Deep Learning for Predictive Business Process Monitoring: Review and Benchmark

Efrén Rama-Maneiro, Juan C. Vidal, and Manuel Lama

Abstract—Predictive monitoring of business processes is concerned with the prediction of ongoing cases on a business process. Lately, the popularity of deep learning techniques has propitiated an ever-growing set of approaches focused on predictive monitoring based on these techniques. However, the high disparity of event logs and experimental setups used to evaluate these approaches makes it especially difficult to make a fair comparison. Furthermore, it also difficults the selection of the most suitable approach to solve a specific problem. In this article, we provide both a systematic literature review of approaches that use deep learning to tackle the predictive monitoring tasks. In addition, we performed an exhaustive experimental evaluation of 10 different approaches over 12 publicly available event logs.

Index Terms—Process mining, business process monitoring, neural networks, systematic literature review, deep learning

1 Introduction

A business process is a series of tasks performed by a set of resources, which can be human, software or hardware, to achieve a goal [1]. The execution of a business process is recorded in an event log [2], storing for each event, at least, its case identifier, the activity performed, and its timestamp. Optionally, case attributes which are shared by events of the same case, or event attributes, specific to each event, are also recorded.

Process mining is the discipline concerned with the analysis of these logs, tackling it from different perspectives such as discovering the underlying process model from the log, checking that the executions registered in the log are conformant with the process model, or extracting or inferring analytics that enhance the description of what has happened in the process executions [2]. In this paper, we focus on predictive monitoring [3], a process mining technique which aims to predict how an ongoing process case is going to unfold using the information of the event log.

Making predictions is useful for foreseeing problems before they occur, helping reallocate resources before they are wasted, or to make recommendations. Taking as an example the event log of Table 1, we might want to forecast which would be the next medical trial applied to a patient but also, the remaining time until a patient is discharged, or whether a patient is going to be intervened or not. The learning phase of a classifier trained to forecast these problems is performed on partial traces (prefixes). A wide array of traditional machine learning techniques has been applied to predictive monitoring such as Decision Trees [4], Factorization Machines [5], Probabilistic Finite Automatons [6], [7], or Support Vector Machines [8], [9].

More recently, deep neural networks have gained a lot of attention in process mining due to their success in fields such as computer vision [10], [11] or natural language processing [12], [13]. First, Recurrent Neural Networks (RNNs) were found to be a good fit to approach predictive monitoring [14], [15] because of the sequential nature of event logs. However, nowadays, many other deep learning approaches to predictive monitoring can be found in the literature [16], [17], [18], [19], improving in most cases classical machine learning approaches [20]. Unfortunately, the high number of neural network architectures, the different ways of encoding the partial traces and events, the number of predictive tasks available, and, sometimes, the difficulty of quantifying the differences and contributions between the works may complicate the task of defining what has already been done, what can be researched and future research directions. Furthermore, and more importantly, approaches are applied to a reduced number of datasets and with significantly different experimental setups. This makes it difficult to new researchers to compare their new approach with existent state-of-the-art works.

To solve these issues, in this paper, we identify the most relevant deep learning approaches for predictive monitoring.

- *Efrén Rama-Maneiro and Manuel Lama are with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15705 Santiago de Compostela, Spain. E-mail: {efren.rama.maneiro, manuel.lama}@usc.es.*
- *Juan C. Vidal is with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15705 Santiago de Compostela, Spain, and also with Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, 15705 Galicia Santiago de Compostela, Spain. E-mail: juan.vidal@usc.es.*

Manuscript received 23 September 2020; revised 1 December 2021; accepted 22 December 2021. Date of publication 31 December 2021; date of current version 6 February 2023.
This work was support in part by Consellería de Educación, Universidade e Formación Profesional under Grant 2019-2022 ED431G-2019/04, in part by European Regional Development Fund (ERDF), which acknowledges the CiTIUS - Centro Singular de Investigación en Tecnoloxías Intelixentes da Universidade de Santiago de Compostela as a Research Center of the Galician University System, and in part by the Spanish Ministry of Science and Innovation under Grants PDC2021-121072-C21 and PID2020-112623GB-I00. E. Rama-Maneiro is supported by the Spanish Ministry of Education, under the FPU national plan under Grant FPU18/05687.
(Corresponding author: Efrén Rama-Maneiro.)
Recommended for acceptance by C. Miao.
Digital Object Identifier no. 10.1109/TSC.2021.3139807

TABLE 1
Excerpt of a Business Event Log About the Medial Trials Performed in a Hospital

Case ID	Activity	Timestamp	Resource
Case2118	Inclusion	14-01-2010 07:52:50	Peter
Case2118	TAC	09-02-2010 13:01:11	Joseph
Case2118	Blood Analysis	17-02-2010 07:44:53	Joseph
Case2118	Intervention	17-02-2020 07:44:59	Joseph
Case2118	Discharge patient	18-02-2020 09:00:10	Joseph
Case2088	Inclusion	04-02-2010 08:37:45	Peter
Case2088	Pneumothorax scan	04-02-2010 09:01:28	Peter
Case2088	TAC	04-02-2010 09:01:35	Peter
Case2088	Blood Analysis	16-03-2010 13:08:40	Peter
Case2088	Discharge patient	31-03-2010 11:08:53	Dio

Then, using these approaches as a starting point, we select those that have their implementation available and perform an experimentation and a statistical comparison in a fair setting using a wide number of event logs. Then, we use the results to highlight the most relevant differences between them. We have made public the source code[1], trained models and results of the experimentation in the hope that it serves as a starting point for new researchers in predictive monitoring.

The rest of the paper is structured as follows: Section 2 formally defines the most common predictive monitoring problems. Section 3 highlights the difference between this survey and other predictive monitoring surveys. Section 4 highlights the search methodology to perform the systematic literature review. Section 5 shows the classification and taxonomy of the retrieved studies, explaining its main achievements. Section 6 shows the experimental setup, results, and discussion of the benchmarking of the available approaches. Finally, Section 6 concludes the paper highlighting future lines of work.

2 Definitions

Given an event prefix of a running case, predictive monitoring is concerned with forecasting how different aspects of the next event or sequence of events will unfold until the end of the case. Formally, let A, T, O be the set all possible activities, timestamps, and outcomes. In order to define the attributes of a log and their bindings, we will first introduce the concept of datatype. Let S be a set of datatype names and $\Psi : S \rightarrow V_S$ a function that maps each datatype to the set V_S of values in an algebra. Let D be the set of attribute names of the log and $\lambda : D \rightarrow S$ a function that maps each attribute to its datatype. We define an attribute binding as a function $\Upsilon : D \rightarrow V_S$ such that if $\Upsilon(d) = v$ for $d \in D$ then $\lambda(d) = s$, $\Psi(s) = V_s$, and $v \in V_s$.

Let also $e = (a, t, (d_1, v_1), \ldots, (d_n, v_n))$ be an event where $a \in A$, $t \in T$, $d_n \in D$, and $v_n \in V_S$. Let $hd^k(\sigma)$ be an event prefix such as $hd^k(\sigma) = \langle e_1, \ldots, e_k \rangle$, e'_{k+1} be a predicted $k+1$ event by a function Ω, π_T a projection function that maps an event to its timestamp $\pi_T(e) = t \mid t \in T$, and let $\oplus$ be the concatenation operator between two traces. Finally, to simplify the notation, the superscript′ indicates that the annotated symbol is a prediction. Depending on the predictive task at hand, we can define the following functions Ω:

Definition 1. *The next activity prediction problem can be defined as $\Omega_A(hd^k(\sigma)) = a'$, where $a' \in A$ is the predicted activity.*

Definition 2. *The next attribute prediction problem can be defined as $\Omega_{(d,V_S)}(hd^k(\sigma)) = v'$, where $d \in D$, $\Upsilon(d) = v'$, and $v' \in V_S$ reflect the predicted attribute.*

Definition 3. *The next timestamp prediction problem can be defined as $\Omega_T(hd^k(\sigma)) = \pi_T(e_k) + t'$, where $t' \in \mathbb{R}$ is the predicted duration, and e_k denotes the last event of the prefix.*

Definition 4. *The outcome of an event prefix can be predicted as $\Omega_O(hd^k(\sigma)) = o'$, where $o' \in O$ is the predicted outcome.*

The activity suffix prediction problem can be defined in two different ways: as the recursive prediction of A, i.e, using the newly predicted activities as the inputs for the next prediction until the end of the case ("[EOC]") is reached (Definition 5) or directly predicting the activity suffix (Definition 6).

Definition 5. *An activity suffix can be recursively predicted as $\Omega_{SA} = \langle \Omega_A(\sigma') = a' | \sigma' = hd^k(\sigma) \oplus \langle e'_{k+1}, \ldots, e'_{i-1} \rangle \rangle$ while $a' \; / = [EOC]$ being $a' \in A$.*

Definition 6. *An activity suffix can be directly predicted as $\Omega_{SA}(hd^k(\sigma)) = \langle a'_{k+1} \ldots a'_n \rangle$, where $a'_i \in A \; \forall i \in [k+1, n]$.*

Functions for predicting an attribute suffix and remaining time can be defined analogously.

Definition 7. *An attribute suffix can be recursively predicted as $\Omega_{Sd} = \langle \Omega_{(d,V_S)}(\sigma') = v' | \sigma' = hd^k(\sigma) \oplus \langle e'_{k+1}, \ldots, e'_{i-1} \rangle \rangle$ while $a' \neq [EOC]$, being $d \in D$, $\Upsilon(d) = v'$, and $v' \in V_S$*

Definition 8. *An attribute suffix can be directly predicted as $\Omega_{Sd}(hd^k(\sigma)) = \langle v'_{k+1} \ldots v'_n \rangle$, where $v'_i \in V_S \forall i \in [k+1, n]$.*

Definition 9. *Let θ be the sequence of predicted next timestamps such as $\theta = \langle \Omega_T(\sigma') = t' + \pi_T(e'_{i-1}) | \sigma' = hd^k(\sigma) \oplus \langle e'_{k+1}, \ldots, e'_{i-1} \rangle \rangle$ while $a' \neq [EOC]$, then the remaining time can be calculated as $\Omega_{RT}(hd^k(\sigma)) = \sum_{i=k}^{n} t'_i$, where $\pi_T(e'_{i-1})$ denotes the timestamp of the last event of the prefix, $t' \in \mathbb{R}$ is the predicted next timestamp, and $a' \in A$.*

Definition 10. *The remaining time can be directly calculated as $\Omega_{RT}(hd^k(\sigma)) = rt'$, where $rt' \in \mathbb{R}$, and it is the predicted remaining time for the whole prefix.*

3 Related Work

Several authors have addressed the problem of reviewing the current state of the art in predictive monitoring. In [21], the authors review a set of studies and classify them in process-aware methods and non-process aware methods, depending on whether they require a process model as an input or not, and also whether they treat predictive monitoring as a regression problem or as a classification problem. In [22], the authors perform a systematic literature review with the aim of helping companies to select their best suited predictive monitoring framework according to multiple dimensions such as the predictions made by the approach,

1. The code link is available in the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2021.3139807.

TABLE 2
Morphological Box of Every Possibility for Each Characteristic

Network type	Sequence encoding	Event Encoding	Input data	Time features	Prediction
Long Short Term Memory (LSTM)	Continuous (CONT)	Embedding (EMB)	Activity (ACT)	Time since start of the case (TSSC)	Activity (ACT)
Autoencoder (AE)	Prefixes padded (PRFX)	One-hot encoding (OH)	Linear Temporal Logic (LTL)	Time since previous event (TSP)	Activity Suffix (SFX)
Deep feedforward network (DFNN)	N-gram (N-GRAM)	Pretrained embedding (P-EMB)	Attributes (ATTR)	Day of week (WK)	Next timestamp (NT)
Differentiable Neural Computer (DNC)	Single event (SE)	Frequency based (FB)	Clustered attributes (C-ATTR)	Time since midnight (TMD)	Remaining time (RT)
Gated Recurrent Unit (GRU)	Timed state sample (TSS)		Process model (PM)	Decayed time since previous place activation (DTPA)	Outcome (OUT)
Bidirectional LSTM (B-LSTM)			Role (R)		Resource (RES)
Transformer (TRANS)					Role (ROLE)
					Activity prediction in checkpoints (ACT [CP])
					Role suffix (RLSFX)
					Attribute suffix (ATTRSFX)

the domain where it is applied, the algorithm developed, and the input used. In [23], the authors perform a study of three deep learning approaches for predictive monitoring, taking into account different dimensions, such as the encoding scheme or the prediction target.

