2023 5th International Conference on Process Mining (ICPM) | 979-8-3503-5839-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPM60904.2023.10271993

Runtime Integration of Machine Learning and Simulation for Business Processes

Francesca Meneghello
Fondazione Bruno Kessler, Trento, Italy
Sapienza University of Rome, Rome, Italy
fmeneghello@fbk.eu

Chiara Di Francescomarino
University of Trento, Trento, Italy
c.difrancescomarino@unitn.it

Chiara Ghidini
Fondazione Bruno Kessler, Trento, Italy
ghidini@fbk.eu

***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 but to complement it with information provided by DL models. 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 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. The runtime-integration enables us to also incorporate the queue as an intercase feature in the DL model, thus further improving the performance in process scenarios where the queue plays an important role.**

***Index Terms*—Business Process Simulation, Deep Learning, Hybrid Simulation**

I. INTRODUCTION

Business process simulation (BPS) [1] provides a widely used and flexible approach to analyse and improve business processes. It exploits a process simulation model, that is, a process model extended with additional information for a probabilistic characterization of the different run-time aspects such as case arrival rate, task durations, routing probabilities, resource utilization, and so on, to produce 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 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 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). Event logs can therefore provide knowledge to process analysts and drive them into the manual construction of the simulation model, or 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, 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 behaviours such as multitasking, batching, fatigue effects, and inter-process resource sharing, among others [8].[1]

The recent development of Deep Learning (DL) techniques applied to Predictive Process Monitoring [9] offers an alternative way to "create" 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 [10], [11]. Unlike simulation models, DL models are problematic to be used for analyzing and improving business processes. Indeed DL models cannot expose an explicit process model that can be exploited to understand the explicit process behavior or easily modified so as to simulate their effects. However, these models are extremely powerful in learning the patterns characterizing the precise relationship between the different trace elements, and thus, the true relationship between the covariates and the distribution of the output variable in a simulation model.

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 [12]–[15]. 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 between the covariates and the distribution of the output variable, thus avoiding the oversimplifications of the simulation model mentioned above.

We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

[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.

A first attempt to combine Deep Learning and Discrete event simulation is presented in [16], 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.

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 "a posteriori" with the predictions of a DL model as in [16], RIMS provides a tight integration of the predictions of the DL model **at runtime** during the simulation (Section V). This *runtime-integration* enables us to fully exploit the specific predictions on elements of the time perspective (waiting times and processing times) at simulation time, thus enhancing the performance of the overall system both w.r.t. the single techniques (BPS, and DL) separately and the *post-integration* approach of *Dsim* (see the Running Example in Section IV and the Evaluation in Section VI). The *runtime-integration* enables us also to incorporate the queue as an intercase feature in the DL model, and to obtain an augmented version of RIMS which further improves its performances in process scenarios where the queue plays an important role. The structure of the paper is complemented by a Background section (Section II), Related Work (Section III) and Conclusions (Section VIII).

II. BACKGROUND

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

A. 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 $\sigma = \langle e_1, ...e_m \rangle$. An event $e_i = (\alpha, t, p1, ..., pn)$ is a complex structure that includes the activity label $\alpha = e_i.act$, its timestamp $e_i.time = t$, indicating the time in which the event has occurred, and possibly data payloads $p1, ..., pn$ consisting of attributes, such as, the resource(s) involved in the execution of an activity, or other data recorded during the event. Non-instantaneous events can be characterised 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.

B. 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 BPMN model), 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.

C. Predictive Process Monitoring

Predictive Process Monitoring (PPM) [9] 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 [18], predicting the time to complete a task or the time until the completion of a case [19], predicting the next activity or the continuation of a running incomplete case [10] or entire traces [20]. In this paper, we focus on the prediction of timestamps through an LSTM (Long Short-Term Memory) model, a DL model widely applied in this field given its capability to deal with sequences of elements, as in the case of traces.

