



# Proactive conformance checking: An approach for predicting deviations in business processes

Michael Grohs<sup>a,\*</sup>, Peter Pfeiffer<sup>b,\*</sup>, Jana-Rebecca Rehse<sup>a,\*</sup>

<sup>a</sup> University of Mannheim, L 15, 1-6, Mannheim, 68161, Baden-Württemberg, Germany

<sup>b</sup> German Research Center for Artificial Intelligence (DFKI) & Saarland University, Campus D 3.2, Saarbrücken, 66123, Saarland, Germany

## ARTICLE INFO

### Keywords:

Process mining  
Deviation prediction  
Predictive process monitoring  
Conformance checking

## ABSTRACT

Modern business processes are subject to an increasing number of external and internal regulations. Compliance with these regulations is crucial for the success of organizations. To ensure this compliance, process managers can identify and mitigate deviations between the predefined process behavior and the executed process instances by means of conformance checking techniques. However, these techniques are inherently reactive, meaning that they can only detect deviations after they have occurred. It would be desirable to detect and mitigate deviations before they occur, enabling managers to proactively ensure compliance of running process instances. In this paper, we propose Business Process Deviation Prediction (BPDP), a novel predictive approach that relies on a supervised machine learning model to predict which deviations can be expected in the future of running process instances. BPDP is able to predict individual deviations as well as deviation patterns. Further, it provides the user with a list of potential reasons for predicted deviations. Our evaluation shows that BPDP outperforms existing methods for deviation prediction. Following the idea of action-oriented process mining, BPDP thus enables process managers to prevent deviations in early stages of running process instances.

## 1. Introduction

Modern business processes are subject to an increasing number of external and internal regulations, which, for example, ensure the protection of customer data [1], prevent fraud in internal transactions [2], or reduce accidents through human error [3]. Maintaining compliance with these regulations fosters trust among organizational stakeholders and mitigates risks, such as legal liabilities and reputational damage [4,5]. Furthermore, it increases operational efficiency by ensuring efficient process execution and resource allocation [6]. Thus, effective mechanisms for monitoring and enforcing compliance are essential for successful business process management.

One approach to ensure process compliance is by means of conformance checking [7], a well-established sub-discipline of process mining [8]. Conformance checking compares actual process executions, captured in the form of event logs, with predefined normative process models. By detecting inconsistencies, so-called deviations, between observed and expected process behavior, it provides valuable insights into process compliance [9]. This automated approach to compliance checking enables process managers to identify deviations quickly and comprehensively, facilitating prompt corrective actions to realign operations with the respective regulations.

However, the nature of current conformance checking techniques makes compliance checking inherently reactive [10]: The deviation, and the ensuing loss of compliance, must occur to be detected by a conformance checking technique. This limits a process manager's ability to detect and address compliance issues in real-time [11]. Online conformance checking [12] and anomaly detection [13] techniques can analyze process instances at runtime, which reduces the time span between the occurrence of a deviation and the notification of the process manager about that occurrence. Nevertheless, these techniques can also only identify deviations after they have occurred.

To ensure the compliance of the process, however, it would be better if conformance checking techniques were proactive, i.e., able to identify deviations before they actually occur. If a process manager is informed about an impending deviation in a running process instance, they can take the necessary actions to prevent the deviation or at least mitigate its impact [11]. To obtain this information, we can leverage predictive process monitoring (PPM) to predict which deviations will occur in a running process instance. PPM is a branch of process mining that aims to enable proactive management of business processes by predicting future behavior of incomplete process traces [14]. Although proactive compliance management is often mentioned as an important benefit of PPM [15], existing PPM approaches have not focused on this

\* Corresponding authors.

E-mail addresses: [michael.grohs@uni-mannheim.de](mailto:michael.grohs@uni-mannheim.de) (M. Grohs), [peter.pfeiffer@dfki.de](mailto:peter.pfeiffer@dfki.de) (P. Pfeiffer), [rehse@uni-mannheim.de](mailto:rehse@uni-mannheim.de) (J.-R. Rehse).

application [14], instead predicting, e.g., the next event in a trace [15] or the process outcome [16]. The few existing approaches that are capable of predicting the occurrence of deviations either lack the ability to predict which activity will deviate and to do this sufficiently early [10] or cannot deal with the imbalanced nature of this task [11].

To advance proactive conformance checking, this paper introduces a novel PPM approach that can predict which deviations will occur in the future of a running process instance. This approach, called *Business Process Deviation Prediction (BPDP)*, uses a supervised machine learning (ML) model to predict, in an early process state, whether an instance will deviate and to which specific activities the deviations will relate. BPDP is designed to tackle the specific challenges of deviation prediction, which include the need to consider explicit process knowledge as well as context attributes, a high imbalance of deviations in the training data, and the frequent co-occurrence of certain deviation types. In addition, it is equipped with multiple features that support the proactive management of potentially deviating process instances, such as a list of potential reasons for predicted deviations.

This paper is an extended and revised version of our original conference publication [17]. It extends the previous paper in three main aspects:

- (1) *Deviation Pattern Prediction*: Whereas the original BPDP could only predict individual deviating activities, this version can also predict deviation patterns, i.e., frequently co-occurring individual deviations.
- (2) *Prediction Model Design*: To ensure optimal performance, we experimentally assess how different designs impact BPDP's predictive strength:
  - (a) We compare the separate classifiers from the original BPDP with two learning architectures that predict all deviations collectively.
  - (b) We compare the feed-forward network from the original BPDP with a more sophisticated LSTM model.
- (3) *Prediction GUI*: To illustrate how BPDP can support process managers in practice, we provide a prototypical graphical user interface.

The paper is structured as follows: In Section 2, we illustrate the challenges of deviation prediction. We discuss related work in Section 3. The BPDP approach is introduced in Section 4. Section 5 experimentally determines the best learning strategy of the approach. We benchmark its predictive strength in Section 6 and evaluate its suitability to the use case in Section 7. After discussing the evaluation results in Section 8, we conclude the paper in Section 9.

## 2. Problem illustration

Approaches for deviation prediction must address the specific challenges of this task. To illustrate these challenges, consider the loan application process in Fig. 1. It is a subset of the to-be process model for the BPI Challenge 2012, including only activities starting with A\_ (BPIC12 A). In this process, deviations from the to-be model occur. Each deviation is the manifestation of a deviation type, characterized as an erroneous or missing activity within a trace, as per the to-be

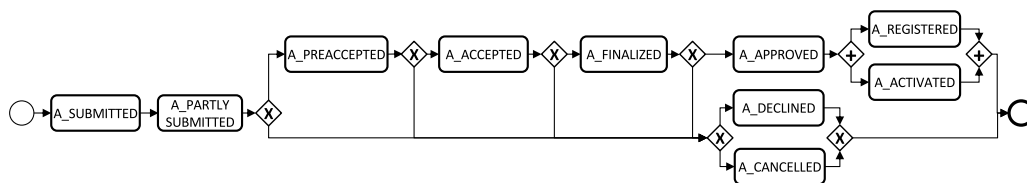


Fig. 1. Deviation prediction running example.

model. Multiple deviation types can co-occur in the same trace, forming a deviation pattern. Deviation patterns often originate from the same underlying issue, such as the swapping of two activities.

Now consider the traces  $t_1$  and  $t_2$  in Figs. 2(a) and 2(b), extracted from the BPIC12 A event log.  $t_1$  consists of eight activities executed by three different resources (Elsa, Olaf, and Anna) and handles a loan with a requested amount of 25,000 €.  $t_2$  also consists of eight activities executed by two resources (Anna and Olaf) and handles a requested amount of 10,000 €.

$t_2$  conforms to the model, so there are no deviations. In  $t_1$ , A\_APPROVED and A\_REGISTERED are swapped, which results in two deviation types:

- $d_1$  : A\_APPROVED is missing from its model-prescribed position
- $d_2$  : A\_APPROVED is executed at a wrong position in the trace

To illustrate the prediction of these deviations, consider the predictions for trace prefixes of  $t_1$  and  $t_2$  in Figs. 2(c) and 2(d).  $t_2$  conforms to the to-be model, so no deviation should be predicted for any of its prefixes. In  $t_1$ ,  $d_1$  occurs at position 6 and should therefore be predicted for each prefix up to length 5.  $d_2$  occurs after the execution of A\_REGISTERED at position 7. Thus, it should be predicted for each prefix of  $t_1$  up to length 6. However, correctly predicting those deviations poses multiple challenges.

- C1: Explicit Process Knowledge.** By definition, the identification of deviations in an event log depends on the availability of a predefined to-be process, documented, e.g., as a process model. For instance, we need to know that A\_APPROVED is supposed to be executed before A\_REGISTERED and that not doing so results in a deviation ( $d_1$  in this case). Thus, deviation prediction requires explicit knowledge about the expected process behavior to define the deviation types that serve as prediction targets. This consideration of explicit process knowledge, instead of, e.g., statistical information on unusual behavior as done in anomaly detection [13], also enables us to clearly distinguish between deviating and infrequent behavior.
- C2: Deviation Patterns.** Deviation types are defined on the activity level, so we consider each deviating activity separately. However, the same underlying behavior of the process can cause several deviation types to simultaneously occur in the process instance. For example, in  $t_1$ , A\_APPROVED and A\_REGISTERED have been swapped, which leads to the occurrence of both  $d_1$  and  $d_2$ . We refer to such a set of co-occurring deviation types as a deviation pattern. A predictive approach should be able to predict complete deviation patterns to fully capture the underlying process behavior.
- C3: Prediction Targets.** From the perspective of supervised ML, predicting deviations and deviation patterns is challenging due to its multi-label, imbalanced, and dynamic nature.

- C3.1: Multi-label Targets.** A trace may deviate from the predefined behavior in more than one way. For example,  $t_1$  deviates in  $d_1$  and  $d_2$ . Similarly, multiple deviation patterns can occur in one trace. Thus, for one running process instance, a classifier has to make separate (binary) predictions per deviation type and per deviation pattern.

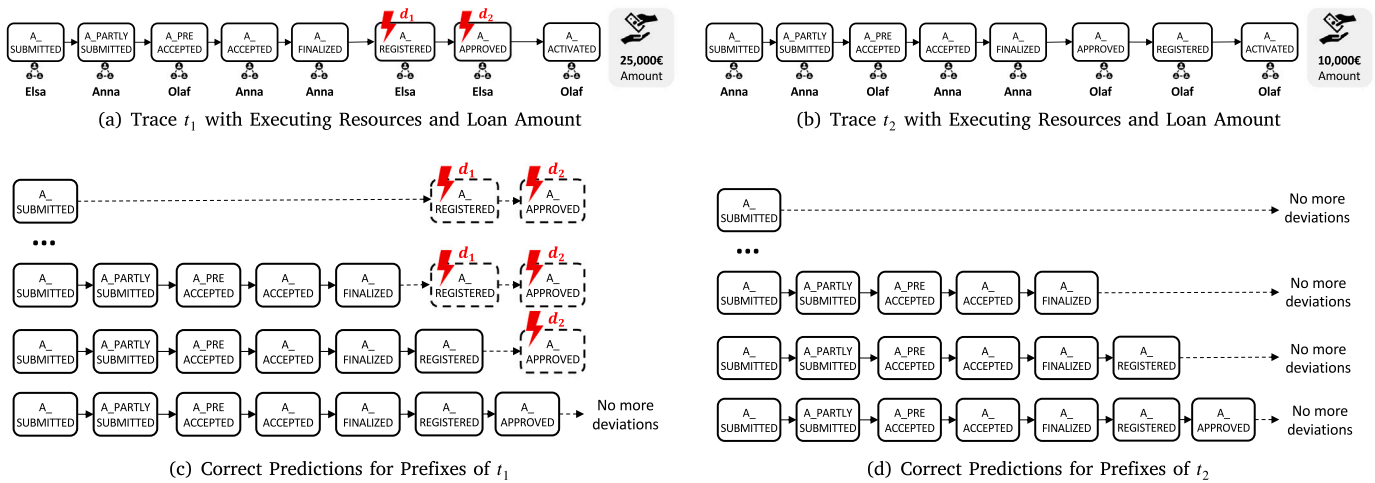


Fig. 2. Exemplary Traces  $t_1$  and  $t_2$  with Correct Predictions.