Regarding to benchmarking studies, in [24], the authors compare the performance of 20 different supervised learning classification techniques over 6 different event logs to predict the next event in a business process. They conclude that the best machine learning classifier, in terms of accuracy, is the credal decision tree.

In [20], a comparison of the following families of techniques for the next activity prediction problem is made: *(i)* recurrent neural networks, *(ii)* markov models, *(iii)* grammar induction techniques, which learn a set of production rules that describe a language (in this case, an event log), *(iv)* process discovery-based techniques, and *(v)* automata based prediction techniques.

In [25], the authors focus their study on the prediction of the remaining time in a business process, comparing several bucketing, encoding, and supervised learning techniques in terms of the Mean Absolute Error of the predictions. They also compare the supervised learning techniques with 3 different process-aware methods. Their conclusion is that LSTMs outperform other approaches for remaining time prediction. In [26], the authors provide a systematic literature review of outcome-oriented predictive monitoring and compare the performance of four different machine learning techniques (random forest, logistic regression, support vector machines and extreme gradient boosting) with multiple encodings to predict the outcome of a business process. In [27], the authors perform a similar analysis for outcome prediction but comparing another set of machine learning techniques (random forest, support vector machines, deep feedforward networks, and LSTMs).

In [28] they perform a similar review to this one, but they do not provide any fair benchmarking of the approaches. In [29] they perform an experimentation to compare CNN, LSTM, and LSTM with an attention mechanism to predict the outcome of a running case. They conclude that CNNs perform similar to LSTMs, and, since the former is computationally faster, it should be the architecture to use. In [30] they introduce key-value-predict attention networks to improve the performance of LSTMs, and introduce the usage of process data properties as additional features. However, their experimentation is limited to a few approaches and not every approach is tested in every proposed dataset.

Only [23] and [28] have similar objectives as the review conducted in this paper. In [23], the analysis is limited to the first three deep learning approaches in predictive monitoring, but without any benchmarking, so the results reported are taken directly from the papers. In [28], even though a more thoroughly review is provided, the analysis also lacks any benchmarking between the approaches.

Other authors include Deep Learning approaches when comparing their solution with the state-of-the-art in predictive monitoring. However, they limit their comparison to evaluating LSTM [20], [25], [27], and GRU [20] alongside other machine learning algorithms. No other review study benchmarks the original deep learning implementations of the state-of-the-art approaches under controlled conditions to provide a fair comparison between them.

4 Analysis and Classification

We performed an analysis and classification of the predictive monitoring that use deep learning[2]. The results of this classification are available in Table 3. Table 2 reflects every possibility for each of the classification criteria.

- *Input data*: data from the event logs used to train the predictive model.
- *Predictions*: the elements of the events that the neural network is trained to forecast.

2. The details of the search methodology are available in the supplementary material, available online.

TABLE 3
Collected Studies for Predictive Monitoring That Use Deep Learning

Author, Year	Reference	Network type	Sequence encoding	Event encoding	Input data	Time features	Prediction
Evermann *et al.*, 2017	[14]	LSTM	CONT	EMB	ACT	-	ACT
Francescomarino *et al.*, 2017	[15]	LSTM	PRFX	OH	ACT, LTL	TSP, TSSC, WK, TMD	SFX
Mehdiyev *et al.*, 2017	[31]	AE + DFNN	NGRAM	-	ACT, ATTR	-	ACT
Tax *et al.*, 2017	[16]	LSTM	PRFX	OH	ACT	TSP, TSSC, WK, TMD	ACT, NT, SFX, RT
Navarin *et al.*, 2017	[32]	LSTM	PRFX	OH	ACT, ATTR	TSP, TSSC, WK, TMD	RT
Al-Jebrni *et al.*, 2018	[33]	CNN	CONT	EMB	ACT	-	ACT
Khan *et al.*, 2018	[17]	DNC	PRFX	OH	ACT	TSP, TSSC, WK, TMD	ACT, NT, RT, SFX
Metzger *et al.*, 2018	[34]	LSTM	PRFX	OH	ACT	-	OUT
Schönig *et al.* 2018	[35]	LSTM	CONT	OH	ACT, ATTR	-	ACT, RES
Mehdiyev *et al.*, 2018	[36]	AE + LR	NGRAM	-	ACT, ATTR	-	ACT
Camargo *et al.*, 2019	[37]	LSTM	PRFX	P-EMB	ACT, R	TSP	ACT, ROLE, NT, RT, SFX, RLSFX
Lin *et al.*, 2019	[38]	LSTM	CONT	EMB	ACT, ATTR	-	ACT, ATTR, SFX, ATTRSFX
Hinkka *et al.*, 2019	[39]	GRU	PRFX	OH	ACT, ATTR, C-ATTR	-	ACT [CP]
Theis *et al.*, 2019	[40]	DFNN	TSS	FB	ACT, PM, ATTR	DTPA	ACT
Wahid *et al.*, 2019	[41]	DFNN	SE	EMB	ACT, ATTR	TSP, TSSC	RT
Pasquadibisceglie *et al.*, 2019	[19]	CNN	PRFX	FB	ACT	TSSC	ACT
Wang *et al.*, 2019	[42]	B-LSTM	PRFX	OH	ACT, ATTR	-	OUT
Mauro *et al.*, 2019	[18]	CNN	PRFX	EMB	ACT	TSP	ACT
Folino *et al.*, 2019	[43]	LSTM	PRFX	OH, EMB	ACT, ATTR	-	OUT
Hinkka *et al.*, 2019	[44]	GRU/ LSTM	CONT	OH	ACT	-	OUT
Philipp *et al.*, 2020	[45]	TRANS	CONT	EMB	ACT	-	ACT

- *Neural Network Type*: type of neural network used such as feedforward, autoencoder, convolutional, recurrent or transformer.
- *Sequence encoding*: how event prefixes are converted into learnable tensors by the neural net.
- *Event encoding*: how each individual categorical and continuous variable is encoded in a feature vector.

4.1 Input Data

The selection of input data used in a neural network is one of the most crucial decisions to make when designing a predictive monitoring approach based on deep learning. In general, the most important information attributes in an event log, for the majority of prediction problems, would be the case identifier, the activities and the timestamp. If the prediction problem is an attribute or outcome, the required information will be dependent on the attribute in case.

Since most event logs used in the literature are real-life logs, these are prone to have noisy data. We refer to noisy data as a set of problems that includes missing data, duplicated tasks, inconsistent tasks, or other incorrect behavior [46]. Given that most logs used in predictive monitoring are real use cases, noisy logs are a prominent problem in process mining, even though most predictive monitoring approaches disregard this problem, or only try to minimize this problem by filtering traces using some condition. Missing data, as the absence of data values for a given variable in an observation [47], is the most paradigmatic case of noise in process mining. This problem can take multiple dimensions. On the worst case, a task could be missing from the event log, so no information would be recorded in that case. On the best case, an event-level attribute could be missing for a given event. For example, the resource that executes the activity of an event could not be present. In the latter case, imputation techniques could be used directly over the event log to help improve the prediction performance [48].

The time information available in the log requires some preprocessing to be exploited since it can not be used directly as inputs of the neural network. The most usual preprocessing applied to timestamps involves computing the difference between the timestamp of the current event and the timestamp of the previous event [15], [16], [17], [18], [32], [37], [41]. Another used feature involves computing the difference between the current event timestamp and the timestamp from the beginning of the case [15], [16], [17], [32], [41]. Other potentially useful features are the day of the week [15], [16], [17], [32] or the time until the midnight [15], [16], [17], [32], [41]. Additionally, the difference between the current timestamp and the timestamp of the last activation on a Petri net can be applied to a decay function which makes the calculated time decrease as the place is not activated [40].

As far as the attributes of the log are concerned, it is possible that not every attribute adds significant information to the predictive problem [31], [32], [35], [36], [38], [39], [40], [41], [42], [43]. For example, and returning to the example in Table 1, if the e-mail information is present as part of an additional trace-level attribute, it will not be useful to predict the next medical test. [38] trains a predictive model where an alignment weight vector learns the importance of each attribute. [39] clusters the attributes together using the x-means algorithm. The belonging to a cluster is added as an additional feature to the feature vector of each event. Furthermore, resources in an event log are often noisy since some resources could potentially appear only once in the whole event log, even though they adhere to an organizational scheme. To solve this problem, [37] groups resources depending on their activity execution profiles. Note that, as shown in Table 3 this is the only approach that distinguishes the resources from other attributes of the event log. As far as the approaches that use time features [15], [16], [17], [18], [19], [37], [40], [41], they face the problem of a high variability in the time between the events, so these time features may complicate the training phase.

4.2 Predictions

Regarding the prediction targets, there are multiple possibilities:

- *Activity* [14], [16], [17], [18], [19], [31], [33], [35], [36], [37], [38], [39], [40], [45]: the next activity of a running case.
- *Activity suffix* [15], [16], [17], [37], [38]: the sequence of activities given a running case.
- *Next timestamp* [16], [17], [37]: the difference between the next timestamp of an event and the timestamp of the current event.
- *Remaining time* [16], [17], [32], [37], [41]: the difference between the timestamp of the last case of a trace and the current timestamp of an event.
- *Outcome* [34], [42], [43], [44]: refers to the outcome of a given running case.
- *Attributes* [37], [38]: other event-level attributes present in the log. We include here the roles from [37] since they are derived from the resources of the log, which are, in turn, attributes of the log.
- *Attribute suffix* [37], [38]: the approaches that predict the activity suffix and use other attributes of the log as inputs must predict also the sequence of attributes since the inputs for these attributes in the predicted events would be missing otherwise.

The most common prediction problem is the prediction of the next activity given a partial event prefix. When this is the case, other prediction tasks are taken up as auxiliary tasks that may help improve the prediction performance [49]. However, when the prediction problem is the remaining time, and it is approached as a direct prediction, it is the only predictive task at hand. Note that no surveyed approach has tried the direct prediction of activity and attribute suffixes, because, contrary to the remaining time prediction, it has the disadvantage of assigning completely different labels to potentially similar suffixes.

As far as the outcome prediction problem is concerned, most studies assume that the log is fully labeled and, thus, the problem is treated in a supervised manner. The exception to this is [43], where they do not make such assumptions and, instead, consider the log as a partially labeled dataset. Thus, the problem is treated in a semi-supervised manner which has two stages: *(i)* an LSTM is trained on the full event log to predict the next activity, and *(ii)* another LSTM is initialized with the parameters learned by the previous model, and it is trained over the log that is labeled to predict the outcome of a running case.

When the objective is to predict the activity suffix from an event prefix, the next event must be sampled from the output probability distribution of the last neural network's prediction layer. A simple choice would be selecting the event with the highest probability in the vector [16]. However, this solution often makes the predictions to be very repetitive on the same top probability activity when the process is complex enough. Even more, it could happen that the end of the trace is never predicted with this method, which forces to add a maximum trace length constraint to avoid the prediction to be infinite. Another solution is to perform a *beam search* that explores a limited space of solutions by retaining the b traces that have the highest composed probability of events. This is the approach used by [15] in which the generation procedure stops when a complete candidate trace is compliant with a series of mined LTL rules is found. They also decrease the probability of subsequent predictions of the same event. In [37], another approach is taken, in which the next event is sampled randomly following the probability distribution of the neural network. Note that there exist more sampling methods, such as *top-k sampling* [50] or *nucleus sampling* [51].

