, either by completing trace prefixes via the prediction of next activities or by complementing an existing trace with predicted attributes such as the duration time of an activity [12,13]. Unlike simulation models, which are widely used for the analysis and improvement of business processes due to their properties, DL models pose challenges when applied for the same purpose. Indeed, DL models cannot transparently expose an explicit process model that can be leveraged to understand the behavior of the process or readily adjusted to simulate the impact of modifications on the simulation output. However, DL models are extremely powerful in learning the patterns characterizing the precise relationship between the different trace elements generating a high-quality distribution of predicted values that can, in turn, lead to the production of high-quality simulation outputs.

Starting from these observations, recent research in Computer Science has investigated the use of Deep Learning (DL) techniques to complement outcomes or decisions within a Discrete Event Simulation (DES) model [14–17]. The main idea here is to maintain a white box simulation model but to complement it with information provided by DL models. As mentioned above, these models are indeed extremely powerful in learning the true relationship among trace elements as expressed by the covariates and the distribution of the output variables, thus avoiding the oversimplifications of the simulation model mentioned above. A first attempt to combine Deep Learning and Discrete Event Simulation with the goal of achieving accurate simulations from a temporal perspective is presented in [10], where the *Dsim* tool is used in a **post-integration** fashion: first an entire simulation is performed, and then a DL model is used to add waiting times and processing times to the events produced by the simulation model.

As a consequence of the post-integration, the features used for the temporal predictive models become inaccurate, especially the inter-case features, and the queue order for resource allocation is not always respected, as shown in Section 3. Furthermore, the post-integration approach does not allow for the integration of predictive models at process decision points, as this would require potentially to alterate the control flow of traces defined during the simulation phase with each prediction. de Leoni et al. [18], instead, proposed a simulation model that integrates a logistic regression model to predict the next activity at each decision point based on the process state. However, the evaluation of the simulation model proposed in [18] focuses solely on the control flow perspective, without considering other perspectives, such as the time perspective. These perspectives can be deeply interconnected, as demonstrated in Section 7.

By introducing Rims (Runtime Integration of Machine Learning and Simulation) [19] we aim to overcome these issues. Instead of complementing the outcome of a complete simulation "a posteriori" with the predictions of a DL model as in [10], Rims provides a tight integration of the predictions of the DL model at runtime during the simulation. The *runtime-integration*, in fact, enables us to fully exploit the predictions related to the time perspective, that is *waiting time* and *processing time*, at simulation time, thus enhancing the performance of the overall system both with respect to the single techniques (BPS, and DL) separately and the *post-integration* approach of *Dsim*. The *runtime-integration* enables us also to leverage information related to the queue e.g., the time an event spends waiting for a resource and/or the number of events competing for that resource, as an intercase feature in the DL model, and to obtain an augmented version of Rims which further improves its performance in process scenarios where the queue plays an important role. Besides time-related predictions, Rims also targets the control-flow dimension, by integrating predictions related to *the next activity at decision points*, thus addressing the limitations of DDPS in terms of decision point implementations. The next activity prediction at decision points allows for a more accurate identification of the branch and hence of the control flow of the simulated trace. In particular, for time prediction, we use a Long Short-Term Memory (LSTM) model, a type of DL model suitable for handling sequences of elements, while for the prediction at decision points, we employ a Decision Tree classifier, as in [20,21].

This paper is an extended and revised version of our previous conference paper [19]. The advancements primarily focus on the following aspects:

- We present a new augmented version of Rims, which integrates predictive models for the control-flow perspective, specifically to predict the next activity from a decision point (Section 4.3). This aims to address the issue of generating infeasible and unrealistic traces, during the simulation.
- We extend the running example (Section 3) to motivate the introduction of predictive models for the control-flow perspective.
- We explore the use of different encodings to train the control-flow predictive models, considering both *intra-case features* and *inter-case features*, along with the type of predictions, single class or probability distribution over classes.
- We expand the evaluation of all proposed baselines and all variants of RIMS by introducing several metrics that comprehensively assess the quality of the simulation model, including control flow, time, and congestion perspectives (Section 6).

The remainder of the paper is structured as follows, we first introduce the main background concepts in Section 2. We elaborate on the motivation using the running example in Section 3. Section 4 describes and formalizes the Rims simulation approach while in Section 5 we position it with existing state-of-the-art techniques. Section 6 we present an evaluation setting in which our approach is compared with the single techniques (BPS, and DL) separately, as well as with *Dsim*, in several real-world and synthetic logs. Finally, Section 7 discusses the results of the evaluation, and Section 8 concludes the paper.

2. Background

In this section we provide the background knowledge necessary to understand the rest of the paper.

[1] While these problems are particularly noticeable in automatic approaches they can also apply to the manual construction if a deep knowledge of the domain at hand is not present.

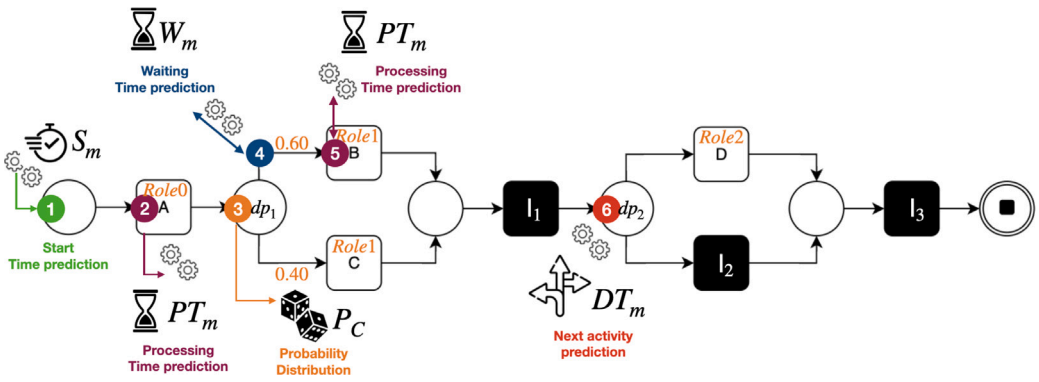


Fig. 1. RIMS system. The DDPS elements are represented by the Petri net model and the simulation parameters for the resources, which are indicated above each activity, as well as the probability for the first decision point dp_1. The DL elements are S_m, PT_m, W_m and DT_m, and the points 1, 2, 4, 5 and 6 represent moments in the process execution where these DL elements are used. Instead, the probabilities defined by the DDPS model are used to determine the branch from dp_1, as indicated by point 3. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.1. Event log

An event log $\mathcal{L}$ records the executions of a business process, that is the execution traces of the process. A trace, in turn, is a sequence of ordered events $\sigma = \langle e_1, e_2, \dots, e_m \rangle$, which may also contain trace attributes $p_1, p_2, \dots, p_n$. These are attributes whose values remain the same for all the events in the sequence. An event $e_i = (\alpha, t, r, d_1, \dots, d_q)$ is a complex structure that includes, besides other event attributes, $d_1, \dots, d_q$, the activity label $\alpha = e_i.act$, its timestamp $t = e_i.time$, indicating when the event occurred, and $r = e_i.r$ representing the resource(s) involved in the activity execution. Non-instantaneous events can be characterized by both the start ($e_i.t_{start}$) and the end ($e_i.t_{end}$) timestamps, denoting the time in which the activity starts and ends, respectively. In particular, we can denote the processing time of an activity as $e_i.t_p = e_i.t_{end} - e_i.t_{start}$ and the waiting time as $e_i.t_w = e_i.t_{start} - e_{i-1}.t_{end}$.

2.2. Simulation model

A BPS model is a tuple $\mathcal{M} = (\mathcal{N}, \mathcal{P})$ composed of a business process model $\mathcal{N}$ (e.g., a Petri net [22]), and a set $\mathcal{P}$ of parameters needed to define the simulation specifications for the different process perspectives, such as the control-flow ($\mathcal{P}_C$), the resource ($\mathcal{P}_R$), as well as the time ($\mathcal{P}_T$) perspective. For instance, case inter-arrival time, activity durations, routing probabilities, resource allocation, and utilization are all examples of simulation model parameters. The simulation model is used to generate the traces composing a simulated log. BPS models are Discrete Event Simulation (DES) models that are stochastically executed by creating new cases according to the inter-arrival time, and by simulating the execution of each case constrained to the control-flow semantics of the process model and according to the allocation of resources. Therefore, an activity is executed only if it is enabled and if there is a resource available that can perform it. Otherwise, the activity waits for another one to release the requested resource and, when it is allocated, it immediately starts. In this way, the DES models assume that waiting times are caused exclusively by resource contention.

2.3. Decision mining

A decision point dp in $\mathcal{N}$ represents any point of the process in which a main path splits into multiple paths. For instance, dp_1 in the process model in Fig. 1 divides the traces into the ones performing B and those performing C. Decision mining (DM), also known as decision point analysis, aims to identify data dependencies influencing trace routing and to associate decision rules with each dp in $\mathcal{N}$. In particular, DM is capable of discovering decision rules from the properties defined over the traces and their events in $\mathcal{L}$, such as trace attributes or events executed before dp in the trace. In BPS, the decision rules discovered by DM techniques can be used as simulation parameters in $\mathcal{M}$ to simulate the paths followed by the traces.

Table 1
Example of a simple event log.

Caseid	Activity	Arrival	Start	End	Role
1	A	24-05-23 08:00	24-05-23 08:00	24-05-23 08:06	*Role0*
2	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:07	*Role0*
3	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:05	*Role0*
3	B	24-05-23 08:05	24-05-23 08:05	24-05-23 08:11	*Role1*
2	C	24-05-23 08:07	24-05-23 08:07	24-05-23 08:14	*Role1*
1	B	24-05-23 08:08	24-05-23 08:11	24-05-23 08:14	*Role1*
1	E	24-05-23 08:14	24-05-23 08:14	24-05-23 08:30	*Role2*
2	D	24-05-23 08:14	24-05-23 08:30	24-05-23 08:40	*Role2*

2.4. Predictive process monitoring

Predictive Process Monitoring (PPM) [11] is a branch of process mining that aims at predicting the future of an ongoing (uncompleted) process execution. Within PPM, there are several different prediction tasks that one could aim to solve: predicting the fulfillment of a predicate [23], predicting the time to complete a task or the time until the completion of a case [24], predicting the next activity or the continuation of a running incomplete case [12] or entire traces [25].

In this work, we focus on predicting timestamps and the next event from a process decision point. For the first type of prediction, we adopt a Long Short-Term Memory (LSTM) model, a DL model widely applied in this field due to its ability to deal with sequences of elements, as in the case of traces. For the second type of prediction, we adopt the decision tree classifier, as in [20,21], due to its white-box nature. Indeed, the individual predictions of a decision tree can be easily explained, i.e., by following the path of the tree constructed according to the rules defined on the features. This also makes it suitable for the development of what-if scenarios, as it is possible to evaluate the impact of changing rules on the nodes of the tree in terms of classification outcome.

3. Running example

Consider the simple event log in Table 1, which contains, for each event, the activity name, the start and end timestamps, as well as the role that has performed the activity.Assume that we are interested in simulating the behavior of the log and that, to this aim, we leverage a DDPS simulation model, as the one reported in Fig. 1. This model has four activities (A,B,C,D) and two XOR splits—one between the activities B and C and one before D. Regarding the resource perspective the model contains three roles: *Role0* with 3 resources available, *Role1* with 2 resources available and *Role2* with 1 resource available. The simulation model, by using a probability distribution at each decision point of the model, would generate the traces reported in the column *Traces* of

Table 2
Traces generated by the DDPS simulation model, and timestamps predicted by using DL models.

Caseid	Traces	Predicted processing times	Predicted waiting times
1	⟨(A, *Role*0), (B, *Role*1)⟩	⟨8, 4⟩	⟨1⟩
2	⟨(A, *Role*0), (C, *Role*1), (D, *Role*2)⟩	⟨5, 7, 8⟩	⟨1, 6⟩
3	⟨(A, *Role*0), (B, *Role*1)⟩	⟨5, 5⟩	⟨0⟩

Table 3
Simulated log generated by *Dsim*.

Caseid	Activity	Arrival	Start	End	Role
1	A	24-05-23 08:00	24-05-23 08:00	24-05-23 08:08	*Role0*
1	B	24-05-23 08:09	24-05-23 08:09	24-05-23 08:13	*Role1*
2	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:06	*Role0*
2	C	24-05-23 08:07	24-05-23 08:07	24-05-23 08:14	*Role1*
2	D	24-05-23 08:14	24-05-23 08:20	24-05-23 08:28	*Role2*
3	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:06	*Role0*
3	B	24-05-23 08:06	24-05-23 08:13	24-05-23 08:18	*Role1*

Table 2. However, the traces generated by a DDPS model could present two problems. First of all they might be *non-accurate in terms of control-flow*. For instance, the simulation model, by leveraging the probability distribution, could generate a trace ⟨(A, *Role*0), (B, *Role*0), (D, *Role*2)⟩, which could result to be infeasible because, for instance, in the real system activity D can be executed only when the last role involved is *Role1* and the previous activity is C. This type of relationship cannot be captured by a pure DDPS model. The second issue of traces generated by DDPS is that the processing and the waiting time in DDPS are typically approximated with fixed times or probability distributions (e.g., exponential, normal), instead of considering also previously performed activities, assigned resources, the current timestamp, and/or other event attributes, thus resulting in *non-accurate* traces *in terms of timing*.