**C3.2: Imbalanced Targets.** Overall, the frequency of deviating traces in a log is low. Also, different deviation types and deviation patterns have different frequencies, leading to highly imbalanced prediction targets. A predictive approach should be able to correctly predict the occurrence of all deviations in imbalanced data.

**C3.3: Dynamic Targets.** The prediction labels change throughout the progression of a trace: once a deviation (pattern) has occurred in the trace, it should no longer be predicted, e.g. not after position 7 in  $t_1$ .

**C4: Context Importance.** Up to the sixth event,  $t_1$  and  $t_2$  share the same control-flow. Nevertheless,  $d_1$  and  $d_2$  should be predicted for all preceding prefixes of  $t_1$  but not for those of  $t_2$ . Thus, a predictive approach must consider trace attributes other than the control-flow. These so-called context attributes can relate to the resources perspective, like the executing resources, or the case perspective, like the requested amount, as shown in Figs. 2(a) and 2(b)). Including context attributes of the running process instance in addition to the control flow increases the likelihood of correctly predicting deviations, because it allows for a more detailed differentiation between prefixes.

**C5: Action Orientation.** To ensure process compliance, deviating process instances are more relevant than conforming ones. For example, accurately predicting the deviations of  $t_1$  alerts the process manager about future problems, such that they can take preventive actions to avoid or mitigate non-conformance. Therefore, early recognition of upcoming deviations is more important than early recognition of conforming behavior. This means that the trained ML model must value a high recall (correctly recognizing all deviations) over a high precision (not misclassifying conforming instances).

### 3. Related work

In this section, we elaborate on related work and explain why existing approaches for deviation prediction do not comprehensively address the challenges from Section 2.

**Deviation Prediction.** As of now, only two existing approaches explicitly address the prediction of deviations for running process instances. The first approach predicts whether the next activity in a trace will be a log or model move, thus detecting deviations from the to-be model [10]. For that purpose, the authors train two specialized classifiers, one for log and one for model moves. However, this approach

does not predict which specific deviation or deviation pattern happens (C2 and C3.1). Furthermore, it cannot handle imbalanced data (C3.2), considers only the control-flow and no context attributes (C4), and predicts deviations only for the next event, thus not enabling early detection of deviations (C5).

The second existing approach for deviation predictions predicts specific deviations in the future of traces by means of a statistical technique called Subjective Logic [11]. The authors create process states for incomplete instances and compare frequencies of past deviating and conforming behavior for each state. Based on those frequencies, they form a so-called belief that a specific deviation type will occur in an ongoing process instance. The predictions per trace are then derived from the beliefs. Prediction targets can be derived using explicit process knowledge (C1) and are allowed to be multi-label targets (C3.1), dynamic targets (C3.3), and also deviation patterns that are known to exist (C2). A limited number of context attributes can be included for prediction (C4), but this is restricted to few attributes that are known to be related to the deviations. However, in their evaluation, the authors use synthetic data with four deviation types that each occur in 20% of the behavior, which are far more deviations than in our real-life scenarios (thus not addressing C3.2). Further, the predictive model does not put specific focus on the deviating class (thus not addressing C5).

In addition to these two approaches, there is a general framework on how to predict process behavior, which includes conformance checking results such as fitness [18]. However, this framework has not been shown to detect deviations in advance. It can therefore be seen more as a conceptual foundation for proactive conformance checking than a concrete approach.

**Deviation Patterns.** Multiple works have addressed the issue that individual deviations can have the same underlying reason and hence co-occur in the form of deviation patterns. Depaire et al. have proposed a general framework to categorize combinations of deviations as deviation patterns [19]. They stress that these patterns can be formed based on skipped and inserted activities but do not propose an approach to discover them. For that purpose, two approaches exist. The first one conducts a conformance check based on log-level event structures [20], which allows for general observations about the differences between the executed and the prescribed process behavior. Thereby, connections between individual deviation types are detected. The second approach discovers sets of strongly correlated deviation types and transforms them into deviation patterns [21].

Alternatively, deviation patterns that are known and expected to occur can be included in a process model and detected by so-called “break-the-glass” alignments [22]. As a further alternative, these deviation patterns can also be detected by means of token-replay [23].

Last, so-called workarounds are related to deviation patterns. These workarounds can be detected semi-automatically [24] or automatically [25]. Although they differ conceptually from deviation patterns, specific workaround types resemble the occurrence of multiple deviation types as found by conformance checking techniques in the same trace. Whereas these approaches conceptually address the identification of deviation patterns, they are not able to predict their appearance.

**Anomaly Detection.** As mentioned above, deviations are closely related to anomalies in process behavior. The main difference is that anomalies are statistically infrequent, whereas deviations are based on explicit process knowledge. Hence, deviations can manifest as anomalies, but this is not guaranteed. Existing approaches for anomaly detection in an event log typically leverage unsupervised machine learning. They train a model to represent the normal, i.e., most frequent process behavior, which is then used to detect anomalous behavior in a concrete trace [13].

Like deviation prediction, anomaly detection is also a highly imbalanced task. To address this, one approach constructs multiple classifiers which are trained with a weighted loss function to account for label imbalance [26]. The prediction of these classifiers are then aggregated with different strategies from the field of ensemble learning, which allow the approach to classify (prefixes of) traces as normal or anomalies under this consideration. However, this approach requires manual labeling from experts and does not predict which type of deviating behavior will occur. One approach is also able to generate alignments between the observed process execution and the “to-be” process model [27] in an unsupervised way. However, this approach cannot predict that a deviation will occur in the future of a running process instance, as they require to compare predicted with observed behavior. This means they can detect a deviation as soon as it happens but not in advance.

**Predictive Process Monitoring.** The goal of PPM is to predict the future behavior of a running process instance, thereby enabling proactive process management [28]. Trained on historic event log data, PPM approaches can be clustered into three big groups, depending on their prediction target [14].

The first group of approaches predict the next event in the trace [15]. Applied consecutively, they can also predict a complete trace suffix. Because they do not consider external process knowledge, they cannot explicitly predict deviations from a to-be model. Nevertheless, suffix prediction can be used for deviation prediction if the predicted suffixes are aligned with a to-be model. This approach to deviation prediction does not account for label imbalance (C3.2) and is not trained with specific focus on the deviating class (C5). Still, we will consider it as a benchmark for deviation prediction.

Second, approaches for outcome prediction predict the end state of a running process instance, e.g., whether a loan application is accepted or rejected. Outcome prediction is related to deviation prediction as exceptional process outcomes, such as error states, can be seen as rule violations [16], but it does not concern individual deviations. The third group of approaches predict numerical characteristics of running instances, such as runtime [14], which is not directly related to conformance.

**Action-Oriented Process Mining.** Finally, action-oriented process mining is a relatively new research stream [29]. It describes the general idea to automatically generate actions in case of constraint violations, which can be detected using conformance checking and are supposed to enable process managers to prevent deviation occurrences. Based on such constraint violations and their temporal dependencies, complete action plans can be derived [30]. These constraint violations can also be predictive in nature [29]. In our context, these approaches for action-oriented process mining are closely related to challenge C5 which acknowledges that deviation predictions are only helpful if process managers can take preventive actions. However, there is no known approach in this research stream that is able to actually predict the occurrence of deviations and recommend corresponding preventive actions.

## 4. Approach

In this section, we introduce our novel approach for deviation prediction, called Business Process Deviation Prediction (BPDP). BPDP predicts which, if any, deviation types and deviation patterns will occur in the future of an ongoing process instance, thereby tackling the challenges of deviation prediction, presented in Section 2. In the following, we present the approach step-by-step.

### 4.1. Overall design

The BPDP approach, illustrated in Fig. 3, is split into an *offline* component, in which the models are trained, and an *online* component, in which they are used. In the following, we focus on the offline component and explain its steps. As input, BPDP receives a to-be model  $B$  of a given process and a corresponding event log  $L$ . After labeling (Section 4.2) and encoding (Section 4.3) each trace prefix, we train a classifier, called Individual Deviation Predictor (IDP), that can predict individual deviations for ongoing traces (Section 4.4). Then, we train an additional classifier, called Deviation Pattern Predictor (DPP), that builds upon IDP to predict deviation patterns (Section 4.5).

For presenting the approach steps, we require the following preliminaries. The to-be model  $B$  defines the desired execution dependencies between the activities of a process. For our purposes, it is sufficient to abstract from specific process modeling languages and focus on the behavior defined by a model. That is, given a set of activities, a to-be model  $B$  defines a set of desired execution sequences that lead the process from an initial to a final state. The event log  $L$  contains a collection of traces. A trace  $t \in L$  is a chronologically ordered sequence of events  $\langle e_1, e_2, \dots, e_n \rangle$  of length  $n$ . An event  $e$  denotes the execution of a certain activity. It is represented as a tuple of attributes from which two, *activity* and *traceID*, are mandatory and others are optional, e.g., *timestamp*, *resource*, *cost*. All events in one trace have the same *traceID*. A trace represents one execution of the process, called a process instance. Similar to events, traces can also have additional attributes, e.g., a loan amount. Running instances are represented by trace prefixes. A prefix of trace  $t$  of length  $p$  is defined as a subsequence  $\langle e_1, \dots, e_p \rangle$ , with  $1 \leq p \leq n$ .

### 4.2. Labeling

We approach deviation prediction as a supervised ML task, separated into predicting individual deviations with IDP and predicting deviation patterns with DPP. Both require target labels for each trace prefix, which are later used to train the classifiers. In the following, we explain how we assign target labels for individual deviations and deviation patterns.

**Individual Deviations.** First, we determine the set of possible target labels  $D_{L,B}$  as all deviations that can occur between  $L$  and  $B$ . Therefore, for each trace  $t \in L$ , we compute the trace alignment [7] between  $t$  and  $B$ , using the pm4py alignment algorithm [31]. In a trace alignment, a log move  $(ac, \gg)$  indicates that an activity  $ac$  executed in the trace should not have been executed according to the model (e.g., wrong execution  $d_2$  in Fig. 2(a)). A model move  $(\gg, ac)$  indicates that activity  $ac$  is prescribed by the model, but missing from the trace (e.g., missing execution  $d_1$  in Fig. 2(a)). If an executed activity in the trace conforms with the model, the alignment contains a synchronous move  $(ac, ac)$ . Table 1 shows an example of an alignment, with one move per column.<sup>1</sup> It contains synchronous moves on activities W and X ( $(W, W)$  and  $(X, X)$ ), a model move on activity Y  $(\gg, Y)$ , and a log move on activity Z  $(Z, \gg)$ . The set of model and log moves in the alignments of all traces in an event log defines the set of deviation types  $D_{L,B} = \{d_1, \dots, d_m\}$ , where  $m$  is the total number of deviation types.

<sup>1</sup> In this paper, alignment visualizations place the trace above the model sequence.

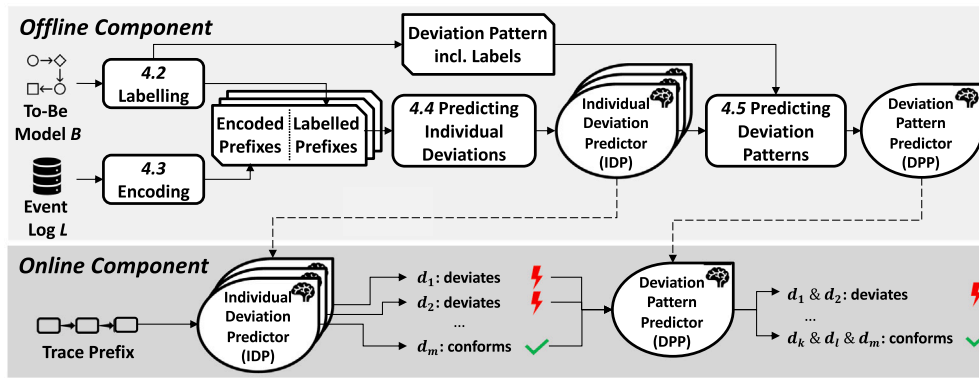


Fig. 3. Overview of the BDPD approach.