When predicting an activity suffix, every feature used as input, such as attributes or time features, must be predicted in subsequent steps. These predictions are often approached using the same neural network to predict multiple features simultaneously. Predicting multiple outputs at once in a neural network is also called *multitask learning* [52]. It has been shown that predicting multiple tasks at once helps to enhance the generalization of the neural network due to the extra information available by the results of multiple prediction tasks [52], i.e, assuming a hard parameter sharing model, the shared layers of the neural network would be affected by the gradients of all prediction tasks. Given that every prediction task is, by its nature, noisy, training a model for the main prediction task ideally should be able to not overfit that noise. For example, assuming that we are interested in forecasting the next activity, we could use the next timestamp and next resource prediction problems as auxiliary tasks. Since each task has a different noise pattern, a model that jointly learns multiple tasks would "average" those noise patterns, achieving thus a better representation of the main prediction task [53]. This effect is especially relevant in predictive monitoring, where the event logs are often scarce of data. Each prediction task has attached its own loss, and the set of losses must be combined so that they can be minimized. In predictive monitoring, the most usual form to combine the losses is depicted in Equation (1), where L_t is the total combined loss, $|T|$ is the total number of tasks, and L_i is the loss of an individual task.

$$L_t = \sum_{i=1}^{|T|} L_i \tag{1}$$

This form of combining the task losses poses the problem that different tasks could have different magnitudes, and one task could dominate others just for a bigger value, which explains why sometimes multitask learning is avoided in predictive monitoring when the prediction target is only the next activity and not the activity suffix. Some solutions to this problem pose by manually setting a different weight for each of the tasks or by automatically learning the weights for those tasks [54].

4.3 Neural Network Types

4.3.1 Feedforward Networks

Feedforward networks [55] are the most basic models for deep learning. In this type of neural network, the information flows through it without any recurrence. This lack of recurrence makes them well suited for tabular data [41] but not in process mining, where event logs have dependencies between the activities that are not exploited by this type of network. Thus, feedforward networks are not often used in predictive monitoring. However, this type of neural network is commonly used in combination with other methods, such as autoencoders [31], [36] or after extracting features from the process model [40].

Every method described in this section uses an algorithm called *backpropagation* which computes the gradients of a loss function with respect to the weights of the neural network. This allows the information from the loss function to flow "backwards" through the network to compute the gradient [55].

4.3.2 Autoencoders

The predictive monitoring approaches presented in [31] and [36] use autoencoders as their type of neural network. This kind of neural network learns how to reconstruct its own input. An autoencoder has two main parts: the encoder, which learns to map its input x into a hidden representation h, and the decoder, which learns to map a given hidden representation h back into the original input x'. The loss function is configured to penalize x' from being different from x. Formally, we would train an encoder e and a decoder d such as

$$d(e(x)) = x' \tag{2}$$

The autoencoders usually trained are *undercomplete*, which means that the dimension of the hidden representation is less than the input dimension. This forces the autoencoder to discern the most useful features of the input in a smaller hidden representation. As shown in Table 3, this approach is followed by [31], [36] in which their inputs to the first autoencoder are the hashed n-grams for each event prefix used to train the network. These approaches use the two-step procedure to train an autoencoder. First, an autoencoder is trained to represent the most useful features by reducing the dimensionality of its input. Then the already trained encoder is used to map the input to a hidden representation. Then, a feedforward network is attached using the hidden representation of the encoder as an input, which is in charge of making the final predictions.

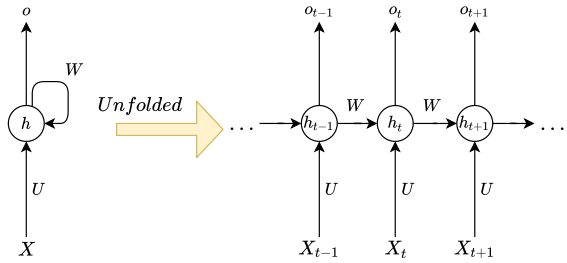

Fig. 1. Graphical representation of a vanilla recurrent neural network. In the left part, the general model is presented. In the right part, the computational graph is unfolded three timesteps.

In the context of predictive monitoring, an autoencoder approach may be useful when the number of attributes of the log is very high since it could help to reduce the dimensionality of the input data by selecting the most important features for each event. However, the main disadvantage of this approach resides in that they disregard longer dependencies between events of the prefix.

The autoencoder architecture presented here could be improved by adding a sparsity penalty to the loss function [56] or by corrupting the inputs with noise [57], but these improvements have not yet been explored in predictive monitoring.

4.3.3 Recurrent Neural Network

Many deep predictive monitoring approaches [14], [15], [16], [17], [32], [34], [35], [37], [38], [39], [42], [43], [44], [44], [45], [58] are based on Recurrent Neural Networks (RNN) [55]. RNNs are neural networks specialized in processing sequential data, that is, they operate with a sequence of vectors $x_1, \ldots, x_\tau$, where τ is the sequence length. The ability of processing sequence-like data makes this type of neural networks very useful to predictive monitoring.

The vanilla implementation of the RNN depicted in Fig. 1 poses an important problem: the gradients propagated using Back Propagation Through Time (BPTT), which is an algorithm that allows applying the backpropagation algorithm to RNNs [59], either vanish or explode when trying to learn long dependencies. There exists multiple alternative models that have been proposed to alleviate this problem: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU) and Memory Augmented Networks (MANN).

LSTM and GRU. As shown in Table 3, LSTMs [14], [15], [16], [32], [34], [35], [37], [38], [43], [44], [58] and GRUs [39], [44] have been widely applied in predictive monitoring and are two of the most popular architectures in this field. LSTMs [60] and GRUs [61] create paths through time that allow the gradients to flow deeper in the input prefix than in a vanilla RNN. Thus, instead of using the previous state directly, h_{t-1}, LSTM, and GRUs use a memory cell C_t that has an internal recurrence and the usual recurrence of vanilla RNN.

In the case of LSTMs, this internal recurrence is controlled by three different gates, f_t, o_t, and i_t, which control the flow of information inside the cell. f_t is called the "forget gate", which filters what information is thrown away from the cell state; i_t is the "input gate", which controls what information is going to be updated; and o_t is the "output gate", which decides what information is exposed from the

cell. The definition of the formulas that define an LSTM is as follows:

$$\begin{aligned} f_t &= \sigma(b_f + U_f x_t + W_f h_{t-1}) \\ i_t &= \sigma(b_i + U_i x_t + W_i h_{t-1}) \\ o_t &= \sigma(b_o + U_o x_t + W_o h_{t-1}) \\ \tilde{C}_t &= tanh(b_C + U_C x_t + W_C h_{t-1}) \\ C_t &= f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \\ h_t &= o_t \circ tanh(C_t) \end{aligned} \tag{3}$$

In the previous equations, x_t represents the input to the LSTM in the timestep t, b is a bias vector; U and W are trainable weight matrices; h_{t-1} represents the previous hidden state; $\tilde{C}_t$ is the calculation of the cell state for the current timestep; and, finally C_t is the combination of the past information of the cell with the current information of the cell. The $\circ$ operation denotes the Hadamard product between two matrices.

GRUs are similar to LSTMs with the main difference that they do not have an output gate. Formally, GRUs can be formally defined as follows:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ \tilde{h_t} &= tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_z) \\ h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t} \end{aligned} \tag{4}$$

In the previous equations, z refers to the "update gate", which controls the amount of information that flows from the past to the future; r is called the "reset gate", which filters how much information from the past is forgotten; $\tilde{h}$ represents the calculation of the current memory; and h_t corresponds to the final calculation of the memory of the cell, which can be interpreted as how much information is retained from the past and how much information is updated.

In practice, there is almost no difference in the performance of GRUs against LSTMs, but the former has the advantage of a faster training [62], [63].

Bidirectional Recurrent Neural Network. [42] uses a bi-directional LSTM with an attention mechanism to predict the outcome of an input event prefix. Bi-directional recurrent neural networks, such as [42], consist on applying one RNN forward and another RNN backwards, concatenating their hidden states of each timestep. Moreover, the attention mechanism, originally devised in [64] and [65], allows learning an alignment vector to weight the importance of each event in the prediction.

Memory Augmented Neural Network. In the predictive monitoring approach of [17], an architecture that belongs to the family of the Memory Augmented Neural Networks (MANN) is proposed. The family of MANN architectures may be useful in predictive monitoring for learning longer dependencies when the traces of the log are very long or when cycles of the same event may make the LSTM and GRU to "forget" events in the beginning of the prefix. However, these architectures are expensive to train and often very sensitive to the hyperparameters used. MANNs use an external memory unit to enhance the learning of longer term dependencies in event prefixes. The controller is often a Feedforward or a RNN that reads the inputs and, with the help of data from the memory, produces the corresponding outputs.

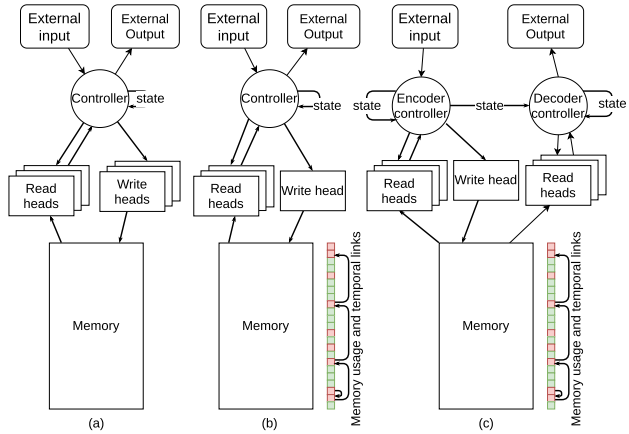

Fig. 2. Comparison between the Neural Turing Machine [66] (Fig. 2a) the Differentiable Neural Computer [68] (Fig. 2b) and the variation of the DNC proposed by Khan *et al.* [17] (Fig. 2).

The oldest MANN architecture is the Neural Turing Machines [66] (NTM). Instead of depending on a single cell for having information from the past, the NTMs use an addressing mechanism to access to this external memory cells. This addressing is based on an attention mechanism that provides a weight vector, w, which highlights the region of the memory more relevant for reading or writing at each timestep. This addressing mechanism allows the neural network to both interact with contiguous regions of memory and jump random addresses. One possible implementation of the memory addressing would use as keys the internal state of the controller LSTM in a certain timestep [67].

This kind of neural network is able to learn longer-term dependencies than its LSTM counterpart. However, the NTMs suffer from training issues (slow convergence, NaNs in gradients, etc.). The Differentiable Neural Computer [68] (DNC) further improves the memory management of the NTM by allowing freeing allocated blocks, keeping track of the writes in memory and avoiding overlapping between memory blocks.

In [17], a variation of the DNC is proposed. In this architecture, the controller is separated into two controllers, the encoder controller and the decoder controller, where both controllers are LSTMs. The encoder reads the input event prefix reading and writing the contents of the memory when necessary. Then, the decoder is initialized with the last state of the LSTM encoder, and the suffix is then predicted. One notable aspect of this architecture is that the decoder is prevented from writing into memory, so this architecture is *write-protected*. Fig. 2 shows a comparison between the NTM from [66], the DNC implementation from [68], and the DNC implementation from [17].

Transformers. The predictive monitoring approach of [45] uses a different architecture named the *transformer*. These type of architectures, originally proposed in [12], substitute the recurrence entirely by attention mechanisms. They rely on performing an attention operation over different parts of the sequence simultaneously (multi-head attention). In [45],

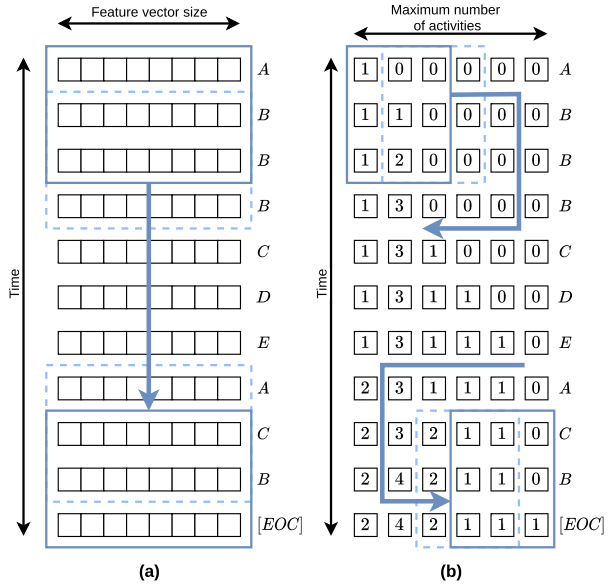

Fig. 3. Differences between 1D convolutions [18], [33] (Fig. 3a) and 2D convolutions [19] (Fig. 3b) for processing the trace $ABBBCDEACB[EOC]$ assuming a total of 6 different activities, including the end of case, on the whole log. The blue square represents the filter and the blue arrow represents the direction of movement over the input matrix. For 1D convolutions, the filter is slid over the time dimension using the full width of the feature vector corresponding to the activities. For 2D convolutions, the filter is slid from right to left and from top to bottom using a smaller kernel size.

instead of training an encoder-decoder like in the original proposal of the architecture, they only use the decoder part of the Transformer. Even though the transformer allows a faster training and inference due to the usage of only attention modules, it is still unclear how would the transformer deal with multiple heterogeneous input data, that is, when the inputs to the transformer model are both categorical and continuous data, such as the resources or time-related measures from the events.

