Process Model Forecasting Using Deep Temporal Learning*

Wenjun Zhou, Artem Polyvyanyy, and James Bailey

The University of Melbourne, Victoria VIC 3010, Australia
{wenjun.zhou;artem.polyvyanyy;baileyj}@unimelb.edu.au

Abstract. Process discovery studies ways to construct process models from event logs of historical executions of a system. While discovered models aim to describe the system, process model forecasting aims to construct models that faithfully describe the executions the system will perform in a given period in the future, informing timely system improvements. Existing approaches tackle the problem of process model forecasting by decomposing it into multiple univariate time series forecasting problems. They forecast each directly-follows constraint over a pair of process activities separately and then aggregate these individual forecasts into the resulting process model. In this paper, we propose a deep learning-based approach that leverages multivariate time series forecasting to solve the process model forecasting problem. Our method learns dependencies across all activity constraints simultaneously, generating an integrated forecast of the entire model at once. Through evaluation over industrial event logs, we demonstrate that this approach significantly outperforms existing baselines and statistical multivariate methods in accuracy. Additionally, we introduce a new measure to evaluate the structural correctness of the forecasted models. In the context of information systems engineering, our work addresses the challenge of predicting process models to support future process planning and optimization.

Keywords: Process mining · process model forecasting · deep learning

1 Introduction

Business Process Management (BPM) is a key area within information systems engineering, focusing on the design, execution, and optimization of business processes to improve organizational efficiency and effectiveness [28]. Process mining is a subarea in BPM that studies ways to use event logs recorded by information systems to understand and improve these systems [25]. An *event log* is a collection of traces, each recorded as a sequence of executed activities by a system, for instance, during an execution of a business process. Within the process mining discipline, *process discovery* addresses the problem of constructing a process model from an event log of a system that *describes the executions the system can support*, where a process model is a conceptual model composed of activities, routing decisions, and control flow that captures the ordering constraints over the activities and decisions. A *directly-follows graph* (DFG) is a process model often constructed by process discovery algorithms. It is a directed graph

* This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

with nodes capturing activities, directed arcs specifying possible orders in which the activities can be executed, and numbers on nodes and arcs suggesting the frequencies with which the corresponding concepts can be executed. DFGs are discovered by most of the commercial process mining tools [26].

Recently, the problem of process model forecasting has been introduced [21]. Given an event log of a system and a time interval in the future, *process model forecasting* studies ways to construct a process model that *describes the executions the system will perform in the given time interval* [29]. Such a forecast, if accurate, can enhance organizations' understanding of their future business processes, allowing targeted planning for redesign and support initiatives. The state-of-the-art technique for process model forecasting tackles this problem by dividing it into multiple univariate time series forecasting sub-problems, one for each directly-follows (DF) constraint (an arc in a DFG), solving these sub-problems, and aggregating the results into the final forecasted DFG [8].

In this paper, we present a Deep Learning (DL) approach —*DeePMF* for process model forecasting that leverages multivariate time series forecasting. Instead of forecasting each DF constraint separately, *DeePMF* learns dependencies across all constraints simultaneously and then generates the forecasted process model at once. We demonstrate that this approach significantly outperforms existing baselines and statistical multivariate time series methods in accuracy. We also introduced a new measure of structural correctness of forecasted models and confirm that *DeePMF* constructs models of good structural characteristics.

Specifically, this paper makes these contributions:

1. A *sparsity test* for event logs that helps determine if an event log could be used to forecast accurate process models;
2. A measure of the level of *consistency* of a DFG that quantifies by how much the sum of incoming and outgoing arc frequencies differs for its activity nodes;
3. The *DeePMF approach to process model forecasting* grounded in DL multivariate time series forecasting techniques that delivers the state-of-the-art forecasting accuracy across multiple real-life datasets;
4. A comprehensive *evaluation* of *DeePMF* over a wide range of industrial event logs that confirms the effectiveness of our approach, suggesting that the transformer architecture often leads to better forecasts.

The next section discusses related work. Section 3 provides the concepts and background knowledge that supports the understanding of the subsequent sections. Section 4 presents our process model forecasting approach, while Section 5 presents evaluation setup and results. Section 6 discusses limitations and ideas for future work before Section 7 draws final conclusions.

2 Related Work

Predictive Process Monitoring (PPM) studies ways to predict future states, outcomes, and key performance indicators of business processes based on data from event logs [10]. PPM techniques learn historical patterns and then extrapolate the learned principles beyond a given event log. As part of the next process state prediction, PPM techniques can predict the next activity, or groups of activities, that will be performed in a given incomplete business process execution. Existing techniques that tackle this problem achieve

high prediction accuracy using conventional statistical and process analysis [18, 24, 27], and DL [11, 12, 15, 23] methods. Rather than predicting aspects of a currently running business process, process model forecasting (PMF) aims to construct a model that describes future executions from a requested period [29]. This fundamental difference between PPM and PMF makes the artifacts they produce not directly comparable.

Our process model forecasting work is inspired by the work by De Smedt et al. [7]. They compared the effectiveness of the statistical time series forecasting techniques for forecasting DF constraints. The forecasting technique proceeds by splitting the event log into equitemporal or equisized periods, calculating frequencies of observed DF constraints for each period, and forecasting each DF constraint using univariate time series forecasting for the series of its frequencies stemming from the different periods of the event log. They evaluated five statistical time series forecasting techniques, namely naïve average (Naïve), auto-regressive integrated moving average (ARIMA) with the order of $(2, 1, 2)$, auto-regressive (AR) with the order of (2), Holt-Winters' model (HW), and generalized auto-regressive conditional heteroskedasticity (GARCH). They then evaluated the mean percentage error in terms of entropic relevance [2] between the ground truth future DFGs and the DFGs assembled from the forecasted constraints. In a follow-up work, they evaluate the vectorized auto-regressive (VAR) model with the order of (1) [8]. In our work, we forecast all DF constraints, and thus the DFG that describes the requested future executions, at once using multivariate time series forecasting and demonstrate that this approach leads to forecasted DFGs of superior accuracy. De Smedt et al. [8] report that techniques that perform the best on average are HW, AR, and the naïve average, while VAR performs worse than other techniques except for one dataset. In our experiments, we replicated ARIMA with the working order of $(1, 1, 1)$, AR with the order of (2), Naïve, and HW as univariate baselines, included VAR with the order of (1) as a multivariate baseline, and further introduced identity function (Identity) as another baseline.

While our techniques demonstrate improved forecasting of DF constraints, their interpretability is an ongoing research. van der Aalst [26] highlights potential inconsistencies that can arise after filtering DFGs, leading to misinterpretations by analysts. Leemans et al. [19] adapts and discusses the soundness property for DFGs, aiming to ensure their correctness. We propose *consistency* as a new measure of DFGs correctness that evaluates how well the forecasted DF constraints align with human interpretability, focusing on the balanced in-flow and out-flow of arc frequencies across nodes.