Table 1

Exemplary alignment.

$t$	W	X	»	Z
$B$	W	X	Y	»

Table 2

Two optimal alignments of  $t_1$ .(a) One optimal alignment of  $t_1$ 

SUB	PAR	PRE	ACC	FIN	»	REG	APP	ACT
SUB	PAR	PRE	ACC	FIN	APP	REG	»	ACT

(b) Alternative optimal alignment of  $t_1$ 

SUB	PAR	PRE	ACC	FIN	REG	APP	»	ACT
SUB	PAR	PRE	ACC	FIN	»	APP	REG	ACT

SUB = A\_SUBMITTED; PAR = A\_PARTLY SUBMITTED; PRE = A\_PREACCEPTED;  
 ACC = A\_ACCEPTED; FIN = A\_FINALIZED; REG = A\_REGISTERED;  
 APP = A\_APPROVED; ACT = A\_ACTIVATED.

Computing alignments is a non-deterministic optimization problem, which can cause problems when training and evaluating an ML model to predict individual deviations. As an example, consider Table 2, which shows two optimal alignments for the trace  $t_1$  with deviation types  $d_1$  and  $d_2$ . In the first alignment (Table 2a), the algorithm found a model and a log move on APP. In the second alignment (Table 2b), the algorithm found a log and a model move on REG. Both alignments are optimal under the standard cost function. If the same deviation types occur in different traces, it can therefore happen that the alignments for these traces contain two different sets of moves. This can lead to situations where the same trace prefix has two different deviation type target labels, which confuses the model during training and negatively impacts its performance during evaluation. Thus, we want the set  $D_{L,B}$  to contain the same set of moves for the same deviation types across traces. This can be achieved heuristically by minimizing the log-wide number of deviation types in  $D_{L,B}$ . Then, for example, only two rather than four moves are used for  $d_1$  and  $d_2$  in all traces. We calculate the log-wide alignments 100 times and fix the results with the fewest deviation types in  $D_{L,B}$  across traces, also ensuring reproducibility.

For each deviation type  $d \in D_{L,B}$ , we then individually label each prefix with either 1 or 0. If  $d$  occurs in the future of the prefix, we label this prefix with 1.<sup>2</sup> Once  $d$  no longer occurs in the future of the prefix, the remaining prefixes of this trace are all labeled with 0. This dynamic labeling addresses C3.3. If  $d$  does not occur in a trace, all prefixes are labeled with 0. Thus, if  $d$  occurs (the last time) at position  $j$  of a complete trace  $t$  of length  $n$ , we will obtain  $j - 1$  prefixes labeled

with 1 and  $n - (j - 1)$  prefixes labeled with 0 where  $1 < j \leq n$ . Note that a model move can still occur after the trace has ended. Thus, we also consider the full trace as its own prefix, which might be labeled with 1. As an example, Table 3 shows the labels of all trace prefixes of  $t_1$  with respect to  $d_1$  and  $d_2$ .

**Deviation Patterns.** To predict deviation patterns, we construct new target labels from the set of deviation types. Deviation patterns are defined as sets of frequently co-occurring deviation types. To find those patterns, we rely on the pairwise correlation coefficients of the deviation types. Concretely, two deviations are frequently co-occurring if their pairwise correlation coefficient is higher than a given threshold  $ths_{corr}$ . Because a deviation pattern can consist of more than two deviation types, we consider the pairwise correlation coefficients between all deviation types and construct maximal sets of deviations, such that each pairwise correlation coefficient exceeds  $ths_{corr}$ . Through that, we ensure that all correlated deviation types are summarized in one large deviation pattern instead of multiple smaller ones. We set the default value of  $ths_{corr}$  to 0.5 as this is a commonly used threshold to determine highly correlated variables [32]. However, we allow the user to adjust  $ths_{corr}$  if the process at hand requires a different threshold.

Following this procedure, we obtain a set  $C_{L,B} = \{c_1, \dots, c_k\}$ , where each element  $c_i \subseteq D_{L,B}$  is a set of individual deviation types. More concretely, deviation patterns are defined as non-empty, non-singleton subsets of the power set of  $D_{L,B}$ , i.e.,  $\{c_i \in 2^{D_{L,B}} : |c_i| \geq 2\}$ . Labels are assigned analogously to the individual deviation labels: We label a prefix with 1 for a deviation pattern  $c_i \in C_{L,B}$  as long as all deviations that make up the pattern occur in the future of the prefix, and 0 otherwise. As an example, consider the deviation pattern  $c_1 = \{d_1, d_2\}$  and all prefixes of  $t_1$  in Table 3.

#### 4.3. Encoding

Incorporating contextual information is highly important for deviation prediction (C4). Therefore, we must encode the available contextual information of the trace prefix to feed it into the ML model. Whereas we experimented with different encoding approaches in the previous version of the paper [17], we decided to stay with the complex index-based encoding (CIBE) [33], as it turned out to work best. In this encoding method, a feature vector is created based on trace and event attributes. This vector contains trace features to represent all available trace attributes. Further, it contains event features to represent the executed activity, executing resource, and temporal features for each event in the trace.

#### 4.4. Predicting individual deviations

In this section, we define the prediction model IDP that predicts whether individual deviations will occur in the future of an ongoing process instance. IDP is realized as a neural network trained with

<sup>2</sup> Note that a deviation can occur multiple times in the same prefix.

**Table 3**  
Labels of individual deviation types  $d_1$  and  $d_2$  as well as of deviation pattern  $c_1$  for prefixes of  $t_1$ .

Trace prefix	Labels		
	Individual deviations		Deviation patterns
	$d_1 : (\gg, \text{APP})$	$d_2 : (\text{APP}, \gg)$	
(SUB)	1	1	1
(SUB, PAR)	1	1	1
(SUB, PAR, PRE)	1	1	1
(SUB, PAR, PRE, ACC)	1	1	1
(SUB, PAR, PRE, ACC, FIN)	1	1	1
(SUB, PAR, PRE, ACC, FIN, REG)	0	1	0
(SUB, PAR, PRE, ACC, FIN, REG, APP)	0	0	0
(SUB, PAR, PRE, ACC, FIN, REG, APP, ACT)	0	0	0

SUB = A\_SUBMITTED; PAR = A\_PARTLY SUBMITTED;  
PRE = A\_PREACCEPTED; ACC = A\_ACCEPTED; FIN = A\_FINALIZED;  
REG = A\_REGISTERED; APP = A\_APPROVED; ACT = A\_ACTIVATED.

encoded and labeled prefixes. It has a *network architecture* with connected layers of neurons and *undersamples* the training set to cope with label imbalance (C3.2). Finally, IDP uses a specific *loss function* to pay attention to the deviating class (C5). In the following, we elaborate on these three aspects.

**Network Architecture.** The neural network architecture consists of an input layer with a number of neurons dependent on the number of encoded features in CIBE. These inputs are fed into hidden layers and finally into one classification layer to predict the deviation types. The result is one class probability per deviation type  $d_i$  from which the predictions can be inferred. As usual for deep learning networks, there are several possibilities to construct hidden layers, each with potential advantages to cope with the challenges of deviation prediction. To determine which designs is suited best, we conduct empirical experiments with different network implementations for IDP in combination with hyperparameter optimization in Section 5.

**Undersampling.** Because the majority of process behavior typically conforms with the predefined process [10,34], we encounter a high imbalance of labels in the data used for training IDP. For example, in the BPIC12 A log, deviation types  $d_1$  and  $d_2$  only occur in 9% of cases and a third deviation type ( $\gg, A\_DECLINED$ ) occurs only in 3% of cases. To cope with this imbalance, we apply undersampling to the majority class, which tends to be repetitive due to similarities of many conforming traces. Concretely, we apply the one-sided selection algorithm, which overcomes limitations from other undersampling approaches [35]. This algorithm does not completely balance the training set, but retains important information from the majority class.

**Loss Function.** To achieve high recall of predicted deviations, we must ensure that IDP correctly recognizes all deviating instances, at the risk of misclassifying some conforming ones. We address this by using a weighted cross-entropy loss (WCEL) function, which allows setting higher weights on the deviating class to prioritize its prediction.

We train the network with the undersampled training set and WCEL, imposing early stopping. We do not undersample our test set.

#### 4.5. Predicting deviation patterns

In the final step, we focus on predicting deviation patterns. Thus, we train DPP, which addresses C2 by using the output of IDP to predict the deviation pattern(s) in the future of ongoing process instances.

**Network Architecture.** The DPP classifier is based on the IDP classifier. Following the concept of ensemble learning [36], the IDP classifier is used as Level-0 classifier, whose output is then used in a Level-1 classifier capable of predicting all deviation patterns  $C_{L,B}$ , as defined in Section 4.2. In particular, the output of IDP is one class probability per deviation type  $d_i$ , stored in one vector of class probabilities  $P_D$ . Thus, the input layer of DPP consists of one neuron for each deviation type trained in the previous step and is fed with the vectors  $P_D$ . This input

layer is connected to hidden layers and an output layer with  $k$  neurons, i.e., the number of deviation patterns in  $C_{L,B}$ .

**Undersampling & Loss Function.** We follow the idea of IDP and undersample the training set for DPP, which consists of the output of IDP as features and the labels for all deviation patterns as prediction targets. On this training set, we apply one-sided selection to address the imbalance in the prediction targets, which is even stronger for deviation patterns as fewer prefixes will be labeled 1 when considering multiple deviation type labels simultaneously. Again, we do not undersample our test set. Further, we also apply a WCEL loss function during training to focus on the deviating class(es). As output of this step, our approach predicts which deviation patterns will occur in the future of running instances, allowing the user to completely depict the underlying non-conformity.

To determine the best implementation of our approach that addresses the challenges of the deviation prediction task, we experimentally assess different network designs in the next section.

## 5. Experimental selection of BPDP's best network architecture

Whereas the input for the deviation prediction network is determined by the CIBE encoding, the hidden architecture and the classification layers of BPDP could utilize different designs. More specifically, the task can be interpreted with structurally different ML models. To determine which design best tackles the challenges of deviation prediction, we empirically assess different implementations of our approach. In the following, we introduce the experimental setup and the investigated network implementations before assessing their performance and concluding with the best model architecture. All resources to reproduce our results can be found online.<sup>3</sup>

### 5.1. Experimental setup

To conduct the experiments, we evaluate the performance of the respective experimental versions on seven public event logs. We split the traces in the log randomly into train ( $\frac{2}{3}$ ) and test ( $\frac{1}{3}$ ) set. Note that deviation types, that occur in one trace only, have to be removed from the deviation types that we predict. Otherwise, they would either be part of the training or test split, resulting in situations where we either cannot learn or evaluate them. Thus, we use all deviation types that occur in at least two traces and ensure that both the training and test set contain at least one deviating trace.

**Data.** We apply our approach to event logs from the BPI Challenge 2012<sup>4</sup> (BPIC 12; sub-processes with A\_ and O\_ activities only) the

<sup>3</sup> <https://gitlab.uni-mannheim.de/jpmac/bpdp>.

<sup>4</sup> <https://doi.org/10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F>.

**Table 4**  
Descriptive statistics for event logs used for experiments.

Log $L$		Traces	Events	Trace Attr.	Trace length			Dev. Types	Dev. Occurrences			Dev. Patterns
					min.	avg.	max.		min.	avg.	max.	
BPIC 12	A	13,087	60,849	1	3	4.7	8	3	399	927	1191	1
	O	5,015	31,244	1	3	6.2	30	8	20	984	1761	1
BPIC 20	Dom.	10,500	56,437	4	1	5.4	24	19	1	252	254	2
	Int.	6,449	72,151	17	3	11.2	27	46	1	292	1701	7
	RfP	6,886	36,796	8	1	5.3	20	23	1	116	1027	3
	Prep.	2,099	18,246	16	1	8.7	21	41	1	64	530	8
MobIS		3,354	55,809	1	11	16.6	49	21	1	182	1011	6

BPI Challenge 2020<sup>5</sup> (BPIC 20; Domestic Declarations (Dom.), International Declarations (Int.), Request for Payment (RfP), and Prepaid Travel Costs (Prep.) processes) and the MobIS Challenge<sup>6</sup> (MobIS; only complete cases). To-be models for BPIC 12 and MobIS are provided; for BPIC 20, they were taken from [34]. We use these logs as they differ in size, attributes, trace length, deviation types, class imbalance, and deviation patterns as shown in Table 4. Each pattern consists of two or more deviation types. For example, the deviation pattern in BPIC12 A is the swap of the activities A\_APPROVED and A\_REGISTERED, corresponding to the deviation types  $d_1$  and  $d_2$  shown in Section 2.

