Information Systems 117 (2023) 102248

Contents lists available at ScienceDirect

Information Systems

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

Learning business process simulation models: A Hybrid process mining and deep learning approach ☆

Manuel Camargo [a,b,c,*], Daniel Báron [b], Marlon Dumas [a], Oscar González-Rojas [b]

[a] University of Tartu, Tartu, Estonia
[b] Universidad de los Andes, Bogotá, Colombia
[c] Apromore, Tartu, Estonia

ARTICLE INFO

Article history:
Received 16 December 2022
Received in revised form 18 April 2023
Accepted 24 June 2023
Available online 28 June 2023
Recommended by Gottfried Vossen

Keywords:
Process mining
Data-driven simulation
Deep learning

ABSTRACT

Business process simulation is a well-known approach to estimate the impact of changes to a process with respect to time and cost measures – a practice known as what-if process analysis. The usefulness of such estimations hinges on the accuracy of the underlying simulation model. Data-Driven Simulation (DDS) methods leverage process mining techniques to learn business process simulation models from event logs. Empirical studies have shown that, while DDS models adequately capture the observed sequences of activities and their frequencies, they fail to accurately capture the temporal dynamics of real-life processes. In contrast, generative Deep Learning (DL) models are better able to capture such temporal dynamics. The drawback of DL models is that users cannot alter them for what-if analysis due to their black-box nature. This paper presents a hybrid approach to learn process simulation models from event logs wherein a (stochastic) process model is extracted via DDS techniques, and then combined with a DL model to generate timestamped event sequences. The proposed approach allows us to simulate different types of changes, including the addition of new activity types to a process. This latter capability is achieved by encoding the activities by means of embeddings, rather than representing them as one-hot-encoded categories. An experimental evaluation shows that the resulting hybrid simulation models match the temporal accuracy of pure DL models, while partially retaining the what-if analysis capability of DDS approaches. The evaluation also sheds light into the relative performance of multiple embedding approaches to represent the activities.

1. Introduction

Business Process Simulation (BPS) models allow analysts to estimate the impact of changes to a process with respect to temporal and cost measures – a practice known as "what-if" process analysis [1]. However, the construction and tuning of BPS models is error-prone, as it requires careful attention to numerous pitfalls [2]. Moreover, the accuracy of manually tuned BPS models is limited by the completeness of the process model used as a starting point, yet manually designed models often do not capture exceptional paths.

Previous studies have proposed to extract BPS models from execution data (event logs) via process mining techniques [3]. While Data-Driven Simulation (DDS) models extracted in this way can be tuned to accurately capture the control-flow and temporal behavior of a process [4], they suffer from fundamental limitations stemming from the assumptions about the resources' behavior. One such assumption is that all waiting times are due to resource contention (i.e. a resource not starting a task because it is busy with another task). Another assumption is that resources exhibit robotic behavior: if a resource is available, and it may perform an enabled activity instance, the resource will immediately start it. In other words, these approaches do not take into account behaviors such as multitasking, batching, fatigue effects, and inter-process resource sharing, among others [5].

Other studies have shown that Deep Learning (DL) generative models trained from logs can accurately predict the next event in a case and its timestamp or the suffix of a case starting from a given prefix [6,7]. Suitably trained DL generative models can also be used to generate entire traces and even entire logs [8], which effectively allows us to use a DL generative model as a simulation model. Camargo et al. [9] empirically show that DL models are more accurate than DDS models when it comes to generating logs consisting of activity sequences with start and end timestamps. In particular, generative DL models can emulate delays between activities that DDS models do not capture.

☆ Work funded by European Research Council (PIX Project).
* Corresponding author at: University of Tartu, Tartu, Estonia.
E-mail addresses: manuel.camargo@apromore.com (M. Camargo), df.baron10@uniandes.edu.co (D. Báron), marlon.dumas@ut.ee (M. Dumas), o-gonza1@uniandes.edu.co (O. González-Rojas).

https://doi.org/10.1016/j.is.2023.102248

A drawback of using DL models in this context is that, unlike DDS models, they are not well-suited for what-if analysis due to their black-box nature. While the parameters of a DDS model are directly related to business process concepts, such as case creation, decision points, activity processing time, and resources, those of a DL model are neural network weights, which do not have a direct relation to such concepts. Furthermore, the starting point of a simulation model is a graphical process model, which business process analysts are usually familiar with, whereas the structure of a DL model consist of layers of neurons. While a process analyst would be able to describe a change in the process in terms of changes in the graphical process model and in the simulation parameters, it is unlikely they would be able to do so for a DL model, as the relation between a change in the input (e.g. neural network weights) and a change in the output (e.g. waiting times of the activities) would not be evident.

In light of the relative strengths and weaknesses of these two families of generative models, we propose to combine these approaches in a way that leverages their strengths. Specifically, we hypothesize that DL techniques can enhance the temporal accuracy of BPS models derived from data. Indeed, DL models have the ability to estimate waiting times and processing times of activity instances, based on the observed sequencing of activities in a trace and temporal and contextual parameters, such as the number of active cases, without making specific assumptions about the behavior of resources (e.g. whether resources act robotically or not).

Concretely, this paper presents a method, namely DeepSimulator, that combines DDS and DL methods to discover a BPS model from a log. The idea is to use an automated process discovery technique to extract a process model with branching probabilities (a.k.a. stochastic process model [10]) and to delegate the generation of activity start and end times to a DL model. The proposed method allows us to simulate different types of changes, such as increasing the arrival rate of cases, changing the order of activities, changing the branching probabilities, deleting activities, or adding activities in a process. Among these changes, the insertion of new activities poses a particular challenge in the context of what-if analysis, because the deep learning model is trained on a dataset that does not contain said activities and hence it does not have information about their temporal dynamics (waiting times, processing times). To tackle this challenge, the proposed approach represents the activities of the process by means of an embedding (i.e. a mapping from activities to dense vectors), rather than representing them as one-hot-encoded categories. In this way, when a new activity is added to the model, and provided that the user provides information about the resources associated to this new activity and its processing time distribution, we can map the new activity to its vector representation and thus allow the deep learning model to generate waiting and processing times for it.

An experimental evaluation shows that the resulting hybrid simulation models match the temporal accuracy of pure DL models, while partially retaining the what-if analysis capability of DDS approaches. The evaluation also sheds light into the relative performance of multiple embedding approaches to represent the activities.

This article is an extended and revised version of a conference paper [11]. With respect to the conference version, the key addition is a new approach to learn an embedding for representing activity types, inspired by the popular Word2Vec approach. This embedding approach takes into account the context in which each activity occurs within each trace (relative to other activities) as well as the context in which each resource who performs an activity occurs in each trace (relative to other resources). The experimental evaluation has been extended to compare the accuracy of simulation models trained with different types of activity embeddings.

The paper is structured as follows. Section 2 discusses methods to learn generative models from logs using DDS and DL techniques. Section 3 presents the proposed method while Section 4 presents an empirical evaluation thereof. Finally, Section 5 draws conclusions and sketches future work.

2. Related work

2.1. Data-driven simulation of business processes

Previous studies on DDS methods can be classified in two categories. A subset of previous studies have proposed conceptual frameworks and guidelines to manually derive, validate, and tune BPS parameters from event logs [3,12], without seeking to automate the extraction process. Other studies have proposed methods that automate the extraction and/or tuning of simulation parameters from logs. In this paper, we focus on automated methods. One of the earliest such methods is that of Rozinat et al. [13], who propose a semi-automated approach to extract BPS models based on Colored Petri Nets. Later, Khodyrev et al. [14] proposed an approach to extract BPS models from data, although they leave aside the resource perspective (i.e. the discovery of resource pools). More recently, Pourbafrani et al. [15] present an approach for generating DDS models based on time-aware process trees. In all of the above studies, the responsibility for tuning these parameters is left to the user. This limitation is addressed in the Simod method [4], which automates the extraction of BPS models by employing Bayesian optimization to tune the hyperparameters used to discover the process model and resource pools as well as the statistical parameters of the BPS model (branching probabilities, activity processing times, and inter-case arrival times). This tuning phase seeks to optimize the similarity between the logs produced by the extracted BPS model and (a testing fold of) the original log.

In the experimental evaluation reported later in this paper, we use Simod as a baseline, due to its fully automated approach to discover and tune business process simulation models from event logs. This decision minimizes the introduction of biases in the evaluation process, compared to alternative approaches that would require us to make simulation design choices and/or tuning.

2.2. Generative DL models of business processes

A Deep Learning (DL) model is a network of interconnected layers of neurons (perceptrons) that collectively perform non-linear data transformations [16]. The objective of these transformations is to train the network to learn the patterns observed in the data. In theory, the more layers of neurons in the system, the more it will detect higher-level patterns via composition of complex functions [16]. A wide range of neural network architectures have been proposed, e.g. feed-forward networks, Convolutional Neural Networks, Variational Auto-Encoders, Generative Adversarial Networks (GAN) (often in a combination with other architectures), and Recurrent Neural Networks (RNN). The latter type of architecture is specifically designed to handle sequential data.