III. 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.

A. 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 [11] or to generate the entire trace from scratch [20]. Camargo et al. [21] 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.

B. Hybrid Simulation Model

Recent research has investigated the use of machine learning or neural networks to predict outcomes or decisions within a DES model in different fields [12]–[15]. Bergmann et

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*

TABLE I: Example of a simple event log.

Caseid	Traces	Predicted processing time	Predicted waiting time
1	$\langle(\mathsf{A}, \textit{Role0}), (\mathsf{B}, \textit{Role1})\rangle$	$\langle 8, 4\rangle$	$\langle 1\rangle$
2	$\langle(\mathsf{A}, \textit{Role0}), (\mathsf{C}, \textit{Role1})\rangle$	$\langle 5, 7\rangle$	$\langle 1\rangle$
3	$\langle(\mathsf{A}, \textit{Role0}), (\mathsf{B}, \textit{Role1})\rangle$	$\langle 5, 5\rangle$	$\langle 0\rangle$

TABLE II: Traces generated by the DDPS simulation model, and timestamps predicted by using DL models.

al. [13] propose the use of neural networks and conventional simulation methods to improve decision-making when system knowledge is insufficient. De la Fuente et al. [15] used 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. [14] propose combining simulation and machine learning for response time prediction for processor sharing queues. Specifically, the proposed approach uses the data generated by the discrete-event simulation to train several machine learning methods that are then compared. Although the integration of methods has been studied in several research areas, in BPS only Camargo et al. [16] proposed a combination of DDPS and DL models to discover a BPS model from a log. Specifically, the framework called *Dsim* applied 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. Differently from *Dsim*, in this paper, RIMS provides a tight integration of the predictions of the DL model at runtime during the simulation.

IV. RUNNING EXAMPLE

Consider the simple event log in Table I, which contains for each event the activity name, the start and end timestamps, as well as the role that has performed the activity.

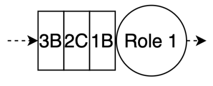
Assume that we are interested in simulating the behaviour of the log and that, to this aim, we leverage the DDPS simulation model reported in Fig. 3. This model has three activities (A,B,C), an exclusive gateway between the activities B and C, and two roles: *Role0* with 3 resources available and *Role1* with 2 resources available. The simulation model would generate traces as the ones reported in the column *Traces* of Table II, which are not accurate.

In order to get accurate simulations, state-of-the-art hybrid approaches, such as *Dsim*, combine simulated traces and predictions on the remaining and waiting 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

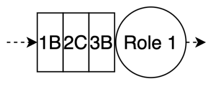
Caseid	Activity	Arrival	Start	End	Role
1	A	24-05-23 08:00	24-05-23 08:00	24-05-23 08:08	*Role0*
1	B	24-05-23 08:09	24-05-23 08:09	24-05-23 08:13	*Role1*
2	A	24-05-23 08:01	24-05-23 08:01	24-05-23 08:06	*Role0*
2	C	24-05-23 08:07	24-05-23 08:07	24-05-23 08:14	*Role1*
3	A	24-05-23 08:01	24-05-23 08:01	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 III: Simulated log generated by *Dsim*.

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, reported in Table II 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 III[2]. Thus, for example, case 1 starts with activity A and it can immediately start because *Role0* has 3 available resources. It ends 8 minutes later as this is the predicted processing time for that event (see Table II). Activity B arrives 1 minute 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 minutes later, as this is the predicted processing time for that event. Similarly for the other traces. One important aspect to note is that this type of 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. 1a: first B of case 1, then C of case 2 and, finally, B of case 3. However, this is not the queue order followed in the real log, due to the addition of the predicted times trace by trace and not per event. Table I shows that resources with *Role1* execute first activity B of case 3, then activity C of case 2 and, finally, activity B of case 1.



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



(b) Queue generated with the RIMS approach.

Fig. 1: 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

[2] Assuming given arrival times.

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*

TABLE IV: 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.