**Metrics.** To evaluate whether the investigated versions of our approach correctly predict both deviations and conformity in running instances, we measure prediction quality on prefix level using precision, recall, and AUC<sub>ROC</sub> (area under curve for the receiver operating characteristic). Given true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions, precision (Prec.) and recall (Rec.) are defined in Eq. (1).

$$Prec. = \frac{TP}{TP + FP} \quad Rec. = \frac{TP}{TP + FN} \quad (1)$$

We calculate precision and recall for the deviating (*Dev*) and non-deviating class (*No Dev*) separately to get a detailed quality assessment. This differentiation is important because classes are imbalanced and a high *Dev* recall is of particular interest (see C5). For imbalanced data, a low *Dev* precision does not necessarily indicate a low *No Dev* recall. To account for this data imbalance, we consider AUC<sub>ROC</sub>, which is commonly used in tasks with imbalanced target labels. AUC<sub>ROC</sub> measures the predictive strength by comparing the TP rate to the FP rate. It quantifies this strength between 0 and 1. A value above 0.7 is considered acceptable, anything above 0.8 is excellent [37]. To assign each deviation type equal importance, regardless of class imbalance, we report the macro average (i.e., unweighted average over all deviation types) of precision, recall, and AUC<sub>ROC</sub>.

## 5.2. Investigated Network Implementations of IDP

To assess which neural network implementation of our approach is suited best for deviation prediction, we adapt the architecture of the individual deviation predictor IDP. Two are based on separate classifiers for each deviation type (IDP-separate) and two are predicting all deviation types collectively (IDP-collective). Thus, we test four network architecture designs of IDP:

- (1) *IDP-separate*<sup>FFN</sup>: multiple separate feed-forward (FF) classifiers, each specialized to predict one deviation type individually
- (2) *IDP-separate*<sup>LSTM</sup>: specialized classifiers as in *IDP-separate*<sup>FFN</sup> but based on Long Short-Term Memory (LSTM) models
- (3) *IDP-collective*<sup>FFN</sup>: single feed-forward classifier to predict all deviation types collectively
- (4) *IDP-collective*<sup>LSTM</sup>: collective classifier as in *IDP-collective*<sup>FFN</sup> but based on LSTMs

Each of these IDP designs can potentially address the challenges of deviation prediction. The specialized classifiers of IDP-separate can be trained for each deviation type individually, possibly best coping with high label imbalance, which differs between deviation types. In contrast, predicting all deviations at once trains the model to learn dependencies between deviations, e.g., in the form of deviation patterns. This could improve the performance of the collective prediction in IDP-collective compared to IDP-separate. We also investigate whether a LSTM model design for IDP-separate and IDP-collective is performing better than feed-forward models. Compared to FF models, LSTMs are generally better suited to handle sequential data, e.g., traces in event logs [38]. However, given the importance of context for deviation prediction (see C4), we cannot consider only (sequential) traces, but also have to include non-sequential contextual data, such as trace attributes. Hence, we experimentally assess both architectures.

In the following, we illustrate the specific network architecture of the four IDP-designs and the consequences for undersampling and WCEL function.

### 5.2.1. IDP-separate<sup>FFN</sup>

This design consists of multiple FF classifiers, each specialized to predict one deviation type  $d_i$ . The input layer of each classifier has a number of neurons equal to the number of encoded features in CIBE. It is connected to two hidden layers of 256 neurons each. The second hidden layer is connected to one output layer with two neurons activated by a Softmax function that returns a class probability for  $d_i$ , performing binary classification. The architecture is illustrated in Fig. 4.

**Undersampling.** In this design, we can undersample each deviation type individually by applying one-sided selection per deviation type to balance the training set. The algorithm deletes majority samples depending on the label distribution of only one deviation type in the event log at once.

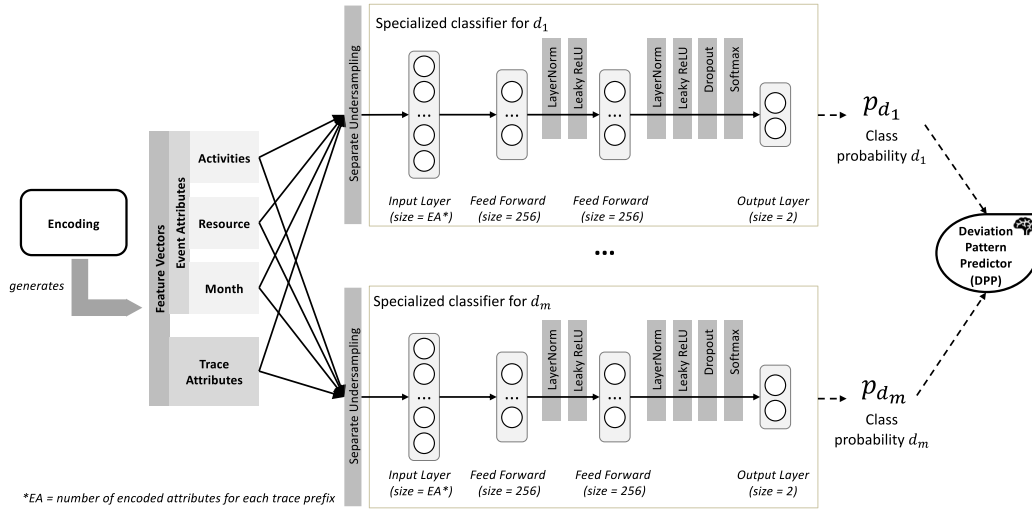
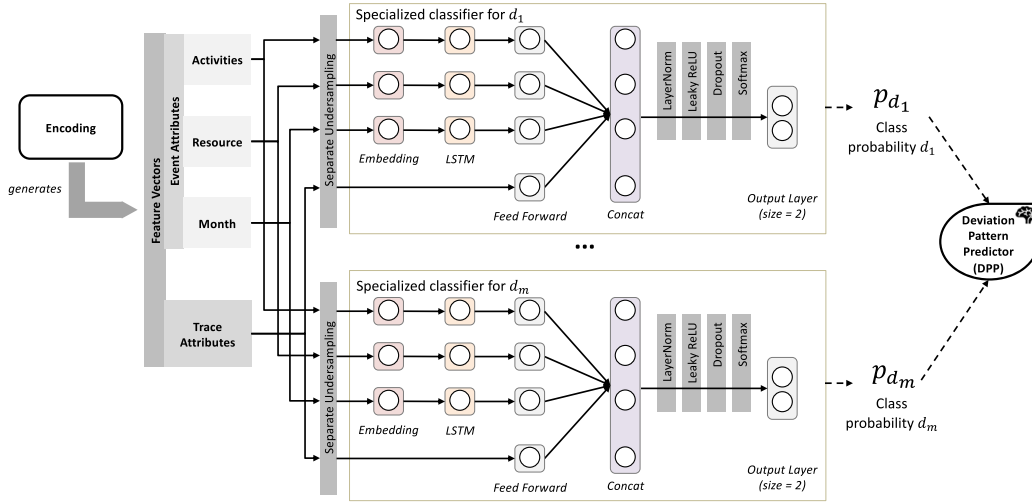
**Loss Function.** As the output layer of all classifiers consists of two neurons per deviation type activated with a Softmax function, we apply the weighted loss function for each deviation type separately. Concretely, we use weight  $\alpha_{DC} = 16$  for the deviating class and  $\alpha_{CC} = 1$  for the conforming class.

**Hyperparameter Optimization.** For the determination of hyperparameters, we applied a grid search strategy. This method searches through a specified subset of hyperparameters, allowing us to evaluate the performance of the model for each possible combination. Although high on computation time, we chose this strategy because it does not overlook potentially optimal combinations, unlike random search or heuristic methods which might miss critical parameter interactions [39]. We determined the hidden layers, the weight  $\alpha_{DC}$ , and dropout rate through a hyperparameter optimization. Concretely, we used the following hyperparameters, boldness indicating the final choice:

- Hidden layer structure: [32 × 32; 64 × 64; **256 × 256**; 512 × 256 × 256]
- $\alpha_{DC}$ : [4; 8; **16**; 24; 32]
- Dropout: [0.0; **0.1**; 0.2]

<sup>5</sup> <https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51>.

<sup>6</sup> <https://doi.org/10.13140/RG.2.2.11870.28487>.

Fig. 4. Network architecture of IDP-separate<sup>FFN</sup>.Fig. 5. Network architecture of IDP-separate<sup>LSTM</sup>.

### 5.2.2. IDP-separate<sup>LSTM</sup>

We now introduce a Long Short-Term Memory (LSTM) model design for IDP-separate. In particular, we adapt the hidden layer structures as proposed in [40], maintaining the architecture of multiple specialized classifiers for each deviation type. Concretely, instead of the feed-forward model, we use three LSTMs in combination with an embedding layer and a feed-forward layer as illustrated in Fig. 5, where each LSTM is dedicated to one event attribute encoded in CIBE (i.e., activities, resources, or month). To incorporate trace attributes in a non-sequential manner, we use a dedicated feed-forward layer for them. A concatenated representation of all layers is then fed into an output layer with two neurons to perform binary classification for one deviation type.

**Undersampling & Loss Function.** As for IDP-separate<sup>FFN</sup>, we undersample the training set for each deviation type individually. We also apply the WCEL per deviation type with weight  $\alpha_{DC} = 16$ .

**Hyperparameter Optimization.** Again, we conducted a grid search with the following parameters, boldness indicating the final choice:

- Embedding dimension per event attribute: [8; **16**; 32; 64]
- LSTM layer size per event attribute: [8; 16; 32; **64**; 128; 256]
- $\alpha_{DC}$ : [4; 8; **16**; 24; 32]
- Dropout: [0.0; **0.1**; 0.2]

### 5.2.3. IDP-collective<sup>FFN</sup>

This design consists of a single classifier to predict all deviation types collectively. Similar to IDP-separate<sup>FFN</sup>, it uses a feed-forward model with an input layer of size equal to the number of encoded features in CIBE. IDP-collective<sup>FFN</sup> has two larger hidden layers of 2048 and 1024 neurons to acknowledge the difficulty of collective prediction. These hidden layers are connected to an output layer with a number of neurons equal to the number of deviation types  $m$  in the log, activated by a Sigmoid function. This Sigmoid function returns the class probabilities of all deviation types collectively in one vector  $P_D$ . The architecture is illustrated in Fig. 6.

**Undersampling.** For design, we have to undersample all deviation types collectively as this is how they are learned and predicted. Thus, we apply one-sided selection to all multi-label targets. This ensures that we do not reduce the number of deviating traces for any deviation type.

This undersampling strategy constitutes a difference in contrast to IDP-separate which manifests in the training data. Fig. 8 illustrates the effects on the training data for BPIC12 A. We see that before undersampling, the number of deviating traces is much smaller than the number of conforming traces. Applying one-sided selection per deviation type in IDP-separate reduces the number of conforming traces much more than collective undersampling, leading to more equal distribution. Especially for very imbalanced deviation types like ( $\gg$ ,  $A\_DECLINED$ ), this



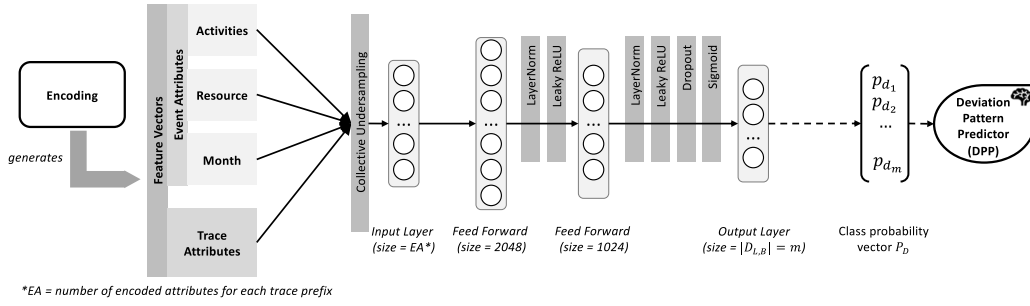


Fig. 6. Network architecture of IDP-collective<sup>FFN</sup>.