4.3.4 Convolutional Neural Network

Convolutional Neural Networks were applied in the approaches of [18], [33] and [19], as shown in Table 3. This type of neural network is specialized in processing grid-like data. Most CNN applied to sequence prediction problems process the data as if it were a one-dimensional (1D) grid [18], [33]. In contrast, some CNNs reengineer their preprocessing of the prefixes to adapt them to a two-dimensional (2D) grid [19]. The two main operations performed by this type of neural network are the *convolution* and the *pooling* operations.

The convolution operation slides the kernel through the input grid across the input grid's width and height. The output of the convolution operation is called the *feature map*. Fig. 3 highlights the differences between a 1D convolution and a 2D convolution. In the case of 1D data, the convolution operation is applied over the tensor full width, which is the feature vector size, and with a certain height (which is the time axis of the tensor, as shown in Fig. 3a). Note that for a 2D convolution the time dimension has to be added as an additional channel of the input, while for the 1D convolution, it would just enlarge the feature vector of each event. Thus, for 2D convolutions, including information about the attributes of the event log requires adding additional channels, which might pose a problem when the number of different attributes is mismatched from the number of different activities (the width of the matrix would not be the same).

After a convolution layer, the most usual next layer is the pooling layer, which applies a statistical summary of its output, often the maximum or average of certain portions of the input grid. This kind of transformation has the advantage of learning invariant features to the position inside the trace (local translation invariance). Furthermore, applying a pooling operation can further reduce the dimensionality of the problem, thus speeding up the training procedure.

The most usual basic architecture for a CNN is a series of building blocks. Each block has a convolutional layer followed by a non-linear activation function, and a pooling layer. The works [19], [33] are based on this neural architecture. [33] follows the 1D convolutional approach, using embeddings for the log categorical variables, whereas [19] uses the 2D convolutional approach, using a frequency-based encoding for continuous and discrete variables, as shown in Fig. 3b. In both studies, the non-linear activation function used between layers is the ReLU function, which allows a faster training and alleviates the vanishing gradient problem [69].

There are two ways to increase the expressiveness of a neural network: increasing its depth or its width. When increasing the depth the vanishing gradient problem arises: the gradient updates in latter layers of the network are too small and impede the network to learn properly [10]. When increasing the width, the number of parameters, and therefore, its computational complexity, grows very rapidly. This latter problem has been tackled in [70], where multiple modules of the neural perform both a convolutional operation with different kernel sizes and a pooling operation simultaneously. This approach is used by [18] but with two main differences: they use 1D convolutions, and they do not perform a 1×1 convolution before applying a convolution with a bigger size.

Comparing CNNs against RNNs for predictive monitoring, the former may have the advantage of a faster training and inference, specially for events logs with longer traces. However, RNNs may capture longer dependencies between the events of the trace since the hidden state for a given event depends on every event before it, whereas on a CNN it only depends on the k most recent events, where k is the size of the kernel.

4.4 Sequence Encoding

In deep learning architectures, traces must be encoded in tensors of fixed size. However, there is a big variability in the length of every trace in a business process, so this step poses an important challenge. Furthermore, this step also conditions how the training targets are fed to the neural network. We have identified the following encoding formats:

- *Continuous* [14], [33], [38], [44], [45]: this encoding technique is inspired by the training of neural language models [71]. Here, the log is viewed as a text, each trace as a sentence of that text, and each activity

as a word of a sentence. In this type of encoding, only a window W of events is considered, and each window can include events from different traces. In case a window is incomplete, it could either be discarded or padded with zeroes. For example, let L be the set of traces $\{[A, B, C], [C, D]\}$. Then, with a window size $W = 3$ we would have the set of windows $\{[A, B, C], [EOC, C, D], [EOC, 0, 0]\}$ as an input. In this case, the training targets fed to the neural network are, in each timestep, the same set of windows shifted one position to the left, i.e, $\{[B, C, EOC], [C, D, EOC], [0, 0, 0]\}$.

- *Prefixes padded* [15], [16], [17], [18], [19], [32], [34], [37], [39], [42], [43]: in this type of encoding, every possible set of event prefixes hd^k where $k \in (1, \ldots, n)$ for each trace is considered. There are two different approaches to apply this encoding. The first one considers only the W most recent events (as in [37]). The second one considers all the events (as in [16]). In both approaches, the event prefixes must be padded with zeroes in case they are shorter than the specified vector length. In the second case, the vector length is often set to the length of the longest trace of the log. The approaches that use this encoding set the training target for each event prefix to the next event that follows in the event prefix, even though the full event suffix of events could also be the training target.
- *N-gram* [31], [36]: this encoding, used by [31], represents each trace as a set of all subsequences up to length k contained in it. The total number of possible sequences for an event log of $|A|$ distinct activities can be calculated using Equation (5).

$$N = \sum_{i=1}^{k} |A|^i \tag{5}$$

Since the space of n-gram combinations is very large, the "hashing trick" [72] (also known as "feature hashing") is used to reduce this dimensionality to a fixed length vector. The hashing trick is defined as in Equation (6):

$$n_i = \sum_{i:h(i)=k} \xi(i) x_i \tag{6}$$

A hash function h is used to determine the k position in the fixed vector that has to be updated with a feature x_i. Another hash function ξ counters the effect of hash collisions by determining the sign of the update.

In predictive monitoring, this type of encoding has only been used for autoencoders [31], [36] so, in these proposals the training targets as well as the inputs of the NN are equal.

- *Single event* [41]: in this encoding, only a single event and its attributes are considered, so the sequence of events in the trace is disregarded. The approach of [41] uses this encoding, setting the training targets as the next event to the event in question.
- *Timed state* [40]: proposed by [40], this encoding represents the inner state of a Petri net after replaying a partial trace in it. Each place of the Petri net is enhanced with a "decay function" that counts the time between the current timestamp and the last time a token was in a given place. The sequence encoded vector is defined as a concatenation of the following hand-crafted vectors: F_t gives the value of the decay function for each place of the Petri net; C_t counts the number of times a token has gone through a place of the Petri net; M_t counts where the tokens are in the Petri net, and R_t: counts the occurrence of other attributes of the trace. The training target is the next event after the replayed activity prefix on the Petri net.

The most used encoding schemes are *Continuous* and *Prefixes padded*, since they are versatile enough to be used with both CNNs and RNNs. The *single event* encoding is used less since it disregards the dependencies between the events of the input prefix. The *timed state* encoding has the advantage of using the model as an input and, potentially, to capture dependencies between the activities that are not present by examining the literal ordering of the events in the prefix. However, this encoding is not directly compatible with more expressive models such as RNNs or CNNs.

Furthermore, these encodings show the two philosophies regarding the existence of long-term dependencies in the logs. On the one hand, the encodings *continuous*, *prefixes padded* from a window of events, and *single event*, implicitly assume that long-term dependencies do not exist or are not relevant in event logs. This is due to the fact that these encoding approaches discard the oldest events in their predictive models. On the other hand, the encodings *prefixes padded* considering every event of the prefix, and *timed state*[3] assume that these long-term dependencies may exist since they consider every event of the prefix, and, thus, they could detect these dependencies if they would exist.

4.5 Event Encoding

Attribute variables can be either categorical variables or continuous variables. On the one hand, continuous variables must be normalized before feeding to the neural network. There are multiple techniques such as log-normalization [37], min-max normalization [37], z-score normalization [73] or tanh-estimators [74]. On the other hand, each categorical variable must be encoded in fixed feature vectors that uniquely represents them. There are various strategies used in the literature for that:

- *One-hot* [15], [16], [17], [32], [34], [35], [39], [39], [42], [43]: categorizing the variable with an integer is not enough since this categorization assumes that the higher the value of the variable is, the more important it is. To avoid that problem, the feature is represented in a binary vector where its size corresponds to the number of possible distinct values for that

3. Even though they consider every single event of the prefix, this encoding still could overwrite features when a loop occurs in the process model, and it does not enforce a strict sequentiality on the representation of the prefix.

TABLE 4
Statistics of the Event Logs Used for Benchmarking

Event log	Num. cases	Num. activities	Num. events	Avg. case length	Max. case length	Avg. event duration	Max. event duration	Avg. case duration	Max. case duration	Variants
Helpdesk	4580	14	21348	4.66	15	11.16	59.92	40.86	59.99	226
BPI 2012	13087	36	262200	20.04	175	0.45	102.85	8.62	137.22	4366
BPI 2012 Complete	13087	23	164506	12.57	96	0.74	30.92	8.61	91.46	4336
BPI 2012 W	9658	19	170107	17.61	156	0.7	102.85	11.69	137.22	2621
BPI 2012 W Complete	9658	6	72413	7.5	74	1.75	30.92	11.4	91.04	2263
BPI 2012 O	5015	7	31244	6.23	30	3.28	69.93	17.18	89.55	168
BPI 2012 A	13087	10	60849	4.65	8	2.21	89.55	8.08	91.46	17
BPI 2013 closed problems	1487	7	6660	4.48	35	51.42	2254.84	178.88	2254.85	327
BPI 2013 incidents	7554	13	65533	8.68	123	1.57	722.25	12.08	771.35	2278
Sepsis	1049	16	15214	14.48	185	2.11	417.26	28.48	422.32	845
Env. permit	1434	27	8577	5.98	25	1.09	268.97	5.41	275.84	116
Nasa	2566	94	73638	28.7	50	0.0	0.0	0.0	0.0	2513

Time related measures are shown in days.

variable, and its position in the vector is a one if the category corresponds with the variable.

- *Embedding* [14], [18], [33], [38], [41], [43], [45]: the embedding encoding creates a matrix $W \in \mathbb{R}^{n \times f}$ where each row corresponds to each of the categories of the variable, and columns correspond to the feature dimension. The parameters of this matrix can be either established randomly or be learned with stochastic gradient descent, so the learned embeddings are optimal for the prediction task at hand.
- *Frequency-based* [19], [40]: this type of encoding [75] indicates how many times the activity i has happened until the current event of the trace. This encoding is useful when temporal information must be added to the encoding of the activities. Note that in the case of [40] the frequency does not represent directly activities but the number of times a token has gone through a place.
- *Pretrained embedding* [37]: instead of directly training the embedding vectors with stochastic gradient descent, the embeddings can be pretrained for another task that gives additional information. For example, in [37], a neural network is trained to learn embeddings by discriminating role-activity pairs. A positive instance means that a role-activity pair exists on the log whereas a negative pair means otherwise. They claim that the embeddings learned this way can discriminate better between different events.

While the *pretrained embedding* and *embedding* approaches have the main purpose of learning a set of embeddings to represent the categorical variables, the former are richer and may provide more information, which eases the convergence of the neural network, as many works in NLP that used similar techniques have shown [76], [77]. Moreover, the *one-hot* encoding is a good solution in terms of easiness of computation and implementation, because it does not use additional parameters, otherwise, the size of the vector could dramatically increase the memory usage of the neural network. Note that associating an integer to a category and using that directly as an input is heavily discouraged, due to imposing an implicit order of the categorical variable that it is not true (i.e, category number 2 would be more important than category number 1) [78].