B and C of these two cases - which is indeed what happens in the reality (see Table I). Table IV shows the log generated by the RIMS approach and Fig. 1b shows the queue of the log on role *Role1*, that is inline with the arrival times of events in the reality (see Table I).

V. THE RIMS SYSTEM

In this section we describe the *runtime-integration* approach RIMS (Section V-A) and a variant of RIMS, RIMS+, which takes into account also the effect of queues in the simulated log (Section V-B).

A. 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 BPMN model;
- $\mathcal{P}_{\neg T}$ is the subset of the simulation parameters that excludes the parameters related to the time perspective, i.e., $\mathcal{P}_{\neg T} = \mathcal{P}/\mathcal{P}_T$
- 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.

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. 3, 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.

2) Training of the models: Differently from DDPS models, $\mathcal{M}^*$ uses, as in [16], the predictions returned by the predictive models for defining the trace start times, the event processing times, as well as the event waiting times. Rather than fitting an interarrival distribution like in DDPS model, S_m, a time series forecasting model developed by Facebook and called Prophet [22], is used for generating the trace start times of the simulator. Prophet, indeed, is able to handle time series data that exhibit non-linear trends, seasonality, and other complex patterns. To train Prophet we used the same configuration used in [16].

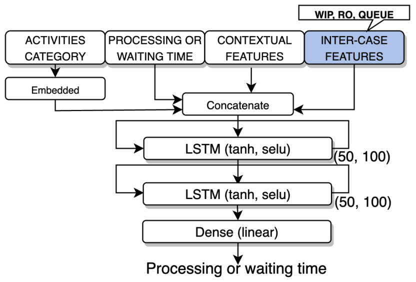


Fig. 2: LSTM architecture from [16] used to train W_m and PT_m. Different intercase features are used in RIMS and RIMS+ than those used in [16].

For the prediction of the waiting and the processing times, RIMS uses two LSTM models with the same architecture used in previous works [16], [20], [21]. The LSTM architecture is shown in Fig. 2: 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 feature ro_τ provides instead the occupancy percentage in terms of resources in use for each role specified in $\mathcal{P}_{\neg T}$ at 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 IV), the intercase-features used to predict the waiting time between the activities A and B of case 1 would be such that $ro_{08:08} = [0, 0.5, 0]$, where 0.5 refers to the occupancy percentage of *Role1* at time 08:08 (as one of the two resources of *Role1* is busy with activity B of case 3). In case of the *post-integration* (Table III), instead, $ro_{08:08} = [0, 0, 0]$, as although one resource of *Role1* 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.

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 [16], i.e., using embeddings for the activity label and extracting n-grams of fixed size for each trace as input sequences.

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. 3, 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 excuting 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, and the intercase features) 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, 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 - by waiting the needed 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. 3 illustrates an example of *runtime-integration*, showing the S_m, PT_m, and W_m models through green, purple and blue points, respectively.