In order to get accurate simulations in terms of time perspective, state-of-the-art hybrid approaches, such as *Dsim*, combine simulated traces and predicted times that are generated by DL models a-posteriori: predicted times are associated one by one to the traces generated by the DDPS model, ensuring compliance in terms of resource allocation. Thus, let us assume we have trained a good DL model that gives us the predicted processing time – one for each event in the trace – and the predicted waiting time, i.e., the waiting time before the next event in the trace, as reported in Table 2 in minutes. The event log generated by the *Dsim* approach combining the simulated traces and the predicted times would be the one reported in Table 3.[2]

Thus, for example, case 1 starts with activity A and it can immediately start because *Role0* has 3 available resources. It ends 8 min later as this is the predicted processing time for that event (see Table 2). Activity B arrives 1 min later (as this is the predicted waiting time between A and B in that trace): it can immediately start as *Role1* has 2 available resources, and it ends 4 min later, as this is the predicted processing time for that event. Finally, the trace ends without performing D. One important aspect to note is that these hybrid approaches do not only leverage the predicted times, as pure generative approaches do, but they also take into account the resource availability. For instance, although activity B of case 3 would be enabled at time *08:06*, the start time is set to *08:13*, since only two resources are available for the role *Role1* – one performing activity C of case 2 and the other one activity B of case 1 – and the first one to be freed is released at time *08:13*. The resulting handling of the queue requests related to *Role1* is hence the one depicted in Fig. 2(a): first B of case 1, then C of case 2 and, finally, B of case 3. However, this does not reflect the queue order of the real log. Indeed, by looking at Table 1, we can see that resources with *Role1* execute first activity B of case 3, then activity C of case 2 and, finally, activity B of case 1, that is the queue situation reported in Fig. 2(b). This limitation of *Dsim*, which does not allow the approach to have a simulation as close as possible to the reality, is due to the insertion of the predicted times trace by trace and not per event, hampering to leverage the full potential of the combination of simulation models and predicted times.

[2] Assuming given arrival times.

(a) Queue generated with the *Dsim* approach. (b) Queue generated with the RIMS approach.

Fig. 2. Queues generated after the simulation with the *Dsim* and RIMS approaches.

The approach presented in this paper, by taking into account predicted times at each step of the simulation rather than once at the end of the trace simulation, is able to overcome this issue. By leveraging processing and waiting time predictions at each step of the simulation, RIMS is indeed able to detect that, although case 1 is the first to start among the three traces, cases 2 and 3 complete their first activity A earlier, so that the two resources of *Role1* should be first assigned to the activity B and C of these two cases—which is indeed what happens in reality (see Table 1). Table 4 shows the log generated by the RIMS approach and Fig. 2(b) shows the queue of the log on role *Role1*, that is inline with the arrival times of events in reality (see Table 1). Moreover, as the predicted times are added during the simulation, they affect all other running traces and, as a result, the intercase features, as the resource occupation, are more accurate. For example, in case of the *runtime-integration* method (Table 4), at time 08:08, both resources for *Role1* result busy (one with the activity B of case 3 and the other with the activity C of case 2). In contrast, in the case of the *post-integration* method (Table 3), at time 08:08 *Role1* appears to have all resources free, as although one resource of it will be busy in activity C of case 2, *Dsim* is not able to detect it as it only considers resource occupation related to traces started before the current trace.

Taking into account predicted times – even at each step of the simulation –, however, does not solve the limitation related to the generation of infeasible simulated traces raised at the beginning of the section. Indeed, since activity D has $1/3$ of probability to occur, the simulation model could still produce incorrect traces, such as ⟨(A, *Role*0), (B, *Role*0), (D, *Role*2)⟩, as mentioned above. To obtain traces close to reality, RIMS considers also the history of the trace. The relations between events can be used to build a predictive model for predicting the next activity at each decision point in the process, such as dp_1 and dp_2 in Fig. 1. Table 4 shows indeed that in the event log generated with RIMS the activity D is performed only under the conditions explained above. In contrast, in Table 3 the generation of activity D for case 2 occurs solely due to a lucky draw from the probability associated with the decision point.

Although the limitations (and corresponding solutions proposed by RIMS) described in this section refer to a small example log with few traces and to a simple simulation model, these issues (and solutions) can be easily generalized to more complex scenarios.

4. The RIMS system

In this section we describe the *runtime-integration* approach RIMS (Section 4.1) and its two variants: RIMS+ (Section 4.2), which takes into account also the effect of queues in the simulated log; and RIMS^{DT} (Section 4.3), which enhances RIMS/RIMS+ with a decision mining approach.

Table 4
Simulated log generated by the RIMS approach. Differences between start and end timestamps with respect to the log produced by the *Dsim* approach are shown in bold.

Caseid	Activity	Arrival	Start	End	Role
1	A	24-05-23 08:00	24-05-23 08:00	24-05-23 08:08	*Role0*
2	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:06	*Role0*
3	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:06	*Role0*
3	B	24-05-23 08:06	**24-05-23 08:06**	**24-05-23 08:11**	*Role1*
2	C	24-05-23 08:07	24-05-23 08:07	24-05-23 08:14	*Role1*
1	B	24-05-23 08:08	**24-05-23 08:11**	**24-05-23 08:15**	*Role1*
2	**D**	24-05-23 08:14	24-05-23 08:20	24-05-23 08:28	*Role2*

4.1. Runtime integration approach

As mentioned before, *runtime-integration* is able to incorporate DL models into a DES model, so that the resulting simulation model is an extension of the BPS model defined as a tuple $\mathcal{M}^* = (\mathcal{N}, \mathcal{P}_{\neg T}, S_m, W_m, PT_m)$ where:

- $\mathcal{N}$ is the Petri net model;
- $\mathcal{P}_{\neg T}$ is the subset of the simulation parameters that excludes the parameters related to the time perspective, i.e., $\mathcal{P}_{\neg T} = \mathcal{P} \setminus \mathcal{P}_T$
- S_m is a predictive model that generates the start time of each trace;
- W_m is a predictive model that predicts the waiting time between an event and the next one;
- PT_m is a predictive model that predicts the processing time of a given event.

RIMS defines $\mathcal{M}^*$ in three main steps: (i) definition of the DDPS elements, (ii) training of the models for the time perspective and, finally, (iii) definition of a simulator able to integrate all the elements.

4.1.1. Definition of the DDPS elements

In this step, we define $\mathcal{N}$, that is the process model and the set of the control-flow and resource simulation parameters $\mathcal{P}_{\neg T}$ over it. $\mathcal{P}_C$ includes branching probabilities for each decision point in $\mathcal{N}$ and $\mathcal{P}_R$ the resource allocation for activities. In particular, a resource pool, namely a role, is assigned to each activity within the process model. In Fig. 1, the elements reported in orange represent possible instances of the simulation parameters in $\mathcal{P}_{\neg T}$. Each activity is associated with a role and a probability is defined for the two branches of the gateway.

4.1.2. Training of the models

Differently from classical DDPS models, which define trace start times, event processing times, and event waiting times as distribution functions, $\mathcal{M}^*$ uses predictions returned by the predictive models, as in [10].

In particular, for the generation of start times, instead of fitting an inter-arrival distribution as classical DDPS models, RIMS involves a time series forecasting model, S_m, developed by Facebook and called Prophet [26] with the same configuration used in [10]. This model is capable of handling time series data that exhibit non-linear trends, seasonality, and other complex patterns.

For the prediction of the waiting and the processing times, RIMS uses two LSTM models with the same architecture used in previous works [10,25,27]. The LSTM architecture is shown in Fig. 3: it is composed of two stacked LSTM layers and a dense output. W_m and PT_m differ only in the features used to train each model.

To predict the waiting time, W_m requires the next activity's label, the end timestamp of the current activity, the day of the week, and intercase features such as the work-in-progress and the resources' occupation. Specifically, the work-in-progress wip_τ represents the number of ongoing traces in the event log or in the simulation at a given time τ. The resources' occupation ro_τ provides instead the occupancy percentage in terms of resources in use for each role specified in $\mathcal{P}_{\neg T}$ at

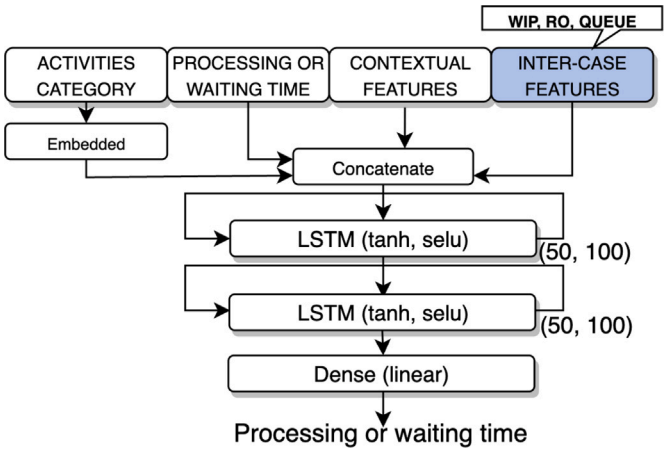


Fig. 3. LSTM architecture from [10] used to train W_m and PT_m. Different intercase features are used in RIMS and RIMS+ than those used in [10].

a given time τ. As the predicted times are added during the simulation, they affect all other running traces and, as a result, the intercase features are more accurate. For example, in case of the *runtime-integration* method (Table 4), the intercase features used to predict the waiting time between the activities A and B of case 1 would be such that $ro_{08:08} = [0, 1, 0]$, where 1 refers to the occupancy percentage of *Role1* at time 08:08 (both resources of *Role1* are busy, one with the activity B of case 3 and one with the activity C of case 2).

For predicting the processing time, the input features of PT_m are the same ones of W_m, except for the activity label, which is the one of the current event, and the time, which is its start timestamp. Finally, to encode the features we implemented the same pipeline as in [10], i.e., using embeddings for the activity label and extracting n-grams of fixed size for each trace as input sequences.

4.1.3. Integration

To integrate all the elements defined in the previous steps, we define a RIMS simulator able to interact with S_m, W_m and PT_m.

The pseudocode of Algorithm 1 illustrates the proposed method. It starts by generating the start times of all the new traces (Line 19) and, once they have been sorted, the *new_trace* function is used to generate the new traces, starting from the first one (Line 21) and proceeding with the others (Lines 22–24), as soon as the time between one trace and another is passed (Line 23).

The *wait_time* function, indeed, interrupts the simulation, even if the clock time is running, thus allowing to simulate the correct inter-arrival time among traces. Indeed, although the traces are independent one from the other, they compete for the same resources and refer to the same simulator clock time. The function *new_trace* simulates a given trace over $\mathcal{N}$ from start to end. To this aim, the function *next_activity* returns, at each step, the next activity in the model chosen by the model simulator according to $\mathcal{P}_C$. For example, in Fig. 1, the orange point represents a decision point where *next_activity* chooses to perform B rather than C, according to $\mathcal{P}_C$. For each activity α, the event executing the activity is created. The role in charge of executing the activity is also associated to the event (Line 5). If the activity is the first one in the model, the starting time of the event is set to the starting time of the trace s (Line 7), otherwise the waiting time required for executing the event e is computed (Lines 9–11). To this aim, the features of e (activity name, current time τ, and the value intercase features at time τ) are extracted and encoded, the W_m model queried and the waiting time t_w predicted (Line 9). Leveraging the *wait_time* function, the event e is enabled after t_w clock times (Line 10). The processing time spent by e is also predicted by querying the PT_m model by leveraging some features computed on e (Line 12). Before the event can be executed,

Algorithm 1 Runtime Integration Algorithm

Input : $[\mathcal{N}, \mathcal{P}_C, \mathcal{P}_R, S_m, PT_m, W_m] \leftarrow \mathcal{M}^*$
Input: Number of traces to generate, O
Input: S_m, PT_m and W_m: the predictive models that predict the trace start time, the event processing time and the inter-event waiting time, respectively
Input: τ global simulation time

1: **function** $new\_trace(\mathcal{N}, \mathcal{P}_C, PT_m, W_m, s)$
2: $\sigma \leftarrow \langle\rangle$
3: **for** α in $next\_activity(\mathcal{N}, \mathcal{P}_C, \sigma)$ **do**
4: $e \leftarrow \{\}$
5: $e.act \leftarrow \alpha$, $e.role \leftarrow get\_role(e.act, \mathcal{P}_R)$
6: **if** $e.act = start$ **then**
7: $e.ts \leftarrow s$
8: **else**
9: $e.t_w \leftarrow predict(W_m, [e.act, \tau, wip_\tau, ro_\tau])$
10: $wait\_time(e.t_w)$, $e.ts \leftarrow \tau$
11: **end if**
12: $e.t_p \leftarrow predict(PT_m, [e.act, e.ts, wip_\tau, ro_\tau])$
13: $e.r \leftarrow wait\_resource(e.role)$, $e.ts \leftarrow \tau$
14: $\sigma \leftarrow append(\sigma, execute(e, e.t_p))$
15: **end for**
16: **return** σ
17: **end function**
18:
19: $ST \leftarrow sort(predict(S_m, O))$
20: $\mathcal{L}_{sim} \leftarrow \emptyset$
21: $\mathcal{L}_{sim} \leftarrow new\_trace(\mathcal{N}, \mathcal{P}_C, PT_m, W_m, ST[0])$
22: **for** $j = 1$ *to* $O - 1$ **do**
23: $wait\_time(ST[j] - ST[j-1])$
24: $\mathcal{L}_{sim} \leftarrow \mathcal{L}_{sim} \cup \{new\_trace(\mathcal{N}, \mathcal{P}_C, PT_m, W_m, ST[j])\}$
25: **end for**

we have to ensure that an available resource from the assigned role can be allocated, otherwise, the activity has to wait in queue until a resource is available (Line 13). The function *wait_resource* returns the first available resource, potentially waiting the proper time if none of them is immediately available. The event can hence be executed and its execution added to the generated trace σ (Line 14). The procedure is then iterated on the next activity. When the end is reached, the simulated trace is returned. Fig. 1 illustrates an example of *runtime-integration*, showing the S_m, PT_m, and W_m models through green, purple and blue points, respectively.

4.2. RIMS+: Enhancing RIMS with queue discovery

In many processes, traces do not operate independently but rather compete for scarce resources [28]. The resource competition among traces generates queues and side-effects on traces' cycle times, namely waiting times. Therefore, to obtain a more accurate prediction of the waiting time between two events, we enrich the set of intercase features with a further one (*queue*) measuring the length of the queue for the required resource (see Fig. 3). Typically, queue mining approaches rely on event logs containing information on queueing dynamics. However, queueing information, such as the time an event spends waiting for a resource and/or the number of events competing for that resource, is often unavailable. Hence, before training the waiting time model enriched with the queue feature, W_m^q, it is necessary to extract the *queue* features from the event log.