DL models have been widely applied in the field of predictive process monitoring. Evermann et al. [7] proposed an RNN architecture to generate the most likely remaining sequence of events (suffix) of an ongoing case. This architecture cannot handle numeric features and thus cannot generate timestamped events (timestamps are numeric). This limitation is shared by the approach in [17] and others reviewed in [18]. Tax et al. [6] use

an RNN architecture known as Long-Short-Term Memory (LSTM) to predict the next event in an ongoing case and its timestamp, and to generate the remaining sequence of timestamped events from a given prefix of a case. However, this approach cannot handle high-dimensional inputs due to its reliance on one-hot encoding of categorical features. Its precision deteriorates as the number of categorical features increases. This limitation is lifted by DeepGenerator [8], which extends the approach in [6] with two mechanisms to handle high-dimensional input: n-grams and embeddings. This approach also addresses the problem of generating long suffixes (not well handled in [6]) and entire traces, by using a random next-event selection approach. It is also able to associate a resource to each event in a trace. More recently, Taymouri et al. [19] proposed a GAN-LSTM architecture to train generative models that produce timestamped activity sequences (without associated resources).

Camargo et al. [9] compare the relative accuracy of DL models against DDS models for generating sequences of the form (activity, start timestamp, end timestamp). This comparison suggests that DL models may outperform DDS models when trained with large logs, while the opposite holds for smaller logs. Camargo et al. [9] additionally show that DL models generally outperform DDS methods when it comes to predicting activity start and end timestamps.

Building upon the aforementioned findings, in this study, we introduce the DeepSimulator method for learning generative models for business process simulation by combining techniques from both DDS and DL models. This approach aims to leverage the accuracy and precision of DL models in generating start and end times of activity instances, while retaining certain characteristics of DDS models to enable what-if analysis, thereby harnessing the strengths of both families of models for improved accuracy.

2.3. Representation learning

DL models require input data to be encoded in numerical vector matrices representing samples of the dynamics observed in the event logs (e.g. distances, directions). Creating this type of input demands the transformation of the source data, commonly organized in tabular formats, where each instance can contain a multitude of categorical and continuous attributes [20]. A key factor for successfully training DL models is to create compact vector representations that avoid an explosion of features when encoded.

Representation learning is the process of extracting and organizing the explanatory factors hidden in the data. This process can be achieved by training a specialized model that maps categories (e.g. activity types, resources) to a numeric vector representation, such that categories that are semantically similar to each other are mapped to vectors that are close to each other in a high-dimensional space. Representation learning has been widely used in the field of process mining as a feature encoding method for predictive process monitoring, clustering, anomaly detection, or conformance checking [21]. Some of the most used representation learning techniques are text embeddings and entity embeddings.

Text embeddings extract vector representations from words, sentences, and documents. Text embeddings consider the dependency between the words in a sentence, establishing the context of every word in a sentence (i.e., the surrounding words in a window size) to produce individual word vectors. In this way, semantically and syntactically similar words tend to be located at closer coordinates in representation space. Word2vec is one of the best-known algorithms for this type of embedding. This technique, which is widely used in NLP, has yet to be extensively evaluated in the context of process mining. The authors in [22] proposes the need of adapting word2vec for the representation of processes at different levels of abstraction: activity (act2vec), trace (trace2vec), log (log2vec), and model (model2vec). At the activity level (act2vec), which is the main research concern in this article, the authors propose the use of word2vec to encode sequences of activity labels. However, no implementation or empirical evaluation is carried out. Moreover, the inclusion of other attributes at the activity level (e.g. resources) different from the activity label is not carried out.

Entity embeddings, on the other hand, take into account the relations between different features in the dataset and are generally used in the context of supervised DL models. DL models have a hierarchical structure in which multiple levels of representation are created, starting from simple representations in the lower layers to creating more abstract and complex ones in the upper ones by applying non-linear transformations. Thanks to this structure, it is possible to place in any DL model a specialized embedded layer in a low-dimensional layer to capture the semantics of the input processed in higher-dimensional layers. This embedded layer can capture the semantics of the input by placing similar inputs close together in the embedding space. Entity embeddings have been widely used in the context of predictive process monitoring [7,8,17,23,24]. In these previous studies, entity embeddings are used as an intermediate step in the training pipeline of the predictive models.

3. Hybrid learning of BPS models

Fig. 1 depicts the architecture of the DeepSimulator approach. The architecture is a pipeline with three phases. The first phase uses PM techniques to learn a model to generate sequences of (non-timestampted) events. The second and third phases enrich these sequences with case start times and activity start and end times. Below, we discuss each phase in turn.

3.1. Phase 1: Activity sequences generation

The aim of this phase is to extract a stochastic process model [10] from the log and to use it to generate sequences of activities that resemble those in the log. A stochastic process model is a process model with branching probabilities assigned to each branch of a decision point. In this paper, we represent process models using the standard BPMN notation. Phase 1 starts with a *control-flow discovery* step (see Fig. 1 step 1.1), where we first discover a plain (non-stochastic) process model using the Split Miner algorithm [25]. This algorithm relies on two parameters: the sensitivity of the parallelism oracle (η) and the level of filtering of directly-follow relations (ϵ). The former parameter determines how likely the algorithm will discover parallel structures, while the latter determines the percentage of directly-follows relations between activity types are captured in the resulting model. Like other automated process discovery algorithms, the Split Miner discovers a process model that does not perfectly fit the log. The discovered process model cannot parse some traces in the log. This hinders the calculation of the branching probabilities. Accordingly, we apply the *trace alignment* algorithm (see Fig. 1 step 1.2) in [26] to compute an alignment for each trace in the log that the model cannot parse. An alignment describes how a trace can be modified to be turned into a trace that can be parsed by the model (via "skip" operations). Based on the alignments, we either *repair* each non-conformant trace, or we *replace* it with a copy of the most similar conformant trace (w.r.t. string-edit distance). The choice between the repair and the replacement approaches is a parameter of the method.

Next, DeepSimulator uses the (conformant) event log to discover the *branching probabilities* (see Fig. 1 step 1.3) for each

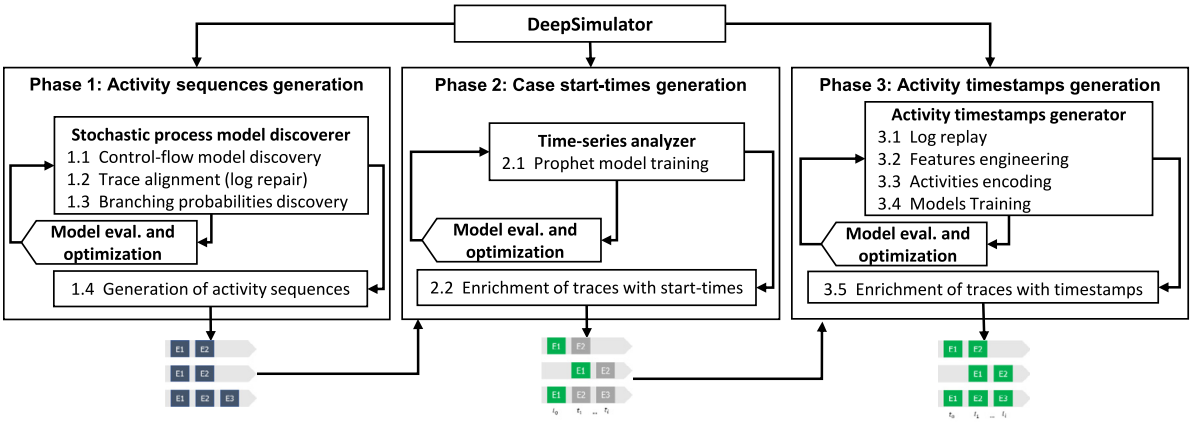

Fig. 1. Overview of the proposed BPS model discovery method.

branching point in the model. Here, DeepSimulator offers two options: (i) assign equal values to each conditional branch; or (ii) compute the branching probabilities by replaying the aligned event against the process model. The first approach may perform better for smaller logs, where the probabilities computed via replay are not always reliable, while the latter may be preferable for larger logs.

Note that we make the assumption that the probabilities associated with each decision gateway are independent. An alternative approach could be to model the choice of branches via decision rules based on data attributes, in such a way that two or more gateways could be inter-dependent. The latter would require an approach to discover such decision rules from the event log. This alternative approach can be considered as a potential avenue for future research.

The DeepSimulator combines the process model and the branching probabilities to assemble a stochastic process model. In this step, the DeepSimulator uses a Bayesian optimization technique to discover the hyperparameter settings (i.e., values of ϵ, η, replace-vs-repair, and equal-vs-computed probabilities) that maximize the similarity between the generated and the ground truth sequences in terms of activity sequences. The optimizer uses a holdout method, and as a loss function, it uses the Control-Flow Log Similarity (CFLS) metric described in [4]. The CFLS metric is the mean string-edit distance between the activity sequences generated by the stochastic process model and the traces in the ground-truth log after their optimal alignment.[1] Finally, in the *sequences' generation* step (see Fig. 1 step 1.4), DeepSimulator uses the resulting stochastic process model to generate a bag of activity sequences without timestamps. This bag is used as the log's base structure in Phase 3.

3.2. Phase 2: Case start times generation

In this phase, the start time of each process instance in the output log is generated, which is crucial for initiating new cases in the Deep Simulator. This step is necessary because the generative models used in phase 3 cannot generate absolute start times for cases, as they are limited to generating relative times at the trace level after case instantiation. As all the timestamps generated by the DL models are relative, generating the first timestamp in this phase is indispensable for calculating the absolute times of the simulated log. Likewise, this phase is critical for calculating the contextual and the inter-case time attributes used by the generative models in phase 3. These features will be described in more detail in the next phase.