5 Benchmark

5.1 Experimental Setup

5.1.1 Datasets

We performed the experiments using 12 real-life event logs from a variety of domains[4]. These event logs were extracted from the *4TU Center for Research Data*[5] and are also available in the repository of our comparison tool. Table 4 shows some relevant statistics from these logs, namely, the number of cases, the number of different activities, the number of events, the average and maximum case length, the maximum and average event duration in days, the average and maximum case duration in days and the number of different variants. Most logs have a high event time variability (difference between average event duration and maximum event duration), and a high trace length variability. The log "Nasa" shows 0 in the time related measures since the time variability in this log is low.

5.1.2 Data Split

We performed a 5-fold cross-validation in which each of the folds is used to report the final performance of the deep learning approach. For each remaining set of 4-folds, we split it in training and validation sets in an 80/20 ratio. These latter partitions are used to train the neural network and to select the best performing model, respectively. We order the timestamps of the events in the trace. If multiple events have the same timestamp, the original order of the events within the log prevails over any other ordering. Each

4. The adaptations made to the approaches, the original code repositories and the attributes used by the approaches are available in the supplementary material, available online.

5. The logs are available in the following repository: https://github.com/ERamaM/PredictiveMonitoringDatasets

of the approaches is tested one time in each of the folds. We refrain from increasing the number of folds, training repetitions or performing a nested cross-validation for two reasons. First, some approaches [17] are computationally expensive to train or perform a hyperparameter optimization that heavily increases training time [18], [37]. Second, and more importantly, the time to perform the testing phase for the suffix prediction tasks when the prediction is not direct (Definitions 5, 7, and 9 from Section 2) is very high due to having to predict every single event for each possible prefix until the end of the case is reached or the maximum trace length is achieved. For some approaches [16] this testing time is orders of magnitude greater than the time required to train the neural network.

Furthermore, many approaches add an "end of case" token at the end of every trace of the log. This modification allows to give a clear stop condition for the next activity and activity suffix prediction problems. Thus, we do not have to rely on inspecting the set of last activities of every trace of the log to know whether a case has finished or not. Furthermore, in this latter case we would not get a definite answer either, because the last activity of the trace could be part of a loop, e.g., a case that always repeats the same activity. To unify the procedure and to make the metrics of next activity prediction comparable, we augmented the log with an end of case activity at the end of each trace for every approach tested.

5.1.3 Metrics

Depending on the predictive task, we use the following metrics for evaluating the performance of the approaches:

- *Next activity prediction*[6]: since the next activity prediction task is a classic classification problem, we use the *accuracy* metric. The accuracy measures the proportion of correct classifications in relation to the number of predictions done. Other measures such as the Matthews Correlation Coefficient [79] or the weighted F1 score are reported by the approaches tested, but we found that the results were aligned with the accuracy measure and do not give additional information. Therefore, these results are not reported in this paper.
- *Activity suffix prediction*: when predicting an activity suffix in the context of predictive monitoring, it is important to take into account that the activities in the process may occur in parallel [16]. Thus, instead of the metrics used for the next activity prediction task, we use the *Damerau-Levenshtein distance* metric. This metric measures the edit distance between two given strings without penalizing too harshly transpositions of tokens, which, in the context of predictive monitoring, could mean a pair of parallel activities. These two strings represent the predicted activity suffix for a given event prefix and its ground truth activity suffix. The Damerau-Levenshtein metric measures the number of insertions, deletions, substitutions, and transpositions needed to transform one string into another. This value is then normalized by the lengths of the two strings, obtaining a value of similarity between 0 and 1.
- *Next timestamp prediction and remaining time prediction*[7]: since the time prediction problem is a regression task, the metric selected for measuring the performance in the next timestamp prediction tasks and remaining time prediction tasks is the *Mean Absolute Error* (MAE). This metric is defined in Equation (7) and has the advantage of not overpenalizing the variability in the observations [80], which is the case in time prediction in predictive process monitoring, where the time between two events in a trace can be potentially large [16], [25].

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{7}$$

Note that the prediction problems of *outcome*, *next attribute* and *attribute suffix* are not part of this experiment. On the one hand, the outcome prediction problems are a special case of predictive monitoring where, unlike the other approaches analyzed in this paper, the solutions are focused on a particular set of logs. Thus, they often define different outcomes, adding additional information from the application domain. Since they are ad-hoc approaches for specific logs and outcomes, they are left out of the scope of this review. On the other hand, the next attribute and attribute suffix prediction problems are also not carried out because, after performing the selection of the approaches to test, we found out that only one predicts attributes.

Independently of the prediction problem and metric used, we apply a two stage statistical test to quantify the differences between the predictive monitoring approaches. Instead of applying some classical null hypothesis significance testing (NHST) approaches, we rely on Bayesian analysis to assess the differences between the approaches, mainly due to the following reasons: *(i)* NHST yields no information about the null hypothesis, so, if the null hypothesis is not rejected, we would not gather any valuable information of the statistical test [81], *(ii)* NHST do not estimate probabilities of the hypotheses, so comparing the classifiers would be more difficult [81], and *(iii)* as far as we know, there is no NHST test that can take into account the variance of the results over multiple folds of a cross-validation testing procedure since those tests work by aggregating the results of the folds for each dataset [82]. We rely on the library `scmamp` [83] for performing these statistical tests.

We propose a two stage statistical comparison procedure. First, we apply a Bayesian analysis technique based on Plackett-Luce model [84] that enables ranking multiple algorithms at the same time, allowing to get an overall comparison and probabilities of being the best classifier for the whole set of tested predictive monitoring approaches. Since the Plackett-Luce approach does not take into account the individual fold results of the cross-validation testing, and to try to solve ties between the approaches, we apply a

6. More metrics for the next activity prediction task are available in the supplementary material, available online.

7. The results of the next timestamp prediction problem are available in the supplementary material, available online.

TABLE 5
Mean Accuracy of the 5-Fold Cross-Validation for the Next Activity Prediction Problem

	BPI 2012	BPI 2012 A	BPI 2012 Complete	BPI 2012 O	BPI 2012 W	BPI 2012 W Complete	BPI 2013 Closed Problems	BPI 2013 Incidents	Env Permit	Helpdesk	Nasa	Sepsis
Camargo	83.28	75.98	77.93	81.35	76.4	68.95	54.67	66.68	**85.78**	82.93	-	-
Evermann	59.33	75.82	62.38	79.42	75.37	67.53	58.83	**66.78**	76.19	**83.66**	20.37	40.0
Hinkka	**86.65**	**81.19**	**80.64**	**87.23**	84.78	**70.54**	**63.47**	**74.69**	84.43	83.08	**88.42**	**63.5**
Khan	42.9	74.9	47.37	66.08	60.15	52.22	43.58	51.91	83.59	79.97	12.71	21.01
Mauro	**84.66**	**79.76**	**80.06**	**82.74**	**85.98**	68.64	24.94	36.67	53.59	31.79	21.03	**61.52**
Pasquadibisceglie	83.25	74.12	74.6	78.88	81.19	68.34	47.45	46.03	**86.69**	**83.93**	88.27	56.15
Tax	**85.46**	**79.53**	**80.38**	**82.29**	85.35	69.79	**64.01**	**70.09**	85.71	**84.19**	**89.44**	**64.22**
Theis et al. (w/o attributes)	82.89	65.5	75.26	78.38	**86.22**	**80.06**	**59.48**	59.41	**86.29**	78.77	**88.96**	55.74
Theis et al. (w/ attributes)	80.96	65.67	75.75	76.89	**86.86**	**83.84**	54.65	51.5	85.12	79.69	86.34	58.14

The best, second best and third best approaches are highlighted in cyan, orange and yellow, respectively.

TABLE 6
Plackett-Luce Probability of Winning and Ranking for the Next Activity Prediction Problem

Approach	Tax	Hinkka	Camargo	Theis (w/o resource)	Pasqua.	Theis (w/ resource)	Evermann	Mauro	Khan
Ranking	1.5288	1.5908	3.2906	5.1021	5.4615	5.5126	6.4451	7.2742	8.7934
Probability	27.69%	26.96%	12.51%	7.47%	6.80%	6.74%	5.36%	4.27%	2.18%

hierarchical Bayesian model to pairwise compare the three best approaches in terms of Plackett-Luce ranking [85].

5.1.4 Approaches and Experimental Setup

We performed the benchmark with 10 different approaches from the state of the art [14], [15], [16], [17], [18], [19], [32], [37], [39], [40]. We discarded the remaining 11 surveyed approaches based on two conditions.

First, we decided to discard the four approaches that predict outcome for the reasons explained in the previous section. Then, we also discarded the remaining seven approaches since they had no code publicly available. For these approaches, we contacted the authors but received no response or the code was no longer available or it could not be shared. Note that we also contacted the authors even if we found blocking issues that prevented us to completely execute the code of the approach.

As far as the hyperparameter optimization is concerned, we decided to apply the hyperparameter optimization procedure only if it were already available in their code. For the remaining approaches, we use the default hyperparameters reported in the papers or in their source code, even though their performance could be further enhanced by an optimization procedure [86]. In fact, only [37] and [18] optimize their hyperparameters using a random search and Hyperopt, respectively.

The experiments were carried out in a server equipped with an Intel Xeon Gold 5220, 192 GB RAM, and 2 Nvidia Tesla V100S 32GB. Trace level attributes have not been taken into consideration. To perform the specific preprocessing for each dataset, we rely on the PM4Py library [87]. As shown in the aforementioned table, the highest count of attributes is present in the datasets "BPI 2013 Incidents" and "Nasa". Note that not every dataset has available the resource associated to the event so, if the approach uses specifically this attribute, it can not be evaluated (which is the case of Camargo *et al.* [37] in datasets "Nasa" and "Sepsis"). If an approach can not be evaluated in a dataset, this forces to discard that dataset from the statistical comparison. Therefore, the provided statistical comparisons are performed without taking into account the "Nasa" and "Sepsis" datasets.

5.2 Results and Discussion

5.2.1 Next Activity Prediction

Table 5 shows the mean accuracy of the 5-fold cross-validation testing for the next activity prediction problem. From a frequentist point of view, Hinkka *et al.* [39] obtain the best result in 5 event logs, Pasquadibisceglie *et al.* [19] obtain the best result in one log, Tax *et al.* [16] obtain the best result in 4 logs, and Theis *et al.* [40] obtain the best result in 2 logs.

From a Bayesian point of view, the best approach according to the Plackett-Luce rankings, shown in Table 6, is Tax *et al.* [16], followed closely by Hinkka *et al.* [39] and, at more distance, by Camargo *et al.* [37]. Fig. 4 shows the credible intervals calculated from the 5% and 95% quantiles alongside with the probability of winning. In this plot, if two approaches are not overlapped, they are statistically significant different. Thus, we can distinguish three blocks of approaches. The approaches of Hinkka *et al.* and Tax *et al.*, that are only overlapped by themselves and by Camargo *et al.* The approach of Camargo *et al.*, which overlaps every other approach, and the rest of approaches that only overlap themselves and Camargo *et al.*

Tables 7, 8, and 9 show the results from the hierarchical Bayesian tests. These tests perform a pairwise comparison between two approaches and report two sets of results. The first set is the global probability of the difference between two algorithms, A and B, being A better than B ($A > B$), A

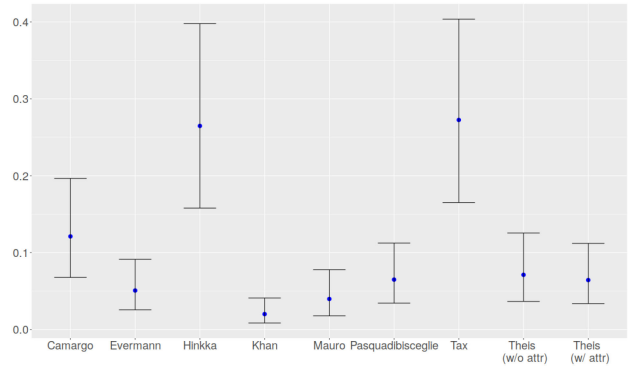

Fig. 4. Credible intervals (5% and 95% quantiles) and expected probability of winning for the next activity prediction problem.

equal to B ($A = B$) or A worse than B ($A < B$). In the table this casuistic is marked as the row "overall". The second set of results performs the same comparison but individually for a dataset. Note that the global probability is not the mean of the probability of each dataset. Thus, assuming a significance level of $> 90\%$ and two algorithms, A and B, we can distinguish the following situations:

1) A is better than B if $(A > B) > 90\%$.
2) A is equal to B if $(A = B) > 90\%$.
3) A is worse than B if $(A < B) > 90\%$.