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 $\alpha$ in $next\_activity(\mathcal{N}, \mathcal{P}_C, \sigma)$ do
4:          $e \leftarrow \{\}$
5:          $e.act \leftarrow \alpha, \quad e.role \leftarrow get\_role(e.act, \mathcal{P}_R)$
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:             $wait\_time(e.t_w), \quad e.ts \leftarrow \tau$
11:         end if
12:         $e.t_p \leftarrow predict(PT_m, [e.act, e.ts, wip_\tau, ro_\tau])$
13:         $e.res \leftarrow wait\_resource(e.role), \quad e.ts \leftarrow \tau$
14:         $\sigma \leftarrow append(\sigma, execute(e, e.t_p))$
15:     end for
16:     return $\sigma$
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
```

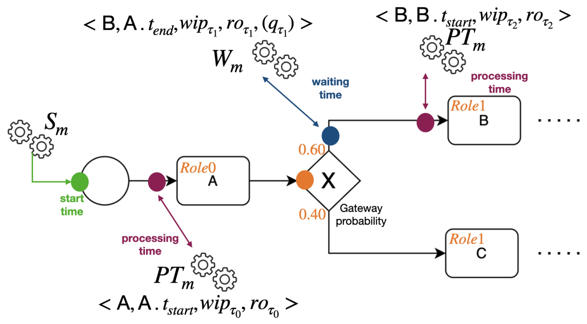

$< B, A.t_{end}, wip_{\tau_1}, ro_{\tau_1}, (q_{\tau_1}) >$
$< B, B.t_{start}, wip_{\tau_2}, ro_{\tau_2} >$
W_m
PT_m
waiting time
processing time
S_m
$Role1$
B
0.60
$Role0$
A
X
start time
0.40
Gateway probability
$Role1$
C
processing time
PT_m
$< A, A.t_{start}, wip_{\tau_0}, ro_{\tau_0} >$

Fig. 3: RIMS and RIMS+ system. The green, blue, and purple points represent examples of interaction with the DL models, and the orange one with DDPS elements.

B. RIMS+*: Enhance* RIMS *with queue discovery*

In many processes, traces do not operate independently but rather compete for scarse resources [23]. 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. 2). Typically, queue mining approaches rely on event logs containing information on queueing dynamics, however, for most event logs, queueing information is 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 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

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 $\sigma$ do
 4:         $e_i.queue \leftarrow retrieve\_queue(e_i.res, \tau)$
 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 $\sigma^*$
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
```

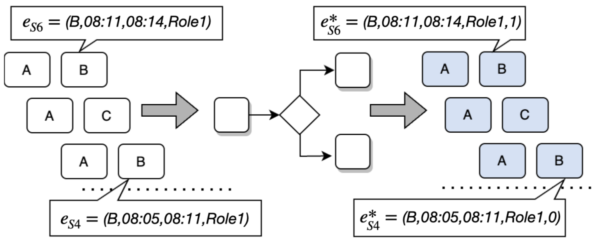

e_{S6} = (B,08:11,08:14,Role1)
e^*_{S6} = (B,08:11,08:14,Role1,1)
A
B
A
C
A
B
e_{S4} = (B,08:05,08:11,Role1)
e^*_{S4} = (B,08:05,08:11,Role1,0)

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

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^q_m is trained for the waiting time prediction (including also the queue length in the encoding). RIMS+ leverages W^q_m to obtain the simulation model $\mathcal{M}^*_q = (\mathcal{N}, \mathcal{P}, S_m, W^q_m, PT_m)$ 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 V.

VI. Evaluation

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

A. Research questions

We aim at investigating the following research questions:

RQ1 How does RIMS perform in terms of simulation quality compared to individual techniques, e.g., DDPS and DL?
RQ2 How does RIMS perform in terms of simulation quality compared to a *post-integration* approach, e.g., *Dsim*?
RQ3 How does RIMS+ improve the quality of simulation compared to approaches that do not use the queue feature?

RQ1 aims to evaluate the advantages of combining the different techniques, DDPS and DL, so as to leverage their strengths and limit their weaknesses. The aim of **RQ2** is to compare RIMS with state-of-the-art approaches integrating prediction and simulation, i.e., the *post-integration Dsim*.

Log	Type	#Traces	#Events	#Activity	Queue mean	Avg. trace length
BPI17W	real	30276	240854	8	1646	7.96
BPI12W	real	8616	59302	6	1115	6.88
Cvs Pharmacy	syn	10000	103906	15	59	10.39
Confidential 2000	syn	1670	44373	29	2.37	26.57
ConsultaDataMining	real	954	4962	16	0	5.2
Production	real	225	4503	24	1.17	20
Confidential 1000	syn	800	21221	29	9.91	26.53
PurchasingExample	syn	608	9119	21	0.30	15
SynLoan	syn	2000	43164	25	68.89	39

TABLE V: Event log and queue statistics. The italic face labels indicate logs used for RIMS+.

Finally, **RQ3** wants to evaluate the effect of the *queue* feature on the waiting time prediction and, hence, on the generated simulation log.