Traditionally, DDS models generate the start time of cases by randomly drawing from an unimodal distribution of the inter-arrival times between consecutive cases. A typical BPS model captures the interarrival times using a negative exponential distribution [1] (i.e., it models the creation of cases as a Poisson process). However, a single distribution is not realistic enough to capture real scenarios. For example, cases might be created more frequently on Mondays than on Thursdays in a claims handling process.

Instead of fitting an interarrival distribution, the DeepSimulator models the case generation as a time series prediction problem as the number of cases generated per hour of the day. This type of modeling allows us to use robust techniques such as ARIMA or ETS tested successfully in several contexts such as Stock Market Analysis or Workload Projections. DeepSimulator uses the Prophet [28] model proposed by Facebook because it is one of the simplest but, at the same time, more accurate predictive models for this type of task. Prophet starts from the time series decomposition into four main components (i.e., trend, seasonality, holidays, and error) and applies specialized techniques to model each component.

The trend component decomposes those non-periodic changes in the time series values, which are modeled using logistic growth models or Piecewise linear models. The seasonality component decomposes the periodic changes repeated at fixed intervals (hours, weeks, months, or years), which are modeled by using the Fourier series. The holidays component represents the effects of holidays that occur on potentially irregular schedules over one or more days. This component is optionally modeled and is defined manually by a domain expert, since it is specific to each time series. The model automatically calculates the error, corresponding to all those unforeseen changes that the model cannot fit.

In the *Prophet model training* step (see Fig. 1 step 2.1), we use a saturated logistic growth model to fit the case generation trend. We chose this model, considering that the time series is limited by a lower and upper bound. The lower bound corresponds to 0, which is the minimum number of cases attended in the process, and the upper bound is theoretically limited by the capacity of the process. The parameter that most significantly affects data trend capture is changepoint-prior-scale, which determines how much the trend changes at the trend change points. This parameter needs tuning, since a too low value may cause under-fitting while a too high value may cause over-fitting. Accordingly, for this parameter, DeepSimulator explores values in the interval [0.001,

[1] We did not use the stochastic conformance checking metrics over Petri nets of [10] since our method handles BPMN models with inclusive join gateways, which cannot be directly transformed to Petri Nets (without exponential blowout) as shown in [27].

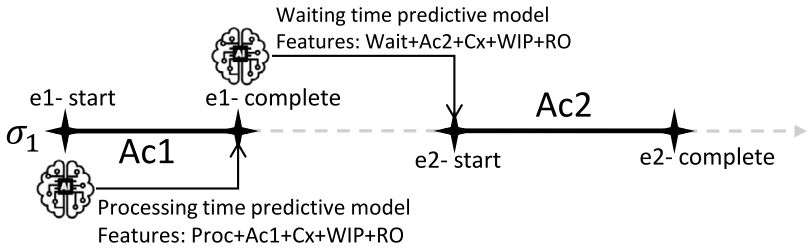

Fig. 2. Predictive models timeline and features. In this figure, Ac1 and Ac2 denote two sequentially executed activities. The predictive model for processing time utilizes inputs such as processing time combined with activity labels, as well as time contextual attributes (Cx), Work-in-progress (WIP), and Resources' Occupation (RO). On the other hand, the predictive model for waiting time employs the duration between the completion of one activity and the initiation of the subsequent one, as opposed to the processing time.

0.5]. Analogously, the parameter that most directly affects the seasonality capture is the seasonality-prior-scale. This parameter affects the flexibility of seasonality learning. If the value is too small, the model tends to focus on small fluctuations, while a large value may cause the model to focus only on large fluctuations. For this parameter, DeepSimulator explores values in [0.01, 10]. We do not define the Holidays component in the Prophet model. The holidays component could be discovered by a calendar discovery technique such as the one proposed in [5], but discovering such calendars is orthogonal to the focus of the present paper.

We use grid search for selecting the best hyperparameters of the Prophet model, as the search space consists of only sixteen configurations (cf. Section 4.3). We rely on the internal mechanisms embedded in Prophet for cross-validation and selection of cutoff points. During the *Enrichment of traces with start-times* step (see Fig. 1 step 2.2), we use the trained Prophet model to determine the number of cases to be created at each hour of the simulation. We then generate the start times, for each simulation hour, by modeling the intercase arrival times via a normal distribution (within the hour).

3.3. Phase 3: activity timestamps generation

We enhance the activity sequences generated in Phase 1 to capture waiting times and processing times in this phase. The DeepSimulator trains two LSTM[2] models to perform two predictive tasks: the processing time of a given activity (herein called the *current activity*) and the waiting time until the start of the next activity. This task differs in two ways from approaches used to predict the next event and its timestamp [8,19]. First, we do not seek to predict the next event, since the sequences of activities are generated by the stochastic process model (cf. Phase 1). Second, we need to support changes in the process model (e.g., adding or removing tasks) for enabling what-if analysis.

Therefore, we train one model specialized in predicting the processing time of the current activity and another specialized in predicting the waiting time until the next activity. Both models differ in the set of features since they act at different moments in the predictive phase, as shown in Fig. 2. The processing time predictive model uses the following features as inputs: the label and processing time of the current activity, the time of day of the current activity's start timestamp, the day of the week, and intercase features such as the Work-in-progress (WIP) of the process and the activity and Resources' Occupation (RO) at the start of the activity. The waiting time predictive model uses the following features as inputs: the next activity's label, the time of day of the current activity's end timestamp, the day of the week, and intercase features such as the WIP of the process and the RO at the end of the current activity.

In the *log replay* step (see Fig. 1 step 3.1), we calculate the waiting and processing times of each activity by replaying each trace in the input log (or in a training subset thereof) against the process model discovered in Phase 1. An activity's processing time is the difference between its end and start timestamps. An activity's waiting time is the difference between its start time and enablement time, i.e., when it was ready to be executed according to the process model. All waiting and processing times are scaled to the range [0...1] by dividing them by the largest values.

In the *feature engineering* step (see Fig. 1 step 3.2), we compute and encode all the remaining features used by the models. We calculate the time of the day as the elapsed seconds from the closest midnight until the event timestamp; this feature is scale over 86,400 s. The day of the week is modeled as a categorical attribute and encoded using one-hot encoding. We include these latter features since they provide contextual information, allowing the model to find seasonal patterns in the data that may affect waiting and processing times. In the same way, considering that the overall process performance is affected by the process' WIP and the RO [30], we use two inter-case features that measure these variations.

The WIP of the process measures the number of active tasks at each moment in the log transversally. The RO measures each resource pool's percentage occupancy in the log, implying that a new feature is created for each pool to record the occupation-specific variations. Since the information about the size and composition of the resource pools is not always included in the logs, we grouped resources into roles by using the algorithm described in [31]. This algorithm discovers resource pools based on the definition of activity execution profiles for each resource and the creation of a correlation matrix of similarity of those profiles. WIP and RO are calculated by replaying over time the log events, recording the variations in both features at every time point.

In the *activities encoding* step (see Fig. 1 step 3.3), we encode the activities of the process by means of an embedding. Embeddings help to prevent exponential feature growth associated with one-hot encoding [8]. In addition, they allow us to make predictions for new categories (i.e. for new activity labels). An embedding maps each category (each activity label) to a point in an n-dimensional space. A suitable embedding is one that maps semantically similar activities to points that are close to each other. If this is the case, the deep learning model is likely to treat the semantically similar activities as equivalent. In other words, a representation that correctly represents how similar or different one activity is to another allows the deep learning model to predict the temporal behavior of activities it has never seen before, so long as they are similar to previously observed

[2] We used LSTM networks as the core of our predictive models since they are a well-known and proven technology to handle sequences, as the nature of a business process event log [29]. The proposed models were based on previous works [6–8] that have extensively explored several architectural options of LSTMs, such as the use of stacked vs. unstacked models or the use of shared layers vs. specialized ones.

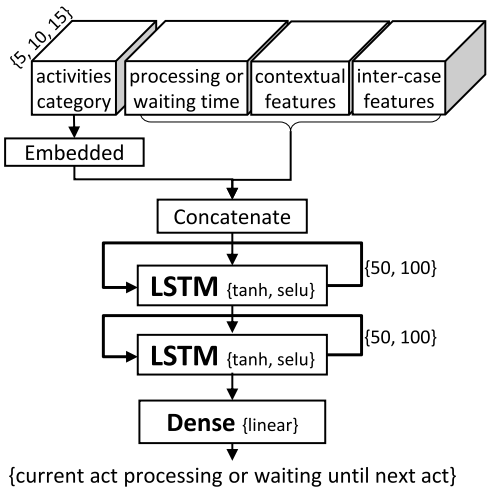

Fig. 3. DL models architectures.

activities. The embedding methods considered in this article are described in Section 3.4.

Once encoded the features, In the *models training* step we extract n-grams of fixed sizes from each trace to create the input sequences to train the model (see Fig. 1 step 3.4). As shown in Fig. 3, both models are composed of two stacked LSTM layers and a dense output. A model receives the sequences as inputs and the expected processing and waiting times as a target. The user can vary the number of units in the LSTM layers, the activation function, the size of the n-gram, and the use of all the RO inter-cases or just the one of the resource pool associated to the execution of the activity.

In the *Enrichment of traces with timestamps* step (see Fig. 1 step 3.5), The output log is generated by assembling each generated sequence (see Phase 1), with the generated case start time (see Phase 2) and the processing and waiting times predicted iteratively (see Phase 3). In each iteration, the trained model predicts times relative to the current activity in seconds, which are transformed into absolute times by adding them to the start time of the case. Then, DeepSimulator generates a simulated log composed of a bag of traces, each trace consisting of a sequence of triplets (activity label, start timestamp, end timestamp).