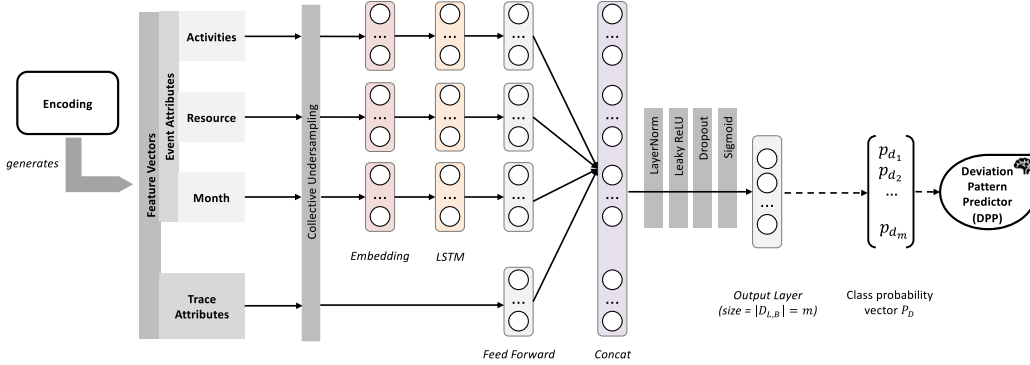


Fig. 7. Network architecture of IDP-collective<sup>LSTM</sup>.

effect is strong. Hence, the resulting training sets for IDP-separate are more balanced between conforming and deviating traces than the training sets for IDP-collective.

**Loss Function.** The output layer consists of  $m$  neurons activated with a Sigmoid function, so we apply the WCEL for all deviation types simultaneously. Concretely, we calculate the label imbalance ratio  $LIR_d$ , defined as the ratio of conforming traces  $CC(d)$  and deviating traces  $DC(d)$  for each  $d \in D_{L,B}$ . The collective weights are then determined as a vector of size  $m$  where each entry refers to one deviation type. Given  $LIR_d$ , the weight for type  $d$  is determined as  $\beta_d = 16^{\frac{1}{2 \times e} + \log(LIR_d)}$ . This way, the weights account for label imbalance within all deviation types, assigning higher weights to more imbalanced ones.

**Hyperparameter Optimization.** We conducted a grid search with the following hyperparameters, boldness indicating the final choice:

- Hidden layer structure: [256 × 256; 256 × 256 × 256; 1024 × 1024; **2048 × 1024**; 2048 × 1,024 × 1024; 2048 × 2,048 × 2048]
- $\beta_d$ : [ $8^{\frac{1}{2 \times e} + \log(LIR_d)}$ ;  **$16^{\frac{1}{2 \times e} + \log(LIR_d)}$** ;  $24^{\frac{1}{2 \times e} + \log(LIR_d)}$ ;  $\log(LIR_d)$ ]
- Dropout: [0.0; **0.1**; 0.2]

5.2.4. IDP-collective<sup>LSTM</sup>

This design, again, uses three LSTMs in combination with an embedding layer and a feed-forward layer as illustrated in Fig. 7, following the same logic as the LSTM-based IDP-separate<sup>LSTM</sup>. In contrast to that design, the output layer does not perform binary classification dedicated to one deviation type. Rather, it collectively predicts all  $m$  deviation types like IDP-collective<sup>FFN</sup>.

**Undersampling & Loss Function.** As for IDP-collective<sup>FFN</sup>, we apply undersampling in form of one-sided selection for all deviation types simultaneously, treating each deviation type with equal importance. Similar, we apply a collective WCEL with a vector based on the label imbalance of the deviation type, determined with  $\beta_d = 16^{\frac{1}{2 \times e} + \log(LIR_d)}$ .

**Hyperparameter Optimization.** Again, we conducted a grid search with the following parameters, boldness indicating the final choice:

- Embedding dimension per event attribute: [8; **16**; 32; 64]
- LSTM layer size per event attribute: [8; 16; 32; 64; **128**; 256]
- $\beta_d$ : [ $8^{\frac{1}{2 \times e} + \log(LIR_d)}$ ;  **$16^{\frac{1}{2 \times e} + \log(LIR_d)}$** ;  $24^{\frac{1}{2 \times e} + \log(LIR_d)}$ ;  $\log(LIR_d)$ ]
- Dropout: [0.0; **0.1**; 0.2]

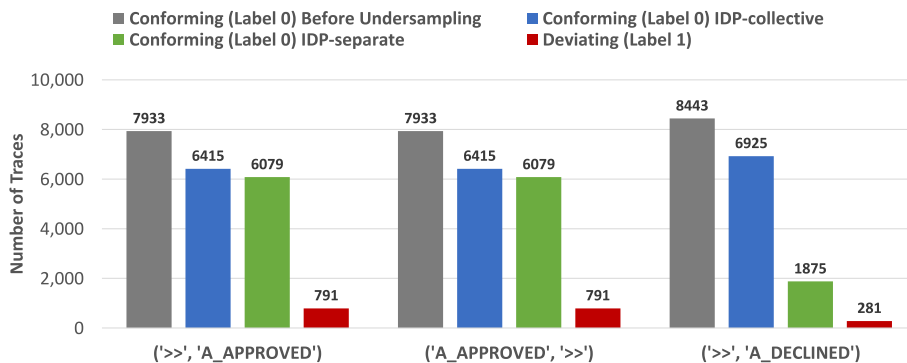


Fig. 8. Effect of undersampling in BPIC12A.

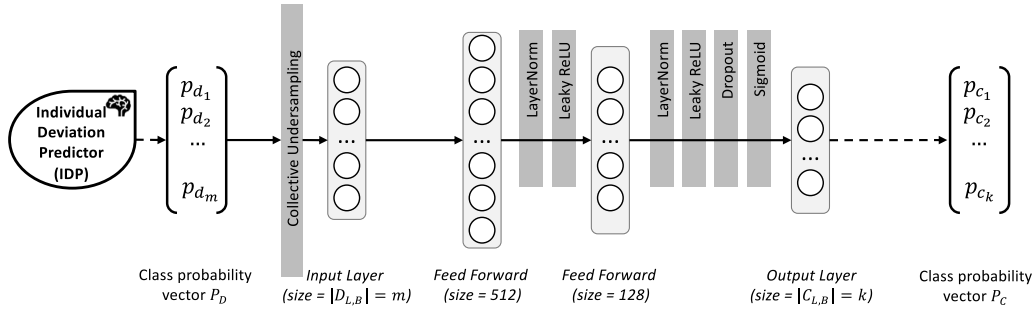


Fig. 9. Network architecture of DPP.

### 5.3. Investigated Network Implementation of DPP

We use the same architecture of the deviation pattern predictor DPP for all adaptations of IDP. We do so because the performance of DPP largely depends on the output of IDP, i.e., a vector  $P_D$  of class probabilities, one probability per deviation type.<sup>7</sup> Furthermore, the same DPP-architecture for all IDP-versions is appropriate since the vectors  $P_D$  are low-dimensional, i.e., with  $m$  values only.

As illustrated in Fig. 9, DPP has an input layer of  $m$  neurons which is fed with the class probability vector  $P_D$ . This input layer is connected to two hidden layers with 512 and 128 neurons, respectively. Finally, the output layer with  $k$  neurons, i.e., the number of deviation patterns in  $C_{L,B}$ , collectively predicts whether deviation patterns will occur, resulting in a vector  $P_C$  of class probabilities with size  $k$ . This architecture is sufficiently large to capture dependencies within the class probabilities but also suited to the low dimensionality of  $P_D$ .

**Undersampling & Loss Function.** We undersample the training data by applying one-sided selection collectively to all deviation patterns. Further, we apply a WCEL for all deviation patterns by determining a vector of size  $k$  where each entry represents the weight of one deviation pattern  $c$  dependent on its imbalance ratio  $LIR_c$ , calculated by  $\beta_c = \frac{1}{16^{\frac{1}{2x_c} + \log(LIR_c)}}$ .

We did not conduct a separate, systematic hyperparameter optimization for DPP due to the low dimensionality of its input, i.e., the  $m$  class probabilities in  $P_D$ . Therefore, no complex network architecture is required. Further, we noted that the performance of DPP was very similar with different hyperparameters based on the same class probabilities from any design of IDP on our experiments. Rather, we found that the probability vector  $P_D$  stemming from the Level-0 classifiers impacts the performance of the Level-1 classifier DPP, which cannot be influenced in an optimization of DPP.

### 5.4. Experimental results

Each network implementation of IDP is trained on the train set to predict all individual deviation types  $D_{L,B}$ . Then, we use the train set vectors of class probabilities  $P_D$  of the IDP versions to train DPP for all deviation patterns  $C_{L,B}$ . For all logs, we test the prediction quality for individual deviation types and deviation patterns on the test set. In the following, we present the performance of the different implementations on the used event logs. Thereby, we differentiate between the performance in predicting individual deviations (Section 5.4.1) and predicting deviation patterns (Section 5.4.2) since both tasks are of importance for proactive conformance checking.

<sup>7</sup> For IDP-separate, the individual class probabilities are concatenated into  $P_D$ .

#### 5.4.1. Predicting individual deviations

Table 5 shows the performance of the four network implementations for the seven event logs. We see that IDP-collective performs worst for all event logs, both as feed-forward version and LSTM version. For all event logs, IDP-separate achieves the highest  $AUC_{ROC}$ . Except for BPIC12 A, it also has the highest *Dev* recall. Thereby, we see that IDP-separate<sup>LSTM</sup> has similar predictive strength as IDP-separate<sup>FFN</sup>. However, *Dev* recall, which is of particular interest (C5), is lower for the LSTM model in all but two logs. Further, in four out of seven logs, the  $AUC_{ROC}$  is worse for the LSTM model, mainly caused by low precision and recall for rare deviation types.

This issue is further illustrated in Fig. 10, which shows a scatter plot depicting the relationship between  $AUC_{ROC}$  and the logarithm of number of prefixes labeled as deviating for any deviation type across all deviation types and event logs. We can see that IDP-separate<sup>LSTM</sup> has an  $AUC_{ROC}$  of less than or equal to 0.5 for 18 deviation types with less than 100 deviating prefixes, whereas this is only true for 4 deviation types in IDP-separate<sup>FFN</sup>. This means that the feed-forward model is much better equipped to predict rarely occurring deviation types. We suspect that this can be attributed to the better handling of trace attributes in the feed-forward model, which are of major importance in predicting these deviation types. Also, as illustrated in our repository, the LSTM models had substantially longer training times, which further reduced their practical applicability.

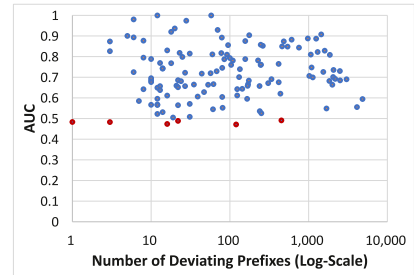
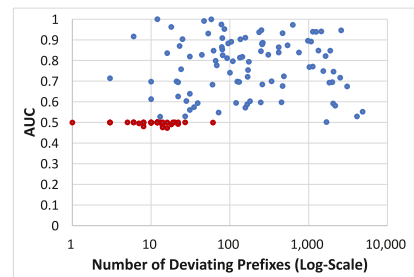
(a) IDP-separate<sup>FFN</sup>(b) IDP-separate<sup>LSTM</sup>

Fig. 10.  $AUC_{ROC}$  vs. support per deviation type in all logs; Color indicates  $AUC_{ROC} \leq 0.5$  (red) or  $> 0.5$  (blue).