The idea is to leverage the simulation to exactly replay the log and retrieve the queue length value at each specific time τ. Given the absence of further information, we make the assumption that resources are requested immediately after the completion of the previous activity, i.e., that resources are always available, if they are not busy with other cases. To this aim, Algorithm 2 takes as input $\mathcal{L}$, i.e., an event log sorted by start times and returns a log $\mathcal{L}_q$ where each event of each trace is enriched with a feature *queue* with the length of the queue

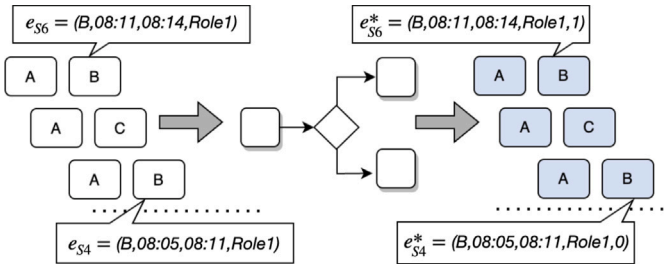


Fig. 4. Example of queue discovery. Algorithm 2 takes as input the three traces from Table 1 and leveraging the simulation returns the same traces enriched with the attribute *queue*.

Algorithm 2 Queue Discovery

Input : $\mathcal{L}$ sorted event log, $\mathcal{P}_R$

1: **function** $new\_trace\_queue(\sigma)$
2: $\sigma^* \leftarrow \langle\rangle$
3: **for** e_i in σ **do**
4: $e_i.queue \leftarrow retrieve\_queue(e_i.r, \tau)$
5: $wait\_time(e_i.t_{start} - e_{i-1}.t_{end})$
6: $execute(e_i, e_i.t_p)$
7: $\sigma^* \leftarrow append(\sigma^*, e_i)$
8: **end for**
9: return σ^*
10: **end function**
11:
12: $\mathcal{L}_q \leftarrow \emptyset$
13: **for** $\sigma \in \mathcal{L}$ **do**
14: $\mathcal{L}_q \leftarrow \mathcal{L}_q \cup new\_trace\_queue(\sigma)$
15: **end for**

at the time of the execution of the event. For each trace σ in $\mathcal{L}$, the function *new_trace_queue* replays the given trace, event by event. First, the function *retrieve_queue* computes the value of the length of the queue at time τ (Line 4), then the elapsed time between the current event and the previous one (i.e., the waiting time) is waited (Line 5), and, in the end, the event e_i is executed (Line 6). Finally, σ^* records the event with the new attribute $e_i.queue$ (Line 7). Once $\mathcal{L}_q$ is obtained, W_m^q is trained for the waiting time prediction (including also the queue length in the encoding). RIMS+ leverages W_m^q to obtain the simulation model $\mathcal{M}_q^* = (\mathcal{N}, \mathcal{P}, S_m, W_m^q, PT_m)$ through Algorithm 1. Fig. 4 describes an example of queue discovery related to the activity B for case 1 and 2 of the log in Table 1.

4.3. Enhancing RIMS/RIMS+ with decision mining

As shown in Section 3, using branching probability to define the control-flow of a simulation model could produce unrealistic process behaviors. The paths of the traces may depend on their attributes and/or the state of the process. Therefore, to enhance the quality of the entire simulation, we rely on predictions that learn the more likely next activity or the branching probability at a decision point from data by taking into account data attributes and the events occurred up to that point in the trace. More specifically, we define RIMS^{DT} that, for each decision point in the process model, may employ a next activity prediction model that at runtime provides predictions on the next activity or the probability distribution.

In the following, we introduce the new simulation parameters of RIMS^{DT} (Section 4.3.1), the training of the predictive models used for the decision points (Section 4.3.2) and, finally, their integration into the simulation (Section 4.3.3).

4.3.1. Redefinition of DDPS elements

RIMS^{DT} defines $\mathcal{M}_{dp}^*$ as a variation of the $\mathcal{M}^*$ simulation model, particularly by redefining $\mathcal{P}_{\neg T}^* = \mathcal{P}_R \cup \mathcal{P}_A^* \cup \mathcal{P}_C^*$ as the subset of sim-

Algorithm 3 Decision Mining

Input : $\mathcal{L}^r$ aligned log, $\mathcal{N}$, $\{dp_1, ...dp_n\} \in \mathcal{N}$, t and $\mathcal{P}_C$

1: **function** *mining_decision_point*(dp, $\mathcal{P}_C$)
2: $\mathcal{L}^r_{dp} \leftarrow filter\_log\_decision\_point(\mathcal{L}^r, dp)$
3: $\mathcal{E}_{dp} \leftarrow encoding\_prefix(\mathcal{L}^r_{dp})$
4: $\mathcal{E}^{train}_{dp}, \mathcal{E}^{test}_{dp} \leftarrow split\_log(\mathcal{E}_{dp})$
5: $DT^{dp}_m \leftarrow training\_dt(\mathcal{E}^{train}_{dp})$
6: **if** $evaluation(DT^{dp}_m, \mathcal{E}^{test}_{dp}) > t$ **then**
7: return DT^{dp}_m
8: **else**
9: return $\mathcal{P}^{dp}_C$
10: **end if**
11: **end function**
12: **for** $dp \in \mathcal{N}$ **do**
13: $\mathcal{P}^*_C \leftarrow \mathcal{P}^*_C \cup mining\_decision\_point(dp, \mathcal{P}^{dp}_C)$
14: **end for**

ulation parameters that excludes those related to the time perspective. The resource allocation $\mathcal{P}_R$ is defined as in $\mathcal{P}_{\neg T}$, while $\mathcal{P}^*_A$ represents the simulation parameters for generating the trace attributes, $p_1, \dots, p_n$, of each simulated trace. As shown in [18,20,21], the use of data attributes has a significant impact on the discovery and prediction of the next activity from a decision point. Therefore, if the traces in $\mathcal{L}$, which are used to construct $\mathcal{M}^*_{dp}$, include data attributes, we define $\mathcal{P}^*_A$ as a set of probability functions capable of estimating each attribute during the simulation.

For each decision point dp_i in $\mathcal{N}$, we train a predictive model DT^i_m, which is included into $\mathcal{P}^*_C$ if its accuracy is higher than a certain threshold; otherwise, the probabilities $\mathcal{P}^i_C$ are used to determine the next activity from the decision point. Hence, $\mathcal{P}^*_C = \{\mathcal{P}^{*1}_C, \dots, \mathcal{P}^{*n}_C\}$, is a set containing $\mathcal{P}^{*i}_C$ for each dp_i in $\mathcal{N}$, with $n = |\mathcal{N}_{dp}|$, which can be either the predictive model predicting the next activity (when its accuracy is higher than the threshold) or alternatively the corresponding branching probability.

The predictive model is defined as a classifier that estimates the next activity, as in [20,21]. The input is represented by features related to the current trace, as well as other simulation state features, while all possible activities occurring directly after the decision point are designated as labels. As predictive model we used the Decision Tree, DT_m which is capable of predicting the next activity in a white-box manner. This allows us to observe the reasons for routing, and furthermore, enables the definition of what-if scenarios based on it. Rims integrates each DT_m at runtime so as to exploit the data attributes and trace history for selecting the next activity to perform.

4.3.2. Data preprocessing and training

In order to train a predictive model DT_m with real data and get predictions for simulated data, we have to align the real data to the simulated one. To this aim, before training DT_m, each trace of $\mathcal{L}$ is aligned with the given process model $\mathcal{N}$ through the alignment method [29,30] and transformed into a format that is compliant to the simulation model. An aligned trace can contain three symbols: SM (Synchronous Move), MM (Model Move) and LM (Log Move). SM denotes a perfect alignment between the trace and the model on the current activity. MM is a deviation between the trace and the execution sequence of $\mathcal{N}$, meaning that the execution of an activity has been skipped in the trace. LM represents that an activity has been executed in the trace, though there is no corresponding activity on the model. Differently from conformance checking approaches, here MMs also include the invisible transitions.[3] Finally, each aligned trace is transformed into a trace compliant to the simulation model through two actions: (i) LM moves are removed; (ii) MMs are added as events with event attributes (e.g., timestamp, resource) set to null. Therefore, for each original trace $\sigma \in \mathcal{L}$, a version of the trace $\sigma^r \in \mathcal{L}^r$ compliant to the simulation model is obtained by aligning σ to the process model $\mathcal{N}$. Fig. 5 illustrates the alignment and transformation preprocessing for case 1 from Table 1. The trace is first aligned to the process model in Fig. 1 and then transformed into a compliant trace. The SMs concern the activities A, B and D, the MMs are I_1 and I_3 and, finally, E is an LM move since the activity is only present in the log.

The model-compliant traces are then used as starting point to train the predictive model DT_m. Specifically, for each decision point dp_i and for each model-compliant trace σ^r, we use the trace prefix up to the decision point as training data and the activity immediately following the decision point as class label. For instance, considering the decision point dp_2 of $\mathcal{N}$ in Fig. 1 and the model-compliant trace σ^r in Fig. 5, the prefix $\langle A, B, I_1 \rangle$ is used as training trace, while activity D serves as class label.

To train the Decision Tree model we apply two different types of encoding: the *last payload intra-case* and the *last payload inter-case* encoding. While the former only includes control flow information of the trace under analysis and data payload information related to its last event, the latter also takes into account concurrent traces. The analysis of both these encodings allows us to understand whether the path of a trace in the process is influenced by other traces. Moreover, the *last payload* allows us to take into account also the data payloads, while preventing the creation of long and redundant encodings in traces that contain a large number of events.

More specifically, given a decision point dp, a model-compliant trace $\sigma^r \in \mathcal{L}^r$ of length m and its prefix σ^r_k, assuming that the first event non-related to an invisible transition that precedes dp occurs at position $k < m$, the *last payload intra-case* encoding is obtained by concatenating the following features: (i) the prefix $\langle e^r_1.act, \dots, e^r_k.act \rangle$ where $k < m$, that is the sequence of activities up to the last activity before dp, (ii) $role(e^r_k.r)$, that is the last role of the resource r involved in the last event; and finally (iii) $\langle \sigma^r.p_1, \dots, \sigma^r.p_n \rangle$, that are the trace attributes. The *last payload inter-case* encoding adds time and inter-case features to the previous encoding, that is $e^r_k.t_{complete}, wip_{\tau_k}, ro_{\tau_k}, q_{\tau_k}$. For instance, given the decision point dp_2 in Fig. 1 and the model-compliant trace of the example in Fig. 5, the last event preceding the decision point that is not an invisible transition is the second event. The considered prefix is therefore $\langle A, B \rangle$ and the last event role and timestamp are the ones related to the second event.

The pseudocode of Algorithm 3 describes how $\mathcal{P}^*_C$ is defined: it takes as input $\mathcal{L}^r$ a log compliant to the simulation model, $\mathcal{P}_C$ and $\mathcal{N}$ as defined in Section 4.1.1. For each decision point dp_i in the process model $\mathcal{N}$ the function *mining_decision_point* builds the corresponding DT^i_m and determines whether $\mathcal{P}^{*i}_C$ corresponds to the branching probability of the decision point dp_i, or to the predictive model DT^i_m. The function, first of all, uses the $filter\_log\_decision\_point$ to retrieve from $\mathcal{L}^r$ only the traces that pass though dp_i (Line 2). Then *encoding* function encodes the filtered log $\mathcal{L}^r_{dp_i}$, according to the specific type of encoding for training DT^i_m (Line 3). The function *split_log*, is used to obtain $\mathcal{E}^{train}_{dp_i}, \mathcal{E}^{test}_{dp_i}$ (Line 4), then *training_dt* uses $\mathcal{E}^{train}_{dp_i}$ to train the Decision Tree model DT^i_m (Line 5). Finally, the discovered model is evaluated with $\mathcal{E}^{test}_{dp_i}$ in order to understand whether its accuracy is higher than a certain threshold and hence the predictive model can be used to predict the next activity after the decision point dp_i. If the accuracy is not high enough, the branching probability $\mathcal{P}^{dp}_C$ is returned. The accuracy of the predictions in the function *evaluation* is measured in terms of F-score macro-average.[4]

[3] The Petri net model may include invisible transitions, namely transitions with no associated label that do not represent an actual activity and are hence considered as instantaneous during the simulation of the corresponding process model $\mathcal{N}$, as they are required for the simulation.

[4] The metrics $macro\_f1\_score = \frac{1}{l}\sum_{j=1}^{l} f1\_score_j$, where l represents the total number of class labels in DT^i_m. We used the $macro\_f1\_score$ to evaluate each DT^i_m to prevent predictions that exclude certain labels due to imbalanced events distribution across possible activities from a decision point dp_i.