3.4. Embedding methods for activity representation

We consider three different embedding methods to learn activity representations. The first one is called dot-product (DOT-PROD). DOT-PROD is an *entity embedding model* that uses an independent neural network to perform a simple classification task. Specifically, this independent neural network is fed with positive and negative examples of associations between activities and resources. The network therefore maps activities executed by the same or by similar sets of resources to nearby points. This mechanism also allows us to add a new point in that space by updating the encoding model without altering the predictive model's input size. Each time an unobserved activity is added to the process model for what-if analysis, we generate positive and negative examples of this new activity and use these examples to determine the coordinates of the new activity label to be encoded in the embedded space. The number of embedded dimensions is calculated as the fourth root of the number of categories to avoid a possible collision between them, according to a common recommendation used in the NLP community.[3] Fig. 4 presents the architecture of the network used in DOT-PROD for training the embedded layers.

The second embedding method we consider is called ACT2V+WR. It is an adaptation of the well-known word representation method Word2vec [32] and it is inspired by the act2vec approach presented in [22]. Word2vec calculates vector representations of words based on each word's context in a corpus of text samples. This means that words that typically appear close in the text will have similar representations, reflecting the fact that they are semantically or syntactically related. The word2vec method is designed for one-dimensional sequences, i.e., sentences made up of words. In our context, the sequences we manipulate are traces, consisting of multidimensional elements, specifically, sequences of activity instances, each composed of an activity label and a resource (in addition to the start and end timestamps). To encode these sequences, we train two separate word2vec models: one for the activity labels and another for the resources. In other words, each trace in the event log is mapped to a sequence of activity labels (to construct the first model) and to a sequence of resources (for the second model). To consolidate the two models, and considering that an activity may be performed by one among multiple resources, we calculate weights corresponding to the frequency in which each resource performs a given activity. The ACT2V+WR method is designed to capture both the context of each activity and the importance of each resource relative to each activity.

If the embedded matrices in our method have to be updated as a result of incorporating a previously unseen activity, we employ a mechanism that involves retraining the model using synthetic sequences. Leveraging the capability of our process model to generate sequences of activities and resources, we generate a set of traces that include the activity to be added. By feeding these synthetic sequences to the model, we can incorporate the new activity without modifying the embeddings of the existing activities. Subsequently, we can inject the updated embedding into the previously trained predictive model without altering the input size. Fig. 5 presents the architecture of the network used in ACT2V+WR for training the embedded layers.

The third proposed method combines the DOT-PROD and ACT2V+WR methods to integrate the perspectives of associations between activities/resources and the context of each element into a single representation. As a first step, the embedding was calculated for each activity using the weighting method described above. Next, the resulting vector representations were taken as the initialization of the weights for the embedding layers in the dot-product model instead of using random values. This allows the dot-product model to start training with values that already consider the context associated with each activity.

4. Evaluation

We empirically compare the DeepSimulator method vs. DDS and DL approaches in terms of the similarity of the simulated logs they generate relative to a fold of the original log (see Section 4.3). We also evaluate how different embedding methods for activity representations can improve the accuracy of the current (AS-IS) generated models (see Section 4.4). In a third experiment, we assess the impact of the stochastic process model and the inter-arrival generation model used by the Deep Simulator, on the accuracy of the Deep Learning models (see Section 4.5). In a fourth experiment, we evaluate the accuracy of DeepSimulator for "what-if" analysis tasks of modifying the case creation intensity and adding new activities to a process (see Section 4.6). We discuss how the embedding method with the best performance contributes to improve the accuracy of the what-if generated models.

[3] https://www.tensorflow.org/guide/feature_columns

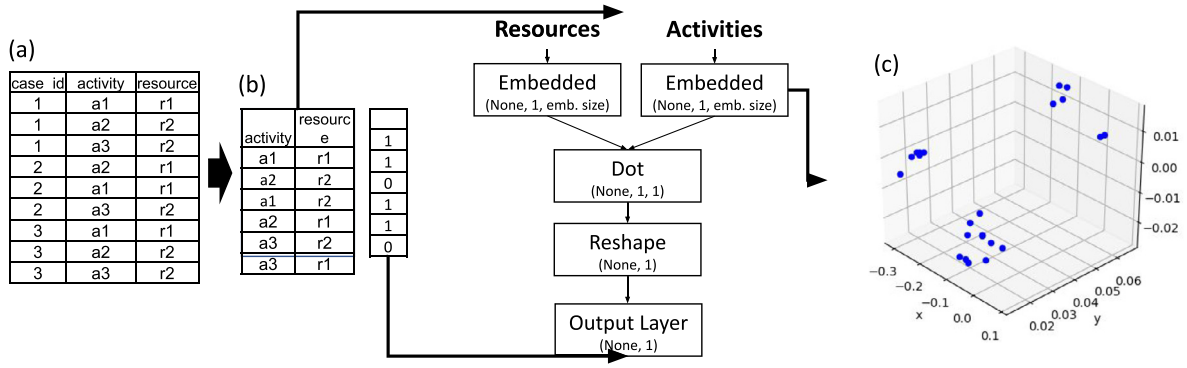

Fig. 4. Embedding network architecture and results: (a) event-log example. (b) The architecture of the model we used to encode the activities. (c) Each point corresponds to one activity; the distances between the points were determined based on the relations the embedding model found in the data.

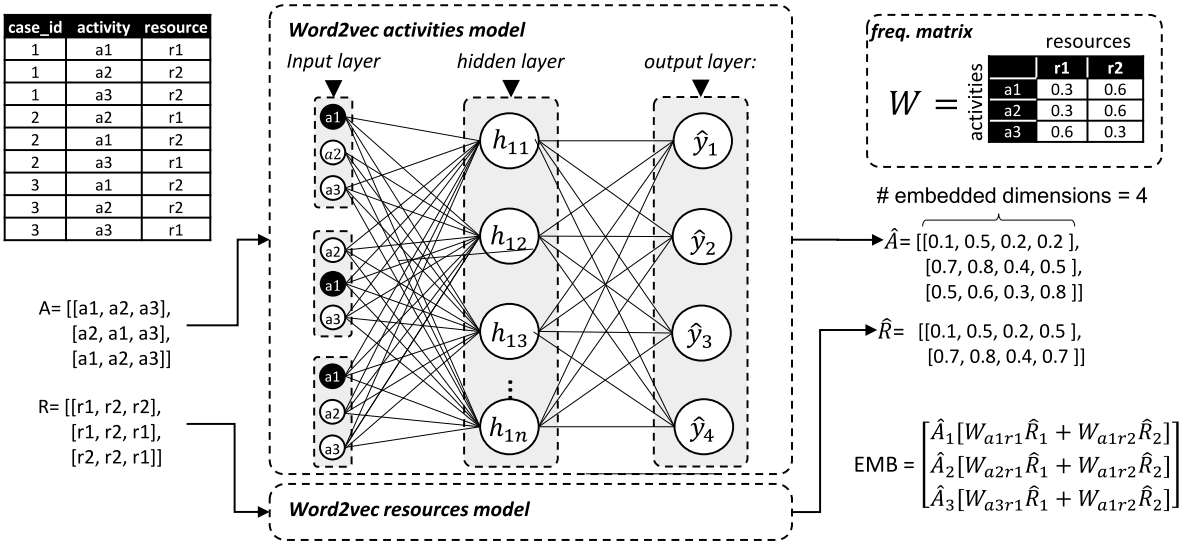

Fig. 5. ACT2vec model architecture.

4.1. Datasets

We evaluated the approaches using 9 logs that contain both start and end timestamps. These logs were obtained from a combination of real-life logs (R), collected from public and private sources, and synthetic logs (S), generated through simulation models of real-world processes. These nine event logs were selected based on their distinct characteristics in control-flow and times and their diverse domains of origin. To account for different data availability scenarios, we further classified these logs into categories of Large or Small, based on the number of cases and activities per case. Below, we provide an overview of the event logs employed.

- The event log of a manufacturing production (MP) process is a publicly available log that contains the steps extracted from an Enterprise Resource Planning (ERP) system.
- The event log of a purchase-to-pay (P2P) process is a publicly available synthetic log generated from a model that is not accessible to the authors.
- The event log from a Colombian University's Academic Credentials Recognition (ACR) process was obtained from its Business Process Management (BPM) system (Bizagi).
- The W subset of the BPI12 event log is a publicly available log of a loan application process from a financial institution in the Netherlands. The W subset specifically includes events corresponding to activities performed by human resources, i.e., activities with a duration.
- The W subset of the BPI17 event log is an updated version of the BPI12 log, extracted based on recommendations from winning teams in the BPIC 2017 challenge.
- For training purposes, we utilized three event logs, each representing a different data size scenario. The INS log is a private log of a real-life insurance claims process, and detailed results for this log will be provided only confidential for confidentiality reasons.
- We also used three synthetic logs generated from simulation models of real-life processes. These models are designed to be complex, incorporating scenarios involving parallelism, resource contention, and scheduled waiting times. The CVS retail pharmacy (CVS) event log represents a large-size training data scenario, derived from a simulation model described in the book "Fundamentals of Business Process Management" [1]. The CFM and CFS event logs were generated from an anonymized confidential process and represent scenarios with large and small training data sizes, respectively.

Table 1 provides descriptive statistics of the logs. The BPI17 W log have the largest number of traces and events, while CFS and P2P have fewer traces but more events per trace.

Table 1
Event logs description.