The results suggest that a separate model per deviation type leads to best performance among the four designs. IDP-collective has the same architecture as IDP-separate but cannot separately undersample the training data, which shows that effective undersampling is required for accurate predictions. Further, IDP-separate can be specialized per deviation type whereas IDP-collective needs to balance predicting all deviations at once. Also, given that IDP-separate<sup>FFN</sup> has the highest Dev recall and AUC<sub>ROC</sub> for the majority of logs, we conclude that a

feed-forward architecture is best suited for deviation prediction, especially for highly imbalanced deviation types.

#### 5.4.2. Predicting deviation patterns

Table 6 shows the performance of the DPP implementation building on the four network implementations of IDP for the seven event logs. We see that Dev recall for deviation patterns is higher when building DPP on IDP-collective, either as FFN- or LSTM-version. However, this

**Table 5**

Performance of IDP-separate and IDP-collective for individual deviation prediction. Boldness indicates best score for this event log.

Log		IDP-separate				IDP-collective			
		IDP-separate <sup>FFN</sup>		IDP-separate <sup>LSTM</sup>		IDP-collective <sup>FFN</sup>		IDP-collective <sup>LSTM</sup>	
		Dev	No Dev	Dev	No Dev	Dev	No Dev	Dev	No Dev
BPIC12A	Prec.	0.1620	0.9669	0.1958	0.9691	0.0877	0.9410	0.0779	0.9432
	Rec.	0.8084	0.6563	0.7980	0.7251	<b>0.9251</b>	0.1186	0.6377	0.3601
	AUC	0.7324		<b>0.7616</b>		0.5218		0.4989	
BPIC12O	Prec.	0.2019	0.9625	0.1942	0.9494	0.7619	0.8830	0.1433	0.9817
	Rec.	<b>0.7981</b>	0.5348	0.6437	0.6195	0.1706	0.9959	0.6874	0.4308
	AUC	<b>0.6665</b>		0.6316		0.5832		0.5591	
Dom.	Prec.	0.1401	0.9982	0.1386	0.9987	0.7258	0.9934	0.0091	0.9953
	Rec.	0.7619	0.8897	<b>0.7975</b>	0.8784	0.1575	0.9995	0.2904	0.7964
	AUC	0.8258		<b>0.8319</b>		0.5784		0.4979	
Int.	Prec.	0.0720	0.9938	0.1151	0.9976	0.3911	0.9849	0.0438	0.9945
	Rec.	0.6333	0.8223	<b>0.6345</b>	0.8629	0.0005	1.0000	0.3131	0.8438
	AUC	0.7270		<b>0.7493</b>		0.5002		0.5689	
RfP	Prec.	0.0402	0.9979	0.1622	0.9988	0.2973	0.9952	0.0741	0.9972
	Rec.	<b>0.6888</b>	0.8353	0.6318	0.8573	0.1083	0.9993	0.3745	0.7736
	AUC	<b>0.7620</b>		0.7493		0.5537		0.5572	
Prep.	Prec.	0.0457	0.9969	0.0609	0.9965	0.0294	0.9896	0.0174	0.9972
	Rec.	<b>0.5566</b>	0.8514	0.4438	0.8652	0.0012	0.9996	0.3539	0.7486
	AUC	<b>0.7040</b>		0.6547		0.5004		0.5339	
MobIS	Prec.	0.0993	0.9748	0.1177	0.9935	0.0612	0.9615	0.0741	0.9972
	Rec.	<b>0.7162</b>	0.5906	0.5455	0.7040	0.0035	0.9982	0.3745	0.7736
	AUC	<b>0.6534</b>		0.6259		0.5007		0.5571	

**Table 6**

Performance of DPP building on the different versions of IDP for deviation pattern prediction. Boldness indicates best score for this event log.

Log		DPP building on							
		IDP-separate				IDP-collective			
		IDP-separate <sup>FFN</sup>		IDP-separate <sup>LSTM</sup>		IDP-collective <sup>FFN</sup>		IDP-collective <sup>LSTM</sup>	
	Dev	No Dev	Dev	No Dev	Dev	No Dev	Dev	No Dev	
BPIC12A	Prec.	0.2927	0.9692	0.1582	0.9829	0.1269	0.9976	0.1164	0.9135
	Rec.	0.8765	0.5250	0.9265	0.4616	<b>0.9945</b>	0.2531	0.4725	0.6083
	AUC	<b>0.7507</b>		0.6941		0.6238		0.5404	
BPIC12O	Prec.	0.1859	0.9914	0.1869	0.9815	0.1836	0.9912	0.1710	0.9997
	Rec.	0.9654	0.4849	0.9210	0.5117	0.9654	0.4768	<b>0.9991</b>	0.4098
	AUC	<b>0.7251</b>		0.7164		0.7211		0.7044	
Dom.	Prec.	0.0543	0.9993	0.0106	0.9993	0.0104	0.9992	0.0075	0.9992
	Rec.	0.5490	0.9522	0.6961	0.4765	<b>0.7215</b>	0.5598	0.5333	0.6388
	AUC	<b>0.7387</b>		0.5756		0.6004		0.4981	
Int.	Prec.	0.0311	0.9986	0.0097	0.9995	0.0047	0.9993	0.0085	0.9995
	Rec.	0.6322	0.8598	0.7037	0.6489	0.5546	0.6312	<b>0.7383</b>	0.6947
	AUC	<b>0.7265</b>		0.6616		0.5408		0.6837	
RfP	Prec.	0.0462	0.9996	0.0055	0.9993	0.0061	1.0000	0.0052	0.9995
	Rec.	0.7292	0.8997	0.5817	0.4761	<b>1.0000</b>	0.4696	0.4902	0.5836
	AUC	<b>0.8013</b>		0.4887		0.6898		0.4853	
Prep.	Prec.	0.0619	0.9973	0.0103	0.9997	0.0129	0.9995	0.0074	0.9988
	Rec.	0.6881	0.8668	0.7624	0.6674	<b>0.8617</b>	0.5980	0.7362	0.5932
	AUC	<b>0.7692</b>		0.6952		0.7100		0.6554	
MobIS	Prec.	0.1257	0.9724	0.1687	0.9556	0.0866	0.9762	0.1374	0.9964
	Rec.	0.5458	0.7694	<b>0.5854</b>	0.7248	0.4729	0.6752	0.5285	0.7211
	AUC	<b>0.6576</b>		0.6551		0.5740		0.6248	

comes at the cost of low *Dev* precision. For example, recall is 1 for DPP building on IDP-collective<sup>FFN</sup> in RfP but precision is only 0.0061. This indicates significant false alarms and is not practically helpful.

In contrast, *Dev* recall of DPP building on IDP-separate<sup>FFN</sup> is also relatively high but is not accompanied by very low precision. This leads to the highest  $AUC_{ROC}$  for this setting for all event logs. Thereby, acceptable values are reached in all logs except MobIS. Consequently, we can conclude that the best network implementation to build DPP on is IDP-separate<sup>FFN</sup>, which already performed best in predicting individual deviations.

### 5.5. Final learning architecture & ML model design

After conducting all experiments, we conclude that the most suited design for IDP is one specialized feed-forward classifier per deviation type and using undersampling and WCEL. The final design of IDP is shown in Fig. 4. On this version of IDP, we build DPP as illustrated in Fig. 9, also using undersampling and a WCEL to collectively predict all deviation patterns. This final architecture of BPDP is used in the next section for benchmarking.

## 6. Benchmark

After experimentally determining the best architecture for our approach, we now benchmark the predictive strength of BPDP by comparing it to existing methods and other machine learning approaches. For that, we use the same seven event logs and metrics as in our experiments, introduced in Section 5.1. We introduce the used baselines in Section 6.1. We then evaluate the predictive strength for individual deviation types (IDP) in Section 6.2 and for deviation patterns (DPP) in Section 6.3.

### 6.1. Baselines

We want to illustrate how the previously determined best neural network architecture performs in comparison to an existing technique and other ML techniques. This is because the deep learning architecture in combination with CIBE is not necessarily the best model for deviation prediction. Other models, including alternative ML designs, could also tackle the challenges of this task. Thus, we compare BPDP to four other approaches: one baseline from an existing approach [11] and three alternative ML approaches in form of CatBoost, MPPN, and suffix prediction. In the following, we describe these baselines.<sup>8</sup>

#### 6.1.1. Subjective logic [11]

To show how BPDP relates to non-ML approaches, we compare it to an existing approach based on subjective logic [11].<sup>9</sup>

#### 6.1.2. CatBoost

To show how BPDP relates to other ML techniques geared towards learning from few examples, we compare it to CatBoost, a gradient boosting technique that builds on decision trees [41]. We use the same WCEL as for BPDP as well as the same architecture for deviation pattern prediction consisting of several Level-0 and one Level-1 classifier. This baseline is developed by us and not based on existing works.

<sup>8</sup> Although the approach by Weinzierl et al. [10] also predicts deviations, its predictions only pertain to the very next event in a binary manner and not per type. As this is a different task, a comparison is not adequate.

<sup>9</sup> Note that we only compare this approach to IDP. Comparing it to DPP would require substantial changes, for which the approach is not intended.

### 6.1.3. Multi-perspective Process Network (MPPN)

To show how BPDP relates to alternative neural network models, we compare it to the MPPN model [42] - a versatile process prediction model. In the previous version of the paper, we pre-trained the MPPN on the next step prediction task and used its trace prefix embeddings as alternative encodings (instead of CIBE) to BPDP. However, training BPDP with CIBE outperformed training BPDP with MPPN embeddings. In this paper, we train the MPPN model end-to-end on the task of deviation prediction. This design features a modular architecture of three parts that can process any combination of attributes found in the event log [42], making it a flexible process representation learning model. Each attribute perspective is transformed into a Gramian Angular Field (GAF) [43], a matrix of size  $n \times n$  for an attribute sequence of length  $n$ , indicating the relation between each pair of attribute values. The first part of the model is a single, pre-trained convolutional neural network (CNN) that extracts features from each Gramian Angular Field. They are processed in a central neural network that enables learning relations between attributes. This central part consists of three fully-connected layers with 128 neurons each. On top of that, a flexible number of small networks, called heads, can be added for classification. We pre-train the MPPN once per event log in predicting attribute values of the next event to make it sensitive in learning general characteristics of the process and dependencies between attributes. Afterward, we fine-tune the model collectively for all  $m$  deviation types, keeping the CNN and fully connected part of the model but replacing the heads. Each head consists of a single fully connected layer that performs binary classification. During fine-tuning, the model should learn dependencies in event attributes and deviation types that lead to non-conforming behavior.

As this design predicts all deviation types collectively, it also undersamples the prefixes collectively. Also, it applies WCEL with the same weight for all deviation types per event log. As the strategy to set the weights based on the imbalance per deviation type (as done for IDP-collective) did not work well, we empirically assess the best weighted loss  $\alpha_{DC}$  per event log as a hyperparameter with the values [16, 32, 64, 96]. Thus, MPPN is optimized for each event log, using higher weights for logs with larger imbalance and lower values for logs with smaller imbalance. This potentially overestimates its performance in comparison to BPDP (which uses the same  $\alpha_{DC}$  for all logs), which we accept since it makes the benchmark more robust.

For deviation pattern prediction, we train MPPN end-to-end instead of building a Level-1 MPPN on top of the Level-0 MPPN. This is technically not possible as the MPPN requires GAFs as input which we cannot transform the probability vector  $P_D$  into. Therefore, we follow the same procedure as deviation prediction and fine-tune the same pre-trained MPPN on the deviation pattern labels. Again, we apply the same undersampling strategy and use loss weight which worked best for the event log.