Algorithm 4 Next activity

Input : $\mathcal{N}$ process model, P_C^*, σ

1: **function** $next\_activity(\mathcal{N}, P_C^*, \sigma)$
2: $p_{act} \leftarrow actual\_place\_model(\mathcal{N}, \sigma)$
3: $\mathcal{D} \leftarrow extract\_decider(P_C^*, p_{act})$
4: **if** $type(D) = DT_m$ **then**
5: $\epsilon \leftarrow encoding(\sigma)$
6: $next \leftarrow predict(\mathcal{D}, \epsilon)$
7: **else**
8: $next \leftarrow apply\_probability(\mathcal{D}, p_{act})$
9: **end if**
10: return $next$
11: **end function**

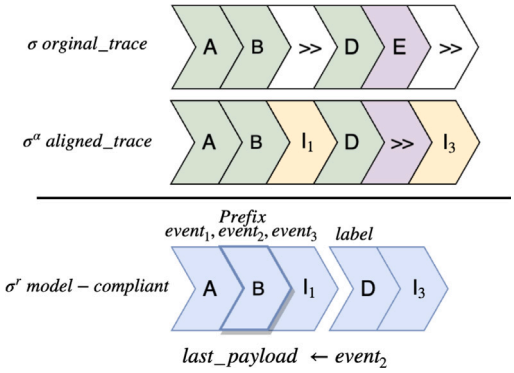


Fig. 5. The color green represents SM moves instead with yellow and purple we have the MM and LM moves, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.3.3. Integration of decision mining

Once P_C^* is defined, it is integrated into RIMS^{DT} by redefining the *next_activity* function of Algorithm 1. This function applies the corresponding probability or predictive model, as detailed in Algorithm 4. The *next_activity* function takes the same inputs as the previous definition, except P_C^*. First of all, *actual_place_model* determines the current position of the running trace σ within the process model by identifying the place (Line 2). Then the function *extract_decider* retrieves from P_C^* the corresponding method for p_{act} (Line 3). If D is a DT_m model, the trace is processed by the encoder and used as input for the predictive model (Line 5–6). Otherwise, the *apply_probability* function determines the next activity based on probabilities (Line 8).

In particular, each $DT_m^i \in P_C^*$ is capable of returning either the predicted class, namely *next*, or the probability distribution of classes. In this latter case the class probability distribution is used by the function *predict* (Line 6) to define *next* as the next activity to perform. The use of predictive probability distributions could prevent unbalanced outcomes, especially when probabilities are close to each other. For instance, Table 5 reports the output (class label and related probabilities) of the Decision Tree DT_m^2 related to dp_2 for a certain trace history and last data payload (e.g., $\langle A, B, Role_1 \rangle$). In this example, the probabilities related to the two class labels I_2 and D are very close, thus revealing that the prediction with the highest probability could not always be correct. If DT_m^2 always predicts only the most probable class, the prediction returned by the Decision Tree will prevent the simulation of traces in which the specific trace history and data payload is followed by the activity I_2 (traces that represent almost half of the traces with the same trace history and data payload in the training set). Leveraging the probability distribution as branching conditions for the choice of the next activity would instead guarantee the simulation of these slightly infrequent cases as well as of infrequent class labels.

Table 5
Output of the Decision Tree DT_m^2 for a given trace history and data payload.

Class label	Probability
D	0.55
I_2	0.45

5. Related work

We can roughly classify the existing literature related to BPS applied to PPM into three groups: DDPS (Data-Driven Process Simulation), DL (Deep Learning) and hybrid approaches. In the next subsection we describe the first two groups, while in the next one we focus on hybrid approaches.

5.1. DDPS and DL simulation models

The first group, namely DDPS, can discover the simulation model from the event log, with an automated [2–5] or semi-automated [6,7] extraction. Typically, a DDPS model is created through two main steps. First, the process model is discovered from the event log, then various parameters are adjusted to optimize the similarity between traces generated by the DDPS model and a subset of traces in the event log (test set). The second group of approaches leverages DL models, widely used in predictive process monitoring, to generate the most likely remaining sequence of events of an ongoing case, as in [13], or to generate the entire trace from scratch [25]. Camargo et al. [27] compare the two different groups of approaches and highlight their strengths and weaknesses. In particular, the DDPSs are suitable to capture the sequence of activities of a process and to define what-if scenarios. Whereas, DL models outperform the former in capturing the time perspective, particularly the waiting times between activities. Hybrid simulation models, as RIMS, aim to leverage the advantages and avoid the weaknesses of the two methods, thereby creating a more accurate simulation model that represents the real process.

5.2. Hybrid simulation models

Recent research has investigated the use of machine learning or neural networks to predict outcomes or decisions within a DES model in different fields [14–17]. Bergmann et al. [15] propose the use of neural networks and conventional simulation methods to improve decision-making when system knowledge is insufficient. De la Fuente et al. [17] use ANN (Artificial Neural Networks) inside a DES model to simulate a simple banking process in which a customer's loan application is accepted or rejected based on the ANN model. Pender et al. [16] propose combining simulation and machine learning techniques to predict response times for processor sharing queues, which are queueing systems where multiple jobs share a common processing resource. Specifically, the proposed approach uses the data generated by the Discrete Event Simulation to train several machine learning methods that are then compared. The integration of predictive models in these works shows how to overcome the unrealistic and oversimplified assumptions typically adopted in DES simulators and/or how to represent complex behaviors that are difficult to capture with simulation parameters.

Although the integration of DDPS and DL methods has been studied in several research areas only Camargo et al. [10] apply it to discover a BPS model from a log. Specifically, the framework called *Dsim* applies process mining techniques to uncover a stochastic process model from a log and then a DL model is employed to assign timestamps to the events generated by the stochastic model. However this method applies the DL model after the DES simulator has generated all the traces, and the consequence of this is, as shown in Section 3, that the order of the queue is not always respected. Our approach aims at overcoming this issue by integrating the DL model inside the DES simulator, so as to return runtime predictions, i.e., predictions provided during the simulation of the case.

Table 6
Event log and queue statistics. The highlighted logs are those considered for Rims+.

Log	Type	#Traces	#Events	#Activity	Queue mean	Avg. trace length
BPI17W	real	30 276	240 854	8	1646	7.96
BPI12W	real	8616	59 302	6	1115	6.88
Cvs Pharmacy	syn	10 000	103 906	15	59	10.39
Confidential 1000	syn	800	21 221	29	9.91	26.53
SynLoan	syn	2000	43 164	25	68.89	39
Confidential 2000	syn	1670	44 373	29	2.37	26.57
ConsultaDataMining	real	954	4962	16	0	5.2
Production	real	225	4503	24	1.17	20
PurchasingExample	syn	608	9119	21	0.30	15

In this paper, Rims provides a tight integration of the predictions of the DL model at runtime during the simulation, thus enabling a correct management of the queue. de Leoni et al. [18] propose a simulation model that incorporates a logistic regression model that from a process state predicts the next activity of each decision point. However, the simulation model is only evaluated in terms of control flow without considering and simulating the other perspectives, which, as we show in Section 7, are interconnected. In addition, the approach requires Data Petri Nets (DPNs), which makes a direct comparison with other methods difficult.[5] In contrast, Rims takes into account in an integrated way the control flow, resource, time and data perspective and can be used also on event logs without data attributes.

6. Evaluation setting

In this section we introduce the research questions, the datasets and the experimental setting (including the metrics) used in the evaluation.

6.1. Research questions

We aim at investigating the following research questions:

RQ1 How does Rims and its variants perform in terms of simulation quality compared to the other state-of-the-art approaches?
RQ2 Which variant of Rims is best suited for event logs based on the queue information present in them?

RQ1 aims to compare all the variants of Rims with state-of-the-art i.e. the single approaches (DDPS and DL) and the hybrid one, *post-integration*. **RQ2** aims to identify the best Rims variant and assess the impact of the queue feature on the quality of the time and control perspectives in the simulation, as well as the impact of introducing predictive models for the control flow.

6.2. Datasets

For the evaluation we considered nine event logs – four real-life and five synthetic logs obtained by simulating real-life processes – containing both start and end timestamps, as this information is necessary to define the predictive models used as simulation time parameters.

Table 6 reports the number of traces, events, activities, and the mean value of *queue* calculated by Algorithm 2.

Additionally, we consider six different semi-synthetic logs, denoted as $\mathcal{L}^i_{syn}$, $i = 1, \dots, 6$, to properly analyze the impact of Rims+ with different levels of queuing. To generate the logs, we define the simulation model $\mathcal{M}^{\mathcal{L}_{syn}}$ inspired by the *BPIC2012* [31] process. Each $\mathcal{L}^i_{syn}$ log contains 4 000 traces and utilize the same $\mathcal{N}^{\mathcal{L}_{syn}}$ and simulation parameters with the exception of $\mathcal{P}_R^{\mathcal{L}_{syn}}$ i.e., a special resource configuration for creating different levels of queues.[6]

[5] A DPN is a Petri net in which transitions can write variables. A transition might have a data-dependent guard and execute write operations on a specified collection of variables, which necessarily need data attributes.

Table 7
Hyperparameters used to optimize the Decision Tree models of Rims^{DT}.

Parameter	Distribution	Value
criterion	*Categorial*	$\{gini, entropy, log\_loss\}$
max_depth	*Uniform*	$[1, \dots, 21]$
min_samples_split	*Uniform*	$[2, \dots, 11]$
min_samples_leaf	*Uniform*	$[3, \dots, 26]$

6.3. Experiment setup

We aim to evaluate the performance of DDPS, generative DL models, *Dsim*, Rims, Rims+ and Rims^{DT}.

To evaluate the DDPS models we used the *Simod* tool [2], which is able to create a BPS model in a fully automated way. For the DL approach, we exploit the DeepGenerator tool [25], which is able, through an LSTM model, to generate complete event logs from zero-size prefixes. The tool *Dsim* [10] is used as *post-integration* approach. In order to carry out a fair comparison, the Petri net model,[7] $\mathcal{N}$, and the simulation parameters, $\mathcal{P}_{\neg T}$, are the same for all methods that require them. In addition, Rims used the same LSTM models as *Dsim* to properly evaluate the effect of different ways of integrating predictions in the simulation. Rims+ is applied only to event logs that exhibit significant queues, as highlighted in Table 6. The level of queue for each log in Table 6 was determined by applying Algorithm 3, as this information is not already included in the logs. Specifically, a log is considered to have significant queue if the average queue mean reaches at least 10% of the average trace length.

To train the LSTM and the Decision Tree DT_m models, we split each event log using the hold-out method and a time split criterion, into a 60%–20%–20% partitioning for training, validation and testing, respectively. The DT_m models are hyperparameter-optimized using *Hyperopt*,[8] across the parameter ranges given in Table 7. The LSTM models are instead optimized using the same hyperparameter optimization used in [10].

The DT_m models involved in Rims^{DT} also use trace attributes in both the *last payload intra-case* and *last payload inter-case* encoding. The simulation parameters $\mathcal{P}^*_A$ of $\mathcal{M}^*_{dp}$ are therefore also needed for the event logs containing trace attributes i.e., *BPI12W*, *SynLoan*, and *BPI17W*. Table 8 describes the random distribution associated with the trace attributes for these three logs.

For each event log in Table 6, we define $\mathcal{M}^*$ (Rims) and $\mathcal{M}^*_{dp}$ (Rims^{DT}). Moreover, for the logs with a significant level of queueing we

[6] The logs $\mathcal{L}^i_{syn}$ and the simulation models $\mathcal{M}^{\mathcal{L}^i_{syn}} = (\mathcal{N}^{\mathcal{L}^i_{syn}}, \mathcal{P}^{\mathcal{L}^i_{syn}})$ used to generate these logs are available, together with the code and all the material used in the evaluation at https://github.com/francescameneghello/RIMS/tree/RIMS_decision_points.

[7] *Simod* and *Dsim* approaches use BPMN as the process model. Thus, before defining $\mathcal{M}$ for Rims, we first translate the BPMN into the corresponding Petri net.

[8] Hyperopt is a Python library for serial and parallel optimization over awkward search spaces, https://hyperopt.github.io/hyperopt/.

Table 8
Data payloads attributes. The non-uniform random variate generation involves generating random variables with specific distributions. For the generation of trace attributes, we use the probability distributions mined from the event log.

Log	Trace attributes	Type	Distribution
BPI12W	Loan amount requested	Continuous	$Exp(\lambda = 14451.153)$
SynLoan	Loan amount requested	Continuous	$Norm(\mu = 15163.62, \sigma = 12325.49)$
BPI17W	Loan amount requested	Continuous	$Exp(\lambda = 16224.34)$
	Loan Goal	Discrete	Non-uniform random variate generation
	TypeApplication	Discrete	Non-uniform random variate generation.

Table 9
Summary of all RIMS variants indicating the perspectives of the process in which the predictive models are used, the optional feature considered (with ✓ and x indicating their presence or absence) and the prediction type.

	Time predictive models		Decision point predictive models			
Rims variant	Included	Queue feature	Included	Intracase feature	Intercase feature	Prediction type
Rims	✓	x	x	x	x	x
Rims+	✓	✓	x	x	x	x
$\text{Rims}^{intra}_{DET}$	✓	x	✓	✓	x	DET
$\text{Rims}^{intra}_{PROB}$	✓	x	✓	✓	x	$PROB$
$\text{Rims}^{inter}_{DET}$	✓	x	✓	x	✓	DET
$\text{Rims}^{inter}_{PROB}$	✓	x	✓	x	✓	$PROB$
$\text{Rims+}^{inter}_{DET}$	✓	✓	✓	x	✓	DET
$\text{Rims+}^{inter}_{PROB}$	✓	✓	✓	x	✓	$PROB$

also define $\mathcal{M}^*_q$ (Rims+) and $\mathcal{M}^*_{q,dp}$ (Rims+^{DT}). In particular, we evaluate four variants of Rims^{DT}, each representing a unique combination of one encoding type – either *last payload intra-case* or *last payload inter-case* – and one DT_m model prediction type: either single class or probability distribution over classes. We denote these four variants with the following names: $\text{Rims}^{intra}_{DET}$, $\text{Rims}^{intra}_{PROB}$, $\text{Rims}^{inter}_{DET}$ and $\text{Rims}^{inter}_{PROB}$, where *intra* and *inter* refer to the use of *last payload intra-case* and *last payload inter-case* encoding, respectively. *DET* and *PROB* indicate single class and probability distribution over classes predictions, respectively.

For Rims+^{DT}, we exclusively conduct the evaluation with the *last inter-case payload* encoding ($\text{Rims+}^{inter}_{DET}$ and $\text{Rims+}^{inter}_{PROB}$), because of the significant impact of inter-case features on the logs used for Rims+ and the absence of additional insights gained from using the last payload intra-case encoding.

Table 9 includes all the variations of Rims applied to event logs, along with details of the respective predictive models integrated for both the time and control-flow perspectives.