Size	Source	Log	# Traces	# Events	#Act.	Avg. activities per trace	Avg. duration	Max. duration	#Train cases	#Test cases	Testing unique variants
LARGE	R	BPI17W	30270	240854	8	7.96	12.66 days	286.07 days	23461	5718	1111
	R	BPI12W	8616	59302	6	6.88	8.91 days	85.87 days	6781	1253	206
	S	CVS[b]	10000	103906	15	10.39	7.58 days	21.0 days	8000	2000	0
	S	CFM[b]	2000	44373	29	26.57	0.76 days	5.83 days	1605	391	108
SMALL	R	INS[a]	1182	23141	9	19.58	70.93 days	599.9 days	946	236	194
	S	CFS[b]	1000	21221	29	26.53	0.83 days	4.09 days	803	191	63
	R	ACR	954	4962	16	5.2	14.89 days	135.84 days	673	173	2
	R	MP	225	4503	24	20.01	20.63 days	87.5 days	180	45	38
	S	P2P	608	9119	21	15	21.46 days	108.31 days	449	98	2

[a]Private logs.
[b]Generated from simulation models of real processes.

4.2. Evaluation measures

To evaluate the accuracy of a model M produced by one of the methods under evaluation, we compute a distance measure between a log generated by model M and a ground-truth log (a testing subset of the original log). In all of our experiments, we use two distance measures: the Mean Absolute Error (MAE) of cycle times and the Earth-Mover's Distance (EMD) of the normalized histograms of activity timestamps grouped by day/hour.

The *cycle time MAE* measures the temporal similarity between two logs at the *trace level*. The absolute error of a pair of traces T1 and T2 is the absolute value of the difference between their cycle times. The cycle time MAE is the mean of the absolute errors over a collection of paired traces. Given this trace distance notion, we pair each trace in the generated log with a trace in the original log using the Hungarian algorithm [33] so that the sum of the trace errors between the paired traces is minimal.

The cycle time MAE is a rough measure of the temporal similarity between the ground-truth and the simulated traces. But it does not consider the start time of each case, nor the start and end timestamps of each activity. To complement MAE, we use the *Earth Mover's Distance (EMD)* between the normalized histograms of the timestamps grouped by day/hour in the ground-truth and the generated logs. 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. The EMD is zero if the observed distributions in the two logs are identical, and it tends to one the more they differ.

4.3. Experiment 1: Accuracy of AS-IS generated models

4.3.1. Setup

This experiment aims to compare the accuracy of DeepSimulator models (herein called DSIM models) vs. DDS and DL models. We use SIMOD [4] as a baseline DDS approach since it is fully automated both w.r.t. parameter discovery and tuning. As DL baselines, we use an adaptation of the LSTM approach proposed by Camargo et al. [8] (herein labeled the LSTM method) as well as the GAN-LSTM approach by Taymouri et al. [19] (herein labeled GAN). Both of these DL approaches have been shown to achieve high accuracy w.r.t. the task of generating timestamped trace suffixes [29]. Fig. 6 summarizes the experimental setup. We use the hold-out method with a temporal split criterion to divide the logs into two main folds: 80% for training-validation and 20% for testing. This means that we only preserve the cases that were entirely executed before the split point and begin after the split for the train/test stages. This is done without trimming cases at the split point. By doing so, we ensure that the models are not using data they are not supposed to know during the training process. From the first fold, we took the first 80% for training and 20% for validation. We use temporal splits to prevent information leakage [8,19].

The DDS technique (SIMOD) is set to explore 15 parameter configurations to tune the stochastic process model. For each configuration, we execute five simulation runs and compute the CFLS measure (cf. Section 3.1) between each simulated log and the validation fold. We select the stochastic model that gives the lowest average CFLS w.r.t. the validation fold. The optimizer is set to explore 20 simulation parameter configurations (i.e. the parameters that Simod uses to model resources and processing times), again using five simulation runs per configuration. We select the configuration with the lowest average EMD (cf. Section 4.2) between the simulated log and the validation fold. We used the parameter ranges given in Table 2 for tuning.

The LSTM technique is hyperparameter-optimized using grid search over a space of 48 possible configurations (see Table 2). For LSTM model training, we use 200 epochs, the cycle time MAE as the model's loss function, Nadam as the optimizer, and early stopping and dropout to avoid model over-training. The GAN technique is configured to dynamically adjust the size of the hidden units in each layer so that their size is twice the input's size, as proposed by the authors [19]. We use 25 training epochs, a batch of size five, and a prefix size of five. DSIM is tuned by randomly exploring 15 parameter configurations with five simulation runs per configuration in the stochastic model discovery phase (cf. Section 3, Phase 1). In Phases 2 and 3, we use grid search to explore the space of hyperparameter configurations specified in Table 2

We generate four models per log: one SIMOD, one LSTM, one GAN, and one DSIM. We then generate five logs per retained model, each with the same number of traces as the original log's testing fold to ensure the comparability. Each generated log is compared with the testing fold using the MAE and EMD measures. We report the mean of each of these measures across 5 runs.

4.3.2. Results

Table 3 show the results grouped by metrics, log size and source type. Note that MAE and EMD are error/distance measures (lower is better). In 3 out of 4 large logs, DSIM outperforms LSTM and SIMOD w.r.t the MAE measure. In the small logs, DSIM attains lower MAE in 3 out of 5 logs and has similar MAE w.r.t. SIMOD in one other log. Similarly, DSIM outperforms SIMOD in 6 of 9 logs w.r.t. EMD, and achieves results similar to SIMOD in two others.[4]

The results suggest that DSIM is often able to outperform the baselines when it comes to replicating the as-is behavior recorded

[4] We cannot measure EMD for the LSTM and GAN models because EMD requires that the timestamps in the log are absolute timestamps, while the LSTM and GAN approaches produce relative timestamps (w.r.t. an unknown case start time). It would be possible to extend the LSTM/GAN approaches to generate logs with absolute timestamps by coupling them with a model to generate start times (e.g. based on Prophet) but this extension is outside the scope of this paper.

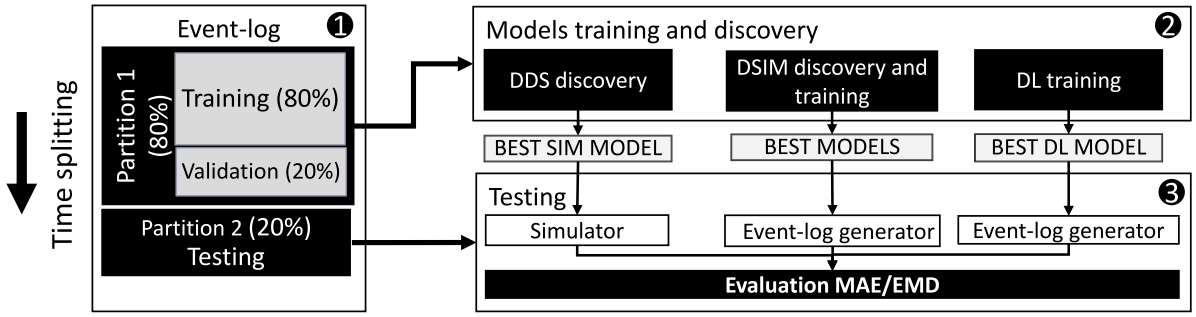

Fig. 6. Setup of experiment 1.

Table 2
Hyperparameters used by optimization techniques.

Model	Stage	Parameter	Distribution	Values
SIMOD	Structure discovery	Parallelism threshold (ϵ)	Uniform	[0…1]
		Percentile for frequency threshold (η)	Uniform	[0…1]
		Conditional branching probabilities	Categorical	{Equiprobable, Discovered}
	Time-related parameters discovery	Log repair technique	Categorical	{Repair, Removal, Replace}
		Resource pools similarity threshold	Uniform	[0…1]
		Resource availability calendar support	Uniform	[0…1]
		Resource availability calendar confidence	Uniform	[0…1]
		Instances creation calendar support	Uniform	[0…1]
		Instances creation calendars confidence	Uniform	[0…1]
LSTM	Training	N-gram size	Categorical	[5, 10, 15]
		Input scaling method	Categorical	{Max, Lognormal}
		# units in hidden layer	Categorical	{50, 100}
		Activation function for hidden layers	Categorical	{selu, tanh}
		Model type	Categorical	{shared_cat, concatenated}
DSIM	Structuregeneration	Parallelism threshold (ϵ)	Uniform	[0…1]
		Percentile for frequency threshold (η)	Uniform	[0…1]
		Conditional branching probabilities	Categorical	{Equiprobable, Discovered}
	Cases start times generation	changepoint-prior-scale	Categorical	{0.001, 0.01, 0.1, 0.5}
		seasonality-prior-scale	Categorical	{0.01, 0.1, 1.0, 10.0}
	Timestampsgeneration	N-gram size	Categorical	{5, 10, 15}
		# units in hidden layer	Categorical	{50, 100}
		Activation function for hidden layers	Categorical	{selu, tanh}
		Single resource-pool intercase feature	Boolean	{True, False}

Table 3
Evaluation results (lower values are better).