B. Datasets

Nine event logs – 4 real-life an 5 synthetic logs obtained by simulating real-life processes – containing both start and end timestamps, have been used in the evaluation. Table V reports the number of traces, events, activities, and the mean value of *queue* calculated by Algorithm 2. Moreover, to analyze the impact of RIMS+ with different levels of queuing, we created 6 logs of 4 000 traces, $\mathcal{L}^i_{syn}$, with the same $\mathcal{N}^{\mathcal{L}_{syn}}$ and $\mathcal{P}^{\mathcal{L}_{syn}}_C$ but with a queue value of $43, 85, 103, 152, 345, 441$, respectively[3].

C. Experiment setup

We aim at evaluating the performance of DDPS, generative DL models, *Dsim*, RIMS and RIMS+. 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 [20], which is able, through an LSTM model, to generate complete event logs from zero-size prefixes. The tool *Dsim* [16] is used as *post-integration* approach. In order to carry out a fair comparison, the BPMN model, $\mathcal{N}$, 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. Finally, RIMS+ is applied only to event logs presenting significant queues.

RIMS is developed with SimPy[4] library, a process-based discrete-event simulation framework in Python. To train the LSTM models, we split each event log using the hold-out method and a time split criterion, into $60\% - 20\% - 20\%$ partioning for training, validation and testing, respectively.

D. Evaulation Metrics

To evaluate and compare the simulation model produced by each approach we used metrics commonly used in BPS,

[3]All the logs $\mathcal{L}^i_{syn}$ and the model $\mathcal{M}^{\mathcal{L}_{syn}} = (\mathcal{N}^{\mathcal{L}_{syn}}\ , \mathcal{P}^{\mathcal{L}_{syn}})$ used to generate these logs are available, together with the code and all the material used in the evaluation, in the code repository, https://github.com/francescameneghello/RIMS.git

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

i.e., the Mean Absolute Error (MAE)[5] of cycle times, and the Earth Mover's Distance (EMD)[6] of the normalized histograms of activity timestamps grouped by day and hour. The two metrics are calculated by comparing the log generated by each model with a ground-truth log, i.e., a testing subset of the event log. All the analyses are performed with a level of confidence of 95%, i.e., there is only a 5% of probability that the results obtained could have an error outside the confidence interval (CI). For each method in Table VI and Table VII, 25 simulations are performed while 10 simulations are computed for each $\mathcal{L}^i_{syn}$ in Fig. 5.

VII. Evaluation results

In this section we show the results of the evaluation and answer the research questions of Section VI-A.

Table VI reports the results of the evaluation of DDPS, LSTM, *Dsim* and RIMS in terms of MAE and EMD. The table shows that in terms of MAE, RIMS outperforms LSTM and DDPS for most logs (**RQ1**), with the exception of *BPI12* where the LSTM approach produces the lowest MAE. This result can be due to the model $\mathcal{N}^{BPI12}$, and to the parameters, $\mathcal{P}^{BPI12}_{\neg T}$, which are used by the other three approaches, i.e., the ones using simulation. Indeed, these models and parameters, differently from the LSTM approach, tend to produce traces with few events. The simulation logs produced by LSTM contain in total about 7 700 events, i.e., a number close to the 7 580 events of the test log, while the other methods produce logs with about 2 400 events. By modifying $\mathcal{N}^{BPI12}$ and $\mathcal{P}^{BPI12}_{\neg T}$, so as to obtain longer traces (*BPI12W**), the performance of RIMS becomes closer to the one of the LSTM. By looking at the EMD metrics for **RQ1**, we do not observe considerable differences since the mean and the CIs of DDPS, LSTM and RIMS are very close, except for the log *Confidential_1000*. For this last log the distribution of events, indeed, exhibits distinct patterns, due to the resources' calendar, which are more accurately captured by DDPS.