TABLE 7
Hierarchical Accuracy Test Results Per Dataset: Hinkka *et al.* (H) Against Tax *et al.* (T)

Log	H > T	H = T	H < T
BPI 2012 Complete	4.14%	95.14%	0.715%
BPI 2012 O	85.43%	13.05%	1.52%
BPI 2013 incidents	92.41%	6.99%	5.99%
BPI 2012 W Comp.	32.44%	65.33%	2.22%
BPI 2012 W	0.27%	94.51%	5.21%
BPI 2012	64.77%	34.99%	0.24%
Env. permit	5.82%	53.56%	40.61%
Helpdesk	5.28%	57.70%	31.01%
BPI 2013 c.p.	28.34%	49.85%	21.81%
Nasa	0.25%	53.41%	46.33%
Sepsis	10.57%	65.17%	24.25%
BPI 2012 A	67.78	30.83%	1.39%
Overall	49.6%	41.5%	8.9%

TABLE 8
Hierarchical Accuracy Test Results per Dataset: Camargo *et al.* (C) Against Hinkka *et al.* (H)

Log	C > H	C = H	C < H
BPI 2012	0.07%	4.34%	95.59%
BPI 2012 A	0.59%	8.5%	90.90%
BPI 2012 Complete	0.0015%	0.21%	99.79%
BPI 2012 O	0.12%	39.75%	60.13%
BPI 2012 W	1.61%	11.72%	86.66%
BPI 2012 W Comp.	3.57%	19.58%	76.86%
BPI 2013 c.p.	0.521%	6.77%	92.71%
BPI 2013 incidents	0.008%	0.48%	99.51%
Env. permit	3.59%	49.09%	47.32%
Helpdesk	0.01%	11.20%	88.79%
Overall	**0.6%**	**0%**	**99.4%**

TABLE 9
Hierarchical Accuracy Test Results Per Dataset: Camargo *et al.* (C) Against Tax *et al.* (T)

Log	C > T	C = T	C < T
BPI 2012	0.07%	4.34%	95.58%
BPI 2012 A	0.59%	8.50%	90.91%
BPI 2012 Complete	0.001%	0.21%	99.79%
BPI 2012 O	0.12%	39.75%	60.13%
BPI 2012 W	1.61%	11.72%	86.66%
BPI 2012 W Comp.	3.57%	19.58%	76.86%
BPI 2013 c.p.	0.52%	6.77%	92.71%
BPI 2013 incidents	0.008%	0.48%	99.51%
Env permit.	3.59%	49.09%	47.33%
Helpdesk	0.01%	11.20%	88.79%
Overall	**0.3%**	**1%**	**98.8%**

4) A is not worse than B if $(A > B) + (A = B) > 90\%$.

The results from Table 7 show that, overall, the approach of Hinkka *et al.* is not worse than the approach from Tax *et al.* (91.1%) but it is not considerably better either. The approach of Hinkka *et al.* outperforms the approach of Tax *et al.* in the dataset "BPI 2013 incidents" (92.41%). This dataset has the highest number of attributes among all the tested datasets (8 attributes), which shows that the clustering information of the approach from Hinkka *et al.* can improve the learning phase of the relationships between the attributes in the log. Furthermore, this approach is not worse than Tax *et al.* in the datasets "BPI 2012 O" (98.48%), "BPI 2012" (97.77%) and "BPI 2012 A" (98.61%). Note that even though from a frequentist point of view the approach of Hinkka *et al.* clearly outperforms the one of Tax *et al.* in the "BPI 2012 O" (4.94%), this high difference is due to the variance between the individual results in each fold of the crossvalidation, i.e, most folds of the approach from Tax *et al.* have an accuracy greater than 82.29%, but the overall performance is decreased due to a bad performing fold. Moreover, the approach of Hinkka *et al.* is equal to the approach of Tax *et al.* in the datasets "BPI 2012 Complete" and "BPI 2012 W". This is an indication that the added features from the clustering phase of Hinkka *et al.* are not detrimental to the learning of the neural network, in terms of overfitting, when the log is complex and has enough data. However, in smaller logs, such as "Env. permit", in logs with complex relationships between the activities, such as "Nasa", and in logs with longer traces, such as "Sepsis", this additional information can be detrimental to the training of the neural network. In this kind of logs, we can affirm that an approach similar to the Tax *et al.* is not worse than the approach of Hinkka *et al.* In the remaining logs, the uncertainty is too big to make any specific recommendation.

In addition, the results from Table 8 show that, overall, the approach of Hinkka *et al.* outperforms the approach of Camargo *et al.* significantly. In fact, Hinkka *et al.* outperforms Camargo *et al.* in every dataset, but in the datasets "BPI 2012 O", "BPI 2012 W", "BPI 2012 W Complete", "Env. permit", and "Helpdesk", where we can only affirm that Hinkka *et al.* is not worse than Camargo *et al.* This shows that, even though the discovery of roles may prove to be

TABLE 10
Mean DL Distance of the 5-Fold Cross-Validation for the Activity Suffix Prediction Problem

	BPI 2012	BPI 2012 A	BPI 2012 Complete	BPI 2012 O	BPI 2012 W	BPI 2012 W Complete	BPI 2013 Closed Problems	BPI 2013 Incidents	Env Permit	Helpdesk	Nasa	Sepsis
Camargo (argmax)	**0.1851**	**0.6536**	**0.2218**	**0.6845**	**0.1941**	0.0501	**0.6641**	0.2607	**0.844**	**0.911**	-	-
Camargo (random)	**0.3891**	**0.6441**	**0.4495**	**0.6042**	**0.311**	**0.3191**	0.5357	**0.5294**	**0.7595**	**0.8524**	-	-
Evermann	**0.1986**	**0.5847**	**0.2693**	**0.5544**	**0.28**	**0.3372**	**0.6416**	**0.473**	0.5713	0.8354	**0.1218**	**0.2693**
Francescomarino	0.1321	0.2871	0.0883	0.4591	0.141	**0.1126**	0.5276	**0.3607**	0.3924	0.4619	**0.0842**	**0.0742**
Tax	0.1409	0.4597	0.1717	0.4972	0.0975	0.0789	**0.5824**	0.3336	**0.8163**	**0.8695**	**0.232**	**0.1158**

The best, second-best and third-best approaches are highlighted in cyan, orange and yellow, respectively.

TABLE 11
Plackett-Luce Probability of Winning and Ranking for the Suffix Prediction Problem

Approach	Camargo (random)	Evermann	Camargo (argmax)	Tax	Francescomarino
Ranking	1.408	2.160	2.616	3.858	4.958
Probability	36.70%	26.21%	21.43%	11.42%	4.24%

useful for leveraging information from the whole set of attributes is a better strategy.

Finally, the results from Table 9 show the comparison between Camargo *et al.* and Tax *et al.* approaches. The approach of Tax *et al.*, overall, outperforms the approach of Camargo *et al.* In fact, Tax *et al.* outperforms Camargo *et al.* in the datasets "BPI 2012", "BPI 2012 A", "BPI 2012 Complete", "BPI 2013 closed problems" and "BPI 2013 incidents". This is due to two main factors. On the one hand, the approach of Tax *et al.* is simpler than the one of Camargo *et al.*, because it only predicts time and activities, while Camargo *et al.* also predicts the role. On the other hand, the approach of Tax *et al.* has its hyperparameters fixed, which allows the neural network to perform consistently among a wide array of event logs since it neither depends on the hyperparameter optimization strategy nor on the number of search trials, even though it may not have the best possible performance. However, Camargo *et al.* tries to optimize its hyperparameters using a random search, but the number of models tested by the approach may not be enough to discover a set of hyperparameters that is optimum for each event log. In fact, the approach of Mauro *et al.* also optimizes hyperparameters, and, while in some logs it works well ("BPI 2012" or "Sepsis"), in some others it underperforms considerably ("BPI 2013 closed problems" or "Helpdesk") due to the same reason: the number of models tested is not enough to achieve a good set of hyperparameters.

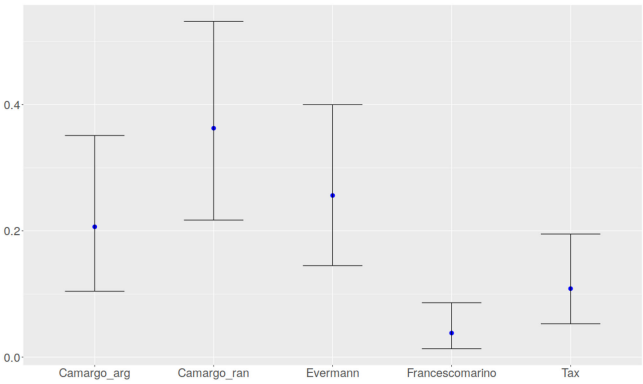

Fig. 5. Credible intervals (5% and 95% quantiles) and expected probability of winning for the suffix prediction problem.

5.2.2 Activity Suffix Prediction

As far as the suffix prediction problem is concerned (Table 10), the best approach according to the Plackett-Luce rankings, shown in Table 11 and Fig. 5, is the approach of Camargo *et al.* [37] with the random sampling procedure (RSP). In fact, according to the results of Table 12, the RSP performs better overall than the "argmax" sampling procedure (ASP). However, note that ASP outperforms RSP in logs where the maximum trace length is short ("Helpdesk", "Env. permit", "BPI 2013 closed problems", and "BPI 2012 O"), while RSP is better in logs with longer traces. We can find the same effect when comparing the approach of Camargo *et al.* "argmax" variant with the one of Evermann, in Table 13. However, as shown in Table 14, Camargo *et al.* outperforms overall the approach of

TABLE 12
Hierarchical Suffix Prediction Test Results Per Dataset: Camargo *et al.* (argmax) (C_a) Against Camargo *et al.* (Random) (C_r)

Log	$C_a > C_r$	$C_a = C_r$	$C_a < C_r$
BPI 2012	0.012%	0.004%	99.98%
BPI 2012 A	48.16%	8.85%	42.99%
BPI 2012 Complete	0.265%	0.054%	99.68%
BPI 2012 O	98.96%	0.267%	0.768&
BPI 2012 W	4.39%	1.02%	94.59%
BPI 2012 W Complete	0.038%	0.007%	99.95%
BPI 2013 c.p.	96.56%	0.59%	2.84%
BPI 2013 incidents	0%	0%	100%
Env. permit	99.70%	0.078%	0.218%
Helpdesk	99.99%	0.004%	0%
Overall	**11.6%**	**0%**	**88.4%**

TABLE 13
Hierarchical Suffix Prediction Time Test Results Per Dataset: Camargo *et al.* (Argmax) (C_a) Against Evermann (E)

Log	$C_a > E$	$C_a = E$	$C_a < E$
BPI 2012	8.57%	15.13%	76.30%
BPI 2012 A	86.42%	3.00%	10.58%
BPI 2012 Complete	9.84%	4.06%	86.10%
BPI 2012 O	0.04%	0.01%	99.95%
BPI 2012 W	8.22%	2.17%	89.62%
BPI 2012 W Complete	0.144%	0.021%	99.83%
BPI 2013 c.p.	68.42%	22.53%	9.04%
BPI 2013 incidents	0.029%	0.001%	99.97%
Env. permit	99.98%	0.001%	0.019%
Helpdesk	99.99%	0.00008%	0.0028%
Overall	**46.3%**	**0%**	**53.7%**

TABLE 14
Hierarchical Suffix Prediction Test Results Per Dataset: Camargo *et al.* (Random) (C_r) Against Evermann (E)

Log	$C_r > E$	$C_r = E$	$C_r < E$
BPI 2012	100%	0%	0%
BPI 2012 A	99.82%	0.0676%	0.001%
BPI 2012 Complete	99.987%	0.001%	0.012%
BPI 2012 O	98.87%	0.53%	0.6%
BPI 2012 W	89.19%	5.06%	5.76%
BPI 2012 W Complete	8.71%	11.37%	79.93%
BPI 2013 c.p.	0.46%	0.10%	99.44%
BPI 2013 incidents	99.80%	0.078%	0.1%
Env. permit	99.94%	0.01%	0.05%
Helpdesk	96.29%	3.02%	0.69%
Overall	**95.9%**	**9%**	**4.1%**

Evermann *et al.*, even though they use the same sampling procedure, since more information is used in the former approach and, thus, a better learning of the suffixes is achieved.