Rims is developed with the SimPy[9] library, a process-based Discrete Event Simulation framework in Python.

6.4. Evaluation metrics

To assess the ability of the approaches proposed in Section 6.3 to replicate the observed behavior of a process, we employed metrics outlined in the paper by Chapela et al. [32]. In particular, they propose multiple metrics to evaluate the quality of simulation models across control flow, temporal, and congestion dimensions. The metrics are calculated by comparing the log $\mathcal{L}_{sim}$ generated by each model with a ground-truth log $\mathcal{L}_{test}$, i.e., a testing subset of the event log.

For evaluating the control-flow dimension of the simulation, we employ the following metrics: Control-flow Log Distance (CFLD) which minimizes the sum of the distances calculated by pairing the traces of $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$ using the Hungarian algorithm [33], and N-Gram Distance (NGD), which compares the Directly-Follows Graph (DFG) of $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$ represented by two histograms, respectively. For the NGD metric, we set the dimension of N to 2 and 3 (2-Gram and 3-Gram), meaning that the metrics evaluates pairs and triplets of activities present in $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$. For all control-flow metrics, the lower the value, the closer the simulated log is to the real one. Since DDPS, *Dsim*, Rims, and Rims+ utilize the same $\mathcal{N}$ process model, to evaluate the control-flow, we compute the metrics only on LSTM, Rims, and Rims^{DT} with its variants.

The evaluation of the temporal dimension includes three metrics: Absolute Event Distribution (AED) for analyzing the trend of the distribution of all events over time; Circadian Event Distribution (CED) for comparing seasonality, particularly the distribution of events by day of the week; and finally, Relative Event Distribution (RED) for analyzing the temporal distribution of events with respect to the origin of the case (i.e., the case arrival).

Finally, the congestion dimension is captured by two metrics that assess the simulation model's ability to replicate process congestion: the Case Arrival Rate (CAR) and the Cycle Time Distribution (CTD) metrics. CAR compares case arrival patterns and the number of arrivals over a set of process timestamps, while CTD compares the cycle times of traces in $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$. Even for time and congestion metrics, the lower the value, the closer the simulated log is to the real one.

While AED and CTD are general metrics – the former captures differences in the distribution of events across the entire time spectrum of $\mathcal{L}_{test}$ and $\mathcal{L}_{sim}$, while the latter considers the entire cycle time of traces – CED, RED and CAR are metrics that focus on specific aspects of the simulated traces (e.g., seasonality, arrival rate). The first group of metrics is hence more suitable for providing an overall evaluation of the simulated traces, while the second group, investigating specific aspects of the simulation, results to be very useful in identifying the reasons for the differences observed in AED and CTD.

For all the time and congestion metrics, the analyzed events of $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$ are represented by two histograms. The latter are then transformed into two distribution functions to compute the *1st Wasserstein Distance (1WD)* [34], which is a computationally efficient variation of the *Earth Mover's Distance (EMD)* [35].

Finally, we define L^{traces}, which compares the length between traces in $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$, since time metrics, especially those concerning congestion, are also influenced by the number of simulated events. L^{traces} is calculated as the average of the absolute errors between the lengths of the traces in $\mathcal{L}_{sim}$ and $\mathcal{L}_{test}$, paired using the Hungarian algorithm [33].

For each method 25 simulations are performed while 10 simulations are computed for each $\mathcal{L}^i_{syn}$ in Fig. 8.

7. Evaluation results

In this section we report the results of the evaluation and answer the research questions of Section 6.1. We divide the analysis of the results for the two research questions, **RQ1** and **RQ2**, based on the presence of queue information in the logs, as this allows for the application of the Rims+ and Rims+^{DT} variants. We begin with an analysis of the general metrics—L^{traces}, CLFD for the control flow, and AED, CTD for the time perspective. This provides an initial overview of the results, enabling a broad comparison of the different methods' performance. We then use specific metrics to delve deeper into the general findings (Figs. A.9, A.11, A.10, A.12).

[9] https://simpy.readthedocs.io/en/latest/

Table 10
Rankings for the CLFD and L^{traces} metrics are defined by calculating p-values on the simulation results across all methods. Table includes the average values obtained in all logs and the corresponding positions in the rankings. The best method(s) are indicated in boldface.

	Method	Confidential 2000	Rank	Purchasing example	Rank	Production	Rank	Consulta DataMining	Rank
CLFD	DDPS	0.2576	5	**0.1874**	**1**	0.7549	6	0.1973	2
	LSTM	0.2307	4	0.4492	8	**0.6784**	**1**	0.1919	2
	Dsim	0.2576	5	**0.1874**	**1**	0.7549	6	0.1973	2
	Rims	0.2576	5	**0.1874**	**1**	0.7549	6	0.1973	2
	$Rims^{intra}_{DET}$	0.2856	8	**0.1880**	**1**	0.7241	2	0.1876	2
	$Rims^{intra}_{PROB}$	**0.2031**	**1**	**0.1870**	**1**	0.7566	4	0.1959	2
	$Rims^{inter}_{DET}$	**0.2053**	**1**	**0.1852**	**1**	0.7339	3	**0.1719**	**1**
	$Rims^{inter}_{PROB}$	**0.2049**	**1**	**0.1856**	**1**	0.7507	4	0.1959	2
L^{traces}	DDPS	4.5528	2	1.6935	5	0.1832	2	1.1081	3
	LSTM	**2.2426**	**1**	5.7648	8	0.6616	7	**0.6813**	**1**
	Dsim	4.5528	2	1.6935	5	0.1832	2	1.1081	3
	Rims	4.5528	2	1.6935	5	0.1832	2	1.1081	3
	$Rims^{intra}_{DET}$	5.8212	8	**1.2626**	**1**	2.0448	8	1.0301	3
	$Rims^{intra}_{PROB}$	5.4767	5	1.5323	3	0.2191	2	1.1344	3
	$Rims^{inter}_{DET}$	5.5879	5	**1.2475**	**1**	0.3680	6	0.8888	2
	$Rims^{inter}_{PROB}$	5.5754	5	1.6063	3	**0.1693**	**1**	1.1341	3

Table 11
Rankings for the AED and CTD metrics are defined by calculating p-values on the simulation results across all methods. Table includes the average values obtained in all logs and the corresponding positions in the rankings. The best method(s) are indicated in boldface.

	Method	Confidential 2000	Rank	Purchasing example	Rank	Production	Rank	Consulta DataMining	Rank
AED	DDPS	400.4493	2	**655.5610**	**1**	1302.7195	8	665.1526	8
	LSTM	780.2948	8	1161.3856	2	164.8390	7	561.9059	7
	Dsim	616.7942	3	1223.9957	3	**123.7305**	**1**	257.8716	6
	Rims	618.6753	3	1265.7655	5	137.4757	3	**238.1313**	**1**
	$Rims^{intra}_{DET}$	**316.4249**	**1**	1254.4631	5	137.4701	3	**240.3870**	**1**
	$Rims^{intra}_{PROB}$	615.9303	3	1265.0231	5	138.3194	3	**239.8577**	**1**
	$Rims^{inter}_{DET}$	616.0715	3	1242.9644	4	**123.6104**	**1**	**242.2535**	**1**
	$Rims^{inter}_{PROB}$	616.4784	3	1265.6600	5	145.3603	6	**238.5318**	**1**
CTD	DDPS	65.1559	7	590.9271	3	86.2018	7	76.2188	7
	LSTM	10.3649	6	638.7778	8	35.7244	5	95.4625	8
	Dsim	**2.2354**	**1**	633.5232	7	**23.1680**	**1**	64.1207	6
	Rims	**2.0388**	**1**	584.6206	3	29.3849	4	**43.2875**	**1**
	$Rims^{intra}_{DET}$	1076.3854	8	**576.3109**	**1**	188.9627	8	**43.1478**	**1**
	$Rims^{intra}_{PROB}$	**1.9731**	**1**	585.6424	3	28.6498	3	**44.4243**	**1**
	$Rims^{inter}_{DET}$	**3.0153**	**1**	**574.2618**	**1**	25.9467	2	**41.4484**	**1**
	$Rims^{inter}_{PROB}$	**2.1393**	**1**	583.1438	3	36.1742	5	**41.3704**	**1**

Tables 10 and 12 show the comparison between the different simulation models with respect to the control-flow perspective.

In particular, for each log, the methods are ranked based on the mean values of the two metrics, L^{traces} and CLFD, calculated across all the simulated logs. Ties in the rankings between two or more methods indicate that the differences are not statistically significant, as determined by a pairwise comparison t-test where the *p*-value exceeds 0.05. Regarding Rims, DDPS, *Dsim* methods in Table 10, like Rims+ in Table 12, report identical values, as all share the same process model and therefore they produce traces with the same control-flow.[10] $Rims^{DT}$ also shares the same process model as the others, but it also leverages the DT models in decision points, and it can hence produce traces with an improved control-flow quality.

In summary this evaluation compares three different techniques for the generation of traces' control flow: a pure DL generative technique (LSTM), a process model with fixed probability transition on decision points (DDPS, *Dsim*, Rims, Rims+) and the integration of a process model with white-box predictive models ($Rims^{DT}$, $Rims+^{DT}$).

[10] Even though Rims, DDPS, *Dsim* and Rims+ report the same values, we include them in Tables 10 and 12, respectively, to ensure accurate computation of the rankings.

Tables 11 and 13 report the comparison of the approach by taking into account the time and the congestion dimension, without and with a significant level of queue, respectively. The rankings are computed in the same manner as with the control flow metrics. For these dimensions, each simulation method reports, in general, different results. Indeed, LSTM and DDPS, during the simulation, employ different methods to estimate processing and waiting times than *Dsim*, Rims and its variants. On the other hand, *Dsim*, Rims and $Rims^{DT}$ utilize the same predictive models, but they differ for the type of integration: *post-integration* for the first one and *runtime-integration* for the last two. Additionally, $Rims^{DT}$ produces variations in the control-flow, as mentioned before, which in turn affect the time and congestion metrics of the entire process. Finally, Rims+ and $Rims+^{DT}$ utilize a different version of predictive model for the waiting time by incorporating the queue feature and apply the *runtime-integration* method.

Fig. 6 summarizes the ranking of the methods across all perspectives – time, congestion and control-flow – for all the event logs used in the evaluations. In terms of time and congestion dimension, AED and CTD are general metrics suitable to provide an overall evaluation of the simulated traces. The former captures differences in the distribution of events across the entire time spectrum of $\mathcal{L}_{test}$ and $\mathcal{L}_{sim}$, while the latter considers the entire cycle time of traces. We consider AED and CTD together to define the optimal method(s) in Fig. 6 and, for

Table 12
Rankings for the CLFD and L^{trace} metrics are defined by calculating p-values on the simulation results across all methods. Table includes the average values obtained in all logs and the corresponding positions in the rankings. The best method(s) are indicated in boldface.

	Method	BPI17W	Rank	BPI12W	Rank	Cvs Pharmacy	Rank	Confidentia 1000	Rank	SynLoan	Rank
CLFD	DDPS	0.4735	8	0.4310	2	0.2602	7	0.2261	6	0.5865	8
	LSTM	0.3970	7	**0.2944**	**1**	0.2715	11	0.2519	10	**0.3699**	**1**
	Dsim	0.4735	8	0.4310	2	0.2602	7	0.2261	6	0.5865	8
	Rims	0.4735	8	0.4310	2	0.2602	7	0.2261	6	0.5865	8
	Rims+	0.4735	8	0.4310	2	0.2602	7	0.2261	6	0.5865	8
	$Rims^{intra}_{DET}$	0.3368	3	0.4944	11	**0.2212**	**1**	0.3351	11	0.3952	2
	$Rims^{intra}_{PROB}$	0.3562	4	0.4320	2	**0.2197**	**1**	0.2018	3	0.4240	5
	$Rims^{inter}_{DET}$	**0.3302**	**1**	0.4799	10	**0.2225**	**1**	**0.1410**	**1**	0.3950	2
	$Rims^{inter}_{PROB}$	0.3563	4	0.4294	2	**0.2211**	**1**	0.2002	3	0.4226	2
	$Rims+^{inter}_{DET}$	**0.3298**	**1**	0.4514	9	**0.2218**	**1**	**0.1376**	**1**	0.3939	5
	$Rims+^{inter}_{PROB}$	0.3555	4	0.4304	2	**0.2196**	**1**	0.2005	3	0.4223	2
L^{traces}	DDPS	2.4658	7	**1.3736**	**1**	**1.1055**	**1**	**3.0221**	**1**	9.8572	8
	LSTM	2.5365	11	1.8841	10	1.9512	11	**2.9801**	**1**	9.1857	4
	Dsim	2.4658	7	**1.3736**	**1**	**1.1055**	**1**	**3.0221**	**1**	9.8572	8
	Rims	2.4658	7	**1.3736**	**1**	**1.1055**	**1**	**3.0221**	**1**	9.8572	8
	Rims+	2.4658	7	**1.3736**	**1**	**1.1055**	**1**	**3.0221**	**1**	9.8572	8
	$Rims^{intra}_{DET}$	**0.9366**	**1**	2.6258	11	1.8064	5	11.2853	11	**8.0797**	**1**
	$Rims^{intra}_{PROB}$	2.0698	5	**1.3590**	**1**	1.8022	5	4.8503	6	9.6933	5
	$Rims^{inter}_{DET}$	1.7046	2	1.6681	8	1.8215	8	4.8025	6	**8.2381**	**1**
	$Rims^{inter}_{PROB}$	2.0787	5	**1.3594**	**1**	1.8131	8	4.8689	6	9.6753	5
	$Rims+^{inter}_{DET}$	1.7006	2	1.6806	8	1.8101	8	4.4989	6	**8.0765**	**1**
	$Rims+^{inter}_{PROB}$	2.0506	4	**1.3878**	**1**	1.7908	4	4.7492	6	9.6920	5

Table 13
Rankings for the AED and CTD metrics are defined by calculating p-values on the simulation results across all methods. Table includes the average values obtained in all logs and the corresponding positions in the rankings. The best method(s) are indicated in boldface.