#### 6.1.4. Suffix prediction

An alternative to explicitly predicting deviations directly is to look for deviations in the predicted continuation of a prefix. The idea is that a next-step prediction approach is able to predict different continuations for prefixes that will or will not deviate in the future, thus implicitly predicting deviations. By aligning the predicted traces, we can detect the deviating ones. Similar to the previous one, we implemented this baseline ourselves. In particular, we used the pre-trained MPPN [42], i.e., not fine-tuned for deviation prediction, to obtain suffixes by predicting new events until the end activity has been reached. We made use of MPPN's ability to predict all next event's attribute values: iteratively, the attribute values of the next event  $e_{n+1}$  are predicted, appended to the existing suffix, and used as input for MPPN again. All complete suffixes are aligned to the process model  $B$ , using the PM4Py implementation [31]. The same approach of fixing the alignments with the fewest different log and model moves across 100 different alignment computations as applied in labeling (Section 4.2) is

used here, too. We infer deviation “predictions” by collecting all model or log moves found in the alignment result. If a deviation of type  $d_i$  is found in the alignment after the current activity of the prefix, we say it was predicted. Similarly, predictions for deviation patterns are inferred by combining all individual predictions in the suffix, i.e., a deviation pattern  $c_i$  is predicted if all deviation types  $d_i \in c_i$  are predicted. Note that by predicting suffixes and aligning them, new model or log moves can be found which do not match any existing deviation type defined beforehand. This can happen if a trace suffix is predicted that

has not been observed before. Such new alignments are ignored during evaluation as no ground truth data to assess against exists.

### 6.2. Benchmark of individual deviation prediction

We now evaluate the predictive strength of BPDP for the prediction of individual deviation types. Table 7 displays average precision, recall, and  $AUC_{ROC}$  for the seven event logs and compares IDP to the four baselines.

**Table 7**  
Precision, Recall, and  $AUC_{ROC}$  for Individual deviation prediction for all event logs. Boldness indicates best score for this event log.

Log		Baselines								IDP	
		[11]		CatBoost		MPPN [42]		Suffix		Dev	No Dev
		Dev	No Dev	Dev	No Dev	Dev	No Dev	Dev	No Dev		
BPIC12A	Prec.	0.2136	0.9214	0.1694	0.9622	0.1040	0.9869	0.3766	0.9256	0.1620	0.9669
	Rec.	0.0678	0.9654	0.7110	0.6797	<b>0.9648</b>	0.3021	0.0967	0.9895	0.8084	0.6563
	AUC	0.5166		0.6954		0.6335		0.5431		<b>0.7324</b>	
BPIC12O	Prec.	0.1462	0.8663	0.2277	0.9566	0.1627	0.9901	0.3282	0.8811	0.2019	0.9625
	Rec.	0.1340	0.8403	0.6034	0.7148	<b>0.8146</b>	0.4240	0.2128	0.8504	0.7981	0.5348
	AUC	0.4872		0.6591		0.6193		0.5316		<b>0.6665</b>	
Dom.	Prec.	0.1961	0.7318	0.4035	0.9977	0.0688	0.9974	0.1934	0.9964	0.1401	0.9982
	Rec.	0.2040	0.7205	0.5110	0.9930	0.3182	0.8982	0.2900	0.9868	<b>0.7619</b>	0.8897
	AUC	0.6372		0.7511		0.6082		0.7023		<b>0.8258</b>	
Int.	Prec.	0.1738	0.8823	0.3648	0.9955	0.0864	0.9981	0.1096	0.9911	0.0720	0.9938
	Rec.	0.1648	0.8684	0.3866	0.9884	0.5367	0.8779	0.2652	0.9732	<b>0.6333</b>	0.8223
	AUC	0.5796		0.6969		0.7073		0.6338		<b>0.7270</b>	
RfP	Prec.	0.1480	0.7352	0.4630	0.9979	0.0686	0.9980	0.2259	0.9972	0.0402	0.9979
	Rec.	0.1244	0.7273	0.3967	0.9965	0.3574	0.8878	0.2064	0.9908	<b>0.6888</b>	0.8353
	AUC	0.5762		0.6961		0.6226		0.6335		<b>0.7620</b>	
Prep.	Prec.	0.1475	0.8705	0.3032	0.9949	0.0811	0.9961	0.1373	0.9943	0.0457	0.9969
	Rec.	0.1067	0.8629	0.2727	0.9946	0.2851	0.9284	0.2447	0.9804	<b>0.5566</b>	0.8514
	AUC	0.5521		0.6333		0.6067		0.6284		<b>0.7040</b>	
MobIS	Prec.	0.1211	0.8355	0.1363	0.9296	0.1555	0.9975	0.1176	0.9697	0.0993	0.9748
	Rec.	0.1245	0.8415	0.2254	0.8907	0.4421	0.7617	0.2063	0.9599	<b>0.7162</b>	0.5906
	AUC	0.5461		0.5729		0.6019		0.5958		<b>0.6534</b>	

**Table 8**  
Precision, Recall, and  $AUC_{ROC}$  for Deviation pattern predictions for all event logs. Boldness indicates best score for this event log.

Log		Baselines						DPP	
		CatBoost		MPPN [42]		Suffix		Dev	No Dev
		Dev	No Dev	Dev	No Dev	Dev	No Dev		
BPIC12A	Prec.	0.1716	0.9177	0.1665	0.9531	0.0000	0.9012	0.2927	0.9692
	Rec.	0.3145	0.8342	0.7413	0.5865	0.0000	0.9965	<b>0.8765</b>	0.5250
	AUC	0.5743		0.6639		0.4983		<b>0.7507</b>	
BPIC12O	Prec.	0.2120	0.9248	0.1870	0.9807	0.0228	0.8608	0.1859	0.9914
	Rec.	0.4769	0.7840	0.8595	0.6559	0.0551	0.7122	<b>0.9654</b>	0.4849
	AUC	0.6304		<b>0.7577</b>		0.3836		0.7251	
Dom.	Prec.	0.2327	0.9951	0.0641	0.9989	0.1794	0.9991	0.0543	0.9993
	Rec.	0.1988	0.9966	0.2976	0.9944	<b>0.6703</b>	0.9857	0.5490	0.9522
	AUC	0.5977		0.6460		<b>0.8245</b>		0.7387	
Int.	Prec.	0.4267	0.9974	0.0926	0.9993	0.0479	0.9979	0.0311	0.9986
	Rec.	0.3736	0.9992	0.5004	0.9609	0.1799	0.9795	<b>0.6322</b>	0.8598
	AUC	0.6863		<b>0.7307</b>		0.5762		0.7265	
RfP	Prec.	0.2957	0.9963	0.0000	0.9993	0.0743	0.9992	0.0462	0.9996
	Rec.	0.2801	0.9092	0.0000	1.0000	0.2923	0.9905	<b>0.7292</b>	0.8997
	AUC	0.6588		0.5000		0.6398		<b>0.8013</b>	
Prep.	Prec.	0.2957	0.9963	0.0549	0.9977	0.0435	0.8965	0.0619	0.9973
	Rec.	0.2801	0.9092	0.2507	0.9723	0.1393	0.8875	<b>0.6881</b>	0.8668
	AUC	0.6588		0.6115		0.5610		<b>0.7692</b>	
MobIS	Prec.	0.2440	0.9539	0.1295	0.9982	0.2576	0.9490	0.1257	0.9724
	Rec.	0.1715	0.9715	<b>0.6038</b>	0.7614	0.3440	0.9518	0.5458	0.7694
	AUC	0.5715		<b>0.6861</b>		0.5119		0.6576	

As we see from the  $AUC_{ROC}$  of around 0.5, subjective logic [11] performs only slightly better than random guessing. This is most likely caused by the high class imbalance in the data, which is much higher than presumed in the original paper. Both CatBoost and suffix prediction reach an acceptable  $AUC_{ROC}$  only for Dom., with CatBoost showing better *Dev* recall. MPPN reaches an acceptable  $AUC_{ROC}$  for the Int. event log and high *Dev* recall for the BPIC12 logs. This indicates that its sophisticated design and the pre-training allows to predict individual deviation collectively in some logs. Thus, we can see that complex ML models can partially address the challenges of predicting deviations collectively. However, MPPN is outperformed by specialized models per deviation type, i.e., CatBoost and IDP, that can deal better with strong class imbalance. For deviation types with strong class imbalance, MPPN performs considerably bad.

IDP outperforms all baselines and reaches mostly acceptable to excellent  $AUC_{ROC}$  values. The high *Dev* recall indicates that a majority of future deviations are predicted in prefixes, which we aimed for to address C5. We also see that the performance of IDP differs between the logs, as shown by  $AUC_{ROC}$ . For BPIC12 A, Dom., Int., RfP, and Prep.,  $AUC_{ROC}$  indicates (very) good predictive strength. For BPIC12O and MobIS, no approach reaches an acceptable  $AUC_{ROC}$ , which is likely due to long traces with differing control-flow in combination with few trace attributes.

With regard to prediction time, all approaches demonstrate reasonable performance with normal computational resources. As shown in our repository, IDP performs comparable to CatBoost and Suffix Prediction.

### 6.3. Benchmark of deviation pattern prediction

After assessing BPDP's ability to predict individual deviations, we now evaluate its strength in predicting deviation patterns. Table 8 displays average precision, recall, and  $AUC_{ROC}$  for the seven event logs and compares the performance of DPP to the three applicable baselines CatBoost, MPPN, and suffix prediction. DPP achieves the best  $AUC_{ROC}$  for three event logs, MPPN also for three event logs, and suffix prediction for one event log. However, DPP is the only approach that does not perform significantly worse in other logs, whereas both MPPN and suffix prediction show at least one log with  $AUC_{ROC}$  of 0.5 or below. Further, DPP achieves an acceptable  $AUC_{ROC}$  for all logs except MobIS, where IDP also did not reach an acceptable  $AUC_{ROC}$ . This shows that performance of DPP depends on the performance of IDP. The input for pattern prediction is the output from the individual deviations, leading to worse performance in case the Level-0 classifier does not perform well. Finally, the high *Dev* recall again indicates that a majority of future deviation patterns are predicted in prefixes, thus effectively addressing C5 for deviation patterns as well. Since DPP reaches the best *Dev* recall in the majority of event logs and, as the only approach, an acceptable  $AUC_{ROC}$  in six logs, we consider it as the overall best performing model.

## 7. Use case evaluation

As we have found BPDP to outperform all baselines, we now further evaluate its performance with regard to the specifics of the context in which deviation prediction is applied. This use case evaluation is twofold:

- (1) In real-life applications, BPDP should predict deviations sufficiently early before they occur. Thus, we evaluate prediction *earliness* in Section 7.1.
- (2) To demonstrate that BPDP can support preventing deviations in the sense of action-oriented process mining, we evaluate the *feature importance for action orientation* in Section 7.2. From this analysis, potentially deviation-causing characteristics can be derived.

Finally, we demonstrate how BPDP can be used in practice to supply process managers with information about upcoming deviations in form of a prediction GUI in Section 7.3. This includes all features of BPDP introduced so far.

In this section, we evaluate IDP only. For earliness, early predictions of individual deviations indicate early predictions of patterns as DPP builds on the output of IDP. Consequently, an evaluation of IDP's earliness suffices to show the use case value. Similarly, the feature importance is useful only if the features contain information on potentially deviation-causing attribute values, which is the case for IDP but not for DPP. For all use case evaluations, we use the event logs from Table 4 and evaluate on the test set only.

### 7.1. Earliness

Earliness of IDP should indicate how soon before the deviation occurrence a prefix is classified correctly. Existing outcome prediction approaches define earliness based on the total trace length [16]. However, as deviations occur within traces, we altered the metric to show how many events in advance a deviation is detected. We only consider traces for which IDP predicted the deviation correctly. For one trace and one deviation type, earliness  $ea(t, d)$  is defined in Eq. (2a). It is based on prediction position  $pp$  and deviation position  $dp$ .  $pp$  is the prefix length where the deviation was predicted first (without any wrong prediction after).  $dp$  is the position in the trace where the deviation occurred. The lower  $ea(t, d)$ , the earlier a deviation is detected.