3 Preliminaries

This section introduces notions used in the discussions in the subsequent sections.

An *event* is a collection of attribute-value pairs comprising at least three elements storing values of case ID, activity, and timestamp attributes [25]. The case ID, activity, and timestamp attributes of an event specify the instance, or *case*, identifier of the business process that triggered the event, the activity that triggered the event, and the timestamp at which the event was recorded. An *event log*, or a *log*, is a collection of events recorded during the execution of multiple instances, or *cases*, of a business process. An example event log L is shown in Table 1, where each row specifies one event with the attribute values specified in the corresponding columns. The activities that triggered all

Case ID	*Activity*	*Timestamp*
1	a	1
2	a	2
1	b	3
2	b	4
2	b	5
2	d	6
1	c	7

Table 1: Event log L.

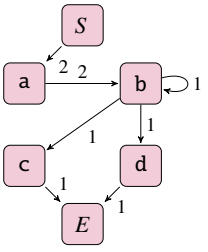

Fig. 1: DFG of L.

	a	b	c	d	E
a	0	2	0	0	0
b	0	1	1	1	0
c	0	0	0	0	1
d	0	0	0	0	1
S	2	0	0	0	0

Fig. 2: DFM of L.

the events with the same case identifier ordered by the timestamps of the corresponding events constitute a *trace*. Event log L contains two traces: $\langle a,b,c\rangle$ and $\langle a,b,b,d\rangle$.

A *Directly-Follows Graph (DFG)* is a process model often constructed from an event log to describe the process that generated the event log [26]. A DFG is also a weighted directed graph with two special nodes denoting the start and end of the process and other nodes annotated with activities. The arcs of a DFG are defined by the DF constraints in the event log, which comprise all pairs of consecutively followed activities in the traces of the event log. For instance, the DF constraints of event log L are defined by the set $\{(S,a),(a,b),(b,b),(b,c),(b,d),(c,E),(d,E)\}$. The start node has no incoming arcs, while the end node has no outgoing arcs. The outgoing arcs of the start node target nodes that represent activities that appear at the start of the traces. The incoming arcs of the end node originate from nodes that represent activities that appear at the end of the traces. In addition, arcs of a DFG are annotated with weights that reflect the frequencies with which the corresponding DF constraints, that is, subsequent executions of the activities, appear in the traces. For instance, the arc $(S,\mathtt{a})$ has a weight of two, as both traces in the event log start with activity a, while arc (a,b) has a weight of two because activity b follows immediately activity a two times in the traces of L. Figure 1 shows the DFG constructed from the traces of event log L.

A *Directly-Follows Matrix (DFM)* is an adjacency matrix used to provide an alternative representation of a DFG. In a DFM, rows represent all the DFG nodes except the end node, while columns represent all the DFG nodes except the start node. Each entry in the matrix specifies the weight of the DFG arc from the corresponding row's node to the corresponding column's node. As the start node has no incoming arcs and the end node has no outgoing arcs, the corresponding column and row are omitted in the DFM. The DFM in Figure 2 is an alternative representation of the DFG in Figure 1.

Time Series Forecasting (TSF) studies techniques to predict future values of the time series given the historical data [6]. A time series is a series of discrete data measurements often ordered in regular time intervals. A univariate TSF uses a single time series data to perform forecasts, while a multivariate TSF analyses the dependencies between multiple time series to generate future values in these series simultaneously. Some established statistical univariate TSF approaches include AR and ARIMA. Popular multivariate TSF techniques include VAR and Vectorized ARIMA (VARIMA) [14, 20, 22].

4 Approach

This section presents the problem of process model forecasting, describes our data selection principle that aims to ensure accurate forecasting, summarizes our forecasting approach, and presents the way we evaluate the results of our forecasts.

4.1 Problem Definition

Let $\mathcal{L}$ be the universe of event logs, let $\mathcal{T}$ be the universe of timestamps, and let $\mathcal{M}$ be the universe of process models. By $\mathcal{P} = \{(s, e) \in \mathcal{T} \times \mathcal{T} \mid e > s\}$ we denote the universe of time periods, where period (s, e) starts at timestamp s and completes at timestamp e.

Given an event log $L \in \mathcal{L}$ and a period $P \in \mathcal{P}$ in the future relevant to L, the *process model forecasting* problem consists in constructing a process model that describes the executions the system that generated L will perform during period P. That is, a solution to process model forecasting can be given as a function $f : \mathcal{L} \times \mathcal{P} \rightarrow \mathcal{M}$, such that for each $(L, (s, e)) \in f$ it holds that $latest(L) \leq s$; by $latest(L)$, we refer to the latest timestamp in L, that is, the maximum timestamp value among timestamps of all the events in L. This is different from the classical process discovery problem studied in process mining that aims to construct a process model that describes all the executions of the system that generated the event log can perform in the period $(-\infty, +\infty)$. In this work, we use process discovery outputs (process models) as inputs to forecasting, and thus the quality of the discovery algorithm is crucial to the success of forecasting.

In this work, we study a restricted version of the process model forecasting problem $\hat{f}$ that aims to construct models that describe executions the system will perform in periods immediately after the latest timestamp in the event log, that is, $\hat{f} : \mathcal{L} \times \mathcal{P} \rightarrow \mathcal{M}$, such that for each $(L, (s, e)) \in \hat{f}$ it holds that $latest(L) = s$.

4.2 Data Selection

It is unrealistic to assume that every event log can support accurate process model forecasting. Event logs may suffer from issues such as insufficient data, poor quality, inconsistency, or sparsity. For example, forecasts based on empty or minimal logs are likely no better than random guesses. In contrast, large event logs collected over extended periods are more likely to capture critical process features—such as trends, recurring irregularities, and seasonality—thus enabling more reliable and meaningful forecasting.

Over the past two years of iterative experimentation and refinement, guided by the design science methodology, we developed the following three criteria for event logs. These criteria are designed to balance the simplicity of imposed requirements with their strong relationship to forecasting accuracy.

1. *Correctness*. An event log must conform to the designated format (e.g., XES).
2. *Completeness*. Every event in an event log must have valid values for three mandatory attributes: case ID, activity, and timestamp.
3. *Density*. An event log should include a sufficient number of occurrences for events representing different activities across its duration.