	Method	BPI17W	Rank	BPI12W*	Rank	Cvs Pharmacy	Rank	Confidential 1000	Rank	SynLoan	Rank
AED	DDPS	4806.0569	10	580 .3735	11	3737.1714	10	395.6612	11	**285.4581**	**1**
	LSTM	36 435.5005	11	182.1496	10	7490.0252	11	358.2496	10	915.9726	11
	Dsim	46.7428	6	**22.9616**	**1**	25.9934	2	**244.7479**	**1**	429.8132	3
	Rims	**28.2686**	**1**	37.9309	4	59.2486	5	**243.7857**	**1**	371.7824	2
	Rims+	106.7241	7	39.7040	4	**14.5868**	**1**	**238.4793**	**1**	565.4652	7
	$Rims^{intra}_{DET}$	33.4366	4	28.8331	2	75.9279	6	**236.6897**	**1**	521.1241	4
	$Rims^{intra}_{PROB}$	**29.2353**	**1**	41.8716	4	75.6858	6	**245.1454**	**1**	578.5477	4
	$Rims^{inter}_{DET}$	37.8911	5	27.8342	2	75.9129	6	**243.7821**	**1**	528.0596	4
	$Rims^{inter}_{PROB}$	**29.5144**	**1**	40.8339	4	75.8363	6	**245.4279**	**1**	580.2814	8
	$Rims+^{inter}_{DET}$	136.5987	9	49.4703	9	26.5468	2	**239.6343**	**1**	718.0791	9
	$Rims+^{inter}_{PROB}$	110.5644	7	40.2435	4	26.5468	2	**240.8168**	**1**	752.6821	10
CTD	DDPS	264.4673	11	192.7671	11	294.9016	11	15.4111	11	637.7109	11
	LSTM	212.4001	10	87.9155	3	177.4086	10	9.0599	8	404.1028	7
	Dsim	115.9858	9	**83.6518**	**1**	38.2325	3	10.0581	10	465.2910	9
	Rims	86.1195	6	100.0458	6	28.6076	2	7.4579	7	355.2428	6
	Rims+	**37.8205**	**1**	**84.4962**	**1**	**12.7126**	**1**	**4.3004**	**1**	229.9056	3
	$Rims^{intra}_{DET}$	69.5044	4	108.5267	10	98.5730	6	**5.0931**	**1**	**215.6133**	**1**
	$Rims^{intra}_{PROB}$	86.8038	6	101.9591	3	98.2224	6	9.3278	6	302.1341	4
	$Rims^{inter}_{DET}$	84.9552	5	100.0842	6	98.4561	6	6.7263	5	223.5069	2
	$Rims^{inter}_{PROB}$	86.9323	6	100.7581	6	98.2924	6	9.3248	8	300.9623	4
	$Rims+^{inter}_{DET}$	65.5271	3	85.2291	3	52.2997	4	**4.1326**	**1**	454.4386	8
	$Rims+^{inter}_{PROB}$	**38.7520**	**1**	86.0765	3	50.5428	4	**4.9638**	**1**	515.8558	10

simplicity, refer to them collectively as time perspectives. The points in Fig. 6 represent the sum of the rankings obtained by various simulation methods across different logs and metrics (AED, CTD, CLFD, and L^{traces}) for the corresponding perspectives. The best simulation method(s) are those closest to the origin of axes, indicating that they achieved low cumulative rankings and consistently top positions.

7.1. Answering RQ1

RQ1 wants to evaluate how Rims and its variants perform compared with the other state-of-the-art techniques, i.e., DL and DDPS.

We start by considering the control-flow perspective. Since DDPS and *Dsim* share the same process model of Rims and Rims+, to answer **RQ1** we only need to compare Rims and $Rims^{DT}$ with the DL technique, that is LSTM. By looking Table 10, $Rims^{DT}$ outperforms the other baselines in terms of CLFD, except for the *Production* log where LSTM is the best method, despite being one of the worst for L^{traces}. This suggests that while the generative model captures the sequences of events within the traces well – as indicated by the lower 2-Gram and 3-Gram metric values in Fig. A.9 – it may not capture the overall process effectively. For *PurchasingExample* log we can observe that the introduction of the DT_m predictive models on the simulation model do

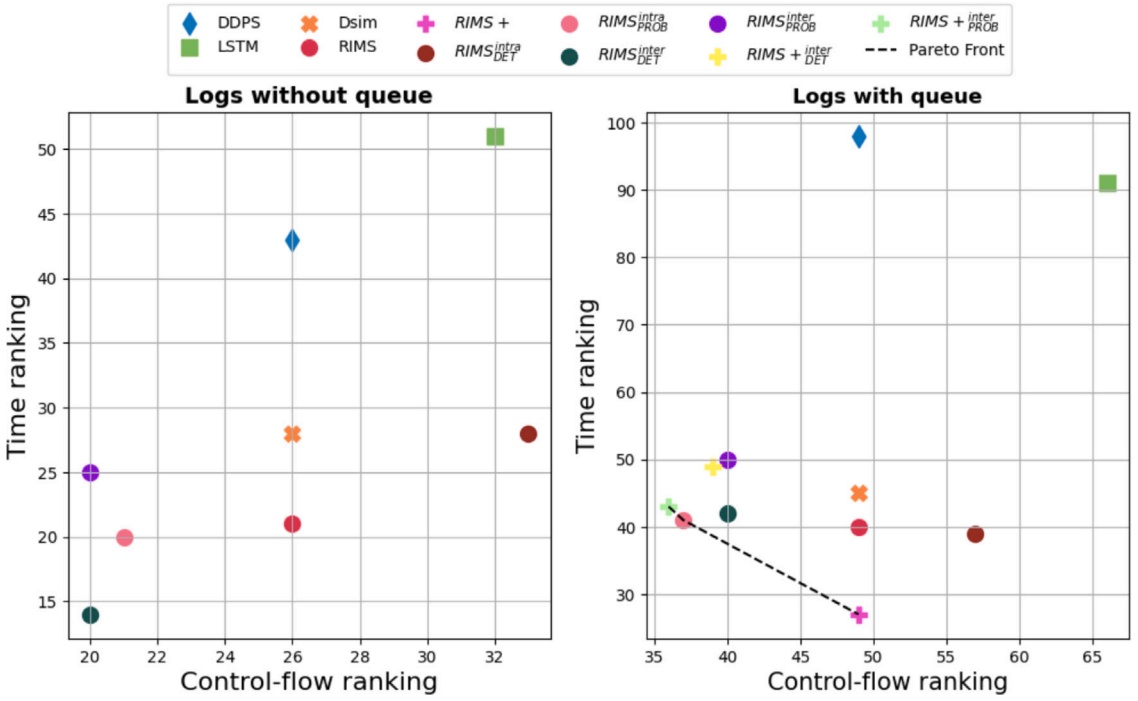


Fig. 6. Pareto front of all baselines and Rims variations considering both time and control-flow perspectives, based on the rankings reported in Tables 10, 11, 12 and 13. Circle symbols indicate the variants of Rims, while plus symbols represent the variants of Rims+.

not affect the results. This can be attributed to the fact that in synthetic logs, the paths of the traces are typically independent of their history or attributes and are generated by applying branching probability. As a result, the predictive models are not able to learn anything additional. Indeed, for *ConsultaDataMining*, which is a real log, only Rims+$^{inter}_{DET}$ differs statistically from the other methods.

Concerning the L^{traces} metric in Table 10, half of logs performed well with LSTM method while the other half with a variants of Rims^{DT}. These results highlight the potential limitations of the process models used by DDPS, *Dsim*, Rims/+ in accurately representing the event frequency in the traces as exhibited in the logs. Limitations that can be partially adjusted by the introduction of DT_m models, as shown with *PurchasingExample* and *Production* logs.

Focusing on the logs with a significant level of queues (Table 12), in terms of CLFD, Rims and its variants outperforms LSTM in the majority of logs with the exception for *BPI12W* and *SynLoan*. In the case of *BPI12W* this is due to the fact that the log is real and with a large number of traces, which ensures an efficient learning of the control flow by the LSTM method. *BPI17W* behaves similarly, as it shares the same characteristics in the log, but in this case the introduction of DT_m is sufficient to overcome the generative method. For *SynLoan*, the explanation is similar to the one given for real logs. This synthetic log, indeed, is built so that at decision points the future path of a trace depend on its history up to that point, and this behavior is not well reproduced by the trivial implementation of decision points in DDPS and Rims.

Nonetheless, for both logs, Rims and its variants closely follow the LSTM approach, demonstrating their effectiveness as a good solution. The only exceptions are Rims$^{intra}_{DET}$ and Rims$^{inter}_{DET}$ in *BPI12W* and *Confidential 1000*, where single class prediction (DET) disadvantage the infrequent behaviors and generate a large number of events, as indicated by the L^{traces}. Also for L^{traces} metrics, we can observe a similar pattern based on the nature of the logs – real or synthetic – where the introduction of DT_m models improves the performance of the simulation model for the real and *SynLoan* logs, but not for the other synthetic logs (Table 12).

Fig. 6 summarizes the rankings obtained by various methods across different logs for the general metrics, CLFD and L^{traces}. Specifically, regarding the logs without queue, Rims$^{inter}_{DET}$ and Rims$^{inter}_{PROB}$ emerge as the two tied winners. From Fig. 6 it is evident that the introduction of DT_m models improves performance, as the Rims^{DT} methods are positioned on the left side of the graph. Overall Rims and its variants outperform LSTM in the control-flow perspective **RQ1**. The only exception is Rims$^{intra}_{DET}$, which as mentioned earlier, is impacted by the main class prediction (DET). When the logs contain a significant level of queuing, the best overall method is Rims+$^{inter}_{PROB}$, as shown in Fig. 6. Additionally, in this scenario, all Rims variants outperform the LSTM baseline

We now analyze the time dimension by looking Tables 11 and 13. Focusing on the logs without a significant level of queues (Table 11), Rims and its variants in terms of AED outperform the baselines with *Confidential 2000* and *ConsultaDataMining*. Rims$^{inter}_{DET}$ is tied with *Dsim* for the *Production* log, as shown by AED, CED, and RED metrics in Fig. A.10, confirming the AED results. Conversely, for *PurchasingExample* log, while DDPS is the best at distributing events over time (AED), it performs worse than Rims$^{inter}_{DET}$ in the specific metrics CED, RED, and CAR (Fig. A.10). Indeed, in terms of CTD, which also incorporates simulation aspects captured by the specific metrics, Rims$^{inter}_{DET}$ surpasses the DDPS method.

By examining the results of Rims$^{inter}_{DET}$ with *Confidential 2000*, we observe a unique behavior: this approach performs the best in terms of AED, but is the worst in terms of CTD, with a significant difference from the other methods. The reason is attributable to the number of events generated by Rims$^{inter}_{DET}$ and the way AED metric is calculated. Indeed, for L^{traces}, Rims$^{inter}_{DET}$ is the worst method, as it generates about 30% more events on average for each trace. AED is computed by use of *1WD* measure which compares empirical distribution functions using histograms – one for $\mathcal{L}_{test}$ and one for $\mathcal{L}_{sim}$ – where the sum of all bins equals 1. This normalization can offset frequency differences between events in $\mathcal{L}_{test}$ and $\mathcal{L}_{sim}$ while rewarding the capability of reproducing

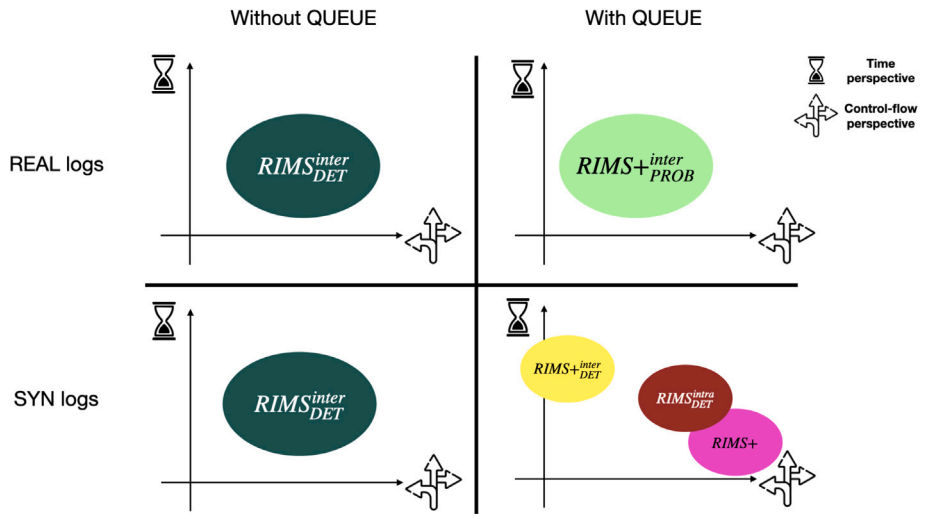


Fig. 7. Comparison of Rims and its variants with respect to the nature of the logs and the presence or absence of significant queue levels. If a box contains only one method, it indicates that a single optimal solution exists; otherwise, all solutions on the Pareto front are included. The lower the value on the axes (x, y), the closer the simulated log is to the real one in terms of control-flow and time perspectives, respectively.

the trend and the seasonality of the process (namely AED, CED and CAR).[11]

Regarding the CTD metrics in Table 11, Rims and its variants outperforms the other approaches for the *ConsultaDataMining* log while for *Confidential 2000* and *PurchasingExample* at least two variants achieve the top ranking. For the *Production* log, *Dsim* is the best method, with Rims$^{inter}_{DET}$ following closely behind.