Size	Type	Log	MAE				EMD	
			GAN	LSTM	SIMOD	DSIM	SIMOD	DSIM
LARGE	R	BPI17 W	828165	603688	961727	*418422*	*0.016873*	0.034584
		BPI12 W	653656	*327350*	662333	548813	0.056789	*0.020098*
	S	CVS	952004	667715	1067258	*158902*	0.018955	*0.000004*
		CFM	956289	15078	252458	*8441*	0.240567	*0.239087*
SMALL	R	INS	1302337	1516368	*1090179*	1190019	0.143675	*0.142097*
		ACR	296094	341694	230363	*165411*	0.207050	*0.205106*
		MP	210714	321147	298641	*157453*	0.062227	*0.050479*
	S	CFS	717266	33016	*15297*	24326	*0.222515*	0.266749
		P2P	2347070	2495593	1892415	*1836863*	*0.130655*	0.132898

in an event log. This conclusion should be tempered in light of two threats to validity: (i) an external threat to validity stemming from the limited number of events logs in the experiment; and (ii) a threat to construct validity created by the fact that the accuracy measures do not necessarily capture all the nuances in the control-flow and temporal behavior captured in the original and simulated event logs.

4.4. Experiment 2: Accuracy improvement by embedding models

4.4.1. Setup

This experiment aims to compare the accuracy of DSIM models using different embedding methods. The main hypothesis is that using information from the activity execution context and the frequency of activity execution by the associated resources can improve the accuracy of the representation of activities in the embedded layers. These, in turn, can improve the accuracy of the predictive models. To test this hypothesis, we used the three embedment models described in Section 3.4, i.e., DOT-PROD, ACT2V+WR, and DOT-PROD+ACT2V+WR. The baseline model in this experiment is DOT-PROD since it is the default model used by DSIM. As in experiment 1, we use the hold-out method with a temporal split criterion to divide the logs into two main folds: 80% for training validation and 20% for testing.

Since our purpose is to evaluate the effect of the embedding methods on the accuracy of the activity timestamps generation,

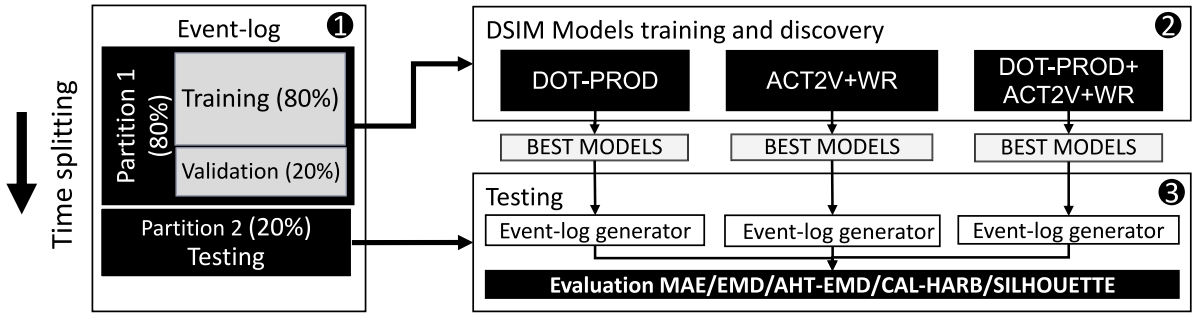

Fig. 7. Setup of experiment 3.

the three variations of the embedding methods used the same DSIM stochastic model (cf. Section 3, Phase 1) and Case start times generation model (cf. Section 3, Phase 2). We train three DSIM models per log, one per embedding model, and then we generate 10 logs per retained model, each with the same number of traces as the original log's testing fold to ensure comparability. Fig. 7 summarizes the experimental setup.

Each generated log is compared with the testing fold using the MAE and EMD measures. Additionally, we report the Absolute Hour Timestamp EMD (AHT-EMD) described in Chapela-Campa et al. [34] to confirm that the results do not depend on the chosen metric. Similarly, we report two metrics used in clustering tasks to evaluate the characteristics of the embeddings generated by these models: The SILHOUETTE distance, which refers to the mean intra-cluster distance, and the Calinski-Harabasz (CAL-HARB) index that measures the Variance Ratio Criterion of the clusters as the sum of between-clusters dispersion and inter-cluster dispersion for all clusters. For both metrics a higher score denotes a better performance. We report the mean of each of these measures across ten runs.

4.4.2. Results

Table 4 presents the results of the accuracy in terms of time metrics (lower is better) and clustering metrics (higher is better) of the three approaches. Regarding the time metrics, DOT-PROD+ACT2V+WR reports the best results in terms of MAE, which is confirmed by the EMD and AHT-EMD metrics. In some cases, the difference is substantial, for example, in the CFM and CVS logs in which the MAE and EMD are more than ten times lower than those obtained using DOT-PROD. On the contrary, DOT-PROD gets the worst results in most experiments except for BPI2012 W and P2P logs, where the difference between the methods is minimal, possibly because there is no distinctive element between the activities in these logs.

Regarding the clustering metrics, the ACT2V+WR and DOT PROD+ACT2V+WR methods obtain better results on average. Regarding CAL-HARB, DOT-PROD+ACT2V+WR got the highest values in 5 of 9 logs. On the contrary, no technique predominates concerning the SILHOUETTE metric. These results indicate that the three methods tend to create defined clusters; however, the groups of activities generated by ACT2V+WR and DOT PROD+ACT2V+WR are denser and well-separated. The results indicate that the union of text and entity embedding techniques produce more precise representations of the activities in the process, which increases the precision of time generation by generative models.

4.5. Experiment 3: Perfect recall evaluation

4.5.1. Setup

This experiment aims to assess the impact of the stochastic process model in phase 1 and the inter-arrival generation model in phase 2, used by the Deep Simulator, on the accuracy of the Deep Learning models used in phase 3. Additionally, the experiment complements the findings of Experiment 2 on different embedding methods and their effect on the accuracy of Deep Learning models. To achieve this, we replicate the actions of a stochastic process model with perfect recall by providing the generative models with sequences of activities and original start times from the test partition of the log, thereby eliminating potential impacts of the stochastic process model and inter-arrival generation model on the generative models' performance.

The baseline for this experiment is the DSIM model discovered in Experiment 1, where models for phases 1, 2, and 3 were discovered, and DOT-PROD was used as the default embedding method. The three embedding models described in Section 3.4, DOT-PROD, ACT2V+WR, and DOT-PROD+ACT2V+WR, were used in conjunction with the "Perfect Recall" approach for phases 1 and 2. Similarly to Experiment 1, a hold-out method with a temporal split criterion was used to divide the logs into two folds: 80% for training validation and 20% for testing. Five logs were generated per retained model, and the mean MAE and EMD measures were evaluated across five runs.

4.5.2. Results

Table 5 presents the accuracy results for the four embedding approaches, measured in time metrics, with lower values indicating better performance. As anticipated, the "perfect recall" approaches yielded the highest results regarding EMR and MAE. This outcome suggests that when both the process model and the instance generation model have perfect recall capabilities, the overall precision of the model can significantly improve. These findings also highlight the potential of exploring other types of models in future work, such as ILP for process model discovery or Auto-Arima or Holt-Winters for inter-arrival generation. However, it is essential to note that the decision to use a BPMN model as the base also considers the model's interpretability, not just its precision. A model that captures all possible states under certain conditions can be challenging to interpret. Thus, the choice between different techniques depends on the specific purpose for which the simulation model is intended. On the other hand, when solely comparing the outcomes of the embedding methods within this setting, the methods that incorporate ACT2V demonstrate superior performance, particularly in relation to the EMD metric. Specifically, eight out of nine logs achieve the best results when ACT2V is employed.

4.6. Experiment 4: Accuracy of what-if generated models

In this experiment, we compare DSIM's ability to simulate a process after a change (what-if analysis). We consider two scenarios. In the first one, we assess DSIM's ability to capture variations in the inter-arrival time between cases (a.k.a. *arrival*

Table 4
Embedding methods times and clustering results.

Metric Type	Mertric	Embedding Method	LARGE				SMALL					Total general
			R		S		R			S		
			BPI12W	BPI17W	CFM	CVS	ACR	INS	MP	CFS	P2P	
Times	MAE	DOT-PROD	***647970***	642761	813012	15858463	273958	3852547	256823	21093	***2067743***	2714930
		ACT2V+WR	652559	699242	101420	557997	526788	55467873	***120482***	***9905***	2198448	6703857
		DOT-PROD+ACT2V+WR	653252	***524604***	***72854***	***324314***	***210749***	***3059536***	155291	22922	2265804	***809925***
	EMD	DOT-PROD	***181.641***	179.457	213.119	4876.941	77.884	1033.574	67.916	6.097	***576.809***	772.483
		ACT2V+WR	182.564	195.771	23.797	133.245	142.088	15584.409	***37.624***	***2.575***	619.556	167.153
		DOT-PROD+ACT2V+WR	181.934	***145.992***	***20.006***	***90.549***	***69.811***	***883.573***	41.384	6.103	626.121	***147.738***
	AHT-EMD	DOT-PROD	80.360	122.643	***503.846***	913.850	249.687	889.247	168.973	245.227	1289.636	446.778
		ACT2V+WR	***78.857***	128.444	599.630	98.428	***197.148***	8962.225	148.749	***239.089***	1293.566	347.989
		DOT-PROD+ACT2V+WR	84.355	***91.543***	607.839	***35.801***	268.602	***780.998***	***114.608***	246.938	***1280.147***	***341.229***
Clustering	CAL-HARB	DOT-PROD	8.570	4.161	***63.924***	22.350	7.719	9.372	27.890	***46.536***	***25.921***	24.049
		ACT2V+WR	***469.380***	226.079	33.988	8.219	6.661	20.224	56.352	18.246	17.396	***95.172***
		DOT-PROD+ACT2V+WR	127.506	***304.947***	37.324	***24.720***	***9.646***	***25.043***	***61.092***	21.333	23.303	70.546
	SILHOUETTE	DOT-PROD	0.433	0.283	***0.763***	***0.562***	0.313	0.391	0.551	***0.736***	***0.618***	0.517
		ACT2V+WR	***0.569***	***0.438***	0.593	0.438	0.416	0.561	0.702	0.506	0.463	0.521
		DOT-PROD+ACT2V+WR	0.533	0.405	0.703	0.406	***0.531***	***0.587***	***0.714***	0.656	0.549	***0.565***

Table 5
Perfect recall evaluation results (lower values are better).