Table VI also shows that RIMS outperforms *Dsim* for real and synthetic logs (**RQ2**). Only for *Production* and *BPI12W**, *Dsim* produces a lower MAE but a slightly higher EMD. For the first log probably the reasons of this performance are related to the size of the test log, which contains only 45 traces, thus hiding the advantages of *runtime-integration*. For *BPI12W**, instead, as observed in Section IV, the *post-integration* may not comply with the order of the queue by producing longer traces and logs. In fact, the average trace duration in *BPI12W** for RIMS is 3.9 days while for *Dsim* is 4.8 days, which is closer to the average trace duration of the test log (8.2 days), although the resulting log lasts 9 days more than the real one.

[5]The absolute error of a pair of traces, $T1$ and $T2$, refers to the absolute value of the difference between their cycle times. The MAE of the cycle time is calculated as the average of the absolute errors across a set of paired traces.

[6]The EMD between two histograms, $H1$ and $H2$, is the minimum number of units that need to be added, removed, or transferred across columns in H1 to transform it into H2 [16]. The two histograms plot the distribution of events where each bar on the x-axis corresponds to a specific date's hour.

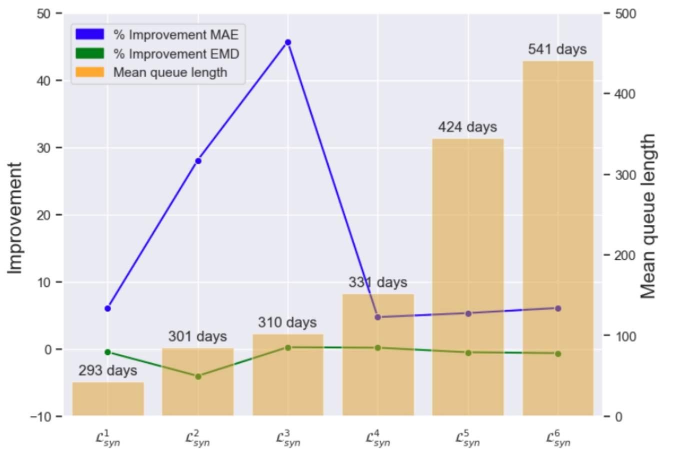


Fig. 5: MAE and EMD improvements by RIMS+ compared to RIMS with different queuing levels related to the same process.

Table VII reports a comparison between RIMS+ and the best among the other simulators on all the logs with a significant queue (see Table V). The table shows that RIMS+ outperforms the other simulators in terms of MAE on all logs (**RQ3**), except for *BPI12*, which, as mentioned above, presents a particular model and set of constraints. On the other hand, RIMS+ slightly worsen the results of the best among the other simulators in terms of EMD on almost all logs. In case of *Confidential 1000*, *Cvs Pharmacy* and *BPI17W*, the RIMS+ EMD is quite close to the one of RIMS but it does not reach the DDPS's or LSTM's performance. Fig. 5 shows how RIMS+ improves the performance of RIMS on logs with queues of increasing length. We can observe that the introduction of the *queue* feature has a high impact in terms of MAE on the first logs, while, although positive, the improvement decreases for higher values of *queue*. In terms of EMD, instead, we have no relevant changes. These results suggest that the *queue* feature improves the MAE performance when significant queues actually occur in the log. However, the obtained improvement deeply increases up to a certain queue length, while becoming stable for queue lengths higher than a certain threshold.

VIII. Conclusion

In this paper we presented and discussed RIMS, a *runtime-integration* technique able to combine DDPS and DL approaches in a successful way, as well as RIMS+, which enriches RIMS with a *queue* feature. Compared to other simulation methods, RIMS/+ is able to accurately delineate the cycle time of each trace. In the future, we plan to study the integration of resource scheduling and different queue policies.

References

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A Reijers, *Fundamentals of business process management*. Springer, 2013.

[2] M. Camargo, M. Dumas, and O. González, "Automated discovery of business process simulation models from event logs," *Decis. Support Syst.*, vol. 134, p. 113284, 2020.