Moving to the logs with queues (Table 13), Rims and its variants consistently outperform DDPS and LSTM in terms of AED, with the only exception being the *SynLoan* log. In this case, DDPS performs best, followed by Rims, which, in terms of CED and RED (Fig. A.12), more accurately represents the seasonal aspects and the distribution of events over time within each case, respectively. Compared to the baseline *Dsim*, Rims and its variants are competitive across most logs, with the exception of BPI12 W, where *Dsim* outperforms, and *Confidential 1000*, where Rims+ demonstrates the impact of queue features. Additionally, for CTD metrics, Rims+ ranks at the top in four out of five logs (Table 13), underlines the importance of queue features. In general, most Rims variants achieve top positions in the rankings across the logs and perform well or are competitive in other specific metrics, as shown in Fig. A.12.

Fig. 6 provides an overview of the results. In particular, we observe that, apart Rims$^{inter}_{DET}$ and Rims+$^{inter}_{DET}$, the Rims variants dominate the other baselines in terms of time perspective, achieving the best performance across all logs with varying characteristics and considering both metrics, AED and CTD.

Finally, examining Fig. 6 allows us to address **RQ1** by considering both aspects of our analysis: time and control-flow perspectives. We begin by noting that, in both cases – logs with and without significant queue levels – the Pareto fronts, which correspond to the sets of undominated methods in both time and control-flow perspectives, consist solely of Rims variants, highlighting the method's overall advantage compared to the current state-of-the-art. For logs without queues, there is a clear winner represented by Rims$^{inter}_{DET}$, and only one method, Rims$^{intra}_{DET}$, is dominated by *Dsim* baseline. The reason for this is related to the type of prediction used by DT_m predictive models involved in the simulation model, which can sometimes yield worse results, as previously mentioned. For logs with a significant level of queues, *Dsim* outperforms Rims$^{intra}_{DET}$ and Rims+$^{intra}_{DET}$ in terms of the time perspective but not for the control-flow perspective, while Rims$^{intra}_{DET}$ exhibits the same behavior as with logs without queues. Instead, Rims+$^{intra}_{DET}$ is strongly penalized by the results obtained with *SynLoan* log. However, all other Rims variants outperform the baselines in logs with a significant presence of queues.

7.2. Answering RQ2

To address **RQ2**, we focus on Rims and its variants, analyzing their performance in relation to different logs and their properties.

Starting with the control-flow perspective, we compare Rims and Rims^{DT}, as Rims+ shares the same process model as the former. This allows us to assess the impact of introducing DT_m on the simulation quality. Fig. 6 illustrates how the introduction of predictive models at decision points within $\mathcal{N}$ improves the accuracy of the simulation model. This is evidenced by the points representing Rims^{DT} in Fig. 6 shifting to the left, closer to the origin, indicating lower values compared to Rims. The only exception is Rims$^{intra}_{DET}$, which is either the worst or among the worst for both logs without queue and with queue, respectively. The DET prediction type in Rims^{DT} achieves both the best and worst results, while the $PROB$ one is more consistent and reliable. Conversely, neither *intra-case* nor *inter-case* features consistently outperform the other. Overall, Rims$^{inter}_{DET}$ and Rims$^{inter}_{PROB}$ are the best methods for the control-flow perspective, with Rims$^{inter}_{DET}$ also performing best for the time perspective with logs that have no significant level of queues. Instead, for logs with queue, Rims$^{inter}_{PROB}$ is the top performer.

Regarding the time perspective, we compare all the variants of Rims. In the case of queuing in logs, Rims+ proves to be the best method, demonstrating how the introduction of the queuing feature in WT_m improves results. On the other hand, Rims^{DT} produces slightly worse results than Rims. For the log without queues, only two variants of Rims^{DT} – Rims$^{inter}_{DET}$ and Rims$^{intra}_{PROB}$ – exhibit improved results (Fig. 6).

Tables 10, 12, 11 and 13 suggest that the results depend critically on two main factors: the presence of a significant level of queues and the nature of the logs, whenever they are real or synthetic. In order to identify a unique optimal Rims variant based on the characteristics of logs, we computed the rankings as the one used to generate Fig. 6 but

[11] Let us consider having the histogram $H^{test} = \langle (t_1, 2), (t_2, 6), (t_3, 4), (t_4, 8) \rangle$ representing $\mathcal{L}_{\text{test}}$, consisting of bins for each timestamp t_i and the corresponding number of events contained in it. While $\langle (t_1, 300), (t_2, 900), (t_3, 600), (t_4, 1200) \rangle$ represents the H^{sim} histogram of $\mathcal{L}_{sim}$. To compute *1WD* measure, the two histograms are converted in a probability distribution where the sum of all bin values is equal to 1. In this case we obtain the same probability distribution, $\langle (t_1, 0.1), (t_2, 0.3), (t_3, 0.2), (t_4, 0.4) \rangle$, and hence *1WD* between the two normalized histograms results in 0, despite the huge difference in the number of events in $\mathcal{L}_{test}$ and $\mathcal{L}_{sim}$.

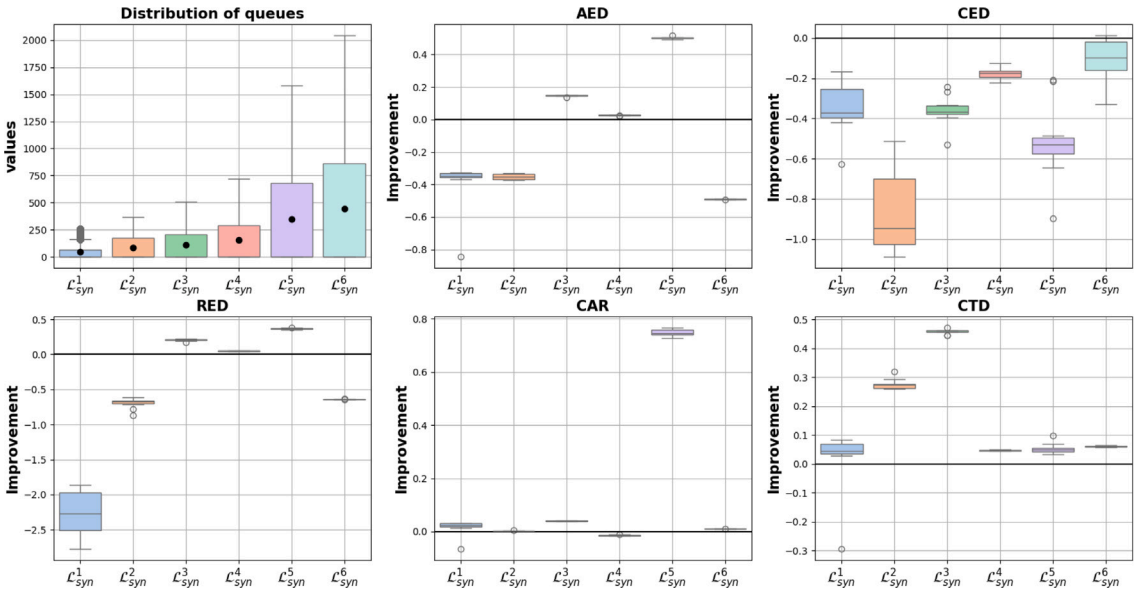


Fig. 8. Comparison of Rims+ and Rims with different queuing levels related to the same process through time and congestion metrics. The improvement is expressed as a percentage. A positive value indicates that the distance of the metrics has been reduced by Rims+, while a negative value indicates the opposite.

only with Rims variants. A qualitative representation of the outcome of this analysis is displayed in Fig. 7, it indicates that in absence of a significant presence of queue $\text{Rims}^{inter}_{DET}$ is the optimal solution considering time and control-flow perspective. In the presence of queues in logs, we identify $\text{Rims+}^{inter}_{PROB}$ as the optimal solution for real logs. In contrast, for synthetic logs, there are three optimal solutions. Therefore, the choice depends on whether one wants to prioritize the time or the control-flow perspective. In general, we can note that for all the four categories in Fig. 7 the optimal solutions are represented by Rims^{DT} variants with the only exception of Rims+. This result highlights the characteristics of synthetic logs, where the control flow typically does not rely on historical data or other attributes. As a result, the introduction of DT_m provides only a partial improvement, unlike the more significant enhancement achieved by incorporating queue features for time predictions.

To better understand the benefits of Rims+, especially in case of logs with queues, we analyze the six semi-synthetic log $\mathcal{L}_{syn}$ with increasing queueing levels, as illustrated in the graph *Distribution of queue* of Fig. 8, that shows the length of the queue that each event in the log has to wait for the assigned resource. The rest of Fig. 8 shows the percentage improvement between Rims+ and Rims (if positive Rims+ outperforms Rims) on logs with increasing queueing levels, for all the time metrics. We can observe that the introduction of the *queue* feature has a high positive impact in terms of CTD – particularly for $\mathcal{L}^3_{syn}$ --while, although positive, the improvement decreases for higher values of *queueing* levels. In terms of AED and RED, we can observe a slight decrease in the performance of Rims+ for $\mathcal{L}^1_{syn}$, $\mathcal{L}^2_{syn}$, and $\mathcal{L}^6_{syn}$, while the results are consistently worse in terms of distribution of events over the week (CED metrics). The decrease in the Rims+ performance in terms of time metrics (especially CED) could be likely due to the LSTM models used for predicting the waiting times, which learn more from the queue features than from the time-related ones, such as the day of the week and the hour of the day.

Summing up, we can state that, although Rims+ is less accurate than Rims in terms of seasonality and weekday distribution, it consistently outperforms Rims in terms of overall cycle time accuracy (CTD) and in many of the cases it is able to improve or have similar performance as Rims in terms of event distribution (AED).

8. Conclusion

In this paper, we presented and discussed Rims, a *runtime-integration* technique capable of successfully integrating DDPS and DL approaches. This runtime-integration enables us to fully exploit the specific DL predictions thus enhancing the performance of the overall system both with respect to the single DDPS and DL techniques separately and the *post-integration* approach. Moreover, Rims+ enables us also to incorporate the queue as an intercase feature in the DL model.

In particular, we integrated predictive models to improve the time and decision mining perspectives of the process, with the aim of avoiding the unrealistic or oversimplified assumptions of DDPS models while preserving the advantages of a white-box simulation model. Specifically, Rims has been enhanced with models predicting the trace arrival rate, the processing times of activities, as well as the waiting times among them. Besides, Rims^{DT} have further enhanced Rims with models predicting the next activity from the decision points of the process model.

We evaluated Rims and its variants using both real-world and synthetic event logs and leveraging several metrics to assess the quality of the simulated log in terms of control-flow, time and congestion.

The results show how Rims and its variants outperform the state-of-the-art in terms of time, congestion, and control-flow perspectives, representing the optimal solutions for logs both with and without a significant level of queuing. In particular, we prove the advantages of *runtime-integration* over *post-integration* and how features dependent on the simulation itself, such as queue features can be leveraged in the integrated predictive models. Finally, the evaluation reveals how the introduction of DT_m predictive models at decision points improves the simulation quality from all perspectives, as shown in Figs. 6 and 7, where the optimal solutions are predominantly Rims^{DT}. Overall, the evaluation results indicate that Rims and its variants consistently produce more accurate simulations across all analyzed event logs, both real and synthetic.

In the future, we aim at extending predictive models to also include the resource perspective that would allow to discover the allocation of real processes and possibly optimizing it. For the evaluation, we used only logs that contain both start and end timestamps, as these are crucial for accurately estimating processing and waiting times. We plan to explore methods for estimating start timestamps in logs where they are missing, thereby expanding the applicability of the proposed approach. Additionally, we intend to investigate other types of predictive decision mining models, such as black-box predictive models. Indeed, although black-box, these models could enhance the quality of the simulated trace control flow and consequently improve

the entire simulation. Finally, we plan to explore the possibility to replace the discovered process model $\mathcal{N}$ with a sequence of activities predicted by a predictive process model, in particular in special cases as the one of the *Production* event log, where the DL approach still performs better than the RIMS^{DT} approach in terms of control flow.

CRediT authorship contribution statement

Francesca Meneghello: Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization. **Chiara Di Francescomarino:** Writing – review & editing, Validation, Supervision, Conceptualization. **Chiara Ghidini:** Supervision, Conceptualization. **Massimiliano Ronzani:** Writing – review & editing, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Boxplot visualizations

In this appendix, we present the detailed boxplot analysis for all the logs listed in Table 6 and for all the metrics described in Section 6.4. Figs. A.9 and A.11 display the control-flow metrics for logs without and with queue, respectively. Figs. A.10 and A.12 display the time metrics for logs without and with queue, respectively. Each boxplot shows the results across all the logs generated by the corresponding method.

Data availability

Data will be made available on request.

References

[1] M. Dumas, M. La Rosa, J. Mendling, H. A Reijers, Fundamentals of Business Process Management, Springer, 2013.

[2] M. Camargo, M. Dumas, O. González, Automated discovery of business process simulation models from event logs, Decis. Support Syst. 134 (2020) 113284.

[3] A. Rozinat, R.S. Mans, M. Song, W.M.P. van der Aalst, Discovering simulation models, Inf. Syst. 34 (2009) 305–327.

[4] I. Khodyrev, S. Popova, Discrete modeling and simulation of business processes using event logs, in: Proc. of the Int. Conference on Computational Science, ICCS 2014, in: Procedia Computer Science, vol. 29, Elsevier, 2014, pp. 322–331.

[5] M. Pourbafrani, S.J. van Zelst, W.M.P. van der Aalst, Supporting automatic system dynamics model generation for simulation in the context of process mining, in: Business Information Systems - BIS 2020, Proc., in: LNBIP, vol. 389, Springer, 2020, pp. 249–263.

[6] N. Martin, B. Depaire, A. Caris, The use of process mining in business process simulation model construction - structuring the field, Bus. Inf. Syst. Eng. 58 (1) (2016) 73–87.

[7] M.T. Wynn, M. Dumas, C.J. Fidge, A.H.M. ter Hofstede, W.M.P. van der Aalst, Business process simulation for operational decision support, in: Business Process Management Workshops, BPM 2007, in: LNCS, vol. 4928, Springer, 2007, pp. 66–77.