Metric	Method	LARGE				SMALL				
		R		S		R			S	
		BPI12W	BPI17W	CFM	CVS	ACR	INS	MP	CFS	P2P
EMD	DOT-PROD	0.0200975	0.0345840	0.2390866	0.0000043	0.2051057	0.1420967	***0.0504794***	0.2667485	***0.1328978***
	PR+DOT-PROD	***0.0101782***	0.0397252	0.2001032	0.0000024	0.2289641	0.1158749	0.0991057	0.1892260	0.1695561
	PR+ACT2V+WR	0.0140482	***0.0227115***	***0.1771678***	***0.0000002***	0.1871419	***0.0807880***	0.0821350	0.2161905	0.1594830
	PR+DOT-PROD+ACT2V+WR	0.0285490	0.0286424	0.2034523	0.0000052	***0.1609440***	0.1000719	0.1047912	***0.1766676***	0.1631321
MAE	DOT-PROD	548813	418422	***8441***	158902	***165411***	1190019	157453	24326	***1836863***
	PR+DOT-PROD	316651	434368	48681	106969	187091	1289121	222449	***15158***	1954229
	PR+ACT2V+WR	***310044***	426786	77338	186057	166240	2686535	***127621***	24889	2369890
	PR+DOT-PROD+ACT2V+WR	321348	***395033***	75151	***93161***	186000	***1056514***	703745	42832	2017980

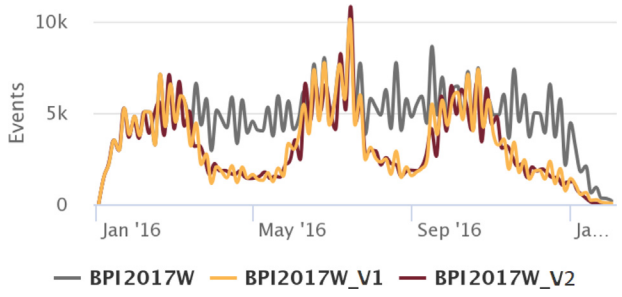

Fig. 8. Original vs modified case creations.

intensity), specifically alternations between periods of lower arrival intensity and periods of higher intensity. In the second experiment, we evaluate the ability of DSIM (vs baselines) to estimate the impact of adding a never-before-observed activity to a process. This scenario is challenging for the DSIM and the DL models because these models need to infer the temporal behavior of the new activity using their embedding layers.

4.6.1. Setup scenario 1

We create two modified versions of the three largest logs (BPI12 W, BPI17 W, and CVS). The first version captures a periodic reduction in the arrival intensity of cases. For this purpose, we divide each log into six batches of the same number of cases, and then two alternating groups of three batches each are created. The first group consists of batches 1, 3, and 5. The batches in this group are left unaltered. These groups represent periods of high arrival intensity. The second group comprises batches 2, 4, and 6. This group is used to emulate periods of low arrival intensity. To do so, we reduce the case arrival rate in these batches by 1/3, by randomly eliminating two out of three cases.

The above altered version of the log captures a situation where the arrival intensity varies, but the waiting times within the cases remain the same. In general, when the arrival intensity goes down, the waiting times should go down. Therefore, the second version of the log captures decreases in waiting times associated with decreases in arrival rates. Accordingly, we take the first altered log, and we reduce the waiting times in group 2 (batches 2, 4, and 6) by 30%. For illustration, Fig. 8 sketches the modifications made to the BPI17 W log.

After altering the logs as presented above, we train and evaluate the DSIM and SIMOD models as in Experiment 1 (see Fig. 6). Since the main goal here in assessing the ability of the evaluated techniques to model the temporal dynamics of case arrivals, we analyze the MAE and EMD metrics to measure the error in cycle times. We also report the Dynamic Time Warping (DTW) distance between the time series of the number of cases generated per hour of the day, between the testing partition and the logs generated.

4.6.2. Setup scenario 2

For each of the synthetic logs (CVS and CFM), we select a random activity A and eliminate all its occurrences from the log. We then train a DSIM simulation model using this modified log (cf. left-hand side of Fig. 9) and the DOT-PROD+ACT2V+WR embedding approach. This method was chosen since it performed better for activities representation (see the second experiment in Section 4.4). Next, we generate synthetic data regarding sequences of activities and resources, including the removed activities (cf. Section 3.4). Using these synthetic samples, we update the embedded dimensions to include the activity A (without modifying the embedding of the remaining activities). We then inject the updated embedding into the previously trained DSIM model (cf. right-hand side of Fig. 9). We calculate the errors of the DSIM model of the "as-is" process (before a change) and the DSIM model of the "what-if" process (after adding an activity). We measure the error using three metrics MAE, RMSE and SMAPE to confirm that the results do not depend on the chosen metric.

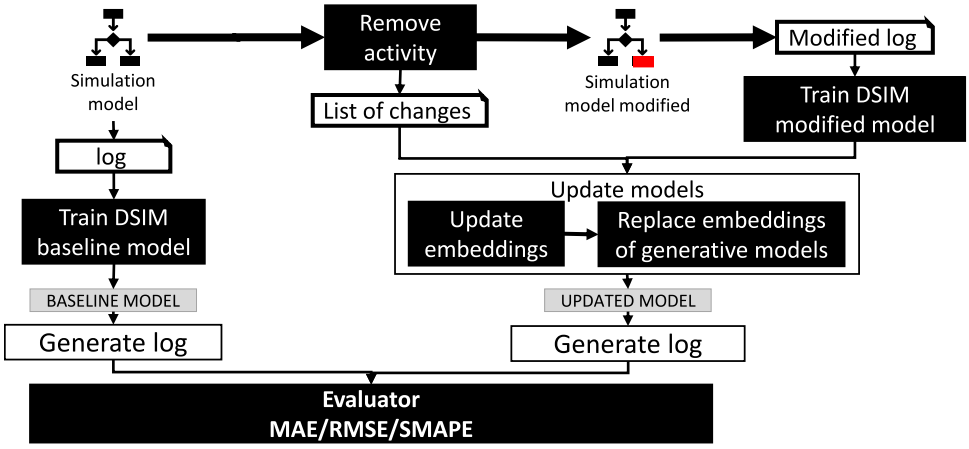

Fig. 9. Pipeline of scenario 2.

Table 6
Results of scenarios 1 and 2.

	Log	MAE		EMD		DTW	
		SIMOD	DSIM	SIMOD	DSIM	SIMOD	DSIM
Scenario 1	*Version 1*						
	BPI17W	971151	***417572***	***0.02222***	0.03593	***3185***	3647
	BPI12W	660211	***534341***	0.11295	***0.04853***	515	***458***
	CVS	1489252	***467572***	0.03213	***0.00001***	3380	***849***
	Version 2						
	BPI17W	895524	***290980***	0.06438	***0.03218***	4528	***3431***
	BPI12W	550266	***524995***	0.25888	***0.22003***	726	***507***
	CVS	540112	***246159***	0.15674	***0.05708***	2453	***1967***
Scenario 2	Log	MAE		RMSE		SMAPE	
		AS-IS	WHAT-IF	AS-IS	WHAT-IF	AS-IS	WHAT-IF
	CFM	***7155***	21258	***22006***	38757	***0.15629***	0.33254
	CVS	***283061***	347773	***357717***	464215	***0.31972***	0.35168

4.6.3. Results scenario 1

Table 6 presents the MAE, EMD and DTW results, in which DSIM has a lower error in cycle times in all cases. In version 2 the MAE logs are considerably reduced compared to those of the version 1. We can explain this result because the DL models in charge of predicting waiting and processing times consider inter-case attributes that capture the workload of the process, allowing their adjustment to workload variations. Regarding the results of EMD and DTW, both metrics follow the same trends. In most cases, DSIM obtains the best results. This trend is more evident in version 2, in which DSIM gets better results in all cases. These results indicate that Prophet and DL models are more effective than the baselines at capturing the temporal variations in waiting times due to a decrease in the arrival intensity. This observation should again be tempered by the threats to validity acknowledged above.

4.6.4. Results scenario 2

Table 6 presents the MAE, RSME, and SMAPE grouped for each log, both for the AS-IS process simulation model vs. the model derived after the addition of the activity (what-if model).

Despite the what-if model has higher MAE than the baseline models in both event logs, the values are not far from those obtained in the model trained with the "as-is" version of the log. This is more evident for the CVS model, which is the larger log used in this experiment. We report a notable improvement in decreasing the error of generated what-if models when adjusting the activities representation with specialized embedding models. For instance, the error of the SMAPE metric decreased from 184% to 35% in comparison with previous results obtained by using a dot product embedding [11]. These results suggest that embedded dimensions incorporated in DSIM can predict the presence of activities that were not present in the training set, and it can estimate until a certain point their temporal behavior without retraining the generative models. This observation suggests that DSIM could be extended with more sophisticated embedding techniques (e.g. transformer models or including other attributes like processing times) to better capture the temporal dynamics of previously unobserved activities (by analogy to activities that have been observed in similar contexts).