These results are explained by the fact that ASP samples the next activity always as the one with the highest probability, while RSP samples the next activity following the probability distribution of the predicted activities. Thus, ASP procedure is often stuck in repeating the same set of activities in logs with longer traces and loops due to selecting always the most probable set of activities, whereas RSP has a better chance to escape these prediction repetitions [51]. In conclusion, RSP is expected to outperform ASP in complex logs with long traces, where the chance of getting stuck predicting the same set of activities is low. However, in more simple logs, with shorter traces, ASP should slightly outperform RSP since the suffixes can be predicted with more precision.

TABLE 16
Plackett-Luce Probability of Winning and Ranking for the Suffix Prediction Problem

Approach	Navarin	Camargo (random)	Camargo (argmax)	Tax	Francescomarino
Ranking	1.003	2.268	2.782	4.225	4.722
Probability	68.60%	14.39%	10.49%	3.79%	2.72%

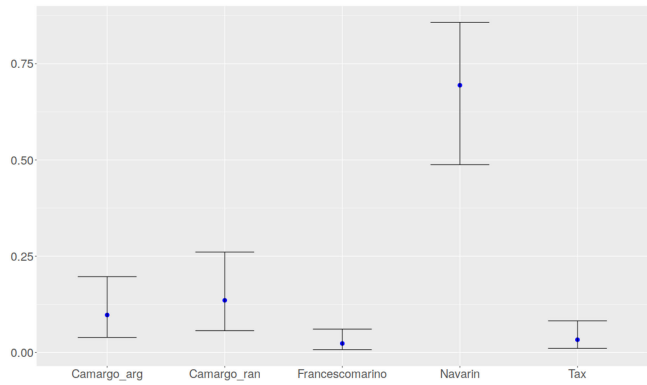

Fig. 6. Credible intervals (5% and 95% quantiles) and expected probability of winning for the remaining time prediction problem.

5.2.3 *Remaining Time Prediction*

As far as the remaining time prediction problem is concerned (Table 15), the best approach according to the Plackett-Luce rankings (Table 16 and Fig. 6) is Navarin *et al.* [32], followed by the sampling approaches "random" and "argmax" of Camargo *et al.* In particular, as shown in the results of Table 18 and 19, the approach of Navarin *et al.* significantly outperforms the approach of Camargo *et al.* in both of their proposed sampling variations. This is due to the fact that the approach of Navarin *et al.* relies on performing a direct prediction of the remaining time for a given prefix, instead of recursively predicting the next timestamp as a suffix generation task. Thus, the prediction errors do not compound, achieving a more precise prediction and allowing the approach of Navarin *et al.* to significantly outperform the other

TABLE 15
Mean Remaining Time of the 5-Fold Cross-Validation for the Remaining Time Prediction Problem

	BPI 2012	BPI 2012 A	BPI 2012 Complete	BPI 2012 O	BPI 2012 W	BPI 2012 W Complete	BPI 2013 Closed Problems	BPI 2013 Incidents	Env Permit	Helpdesk	Sepsis
Camargo (argmax)	**29.905**	**8.989**	**28.045**	**15.25**	**30.484**	**15.263**	257.086	**28.132**	8.29	**18.634**	-
Camargo (random)	**29.599**	**8.976**	**10.468**	**15.247**	**30.564**	**10.417**	257.697	**28.511**	8.277	**18.46**	-
Francescomarino	500.303	29.363	102.998	66.454	248.37	59.68	**191.1**	35.405	**5.613**	159.891	**503.867**
Navarin	**7.364**	**7.774**	**7.2**	**7.586**	**7.845**	**8.189**	**159.164**	**12.366**	**4.124**	**8.186**	**33.973**
Tax	383.054	20.283	195.482	68.155	157.518	127.397	**172.849**	30.082	**4.109**	68.864	**421.6**

The best, second best and third best approaches are highlighted in cyan, orange and yellow, respectively.

TABLE 17
Hierarchical Remaining Time Test Results Per Dataset: Camargo *et al.* (Argmax) (C_a) Against Camargo *et al.* (Random) (C_r)

Log	$C_a > C_r$	$C_a = C_r$	$C_a < C_r$
BPI 2012	5.86%	79.14%	15.00%
BPI 2012 A	0.08%	99.77%	0.14%
BPI 2012 Complete	10.78%	78.99%	10.23%
BPI 2012 O	2.87%	94.24%	2.89%
BPI 2012 W	9.61%	90.22%	0.17%
BPI 2012 W Complete	10.67%	78.96%	10.36%
BPI 2013 c.p.	11.64%	79.11%	9.25%
BPI 2013 incidents	10.78%	79.06%	10.16%
Env. permit	0.055%	99.86%	0.088%
Helpdesk	9.89%	79.67%	10.44%
Overall	**4.7%**	**90%**	**5.3%**

TABLE 18
Hierarchical Remaining Time Test Results Per Dataset: Camargo *et al.* (Argmax) (C_a) Against Navarin (N)

Log	$C_a > N$	$C_a = N$	$C_a < N$
BPI 2012	3.88%	0.3%	95.82%
BPI 2012 A	5.33%	2.02%	92.64%
BPI 2012 Complete	8.58%	0.49%	90.93%
BPI 2012 O	3.73%	0.057%	99.57%
BPI 2012 W	4.10%	0.27%	95.63%
BPI 2012 W Complete	7.05%	0.47%	92.48%
BPI 2013 c.p.	9.93%	0.49%	89.58%
BPI 2013 incidents	5.27%	0.34%	94.40%
Env. permit	6.63%	0.61%	92.76%
Helpdesk	2.07%	0.199%	97.73%
Overall	**0.9%**	**0%**	**99.1%**

ones, as shown in the rankings of Table 16 and in the hierarchical tests of Tables 18 and 19. Furthermore, when comparing the "random" and "argmax" approaches of Camargo *et al*., as shown in Table 17, the results show that they are equivalent to predict the remaining time, suggesting that the "random" sampling does not improve the prediction performance when the remaining time is predicted recursively.

6 Conclusion

In this paper, we presented a systematic literature review of Deep Learning approaches to predictive business process monitoring. We made an analysis and classification of these approaches, supported by an ex-haustive experimental evaluation of 10 approaches on 12 different event logs, according to five different perspectives: *(i)* input data, *(ii)* predictions, *(iii)* neural network type, *(iv)* sequence encoding, and *(v)* event encoding.

TABLE 19
Hierarchical Remaining Time Test Results Per Dataset: Camargo *et al.* (Random) (C_r) Against Navarin (N)

Log	$C_r > N$	$C_r = N$	$C_r < N$
BPI 2012	4.96%	0.52%	94.52%
BPI 2012 A	3.96%	1.72%	94.32%
BPI 2012 Complete	0.028%	0.0095%	99.96
BPI 2012 O	0.45%	0.0812%	99.47%
BPI 2012 W	4.98%	0.527%	94.49%
BPI 2012 W Complete	0.066%	0.0244%	99.91%
BPI 2013 c.p.	9.51%	0.73%	89.85%
BPI 2013 incidents	4.65%	0.52%	94.83%
Env. permit	5.73%	0.69%	93.57%
Helpdesk	2.65%	0.356%	96.99%
Overall	**0%**	**0%**	**100%**

Our experimentation highlights the differences between the approaches when tackling the prediction problems of *next activity prediction*, *activity suffix prediction*, *next timestamp prediction*, and *remaining time prediction*. The experimentation on the *next activity prediction* shows that the usage of the whole set of available attributes [39] is, in general, beneficial, or, at least, not detrimental, as long as the amount of data is sufficient, and the log is not too complex. However, when the log is too complex, a simpler approach, such as Tax *et al*. [16], may capture more effectively the relations among the activities in the trace than other more complex approaches, such as Hinkka *et al*. [39] or Camargo *et al*. [37]. As far as the *activity suffix prediction*, we showed that sampling the highest probable activity ("argmax") [16], [37] is not always better than sampling randomly from the probability vector ("random") [14], [37]. The former approach works better in simpler logs with shorter traces whereas the latter works better in logs more complex and with longer traces, due to the fact that in the latter case the "argmax" approach gets stuck in a local minimum, always predicting the same set of activities and never predicting the end of case. Regarding the *remaining time prediction*, we showed that a direct prediction [32] always outperforms a recursive prediction because it does not accumulate the individual errors of predicting each event of the prefix.

Regarding the neural network type, we found that, overall, RNNs seem to outperform CNNs. This is due to the fact that CNNs also often need to set up pooling layers to improve their performance but doing so adds additional hyperparameters and may impose a suboptimal architecture that may hinder their performance, while RNNs are just often stacked one after another giving good results (even though the approach of Mauro *et al*. [18] seems to tackle this issue, but unsatisfactorily in some logs). However, CNNs perform way faster than RNNs, so they may be the only candidate in applications where speed is a priority (either by the abundance of data or by the need for a quick testing phase). Concerning the sequence and event encoding perspectives, even though the experimentation does not highlight differences on the performance of the multiple possibilities of encoding, there are differences on the efficiency between the encodings. For example, the *one-hot* event encoding uses more memory than the *embedding* event encoding when the number of different activities is very high or the *continuous* sequence encoding has a faster training procedure than the *prefixes padded* sequence encoding.

We expect the experiments serve as a baseline for future works in predictive monitoring. As a future line of work, this experimentation could be enhanced with non-deep learning approaches but also using specific synthetic datasets to further clarify the properties of neural networks for predictive monitoring.

REFERENCES

[1] M. Kirchmer, *High Performance Through Business Process Management*. Berlin, Germany: Springer, 2017.

[2] W. M. P. van der Aalst, *Process Mining*. Berlin, Germany: Springer, 2011.

[3] F. M. Maggi, C. D. Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *Proc. 26th Int. Conf. Adv. Inf. Syst. Eng.*, 2014, pp. 457–472.

[4] C. D. 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, Nov. 2019.

[5] W. L. J. Lee, D. Parra, J. Munoz-Gama, and M. Sepúlveda, "Predicting process behavior meets factorization machines," *Expert Syst. Appl.*, vol. 112, pp. 87–98, 2018.

[6] J. Becker, D. Breuker, P. Delfmann, and M. Matzner, "Designing and implementing a framework for event-based predictive modelling of business processes," in *Proc. Enterprise Model. Inf. Syst. Architectures*, 2014, pp. 71–84.

[7] D. Breuker, M. Matzner, P. Delfmann, J. Becker, "Comprehensible predictive models for business processes," *MIS Quart.*, vol. 40, no. 4, pp. 1009–1034, 2016.

[8] B. Kang, D. Kim, and S.-H. Kang, "Periodic performance prediction for real-time business process monitoring," *Ind. Manage. Data Syst.*, vol. 112, no. 1, pp. 4–23, 2012.

[9] C. Cabanillas, C. D. Ciccio, J. Mendling, and A. Baumgrass, "Predictive task monitoring for business processes," in *Proc. 12th Int. Conf. Bus. Process Manag.*, S. W. Sadiq, P. Soffer, and H. Völzer, Eds., 2014, pp. 424–432.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html

[12] A. Vaswani *et al.*, "Attention is all you need," in *Proc. 30th Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 27th Annu. Conf. Neural Inf. Process. Syst.*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 3104–3112.

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

[15] C. D. Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, and A. Yeshchenko, "An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring," in *Proc. 15th Int. Conf. Bus. Process Manage.*, 2017, pp. 252–268.

[16] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Proc. 29th Int. Conf. Adv. Inf. Syst. Eng.*, 2017, pp. 477–492.