We operationalize the correctness and completeness checks for event logs using the Disco tool [13]. Specifically, we check if there are no errors or warnings reported when

Algorithm 1: SparsityTest

Input: M – a non-empty list M of DFMs, each of size $n \times n$, $n \in \mathbb{N}$; it – individual matrix sparsity threshold; tt – total sparsity threshold

Output: Result of the sparsity test for DFMs M

```
passed ← false;                                  /* initialize test result to false */
count ← 0;                                       /* number of sparse DFMs */
for k ∈ [1 .. |M|] do                            /* for each position k in M */
    sparsity ← Σ_{i=1}^{n} Σ_{j=1}^{n} 𝕀(M(k)_ij) / n²;   /* share of zero entries in M(k) */
    value ← Σ_{i=1}^{n} Σ_{j=1}^{n} M(k)_ij / n²;        /* average value in M(k) */
    if sparsity > it ∧ value < it then           /* if M(k) is sparse ... */
        count ← count + 1;                       /* ... then increment count */
if count / |M| < tt then                         /* if share of sparse DFMs below threshold ... */
    passed ← true;                               /* ... then test passed */
return passed;                                   /* return test result */
```

loading an event log in Disco. To ensure validity, the values of the mandatory event attributes were examined manually. To check the data is sufficient to yield meaningful forecasts, we designed a sparsity test described in Algorithm 1. This test evaluates whether the chosen time window is coarse enough to minimize empty time series DFMs, thereby ensuring the forecasts remain both meaningful and necessary.

In the algorithm, $\mathbb{I}(.)$ is the indicator function, such that $\mathbb{I}(x = 0) = 1$; otherwise $\mathbb{I}(x) = 0$. Also, if M is a sequence, then $M(i)$ is the element at position i in M. The algorithm takes a list of DFMs as input. A *time window*, or a *lag*, represents a period of time. We assume that the duration of an event log (the period between the earliest and the latest timestamps of all its events) is split into a number of consecutive time windows, denoted by $\#lag$, each of the same duration. Given an event log and a lag size, we compute a sequence of DFMs, one DFM for each time window, in which DFMs are ordered according to the order of the corresponding time windows. This procedure is sketched in Figure 3 for a sample event log and $\#lag = 3$; each day defines a time window. The obtained sequence of DFMs is then subject to the sparsity test. The test checks if the number of sparse DFMs in the input sequence is below the total sparsity threshold (tt). A DFM is defined as sparse if the share of its entries that are zeros is above the individual matrix sparsity threshold (it) and the average entry in the DFM is below this threshold. Empirically, we established that DFM sequences that pass the sparsity test for thresholds $it = 0.98$ and $tt = 0.2$ lead to meaningful forecasts.

4.3 Forecasting

We follow a similar approach to process model forecasting as De Smedt et al. [7]. An event log is prepared for training a process forecasting model in the same way as during data selection. Referring to the example in Figure 3, the event log lasts for three days, and to simplify the example, to split the event log, we use $\#lag = 3$. Hence, each lag contains all the events recorded on a particular day. We then use PM4Py [4] to discover DFGs from (fragments of) traces from each lag and represent these DFGs as DFMs.

We use DFM sequences to train DL multivariate time series forecasting models. We identified six DL models used in time series forecasting that can be trained on DFM

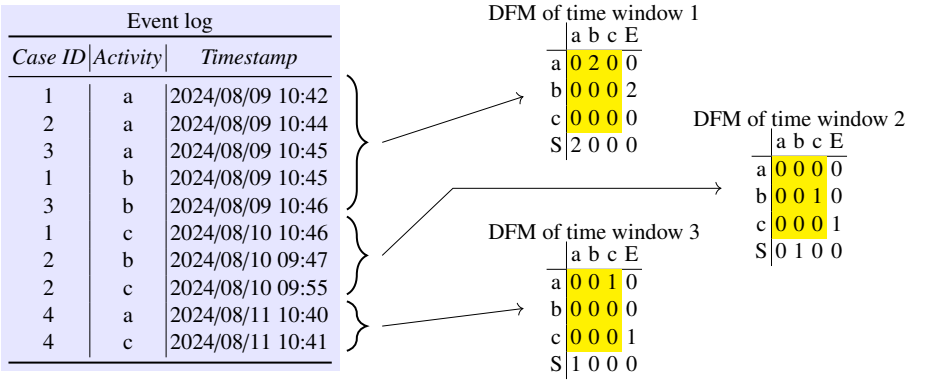

Fig. 3: Event log prepared as DFM time series (#*lag* = 3).

sequences: Vanilla, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), and transformer [16, 17]. The Vanilla model is a fully connected deep neural network (NN) with a sequence of two linear layers, each followed by a ReLU activation function, and it is concluded with an additional linear layer as the output layer, providing the final predictions without an activation function. The CNN model is similar to the Vanilla model and differs in that it uses 1D convolutional layers instead of linear layers, while it still encompasses an additional linear layer as the output layer. The RNN model is an Elman RNN architecture with two RNN layers and a final linear output layer. The GRU and LSTM models use the same architecture as the RNN model. The transformer model is the default PyTorch transformer model without any additional layers.

To ensure that the comparison between NN models and baselines is fair, in this work, we fixed all models' (both NN models and statistical models) horizon (#*horizon*) and look-back window (#*lookback*) to be equal to one; the same configuration was used by De Smedt et al. [8]. In time series forecasting, a *horizon* is the number of time windows the method forecasts, while a look-back window defines how many previous time windows are used by the forecasting model to come up with a forecast.

4.4 Testing

We further describe how we split the data into 10 folds to perform a 10-fold cross-validation of our forecasting approach. After constructing the series of DFMs over time,

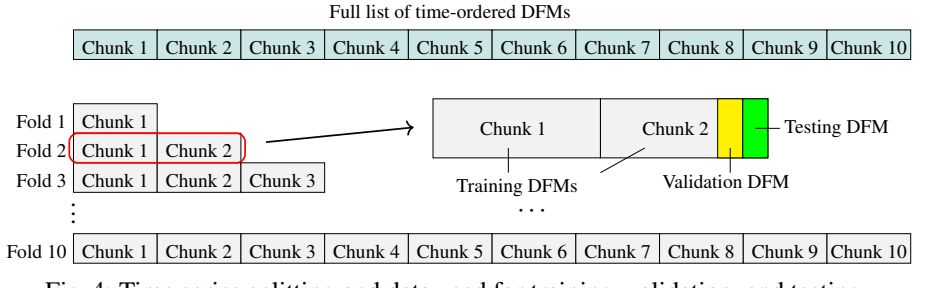

Fig. 4: Time series splitting and data used for training, validation, and testing.

we use these matrices as inputs to train the selected NN models, as well as the baseline models. We first split the ordered time windows from the earliest to the latest into 10 equal chunks. Then, to construct the *i*-th fold, we take all the DFMs starting from the first DFM in the entire series of DFMs up to and including all the DFMs from the *i*-th chunk; this is a standard approach for splitting time series data for cross-fold validation. Figure 4 visualizes this splitting process. Since the forecasting horizon we use is set to one, we use the last DFM in each fold as the testing ground truth, and we set the DFM before the testing DFM as our validation DFM. All the other DFMs are used to train the forecasting NN models. Specifically, for each fold, we train the NN models from scratch and use Optuna [1] to report on the hyperparameter combinations that have the lowest loss on the validation DFM. Since Optuna does not keep the model state, we retrain the model using the reported best hyperparameters after 50 trials of hyperparameter search.