		DDPS			LSTM			Dsim			RIMS		
		$Mean$	CI_l	CI_u	$Mean$	CI_l	CI_u	$Mean$	CI_l	CI_u	$Mean$	CI_l	CI_u
MAE	*Confidential 1000*	55477	51509	59444	32754	32377	33132	36337	35796	36878	**26950**	**25683**	**28217**
	Confidential 2000	234520	179570	289471	37344	37063	37625	8095	7309	8882	**7385**	6381	8390
	Cvs Pharmacy	1061957	1061300	1062613	638665	637616	639714	137636	136104	139168	**102950**	**101130**	**104771**
	PurchasingExample	2127361	2109665	2145058	2299619	2294319	2304919	2280717	2273954	2287481	**2104676**	2082360	2126992
	SynLoan	2295797	2295579	2296015	1446177	1446177	1463812	1675036	1635297	1714775	**1278782**	**1271133**	**1286430**
	Production	312153	307036	317270	128942	117516	140368	**83700**	73572	93828	105988	91713	120264
	ConsultaDataMining	274786	271536	278035	343520	343422	343618	230921	228349	233494	**155994**	141180	170707
	BPI12W	667139	665889	668389	**316642**	**309766**	**323517**	597090	594860	599320	555094	550135	560052
	BPI12W*	694558	694083	695033	316642	309766	323517	**310293**	304608	315977	357940	351301	364579
	BPI17W	952502	949997	955007	757742	653898	861585	417641	416631	418650	**393749**	**391442**	**396056**
EMD	*Confidential 1000*	**0.00115**	**0.00103**	**0.00127**	0.01435	0.01434	0.01437	0.01269	0.01263	0.01274	0.01264	0.01257	0.01270
	Confidential 2000	**0.00109**	**0.00105**	**0.00113**	0.00992	0.00991	0.00993	0.00897	0.00895	0.00899	0.00907	0.00905	0.00909
	Cvs Pharmacy	**0.00093**	**0.00092**	**0.00094**	0.00237	0.00237	0.00237	0.00659	0.00658	0.00659	0.00672	0.00672	0.00673
	PurchasingExample	0.00702	0.00699	0.00706	**0.00525**	**0.00500**	**0.00550**	0.00645	0.00634	0.00656	0.00856	0.00831	0.00880
	SynLoan	0.00799	0.00798	0.00800	0.01926	0.01924	0.01928	0.00473	0.00469	0.00477	**0.00321**	**0.00317**	**0.00325**
	Production	0.01486	0.01438	0.01534	0.00715	0.00647	0.00784	0.00635	0.00589	0.00680	**0.00523**	**0.00487**	**0.00559**
	ConsultaDataMining	0.01573	0.01555	0.01591	0.03830	0.03820	0.03840	0.01233	0.01202	0.01265	**0.00777**	**0.00739**	**0.00816**
	BPI12W	**0.00642**	**0.00640**	**0.00644**	0.00766	0.00765	0.00768	0.00807	0.00806	0.00808	0.00805	0.00804	0.00806
	BPI12W*	**0.00468**	**0.00465**	**0.00470**	0.00766	0.00765	0.00768	0.00799	0.00798	0.00799	0.00798	0.00798	0.00799
	BPI17W	0.00489	0.00488	0.00490	**0.00448**	**0.00446**	**0.00449**	0.00734	0.00728	0.00740	0.00701	0.00694	0.00707

TABLE VI: Mean and CIs of MAE and EMD values related to DDPS, LSTM, *Dsim* and RIMS. The italic font indicates a synthetic log, while the bold font the mean of the best approach and the CIs not overlapped with the others.