[17] A. Khan *et al.*, "Memory-augmented neural networks for predictive process analytics," *CoRR*, vol. abs/1802.00938, 2018.

[18] N. D. Mauro, A. Appice, and T. M. A. Basile, "Activity prediction of business process instances with inception CNN models," in *Proc. 18th Int. Conf. Ital. Assoc. Artif. Intell.*, 2019, pp. 348–361.

[19] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Using convolutional neural networks for predictive process analytics," in *Proc. Int. Conf. Process Mining*,2019, pp. 129–136.

[20] N. Tax, I. Teinemaa, and S. J. van Zelst, "An interdisciplinary comparison of sequence modeling methods for next-element prediction," *Softw. Syst. Model.*, vol. 19, pp. 1345–1365, 2020.

[21] A. E. Marquez-Chamorro , M. Resinas, and A. Ruiz-Cortes , "Predictive monitoring of business processes: A survey," *IEEE Trans. Serv. Comput.*, vol. 11, no. 6, pp. 962–977, Nov./Dec. 2018.

[22] C. D. Francescomarino, C. Ghidini, F. M. Maggi, and F. Milani, "Predictive process monitoring methods: Which one suits me best?," in *Proc. 16th Int. Conf. Bus. Process Manage.*, 2018, pp. 462–479.

[23] N. Harane and S. Rathi, "Comprehensive survey on deep learning approaches in predictive business process monitoring," in *Studies in Computational Intelligence*. Berlin, Germany: Springer, 2020, pp. 115–128.

[24] B. A. Tama and M. Comuzzi, "An empirical comparison of classification techniques for next event prediction using business process event logs," *Expert Syst. Appl.*, vol. 129, pp. 233–245, 2019.

[25] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, and I. Teinemaa, "Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 1–34, 2019.

[26] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, "Outcome-oriented predictive process monitoring," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 2, pp. 1–57, 2019.

[27] W. Kratsch, J. Manderscheid, M. Röglinger, and J. Seyfried, "Machine learning in business process monitoring: A comparison of deep learning and classical approaches used for outcome prediction," *Bus. Inf. Syst. Eng.*, vol. 63, pp. 261–276, 2020.

[28] D. A. Neu, J. Lahann, and P. Fettke, "A systematic literature review on state-of-the-art deep learning methods for process prediction," *Artif. Intell. Rev.*, Mar. 2021, doi: 10.1007/s10462-021-09960-8.

[29] H. Weytjens and J. D. Weerdt, "Process outcome prediction: CNN versus LSTM (with attention)," in *Proc. Bus. Process Manage. Workshops - BPM Int. Workshops, Seville, Spain, Revised Sel. Papers*, 2020, pp. 321–333.

[30] K. Heinrich, P. Zschech, C. Janiesch, and M. Bonin, "Process data properties matter: Introducing gated convolutional neural networks (GCNN) and key-value-predict attention networks (KVP) for next event prediction with deep learning," *Decis. Support Syst.*, vol. 143, Apr. 2021, Art. no. 113494.

[31] N. Mehdiyev, J. Evermann, and P. Fettke, "A multi-stage deep learning approach for business process event prediction," in *Proc. IEEE 19th Conf. Bus. Inform.*, 2017, pp. 119–128.

[32] N. Navarin, B. Vincenzi, M. Polato, and A. Sperduti, "LSTM networks for data-aware remaining time prediction of business process instances," in *Proc. IEEE Symp. Ser. Comput. Intell.*, 2017, pp. 1–7.

[33] A. Al-Jebrni , H. Cai, and L. Jiang, "Predicting the next process event using convolutional neural networks," in *Proc. IEEE Int. Conf. Prog. Inform. Comput.*, 2018, pp. 332–338.

[34] A. Metzger and A. Neubauer, "Considering non-sequential control flows for process prediction with recurrent neural networks," in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl.*, 2018, pp. 268–272.

[35] S. Schönig, R. Jasinski, L. Ackermann, and S. Jablonski, "Deep learning process prediction with discrete and continuous data features," in *Proc. 13th Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2018, pp. 314–319.

[36] N. Mehdiyev, J. Evermann, and P. Fettke, "A novel business process prediction model using a deep learning method," *Bus. Inf. Syst. Eng.*, vol. 62, pp. 143–157, 2018.

[37] M. Camargo, M. Dumas, and O. G. Rojas, "Learning accurate LSTM models of business processes," in *Proc. 17th Int. Conf. Bus. Process Manage.*, 2019, pp. 286–302.

[38] L. Lin, L. Wen, and J. Wang, "Mm-Pred: A deep predictive model for multi-attribute event sequence," in *Proc. SIAM Int. Conf. Data Mining*, 2019, pp. 118–126.

[39] M. Hinkka, T. Lehto, and K. Heljanko, "Exploiting event log event attributes in RNN based prediction," in *Proc. 9th Int. Symp. Data-Driven Process Discov. Anal.*, 2019, pp. 67–85.

[40] J. Theis and H. Darabi, "Decay replay mining to predict next process events," *IEEE Access*, vol. 7, pp. 119787–119803, 2019.

[41] N. A. Wahid, T. N. Adi, H. Bae, and Y. Choi, "Predictive business process monitoring – remaining time prediction using deep neural network with entity embedding," *Procedia Comput. Sci.*, vol. 161, pp. 1080–1088, 2019.

[42] J. Wang, D. Yu, C. Liu, and X. Sun, "Outcome-oriented predictive process monitoring with attention-based bidirectional LSTM neural networks," in *Proc. IEEE Int. Conf. Web Serv.*, 2019, pp. 360–367.

[43] F. Folino, G. Folino, M. Guarascio, and L. Pontieri, "Learning effective neural nets for outcome prediction from partially labelled log data," in *Proc. IEEE 31st Int. Conf. Tools Artif. Intell.*, 2019, pp. 1396–1400.

[44] M. Hinkka, T. Lehto, K. Heljanko, and A. Jung, "Classifying process instances using recurrent neural networks," in *Proc. 16th Int. Conf. Bus. Process Manage. Workshops*, 2018, pp. 313–324.

[45] P. Philipp, R. Jacob, S. Robert, and J. Beyerer, "Predictive analysis of business processes using neural networks with attention mechanism," in *Proc. 2nd Int. Conf. Artif. Intell. Inf. Commun.*, 2020, pp. 225–230.

[46] H.-J. Cheng and A. Kumar, "Process mining on noisy logs — Can log sanitization help to improve performance?," *Decis. Support Syst.*, vol. 79, pp. 138–149, Nov. 2015.

[47] J. W. Graham, "Missing data analysis: Making it work in the real world," *Annu. Rev. Psychol.*, vol. 60, no. 1, pp. 549–576, Jan. 2009.

[48] M. S. Osman, A. M. Abu-Mahfouz , and P. R. Page, "A survey on data imputation techniques: Water distribution system as a use case," *IEEE Access*, vol. 6, pp. 63279–63291, 2018.
[49] M. Crawshaw, "Multi-task learning with deep neural networks: A survey," *CoRR*, vol. abs/2009.09796, 2020.
[50] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, 2018, pp. 889–898.
[51] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," in *Proc. 8th Int. Conf. Learn. Representations*, 2020.
[52] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, 1997.
[53] S. Ruder, "An overview of multi-task learning in deep neural networks," *CoRR*, vol. abs/1706.05098, 2017.
[54] R. Cipolla, Y. Gal, and A. Kendall, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7482–7491.
[55] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[56] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Proc. 19th Int. Conf. Neural Inf. Process. Syst.*, 2006, pp. 1137–1144.
[57] G. Alain and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3563–3593, 2014.
[58] J. Evermann, J. Rehse, and P. Fettke, "A deep learning approach for predicting process behaviour at runtime," in *Proc. 14th Int. Bus. Process Manage. Workshops*, 2016, pp. 327–338.
[59] M. C. Mozer, "A focused backpropagation algorithm for temporal pattern recognition," *Complex Syst.*, vol. 3, pp. 349–381, 1989.
[60] F. Gers, "Learning to forget: Continual prediction with LSTM," in *Proc. 9th Int. Conf. Artif. Neural Netw.*, 1999, pp. 850–855.
[61] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1724–1734.
[62] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.
[63] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of CNN and RNN for natural language processing," *CoRR*, vol. abs/1702.01923, 2017.
[64] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1412–1421.
[65] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.
[66] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014.
[67] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap, "Meta-learning with memory-augmented neural networks," *Proc. 33nd Int. Conf. Mach. Learn.*, vol. 48, pp. 1842–1850, 2016.
[68] A. G. *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.
[69] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[70] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. 28th IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[71] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc.*, T. Kobayashi, K. Hirose, and S. Nakamura, Eds. 2010, pp. 1045–1048.
[72] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1113–1120.
[73] "Score normalization," in *Encyclopedia of Biometrics*. New York, NY, USA: Springer, 2009, pp. 1134–1135.
[74] W. J. Scheirer, A. Rocha, R. J. Micheals, and T. E. Boult, "Robust fusion: Extreme value theory for recognition score normalization," in *Proc. 11th Eur. Conf. Comput. Vis.*, 2010, pp. 481–495.
[75] A. Leontjeva, R. Conforti, C. D. Francescomarino, M. Dumas, and F. M. Maggi, "Complex symbolic sequence encodings for predictive monitoring of business processes," in *Proc. 13th Int. Conf. Bus. Process Manage.*, 2015, pp. 297–313.
[76] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. Learn. Representations*, Y. Bengio and Y. LeCun, Eds., 2013.
[77] M. Peters *et al.*, "Deep contextualized word representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2018, pp. 2227–2237.
[78] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *J. Big Data*, vol. 7, no. 1, p. 28, 2020.
[79] B. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Struct.*, vol. 405, no. 2, pp. 442–451, 1975.
[80] C. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Res.*, vol. 30, pp. 79–82, 2005.
[81] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon, "Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis," *J. Mach. Learn. Res.*, vol. 18, pp. 77:1–77:36, 2017.
[82] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
[83] B. Calvo and G. Santafé, "Scmamp: Statistical comparison of multiple algorithms in multiple problems," *R J.*, vol. 8, no. 1, pp. 248–256, 2016.
[84] B. Calvo, J. Ceberio, and J. A. Lozano, "Bayesian inference for algorithm ranking analysis," in *Proc. Genet. Evol. Comput. Conf. Companion*, 2018, pp. 324–325.
[85] G. Corani, A. Benavoli, J. Demšar, F. Mangili, and M. Zaffalon, "Statistical comparison of classifiers through Bayesian hierarchical modelling," *Mach. Learn.*, vol. 106, no. 11, pp. 1817–1837, May 2017.
[86] M. Hutson, "Core progress in AI has stalled in some fields," *Science*, vol. 368, no. 6494, pp. 927–927, 2020.
[87] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst, "Process mining for python (PM4PY): Bridging the gap between process- and data science," *CoRR*, vol. abs/1905.06169, 2019.

Efrén Rama-Maneiro received the BEng degree in 2018 in computer engineering University of Santiago de Compostela, Spain, the MSc degree in 2019 in Big Data from the University of Santiago de Compostela, where he is currently working toward the PhD degree. He is currently a researcher with the Centro Singular de Investigación en Tecnoloxías Intelixentes, University of Santiago de Compostela. His research interests include process mining and deep learning.

Juan C. Vidal received the BEng degree in computer science from the University of La Coruña, La Coruña, Spain, in 2000 and the PhD degree in artificial intelligence from the University of Santiago de Compostela (USC), Santiago de Compostela, Spain, in 2010. He was an assistant professor with the Department of Electronics and Computer Science, University of Santiago de Compostela (USC), Santiago de Compostela, Spain, from 2010 to 2017. He is currently an associate researcher with the Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), USC. His research interests include process mining, fuzzy logic, machine learning, and linguistic summarization.

Manuel Lama received the PhD degree in physics from the University of Santiago de Compostela in 2000. He is currently an associate professor of artificial intelligence. He has collaborated on more than 30 projects and research contracts financed by public calls, participating as a principal investigator in 20 of them. He has authored or coauthored more than 150 scientific articles with review process in conference and national and international journals. His research interests include process discovery, predictive monitoring, and management of dynamic processes.