To further improve the accuracy of our forecasts, we also experimented with two approaches: modifying every model architecture by adding a ReLU layer at the end of each model (the transformer model also introduced an additional linear layer for applying the ReLU activation) and post-processing the prediction by taking the maximum frequency between zero and the predicted values.

5 Evaluation

This section presents the datasets used in our evaluation, our implementation of the approach and experimental setup, quality measure used to assess the performance of our process model forecasting techniques, and reports the results of the conducted forecasts.

5.1 Datasets

We explained our data selection steps and criteria in Section 4.2 and applied them to all event logs made publicly available by the IEEE Task Force on Process Mining[1]. To increase the number of event logs suitable for analysis, we also manually truncated and filtered some logs that have a long idle period at the beginning or the end of the logging period. We use '_f' to indicate that the log has been truncated and filtered; for the events belonging to the cases outside the included period, we removed those events. During the data preparation stage, we read the event logs and retrieved the earliest and latest timestamps for each dataset. We further sliced the duration of the earliest to the latest timestamp into a number of equal time windows as described in Section 4.3. Then, for the events in each period, we use PM4Py [4] with default settings to discover the DFG of that period, and we further turn these DFGs into equivalent DFMs. The obtained lists of DFMs were used as inputs to our sparcity test (Algorithm 1). Ten event logs passed the test. These are Hospital Billing (hb), Road Traffic Fine Management Process (rtfmp), Sepsis Cases (sepsis), BPI Challenge 2017 (bpic17), the help desk log of an Italian company (helpdesk), BPI Challenge 2019 (bpic19), BPI Challenge 2013 Closed Problems (bpic13c), BPI Challenge 2012 (bpic12), NASA Crew Exploration Vehicle Software Event Log (nasacs), and BPI Challenge 2013 Open Problems (bpic13o) event logs. The characteristics of these event logs are summarized in Table 2. Note that not every event log passes the sparsity test for all chosen lag sizes.

[1] https://www.tf-pm.org/resources/logs

Table 2: Event logs and their characteristics.

Log name	*Events*	*Traces*	*Activities*	*Earliest timestamp*	*Latest timestamp*
hb	451,359	100,000	18	2012-12-13 20:13:18	2016-01-19 18:58:56
rtfmp	561,470	150,370	11	2000-01-01 10:00:00	2013-06-18 08:00:00
sepsis_f	14,766	1,025	16	2013-11-07 18:18:29	2015-02-28 04:00:00
bpic17	1,160,405	31,509	26	2016-01-01 20:51:15	2017-02-02 01:11:03
helpdesk	21,348	4,580	14	2010-01-13 08:40:25	2014-01-03 13:20:58
sepsis	15,214	1,050	16	2013-11-07 18:18:29	2015-06-05 20:25:11
bpic19_f	1,588,420	251,478	11,879	2018-01-01 09:59:00	2019-01-19 00:34:00
bpic13c_f	6,483	1,456	7	2010-01-06 02:42:20	2012-06-01 07:49:06
bpic12	262,200	13,087	36	2011-10-01 08:38:45	2012-03-15 02:04:55
nasacs	36,819	2,566	47	2017-02-14 01:50:52	2017-02-14 01:50:56
bpic13o_f	2,319	812	5	2010-01-14 20:34:54	2012-06-15 20:19:56

5.2 Implementation and Experimental Setup

We split each event log into different numbers of time windows. Specifically, we use #*lag* of 100, 300, 500, 700, and 1,000. All the experiments were conducted on the University of Melbourne supercomputing platform—Spartan. Our experiments were implemented in Python 3.9.19. We configured the Pytorch DL framework in the Anaconda3 (2022.10) environment with CUDA (12.2.0). Table 3 summarizes the platform specification as well as other Python packages and version information.

We used Optuna [1] as a hyperparameter tuning tool. The hyperparameter values we used are listed in Table 3. We fix the other parameters including the number of layers, batch size, and kernel size (for the CNN model) to two, one, and one, respectively. For each NN model, we set Optuna to try 50 trials for each fold, and we picked the hyperparameters that returned the lowest loss on the validation dataset for training the final NN model. For each dataset, the NN models were optimized for 10 folds. The implementation of our experiments, including the data preparation, training, evaluation and result analysis, is publicly available.[2]

Table 3: Platforms, packages and hyperparameters.

Processor: Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz.				
Memory: 1,000GB (only utilised 16GB in our experiments).				**Cores:** 32.
GPU memory: 80GB GPU RAM per GPU.				**GPU type:** A100.
torchaudio: 2.4.0	torchvision: 0.19.0	pm4py: 2.7.11.13	scikit-learn: 1.5.1	optuna: 3.6.1
torch: 2.1.0.dev20230621+cu117		numpy: 1.26.4	pandas: 2.2.2	scipy: 1.13.1

Optimizer	Adam, SGD
Loss function	L1Loss, MSELoss, SmoothL1Loss
Hidden size	121, 196, 256, 324, 1296
Epochs	1000 to 2000
Learning rate	0.001 to 0.01
Dropout probability	0.1 to 0.3

[2] https://github.com/zhoudayun81/DeePMF

5.3 Quality Measures

To evaluate the accuracy of our forecasts, we rely on the commonly used mean absolute error (MAE) measure. Specifically, we measure the errors of the entries in the forecasted DFM (*Forecasted_DFM*) with respect to the corresponding entries in the ground truth DFM (*Ground_Truth_DFM*). We calculate MAE for each fold and average the mean performance of each NN model for the 10 folds for each dataset.

Below, we detail the computation of MAE for an event log with n unique activities:

$$Ground\_Truth\_DFM \in \mathbb{N}^{n \times n}$$

$$Forecasted\_DFM \in \mathbb{Z}^{n \times n}$$

$$\text{MAE} = \frac{1}{(n+1)^2} \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} \left| Ground\_Truth\_DFM_{ij} - Forecasted\_DFM_{ij} \right|$$

We propose a consistency measure of DFG quality that quantifies how well the sum of the frequencies of the arcs entering and leaving its activity nodes match. For the *start* node, which only has outgoing arcs, the frequency should be the same as the frequency of arcs entering the *end* node. We require that a DFG has at least one arc. If a DFG has exactly one arc, this is the arc from its start node to its end node.