		Best			RIMS+		
		$Mean$	CI_l	CI_u	$Mean$	CI_l	CI_u
MAE	*Confidential 1000*	26950	25683	28217	**15489**	**14771**	**16208**
	Cvs Pharmacy	102950	101130	104771	**45756**	**37930**	**53583**
	SynLoan	1278782	1271133	1286430	**827635**	**821431**	**833839**
	BPI12W	**316642**	**309766**	**323517**	454494	452008	456980
	BPI12W*	310293	304608	315977	**301663**	292720	310606
	BPI17W	393749	391442	396056	**308038**	**299884**	**316193**
EMD	*Confidential 1000*	**0.00115**	**0.00103**	**0.00127**	0.01254	0.01247	0.01260
	Cvs Pharmacy	**0.00093**	**0.00092**	**0.00094**	0.00680	0.00679	0.00681
	SynLoan	**0.00321**	**0.00317**	**0.00325**	0.00576	0.00571	0.00582
	BPI12W	**0.00642**	**0.00640**	**0.00644**	0.00813	0.00812	0.00814
	BPI12W*	**0.00468**	**0.00465**	**0.00470**	0.00799	0.00797	0.00801
	BPI17W	**0.00448**	**0.00446**	**0.00449**	0.00847	0.00798	0.00895

TABLE VII: Mean and CIs of MAE and EMD values of the best simulation approach (without the *queue* feature) and of RIMS+. The italic font indicates a synthetic log, while the bold font the mean of the best approach and the CIs not overlapped with the others.

[3] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst, "Discovering simulation models," *Inf. Syst.*, vol. 34, pp. 305–327, 2009.

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

[5] M. Pourbafrani, S. J. van Zelst, and 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.*, ser. LNBIP, vol. 389. Springer, 2020, pp. 249–263.

[6] N. Martin, B. Depaire, and A. Caris, "The use of process mining in business process simulation model construction - structuring the field," *Bus. Inf. Syst. Eng.*, vol. 58, no. 1, pp. 73–87, 2016.

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

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

[9] C. Di Francescomarino and C. Ghidini, "Predictive process monitoring," in *Process Mining Handbook*, ser. LNBIP. Springer, 2022, vol. 448, pp. 320–346.

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

[11] J. Evermann, J. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decis. Support Syst.*, vol. 100, pp. 129–140, 2017.

[12] W. J. Chang and Y. H. Chang, "Design of a patient-centered appointment scheduling with artificial neural network and discrete event simulation," *Journal of Service Science and Management*, vol. 11, no. 01, 2018.

[13] S. Bergmann, S. Stelzer, and S. Straßburger, "On the use of artificial neural networks in simulation-based manufacturing control," *J. Simulation*, vol. 8, no. 1, pp. 76–90, 2014.

[14] J. Pender and 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.

[15] R. D. L. Fuente, I. Erazo, and R. L. Smith, "Enabling intelligent processes in simulation utilizing the tensorflow deep learning resources," in *Winter Simulation Conference, WSC*. IEEE, 2018, pp. 1108–1119.

[16] M. Camargo, M. Dumas, and 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*, ser. LNCS, vol. 13295. Springer, 2022, pp. 55–71.

[17] F. M. Maggi, C. Di Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *Advanced Information Systems Engineering - 26th Int. Conf., CAiSE. Proceedings*, ser. LNCS, vol. 8484. Springer, 2014, pp. 457–472.

[18] C. Di Francescomarino, M. Dumas, F. M. Maggi, and I. Teinemaa, "Clustering-based predictive process monitoring," *IEEE Trans. Serv. Comput.*, vol. 12, no. 6, pp. 896–909, 2019.

[19] B. F. van Dongen, R. A. Crooy, and 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*, ser. LNCS, vol. 5331. Springer, 2008, pp. 319–336.

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

[21] ——, "Discovering generative models from event logs: data-driven simulation vs deep learning," *PeerJ Comput. Sci.*, vol. 7, p. e577, 2021.

[22] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, pp. 37–45, 2018.

[23] A. Senderovich, C. D. Francescomarino, and F. M. Maggi, "From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring," *Inf. Syst.*, vol. 84, pp. 255–264, 2019.