5. Conclusion

This paper presented a method, namely DeepSimulator, to learn BPS models from event logs based on process mining and DL techniques. The design of DeepSimulator draws upon the observation that DDS methods (based on process mining) do not capture delays between activities caused by factors other than resource contention (e.g. fatigue, batching, inter-process dependencies). In contrast, DL techniques can learn temporal patterns without assuming these patterns stem only from resource contention. Accordingly, DeepSimulator discovers a stochastic process model from a log using process mining, and then uses a DL model to add timestamps to the events produced by the stochastic model. The stochastic model can be modified (activities may be added/removed, branching probabilities may be altered), thus enabling some forms of what-if analysis.

The paper reported on an empirical comparison of the proposed technique with respect to: (i) its ability to replicate the observed as-is behavior; and (ii) its ability to estimate the impact of changes (what-if settings). The evaluation in the "as-is"

setting shows that the DeepSimulator method outperforms the baselines (one DDS and two DL methods). This improvement in the generated models is especially notable when using embedding methods that take into account the context of each activity relative to other activities, and the importance of each resource relative to each activity (cf. the ACT2V+WR method and the combined method).

The evaluation in the what-if analysis setting shows that DeepSimulator can better estimate the impact of changes in the arrival rate of new cases (the demand) in settings where such changes have been previously observed in the data. In addition, the use of embeddings allows DeepSimulator to make accurate predictions of temporal behavior for previously unobserved activities.

The multimodel structure proposed in this paper opens up avenues for extending each phase of the method in various directions for future research. In the context of phase one, more advanced techniques for discovering stochastic process models could be explored to enhance the global accuracy of the approach. For instance, investigating the impact of different process model discovery methods on the accuracy of the generated simulation models is a potential research direction. Another direction is to explore alternative approaches to model the branching probabilities in the decision gateways of the stochastic process model. Possible approaches include Laplace smoothing or approaches that assign probabilities based on observed traversal frequencies when a sufficient confidence level can be guaranteed, and fall back to distributing the probabilities equally across the conditional flows when the confidence level is insufficient. Furthermore, the possibility of using data-driven decision rules, instead of probabilities, could be investigated.

Regarding phase two of the approach, future research could involve exploring other time series forecasting techniques, such as Auto-Arima or Holt-Winters, or employing a separate deep learning generative model specialized in inter-arrival time generation.

Lastly, in phase three of the approach, there is potential for extending the investigation of embedding methods to include more sophisticated features, such as attributes related to the processing times of each activity, to capture the temporal dynamics of previously unobserved activities more accurately. Similarly, the challenge of managing what-if scenarios in a process simulation is closely intertwined with the problem of adapting a predictive model to handle concept drifts. Concept drift refers to a situation where the relationship between the input data and the target variable changes over time, thereby impacting the accuracy of a trained predictive model. In both scenarios, i.e., what-if analysis and concept drift, the predictive model must adapt to dynamics not part of the training data. A potential direction for future research could involve investigating techniques that have been previously evaluated for managing concept drift in predictive process monitoring, such as incremental learning [35], adapted within the context of process simulation.

Another avenue for future work is to extend the approach to generate events with a Resource attribute. This entails extending the proposed method to support a broader range of changes, such as adding or removing resources. Yet another future work avenue is to validate the proposed method via case studies, to complement the post-mortem evaluation reported in this paper.

Reproducibility The source code is available at https://github.com/AdaptiveBProcess/DeepSimulator.git. The datasets, models, and evaluation results can be found at https://doi.org/10.5281/zenodo.5734443. The datasets, models, and evaluation results for additional experiments can be found at https://doi.org/10.5281/zenodo.7443872.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Marlon Dumas reports financial support was provided by European Research Council (PIX project).

References

[1] M. Dumas, M. La Rosa, J. Mendling, H.A. Reijers, Fundamentals of Business Process Management, second ed., Springer, 2018.
[2] W.M.P. van der Aalst, Business process simulation survival guide, in: J. vom Brocke, M. Rosemann (Eds.), Handbook on Business Process Management 1: Introduction, Methods, and Information Systems, Springer, 2015, pp. 337–370.
[3] N. Martin, B. Depaire, A. Caris, The use of process mining in business process simulation model construction, Bus. Inf. Syst. Eng. 58 (1) (2016) 73–87.
[4] M. Camargo, M. Dumas, O. González-Rojas, Automated discovery of business process simulation models from event logs, Decis. Support Syst. 134 (2020) 113284.
[5] 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.
[6] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: Proceedings of CAiSE 2017, in: LNCS, Springer, 2017, pp. 477–492.
[7] J. Evermann, J.R. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decis. Support Syst. 100 (2017) 129–140.
[8] M. Camargo, M. Dumas, O. González-Rojas, Learning accurate LSTM models of business processes, in: Proceedings of BPM 2019, in: LNCS, Springer, 2019, pp. 286–302.
[9] M. Camargo, M. Dumas, O. González-Rojas, Discovering generative models from event logs: data-driven simulation vs deep learning, PeerJ Comput. Sci. 7 (2021) e577.
[10] S.J.J. Leemans, W.M.P. van der Aalst, T. Brockhoffb, A. Polyvyanyy, Stochastic process mining: Earth movers' stochastic conformance, Inform. Syst. 102 (2021) 101724.
[11] M. Camargo, M. Dumas, O. González-Rojas, Learning accurate business process simulation models from event logs via automated process discovery and deep learning, in: Proceedings of CAiSE 2022, Vol. 13295, in: LNCS, Springer, 2022, pp. 55–71.
[12] 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: Proceedings of BPM Workshops 2007, in: LNBIP, Springer, 2008, pp. 66–77.
[13] A. Rozinat, R.S. Mans, W.M.P. van der Aalst, Discovering simulation models, Inform. Syst. 34 (3) (2009) 305–327.
[14] I. Khodyrev, S. Popova, Discrete modeling and simulation of business processes using event logs, Procedia Comput. Sci. 29 (2014) 322–331.
[15] 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: Proceedings of BIS 2020, in: LNBIP, 2020, pp. 249–263.
[16] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[17] L. Lin, L. Wen, J. Wang, MM-pred: A deep predictive model for multi-attribute event sequence, in: Proceedings of SIAM 2019, Society for Industrial and Applied Mathematics, 2019, pp. 118–126.
[18] N. Tax, I. Teinemaa, S.J. van Zelst, An interdisciplinary comparison of sequence modeling methods for next-element prediction, Softw. Syst. Model 19 (6) (2020) 1345–1365.
[19] F. Taymouri, M. La Rosa, S. Erfani, Z.D. Bozorgi, I. Verenich, Predictive business process monitoring via generative adversarial nets: The case of next event prediction, in: Proceedings of BPM 2020, in: LNCS, Springer, 2020, pp. 237–256.
[20] N. Lavrač, V. Podpečan, M. Robnik-Šikonja, Representation Learning: Propositionalization and Embeddings, Springer, 2021.
[21] P. Pfeiffer, Business process representation learning, 2022, URL: http://ceur-ws.or.
[22] P.D. Koninck, S. vanden Broucke, J.D. Weerdt, Act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes, in: Proceedings of BPM 2018, in: LNCS, Springer, 2018, pp. 305–321.
[23] A. Al-Jebrni, H. Cai, L. Jiang, Predicting the next process event using convolutional neural networks, in: Proceedings of PIC 2018, IEEE, 2018, pp. 332–338.
[24] N. Di Mauro, A. Appice, T.M.A. Basile, Activity prediction of business process instances with inception CNN models, in: Proceedings of AI*IA 2019, in: LNCS, Springer, 2019, pp. 348–361.

[25] A. Augusto, R. Conforti, M. Dumas, M.L. Rosa, A. Polyvyanyy, Split miner: automated discovery of accurate and simple business process models from event logs, Knowl. Inf. Syst. 59 (2) (2019) 251–284.
[26] D. Reißner, A. Armas-Cervantes, R. Conforti, M. Dumas, D. Fahland, M. La Rosa, Scalable alignment of process models and event logs: An approach based on automata and S-components, Inform. Syst. 94 (2020) 101561.
[27] C. Favre, H. Völzer, The difficulty of replacing an inclusive OR-join, in: Proceedings of BPM 2012, in: LNCS, Springer, 2012, pp. 156–171.
[28] S.J. Taylor, B. Letham, Forecasting at scale, Am. Stat. 72 (1) (2018) 37–45.
[29] E. Rama-Maneiro, J.C. Vidal, M. Lama, Deep learning for predictive business process monitoring: Review and benchmark, 2021, arXiv:2009.13251.
[30] M. Laguna, J. Marklund, Business Process Modeling, Simulation and Design, CRC Press, 2018.
[31] M. Song, W.M.P. van der Aalst, Towards comprehensive support for organizational mining, Decis. Support Syst. 46 (1) (2008) 300–317.
[32] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: Proceedings of ICLR Workshops 2013, 2013, pp. 1–12.
[33] H.W. Kuhn, The hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1955) 83–97.
[34] D. Chapela-Campa, M. Dumas, Modeling extraneous activity delays in business process simulation, 2022, URL: https://arxiv.org/abs/2206.14051.
[35] W. Rizzi, C. Di Francescomarino, C. Ghidini, F.M. Maggi, How do I update my model? On the resilience of predictive process monitoring models to change, Knowl. Inf. Syst. 64 (5) (2022) 1385–1416.