We compute *consistency* of a DFG with n activity nodes over its DFM as follows.

$$\text{row\_sum}_i = \sum_{j=1}^{n+1} \max(DFM_{ji}, 0) \quad \text{(outgoing sum for node } i\text{)} \tag{Eq. 1}$$

$$\text{column\_sum}_i = \sum_{j=1}^{n+1} \max(DFM_{ij}, 0) \quad \text{(incoming sum for node } i\text{)} \tag{Eq. 2}$$

$$\text{Consistency} = \frac{1}{n+1} \sum_{i=1}^{n+1} \frac{\min(\text{row\_sum}_i, \text{column\_sum}_i)}{\max(\text{row\_sum}_i, \text{column\_sum}_i)} \tag{Eq. 3}$$

Firstly, we calculate the sum of outgoing (Eq. 1) and incoming (Eq. 2) arcs for each node i of the DFG. As a forecasted DFM can, in general, contain negative entries, if a negative value is encountered when computing the sum for a row or column, it is replaced with the value of zero. If the outgoing, as well as the incoming arcs' frequencies, sum up to zeros, to avoid the division by zero problem, the ratio for the node in Eq. 3 is accepted to be equal to one. In the DFG, this is interpreted as the node does not exist in the graph. Consequently, it holds that consistency is a value between zero and one, with larger values signifying a higher degree of consistency.

There are several design options to evaluate the quality of the DFG. The reason for our choice of this measure is twofold. First, the measurement is bounded between zero and one. Second, the measurement values can be compared between different datasets.

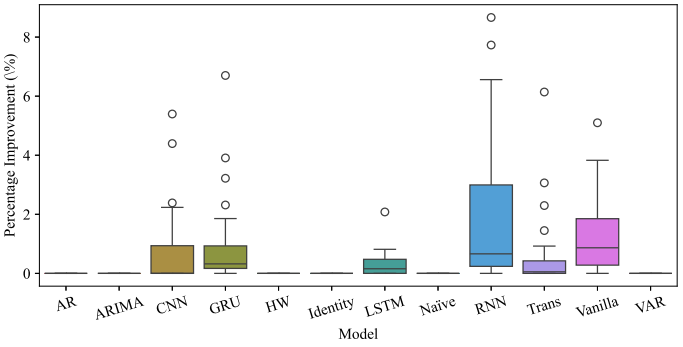

Fig. 5: Percentage improvement of MAE after applying normalization.

5.4 Results

Due to the page limit, we could not fully present the large-scale experiments conducted and the extensive data collected. The interested reader can access the complete experimental outputs and results in our GitHub repository, specifically in the *output* and *result* folders. Several statistical models we explored assume stationarity in the data, a requirement that the majority of the training DFMs do not meet. As a result, some models failed to converge during training. Consequently, we only report results for ARIMA with a modified order $(1, 1, 1)$, AR(2), HW, VAR(1), naïve average (Naïve), and the identity function (Identity). Although we attempted to apply a vectorized ARIMA model (VARIMA), none of the tested orders worked with our datasets.

In Table 4, we report on the mean MAE for each model over 10 folds for each dataset; the number of time windows used to split the dataset is annotated in the brackets next to the dataset name and we bold the best (lowest) values for each dataset. Table 4 provides several insights. First, transformer (Trans) has a superior lower MAE on the majority of the datasets comparing to the baselines, which are mostly more than 50% improvement in MAE comparing to the multivariate baseline VAR. Second, transformer almost consistently performs not good on hb and bpic17 datasets. Third, VAR model yields most of the poorest results and cannot even compete with the simple baselines (Identity and Naïve), hence we do not recommend using VAR for multivariate time series process model forecasting. Finally, when the lag size is small (e.g., $\#lag = 100$), the NN models have less advantage to win over the identity function. The reported results are the best after post-processing the predictions as described in Section 4.3. The percentage improvement of each model after post-processing is described in Figure 5. It is clear that RNN benefits the most from this improvement, which also is an indication that the original RNN model quality can be poor. As we also explored adding a ReLU layer to ensure all the values in the forecasted DFM are positive, it is surprising that this approach deteriorates the forecasting accuracy.

To further analyze the impact of the lag on the forecasting results, we also plot the average rankings for all datasets in different lag groups in Figure 6. Figure 6 implies that with the greater number of time windows used for training, the transformer model has better performance, while the univariate baselines, as well as the identity function, lose their advantage with finer time windows. This makes sense and aligns with the results in Table 4, where the majority of the best results in the baselines are from the

Table 4: Average MAE of All Models (rounded to 3rd place decimal).