[8] W.M. Van Der Aalst, Business process simulation survival guide, in: Handbook on Business Process Management 1: Introduction, Methods, and Information Systems, Springer, 2014, pp. 337–370.

[9] B. Estrada-Torres, M. Camargo, M. Dumas, L. García-Bañuelos, I. Mahdy, M. Yerokhin, Discovering business process simulation models in the presence of multitasking and availability constraints, Data Knowl. Eng. 134 (2021) 101897.

[10] M. Camargo, M. Dumas, O.G. Rojas, Learning accurate business process simulation models from event logs via automated process discovery and deep learning, in: Advanced Information Systems Engineering - 34th International Conference, CAiSE 2022, Proceedings, in: LNCS, vol. 13295, Springer, 2022, pp. 55–71.

[11] C. Di Francescomarino, C. Ghidini, Predictive process monitoring, in: Process Mining Handbook, in: LNBIP, vol. 448, Springer, 2022, pp. 320–346.

[12] N. Tax, I. Verenich, M.L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: Advanced Information Systems Engineering, CAiSE 2017, Proc., in: LNCS, vol. 10253, Springer, 2017, pp. 477–492.

[13] J. Evermann, J. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decis. Support Syst. 100 (2017) 129–140.

[14] W.J. Chang, Y.H. Chang, Design of a patient-centered appointment scheduling with artificial neural network and discrete event simulation, J. Serv. Sci. Manag. 11 (01) (2018).

[15] S. Bergmann, S. Stelzer, S. Straßburger, On the use of artificial neural networks in simulation-based manufacturing control, J. Simul. 8 (1) (2014) 76–90.

[16] J. Pender, E. Zhang, Uniting simulation and machine learning for response time prediction in processor sharing queues, in: Winter Simulation Conference, WSC, IEEE, 2021, pp. 1–12.

[17] R.D.L. Fuente, I. Erazo, R.L. Smith, Enabling intelligent processes in simulation utilizing the TensorFlow deep learning resources, in: Winter Simulation Conference, WSC, IEEE, 2018, pp. 1108–1119.

[18] M. de Leoni, F. Vinci, S.J.J. Leemans, F. Mannhardt, Investigating the influence of data-aware process states on activity probabilities in simulation models: Does accuracy improve? in: Business Process Management - 21st International Conference, BPM 2023, Utrecht, the Netherlands, September 11-15, 2023, Proceedings, in: Lecture Notes in Computer Science, vol. 14159, Springer, 2023, pp. 129–145, http://dx.doi.org/10.1007/978-3-031-41620-0_8.

[19] F. Meneghello, C. Di Francescomarino, C. Ghidini, Runtime integration of machine learning and simulation for business processes, in: 5th International Conference on Process Mining, ICPM 2023, Rome, Italy, October 23-27, 2023, IEEE, 2023, pp. 9–16, http://dx.doi.org/10.1109/ICPM60904.2023.10271993.

[20] A. Rozinat, W.M. van der Aalst, Decision mining in ProM, in: Business Process Management: 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006. Proceedings 4, Springer, 2006, pp. 420–425.

[21] M. De Leoni, W.M. Van Der Aalst, Data-aware process mining: Discovering decisions in processes using alignments, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 1454–1461.

[22] W.M.P. van der Aalst, Foundations of process discovery, in: W.M.P. van der Aalst, J. Carmona (Eds.), Process Mining Handbook, in: LNBIP, vol. 448, Springer, 2022, pp. 37–75.

[23] C. Di Francescomarino, M. Dumas, F.M. Maggi, I. Teinemaa, Clustering-based predictive process monitoring, IEEE Trans. Serv. Comput. 12 (6) (2019) 896–909.

[24] B.F. van Dongen, R.A. Crooy, W.M.P. van der Aalst, Cycle time prediction: When will this case finally be finished? in: On the Move To Meaningful Internet Systems: OTM 2008, Proceedings, Part I, in: LNCS, vol. 5331, Springer, 2008, pp. 319–336.

[25] M. Camargo, M. Dumas, O.G. Rojas, Learning accurate LSTM models of business processes, in: Business Process Management - 17th International Conference, BPM, Proceedings, in: LNCS, vol. 11675, Springer, 2019, pp. 286–302.

[26] S.J. Taylor, B. Letham, Forecasting at scale, Amer. Statist. 72 (2018) 37–45.

[27] M. Camargo, M. Dumas, O.G. Rojas, Discovering generative models from event logs: Data-driven simulation vs deep learning, PeerJ Comput. Sci. 7 (2021) e577.

[28] A. Senderovich, C. Di Francescomarino, F.M. Maggi, From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring, Inf. Syst. 84 (2019) 255–264.

[29] W. Van der Aalst, A. Adriansyah, B. Van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdisc. Rev.: Data Min. Knowl. Discov. 2 (2) (2012) 182–192.

[30] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, Conformance Checking, vol. 56, Springer,[Google Scholar], Switzerland, 2018, p. 12.

[31] B. van Dongen, BPI Challenge 2012, 4TU.ResearchData, 2012.

[32] D. Chapela-Campa, I. Benchekroun, O. Baron, M. Dumas, D. Krass, A. Senderovich, Can I trust my simulation model? Measuring the quality of business process simulation models, in: Business Process Management - 21st International Conference, BPM 2023, Utrecht, the Netherlands, September 11-15, 2023, Proceedings, in: Lecture Notes in Computer Science, vol. 14159, Springer, 2023, pp. 20–37, http://dx.doi.org/10.1007/978-3-031-41620-0_2.

[33] H.W. Kuhn, The Hungarian method for the assignment problem, Naval Res. Logistics Q. 2 (1–2) (1955) 83–97.

[34] S.J.J. Leemans, W.M.P. van der Aalst, T. Brockhoff, A. Polyvyanyy, Stochastic process mining: Earth movers' stochastic conformance, Inf. Syst. 102 (2021) 101724.

[35] E. Levina, P.J. Bickel, The earth mover's distance is the mallows distance: Some insights from statistics, in: Proceedings of the Eighth International Conference on Computer Vision (ICCV-01), Vancouver, British Columbia, Canada, July 7-14, 2001 - Volume 2, IEEE Computer Society, 2001, pp. 251–256.

Fig. A.9. Control-flow analysis for logs without a significant level of queue.

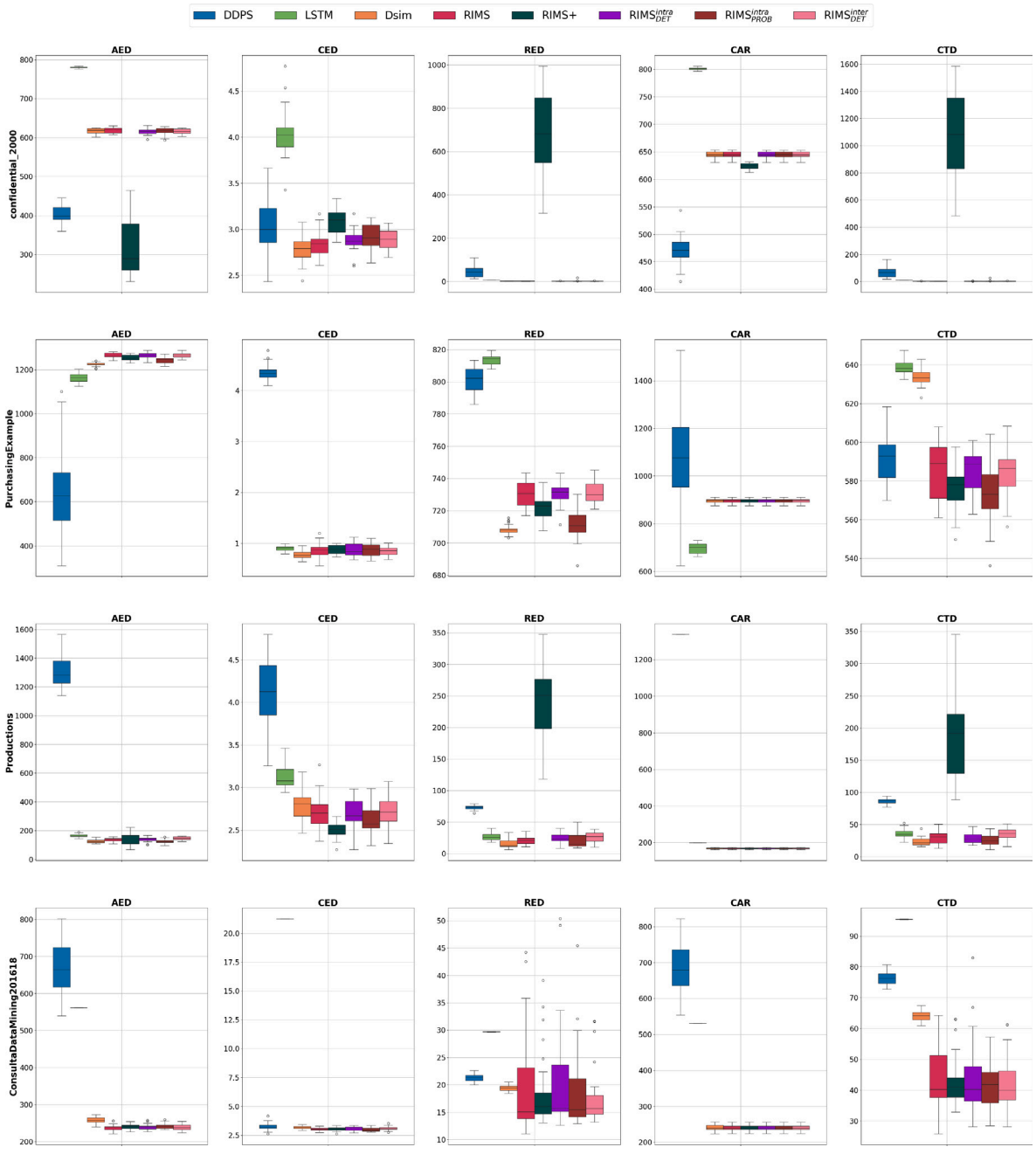


Fig. A.10. Time analysis for logs without a significant level of queue.

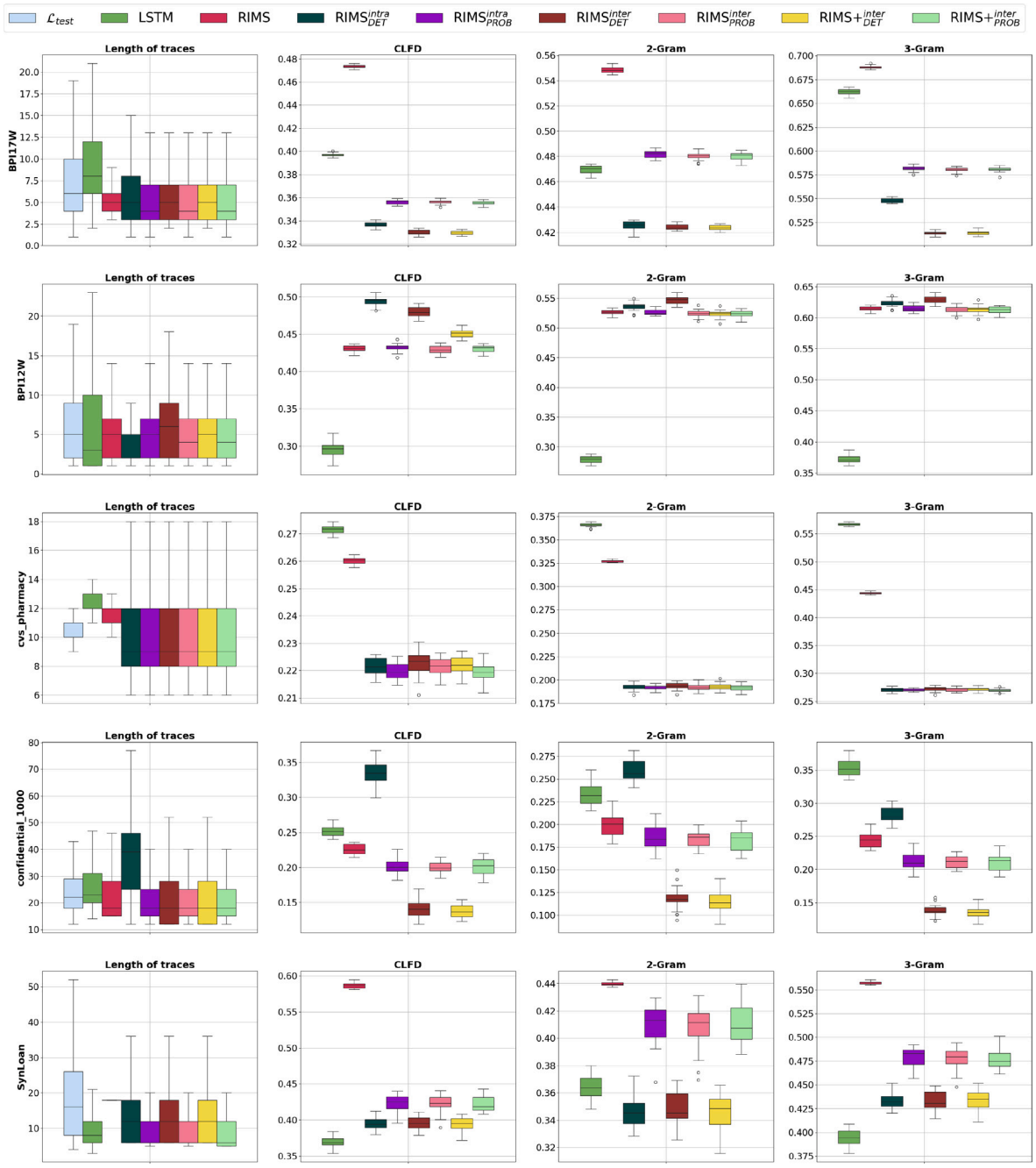


Fig. A.11. Control-flow analysis for logs with a significant level of queue.

Fig. A.12. Time analysis for logs with a significant level of queue.