	DeePMF						*Baselines*					
Dataset	*Trans*	*RNN*	*LSTM*	*GRU*	*CNN*	*Vanilla*	*Identity*	*Naïve*	*VAR*	*ARIMA*	*HW*	*AR*
hb(100)	5.344	4.996	4.153	4.253	15.712	5.788	**3.164**	7.584	10.653	3.188	3.062	3.357
rtfmp(100)	**47.098**	48.581	49.565	53.098	63.303	103.747	73.357	53.288	280.804	54.324	54.276	54.459
sepsis_f(100)	**0.274**	0.304	0.362	0.305	0.309	0.328	0.387	0.325	0.560	0.336	0.328	0.338
bpic17(100)	4.907	4.780	4.485	4.594	10.126	6.075	7.219	6.521	8.414	**4.331**	5.188	4.836
helpdesk(100)	0.683	0.656	0.568	0.612	0.564	0.656	**0.546**	0.784	1.828	0.630	0.613	0.588
sepsis(100)	**0.297**	0.326	0.376	0.352	0.311	0.316	0.359	0.439	0.503	0.335	0.326	0.382
bpic19_f(100)	6.077	**5.733**	6.383	6.328	11.569	10.606	8.761	6.761	12.835	6.265	6.615	5.952
bpic13c_f(100)	**0.602**	0.761	0.809	0.744	0.755	0.777	0.711	1.272	0.941	0.731	0.714	0.730
bpic12(100)	1.432	1.265	1.243	1.420	1.591	1.434	1.511	1.245	2.200	1.225	**1.193**	1.200
nasacs(100)	0.121	0.137	0.134	0.127	0.141	0.135	**0.120**	0.147	0.232	0.125	0.122	0.135
bpic13o_f(100)	**0.642**	0.656	0.664	0.658	1.144	0.847	0.894	1.289	1.041	0.651	0.689	0.770
hb(300)	2.010	1.651	1.580	1.650	1.673	1.683	1.557	2.944	4.237	1.379	**1.358**	1.433
rtfmp(300)	**15.475**	16.617	17.296	16.835	17.656	18.049	24.146	20.801	136.940	18.565	17.549	18.054
sepsis_f(300)	**0.102**	0.161	0.169	0.172	0.173	0.161	0.189	0.165	0.421	0.178	0.174	0.185
bpic17(300)	2.322	2.605	2.817	2.627	2.756	2.767	3.397	2.857	95.430	2.368	**2.292**	2.617
helpdesk(300)	**0.169**	0.312	0.318	0.328	0.262	0.324	0.316	0.339	0.395	0.301	0.297	0.305
sepsis(300)	**0.107**	0.160	0.157	0.158	0.145	0.145	0.178	0.186	0.385	0.164	0.163	0.188
bpic19_f(300)	3.993	**3.535**	3.932	3.590	4.277	4.453	4.913	3.957	1,440.695	3.907	3.994	3.990
bpic13c_f(300)	**0.172**	0.464	0.352	0.394	0.469	0.452	0.650	0.405	0.564	0.422	0.440	0.380
bpic12(300)	**0.454**	0.546	0.482	0.499	0.563	0.565	0.665	0.573	5.596	0.538	0.548	0.520
hb(500)	1.267	1.221	1.110	1.097	1.901	0.925	1.060	1.983	9.989	**0.820**	0.824	0.938
rtfmp(500)	**8.583**	10.252	9.751	10.678	10.367	12.209	17.915	14.020	13.468	11.927	11.387	11.758
sepsis_f(500)	**0.057**	0.110	0.123	0.111	0.101	0.120	0.127	0.115	0.256	0.129	0.127	0.132
bpic17(500)	**1.745**	1.928	1.836	1.929	2.309	2.215	1.958	2.439	54.097	1.831	2.031	2.090
helpdesk(500)	**0.095**	0.221	0.186	0.240	0.177	0.269	0.235	0.238	0.265	0.204	0.198	0.226
sepsis(500)	**0.065**	0.095	0.105	0.107	0.097	0.112	0.136	0.117	0.248	0.113	0.113	0.129
hb(700)	0.781	0.764	0.836	**0.699**	0.907	0.871	0.978	1.554	1.063	0.802	0.806	0.967
rtfmp(700)	**6.852**	8.246	8.838	8.324	8.624	11.401	16.510	11.636	9.358	9.844	9.583	9.870
sepsis_f(700)	**0.034**	0.082	0.088	0.103	0.091	0.096	0.108	0.092	0.171	0.102	0.102	0.106
bpic17(700)	**1.591**	1.631	1.492	1.616	1.941	1.792	1.759	2.261	5.785	1.779	2.065	1.943
helpdesk(700)	**0.063**	0.164	0.164	0.168	0.184	0.180	0.200	0.191	0.196	0.169	0.164	0.186
hb(1000)	0.528	**0.523**	0.708	0.570	0.986	0.633	0.698	1.092	1.035	0.610	0.605	0.641
rtfmp(1000)	**5.458**	6.730	8.117	7.832	7.348	7.109	12.595	9.485	8.221	7.872	8.008	7.680
sepsis_f(1000)	**0.019**	0.064	0.075	0.073	0.068	0.073	0.100	0.072	1.41E+09	0.082	0.083	0.079
bpic17(1000)	1.397	1.442	1.426	**1.276**	1.664	1.916	1.695	1.798	24.706	1.477	1.679	1.617

identity function for smaller lags. This infers that for a greater time span, it may not be suitable to apply time series techniques for process model forecasting, as the nuances are usually hidden by emergent global behaviors. Alternatively, a univariate forecasting model may be sufficient and cost-effective for a smaller lag DFM forecast.

We then ranked the models' performance, calculated the mean ranking and generated the critical difference diagram over all the datasets, refer to Figure 7. The critical difference diagram was proposed by Demšar [9] and further refined by Benavoli et al. [3]. To compute statistics, we used the default value of alpha of 0.05. Figure 7 shows that transformer and RNN DL models (Elman RNN, LSTM, GRU) perform substantially better than the simple baselines and statistical models and significantly better than the VAR model. Although the critical differences between the *DeePMF* models and the univariate baselines are blended, the actual ranking difference between transformer (2.7) and the best baseline ARIMA (5.4) is significant. Looking back at Figures 5 and 7, despite transformer receiving little improvement, it still ranks first among all models,

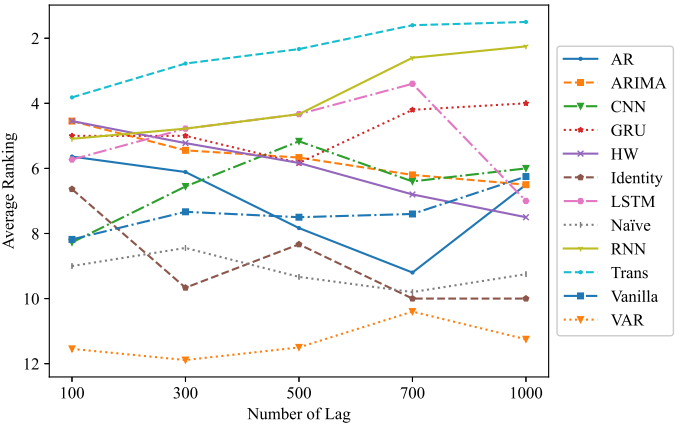

Fig. 6: Average ranking in terms of MAE for each time window size.

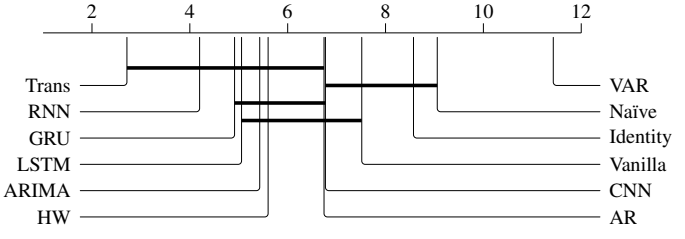

Fig. 7: Critical Difference (CD) diagram for all datasets.

which confirms its superiority in terms of the prediction accuracy and the DFM quality.

To evaluate the quality of the interpretation of the forecasted DFMs, we evaluated the consistency of the forecasts. The results are summarized in Table 5. We bolded the best results, underscored the second-best results, and italicized the worst. As Table 5 shows, the identity function has the highest average consistency, and ideally, this score should be exactly one, as every DFM discovered should be perfectly consistent. However, the DFGs discovered during training based on the bpic17 and nasacs datasets are not always consistent. Hence, they contribute to the reduction of the mean consistency in the results. The naïve average approach has mostly the second-best consistency, while the VAR model performs overall the worst again. It is interesting to see that most baselines (except VAR and ARIMA) can achieve better consistency compared to the *DeePMF* models. This could imply that there is a trade-off between accuracy and consistency for the forecasted process models.

6 Discussion

The results have demonstrated that *DeePMF*, mainly based on the transformer architecture, achieves superior accuracy of process model forecasts. The results may be biased as we did not fully explore all the DL model architectures. For example, we fixed the kernel size and stride for CNN and the number of layers for all DL models. It would

Table 5: Mean consistency of process model forecasts.

	DeePMF						*Baselines*					
	Trans	*RNN*	*LSTM*	*GRU*	*CNN*	*Vanilla*	*Identity*	*Naïve*	*VAR*	*ARIMA*	*HW*	*AR*
hb(100)	0.910	0.864	0.915	0.858	*0.814*	0.860	**1.000**	0.966	0.893	0.918	0.944	0.907
rtfmp(100)	0.876	0.845	0.866	0.854	0.804	0.841	**1.000**	0.982	*0.778*	0.909	0.896	0.903
sepsis_f(100)	0.885	0.883	0.926	0.921	0.899	0.916	**1.000**	0.884	*0.841*	0.903	0.932	0.890
bpic17(100)	0.944	0.945	0.934	0.947	*0.852*	0.914	**0.979**	0.970	0.898	0.934	0.957	0.939
helpdesk(100)	0.969	0.937	0.958	0.933	0.925	0.919	**1.000**	0.941	*0.879*	0.895	0.923	0.930
sepsis(100)	0.910	0.886	0.950	0.941	0.906	0.920	**1.000**	0.872	*0.835*	0.872	0.926	0.870
bpic19_f(100)	0.898	*0.797*	0.889	0.852	0.872	0.865	**1.000**	0.961	0.834	0.900	0.933	0.897
bpic13c_f(100)	0.913	0.884	0.900	0.928	0.884	0.949	**1.000**	0.920	0.893	*0.790*	0.827	0.885
bpic12(100)	0.897	0.902	0.912	0.891	0.917	0.898	**1.000**	0.946	*0.856*	0.950	0.971	0.945
nasacs(100)	0.747	0.764	0.886	0.846	0.789	0.825	**0.996**	*0.704*	0.732	0.794	0.780	0.770
bpic13o_f(100)	0.902	0.913	0.861	0.858	0.850	0.973	**1.000**	0.882	0.940	*0.845*	0.892	0.866
hb(300)	0.934	0.819	0.888	0.881	0.907	0.884	**1.000**	0.917	*0.813*	0.914	0.928	0.915
rtfmp(300)	0.857	0.857	0.874	0.853	0.866	*0.828*	**1.000**	0.955	0.927	0.922	0.911	0.934
sepsis_f(300)	0.887	0.890	0.911	0.915	0.913	0.921	**1.000**	0.924	*0.818*	0.916	0.959	0.927
bpic17(300)	0.918	0.898	0.910	0.898	0.911	0.928	**0.981**	0.964	*0.702*	0.951	0.942	0.940
helpdesk(300)	0.931	0.909	0.963	*0.901*	0.939	0.946	**1.000**	0.984	0.911	0.908	0.928	0.959
sepsis(300)	0.941	0.909	0.949	0.921	0.946	0.931	**1.000**	0.916	*0.788*	0.891	0.928	0.911
bpic19_f(300)	0.879	0.819	0.827	0.845	0.912	0.884	**1.000**	0.954	*0.688*	0.903	0.950	0.913
bpic13c_f(300)	0.949	0.885	0.860	0.878	0.864	0.896	**1.000**	0.932	0.910	*0.779*	0.857	0.912
bpic12(300)	0.903	0.874	0.918	0.906	0.941	0.940	**1.000**	0.936	*0.804*	0.900	0.967	0.955
hb(500)	0.922	*0.837*	0.923	0.877	0.933	0.860	**1.000**	0.945	0.880	0.918	0.940	0.920
rtfmp(500)	0.880	0.896	0.904	0.887	0.872	*0.836*	**1.000**	0.970	0.912	0.945	0.918	0.956
sepsis_f(500)	0.863	0.870	0.853	0.903	0.876	0.890	**1.000**	0.926	*0.840*	0.935	0.949	0.933
bpic17(500)	0.915	0.922	0.875	0.906	0.894	0.906	**0.980**	0.966	*0.730*	0.943	0.940	0.948
helpdesk(500)	0.946	0.967	0.973	0.969	0.943	0.967	**1.000**	0.993	0.939	*0.895*	0.928	0.956
sepsis(500)	0.912	0.886	0.911	0.940	0.933	0.944	**1.000**	0.919	*0.858*	0.898	0.926	0.923
hb(700)	0.909	0.822	0.904	0.914	0.927	0.852	**1.000**	0.939	*0.819*	0.913	0.943	0.917
rtfmp(700)	0.908	0.877	0.908	0.904	0.921	*0.836*	**1.000**	0.948	0.928	0.949	0.916	0.959
sepsis_f(700)	0.913	0.853	0.870	0.843	0.848	0.858	**1.000**	0.829	*0.817*	0.928	0.952	0.939
bpic17(700)	0.887	*0.869*	0.935	0.909	0.873	0.915	**0.980**	0.957	0.871	0.871	0.937	0.944
helpdesk(700)	0.934	0.951	0.934	0.954	0.963	0.954	**1.000**	0.983	0.980	*0.912*	0.931	0.968
hb(1000)	0.921	0.883	0.897	0.915	*0.841*	0.862	**1.000**	0.932	0.862	0.922	0.932	0.918
rtfmp(1000)	0.907	0.871	0.893	0.912	0.930	*0.837*	**1.000**	0.945	0.925	0.959	0.930	0.968
sepsis_f(1000)	0.876	0.849	0.892	0.864	0.833	0.827	**1.000**	*0.787*	0.867	0.909	0.963	0.929
bpic17(1000)	0.897	0.886	0.924	0.892	*0.881*	0.937	**0.982**	0.941	0.919	0.903	0.937	0.946

be interesting to initiate research on each of the DL models and fully explore the architecture's potential for process model forecasting. Our research results can be used as a baseline for such endeavors. For suggestions on hyperparameter selection, unfortunately, we could not find any patterns of the optimal hyperparameters and make a recommendation for their use. As Optuna statistics shows, the best hyperparameter combinations always vary from dataset to fold, and the selected hyperparameters in our experiments seem to be reasonable. Yet, it remains open how much of the differences in the measurements result from choosing the proper configurations and parameters.

The *DeePMF* comes with several natural limitations. Firstly, for training, it requires a reliable process discovery algorithm that can best describe the system behavior for a certain period of time. Secondly, this approach is not able to cater unseen activities as constructed DFMs used for training also fix the number of the possible forecasted

activities. Thirdly, it does not guarantee the quality of the forecasted models, where the best consistency score from NN models is around 0.9.

The improved ranking on finer time windows implies that process model forecasting on large numbers of time windows is promising. Due to the scope constraints, we did not fully explore the optimal time span for forecasting accuracy, which is an interesting direction for future work.

In terms of the training time, despite the DL model training being done with high-end GPUs, it can still take hours to days to train the most optimal NN model. With finer time windows, more training samples are available, and one observes improvements in forecasting accuracy, but it could also take longer to train the model. It is a dilemma to trade off the forecasting accuracy and the time taken to train an NN model. Note that it can be impractical to use *DeePMF* if the time taken to train a forecasting DL model is the same or even longer as the forecasting horizon. However, if this training time dilemma is addressed, organizations and process analysts may use the prediction outcome to help organizational planning such as resource allocation or process model design in the next BPM cycle.

As another direction for future work, it is interesting to explore the impact of larger look-back windows and forecasting horizons on forecasting accuracy and consistency. The *DeePMF* can either be used recursively to forecast a longer period or adapted to use DL recurrent architectures for generating forecasts for longer horizons. Our approach only uses the event log for forecasting. Correlating the event log with other observations could be worthwhile for many applications, and we leave this as future work.

Finally, it is worth noting the benefits of process model forecasting and its potential applications. BPM lifecycle comprises five stages, namely, business process (re)design, implementation, monitoring, adjustment, and diagnosis [25]. It can take at least six months to over two years for a business model to be implemented and run from its initial design [5]. For such an extended period, the process behaviors may evolve, which, by the time of the implementation and monitoring phases of the lifecycle, could potentially make the following redesign phase obsolete. An accurate process model forecast can support resource allocation and planning. Additionally, it can provide analysts and stakeholders with valuable insights into potential process model changes over time. Finally, by incorporating an early process forecast into the redesign initiative, organizations can proactively account for anticipated process evolutions, ensuring that the redesigned process is implemented by the time the forecasted process changes materialize, ultimately facilitating a smoother transition into future business operations.

7 Conclusion

In this paper, we advanced process model forecasting techniques by leveraging deep neural networks. Our experiments demonstrate that deep neural networks offer greater potential than traditional statistical models, with the transformer architecture achieving the highest overall accuracy. To complement this higher accuracy, we introduced a quality measure to assess the consistency of the predicted process models. We explored two methods to further improve forecasting accuracy, finding that post-processing the forecasts can lead to further improvements. By improving forecasting accuracy and consistency, we aim to provide analysts with more reliable and interpretable results.

References

[1] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: KDD, pp. 2623–2631, ACM (2019)

[2] Alkhammash, H., Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: Entropic relevance: A mechanism for measuring stochastic process models discovered from event data. Information Systems **107**, 101922 (2022)

[3] Benavoli, A., Corani, G., Mangili, F.: Should we really use post-hoc tests based on mean-ranks? The Journal of Machine Learning Research **17**(1), 152–161 (2016)

[4] Berti, A., van Zelst, S., Schuster, D.: Pm4py: A process mining library for python. Software Impacts **17**, 100556 (2023)

[5] Bingi, P., Sharma, M.K., Godla, J.K.: Critical issues affecting an ERP implementation. Inf. Syst. Manag. **16**(3), 7–14 (1999)

[6] Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control. Wiley, 4th edn. (2008)

[7] De Smedt, J., Yeshchenko, A., Polyvyanyy, A., De Weerdt, J., Mendling, J.: Process model forecasting using time series analysis of event sequence data. In: Conceptual Modeling (ER 2021), pp. 47–61, Springer International Publishing, Cham (2021)

[8] De Smedt, J., Yeshchenko, A., Polyvyanyy, A., De Weerdt, J., Mendling, J.: Process model forecasting and change exploration using time series analysis of event sequence data. Data & Knowledge Engineering **145**, 102145 (2023)

[9] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research **7**, 1–30 (2006)

[10] Di Francescomarino, C., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: Which one suits me best? In: Business Process Management, pp. 462–479, Springer International Publishing (2018)

[11] Di Francescomarino, C., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A.: An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring. In: Business Process Management, pp. 252–268, Springer International Publishing (2017)

[12] Evermann, J., Rehse, J., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017)

[13] Günther, C., Rozinat, A.: Disco: discover your processes. In: Proceedings of the Demonstration Track of the 10th International Conference on Business Process Management, pp. 40–44, CEUR Workshop Proceedings, CEUR-WS.org (2012)

[14] Hamilton, J.D.: Time Series Analysis. Princeton University Press (1994)

[15] Jalayer, A., Kahani, M., Beheshti, A., Pourmasoumi, A., Motahari-Nezhad, H.R.: Attention mechanism in predictive business process monitoring. In: 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, pp. 181–186 (2020)

[16] Längkvist, M., Karlsson, L., Loutfi, A.: A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters **42** (2014)

[17] Lara-Benítez, P., Carranza-García, M., Riquelme, J.C.: An experimental review on deep learning architectures for time series forecasting. International Journal of Neural Systems **31**(03) (2021)

[18] Le, M., Gabrys, B., Nauck, D.: A hybrid model for business process event prediction. In: Research and Development in Intelligent Systems XXIX, Springer London (2012)

[19] Leemans, S.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: Exploration & a case study. In: 2019 International Conference on Process Mining, pp. 25–32 (2019)

[20] Lütkepohl, H.: New Introduction to Multiple Time Series Analysis. Springer (2005)

[21] Poll, R., Polyvyanyy, A., Rosemann, M., Röglinger, M., Rupprecht, L.: Process forecasting: Towards proactive business process management. In: Business Process Management, pp.

496–512, Springer International Publishing (2018)
[22] Reinsel, G.C.: Elements of Multivariate Time Series Analysis. Springer, 2nd edn. (2003)
[23] Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: Advanced Information Systems Engineering : 29th International Conference, CAiSE 2017, Essen Germany, June 12-16, 2017. Proceedings, pp. 477–492, LNCS, Springer (2017)
[24] Tschumitschew, K., Nauck, D., Klawonn, F.: A slassification algorithm for process sequences based on markov chains and bayesian networks. In: Knowledge-Based and Intelligent Information and Engineering Systems, pp. 141–147, Springer Berlin Heidelberg (2010)
[25] van der Aalst, W.: Process Mining—Data Science in Action. Springer, 2nd edn. (2016)
[26] van der Aalst, W.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321–328 (2019)
[27] van der Aalst, W., Schonenberg, M., Song, M.: Time prediction based on process mining. Information Systems **36**(2), 450–475 (2011)
[28] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, 3rd edn. (2019)
[29] Zhou, W., Polyvyanyy, A., Bailey, J.: Event data and process model forecasting. In: CAiSE Forum, LNBIP, vol. 520, pp. 3–10, Springer (2024)