To construct a baseline for the earliness, we define a theoretically optimal earliness  $ea^{opt}(t, d)$  in Eq. (2b). This lower bound  $ea^{opt}(t, d)$  shows the value of  $ea$  if the deviation would have been recognized at length 1, i.e.,  $pp(t, d) = 1$ . If a deviation happens at position 10,  $ea^{opt}(t, d)$  is 0.1. The earlier the deviation occurs in the trace, the higher the theoretically optimal earliness.

$$ea(t, d) = \frac{pp(t, d)}{dp(t, d)} \quad (2a)$$

$$ea^{opt}(t, d) = \frac{1}{dp(t, d)} \quad (2b)$$

Additionally, we define the reaction time (rt), which indicates the number of events between correct prediction and deviation, according to Eq. (3).

$$rt(t, d) = dp(t, d) - pp(t, d) \quad (3)$$

The average values for  $ea(t, d)$  (also applicable to  $ea^{opt}(t, d)$ ) and  $rt$  for a log  $L$  with  $|L|$  traces and a corresponding to-be model  $B$  are defined in Eqs. (4) and (5).

$$ea_{L,B} = \frac{1}{|D_{L,B}| \times |L|} \sum_{d \in D_{L,B}} \sum_{t \in L} ea(t, d) \quad (4)$$

$$rt_{L,B} = \frac{1}{|D_{L,B}| \times |L|} \sum_{d \in D_{L,B}} \sum_{t \in L} rt(t, d) \quad (5)$$

Table 9 displays the earliness values, including averages (Avg.) and standard deviation (SD) for  $ea_{L,B}$  as well as Avg. - SD for  $rt_{L,B}$ . For event logs where deviations tend to occur early (i.e., high  $ea_{L,B}^{opt}$ ), IDP predicts these deviations relatively early as well. This is particularly apparent for Dom. and RfP. The SD indicates that the spread of earliness values is narrow in relation to the average for all logs except for Int. and Prep., the two logs with the most different deviation types. This is because the SD is rather high for many of the highly imbalanced deviation types in these logs. For event logs where deviations tend to occur later (i.e., low  $ea_{L,B}^{opt}$ ), the earliness is worse, as seen in, e.g., the MobIS log. We suspect that deviations which only occur later in traces are harder to predict in earlier stages due to strong similarities of prefixes in these earlier stages in the MobIS log. Nevertheless, for all event logs,  $rt$  indicates that deviations are, on

**Table 9**

Comparison of prediction earliness to optimal earliness. Portrayed are Averages (Avg.) and Standard Deviation (SD). Lower earliness values  $ea_{L,B}$  and higher reaction time values  $rt_{L,B}$  are preferable.

Log	$ea_{L,B}^{opt}$	$dp_{L,B}$	IDP			
			$ea_{L,B}$		$rt_{L,B}$	
			Avg.	SD	Avg.	Avg. - SD
BPIC12A	0.16	6.21	0.25	0.14	4.65	3.76
BPIC12O	0.18	5.62	0.36	0.14	3.60	2.82
Dom.	0.38	2.67	0.46	0.13	1.44	1.08
Int.	0.19	5.18	0.35	0.24	3.38	2.12
RfP	0.35	2.85	0.46	0.19	1.53	1.00
Prep.	0.22	4.65	0.37	0.21	2.91	1.93
MobilS	0.12	8.41	0.36	0.18	5.36	3.81

average, correctly predicted at least 1 event in advance, giving process managers time to address upcoming deviations. This is still true when deducting the standard deviation of  $rt$  from the average (Avg. - SD) for all logs.

**7.2. Feature importance for action orientation**

To help managers in preventing deviations in the future, we analyze potential reasons for their occurrence. Therefore, we use explainable artificial intelligence (XAI), which can make the predictions of IDP interpretable and identify process features that potentially cause deviations to occur [44]. Concretely, we compute the Shapley values of the predictions [45], which indicate how much impact the value of a certain input feature has on the prediction. In our case, a negative Shapley value indicates that this specific feature value increases the likelihood for not predicting a deviation. A positive Shapley value indicates that the feature value increases the likelihood for predicting a deviation. In general, the higher the absolute Shapley value, the higher the impact of this feature value on the prediction. For IDP, we are interested in features with high positive Shapley values as those indicate deviation occurrences and should therefore be avoided. Features with high negative Shapley values indicate conforming behavior and can also be helpful.

Referring to the running example in Section 2, Fig. 11 shows the Shapley values for the ten most important features to predict deviation type  $d_1$ . For the binary features, the color indicates their presence (red) or absence (blue). For the numerical feature AMOUNT\_REQUIRED, it indicates high values (red) or low ones (blue). We see that a higher amount increases the likelihood of a deviation prediction. The same is true if the trace starts on a Tuesday or Saturday or if the first event is executed in November. On the contrary, if the third event is

executed by resource 112 or if this third event is executed in October, the probability for predicting a deviation is significantly lower.

The Shapley values offer two interesting insights for the use case evaluation. First, they make the predictions of the trained classifiers interpretable. Second, they point towards those process features that should be investigated when trying to prevent deviations from happening. In case of BPIC12 A, these process features are the loan amount as well as the weekday and the month where the trace started. Additionally, using process knowledge of resource 112 could help to define actions that prevent the deviation ( $\gg$ , A\_APPROVED). This information cannot be obtained from classical conformance checking.

**7.3. Prediction GUI**

In Fig. 12, we combine all features of BPDP and illustrate how it can support process managers in handling deviation predictions.

It shows a prototypical graphical user interface with the exemplary prediction results for a running process instance of the BPIC12 A. The interface is separated in four sections: (1) the to-be process model (top), (2) descriptive features for the selected case, including trace attributes and the current control-flow progress (middle), (3) individual deviation predictions from IDP (also annotated in the model), including the prediction probability on the left side as well as the top two features with the highest Shapley values that lead to this prediction, and (4) deviation pattern predictions from DPP for the top two deviation patterns that occur in the process (bottom). The prediction probability of each deviation indicates the models' belief in how likely this deviation will occur. We define this probability as the Softmax-activation of the neuron in the output layer of the neural network that represents the deviating class.

**8. Discussion**

The conducted experiments show that BPDP is well capable of predicting deviations. By tackling the five challenges of deviation prediction with customized design choices, it outperforms the three baselines for all event logs. In the following, we discuss our design choices and potential limitations.

We found that separate classifiers per deviation type outperformed the single classifiers that predict all deviation types collectively. This contradicted our initial presumption that a collective classifier would learn dependencies between deviation types and hence perform better than separate classifiers. The observed differences in performance can be attributed undersampling and weighted loss, which are more effective for separate classifiers. Due to the highly unequal imbalance between the deviation types, undersampling was less effective for collective predictions, but worked very well for IDP-separate which could

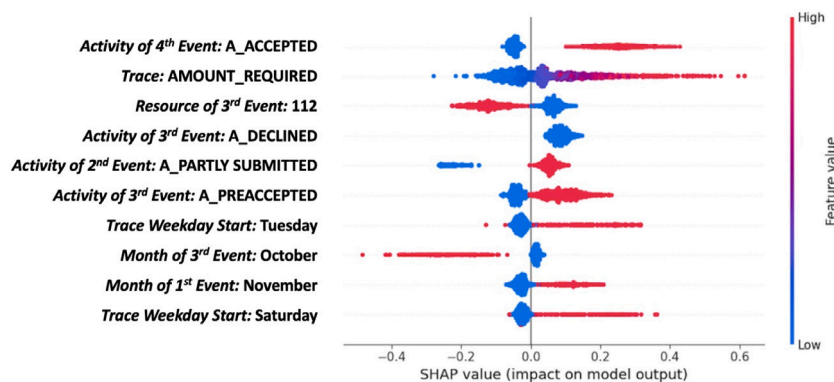


Fig. 11. Shapley values for correct predictions of ( $\gg$ , A\_APPROVED) in BPIC12A.

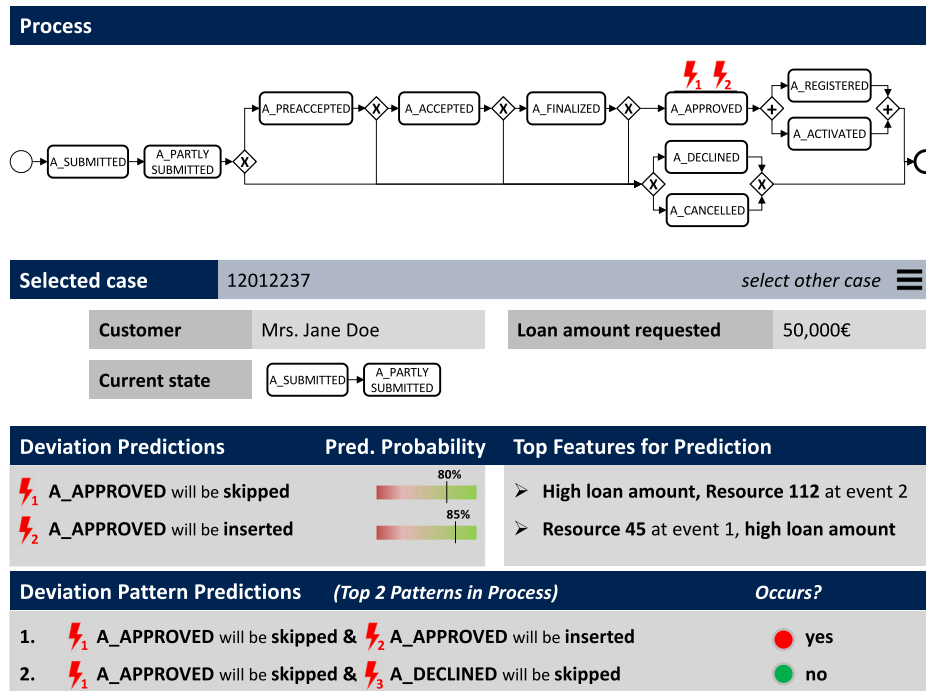


Fig. 12. The prediction GUI.

undersample the logs individually per deviation. The less effective undersampling for IDP-collective also explains the much lower recall of those models, resulting in a worse  $AUC_{ROC}$ . We noted that IDP-collective could not predict the very rare deviation types at all or at least in a sufficiently reliable manner. In contrast, IDP-separate was much more accurate across almost all deviation types.

For logs with exceptionally high imbalance (BPIC120 and MobIS), even separate classifiers plus undersampling were not enough to get acceptable  $AUC_{ROC}$  values. This strategy also came at the cost of lower precision than, e.g., CatBoost, resulting in many false positives. These findings hold for the prediction of individual deviation types but also for deviation patterns because the imbalance of patterns is, by definition, equally high or higher than those of the individual deviation types.

Another observation concerns the impact of additional context in form of trace attributes on the prediction quality. The performance of BPDP is better on the BPIC 20 logs with a high number of trace attributes than on other logs with fewer attributes. We suspect that more attributes lead to more accurate predictions for short prefixes as additional context allows to better differentiate between them. This is supported by our finding that the use of LSTM models to predict individual deviation types does not improve predictive strength, indicating that the ability to handle sequential data does not outweigh the ability to handle static trace attributes.

One drawback of the current design choices is the resulting complexity of the whole approach. For  $m$  deviation types in  $D_{L,B}$ , we have to train  $m$  networks with individual undersampling and loss weighting. In our experiments, we established that this model design best addresses the challenges at hand. Nevertheless, it would be advantageous to have to train only a single model per log, as done with IDP-collective and MPPN. Further, the current two-step approach to first predict individual deviations with IDP and using a second model to predict the patterns could be simplified to a model that predicts deviation types and patterns in one.

Another limitation of BPDP stems from the inherent non-determinism of alignments, which can make the prediction targets non-deterministic as well. In our experiments, we ensured full reproducibility by fixing the alignments per trace. When using BPDP

in practice, however, training data might still be contradictory if differing optimal alignments are returned for similar trace variants. We addressed this problem by fixing the alignments with the smallest number of deviation types to minimize the occurrence of different types for similar deviations. This heuristic approach should rule out non-determinism for most practical use cases.

Further, using oversampling and WCEL effectively led to good recognition of deviating traces but also to many false positives as indicated by low *Dev*-precision. While we choose this intentionally to ensure that all deviations are handed over to managers, it might lead to mistrust in practical applications, as many false alarms handed over to the managers. Potentially, a heuristic based on current position of the ongoing process instance, the prediction probability, and model certainty could be used to trigger manual trace inspection, thus reducing the number of false alarms. Our GUI does provide managers with some information for that, but a concrete heuristic would be preferable.

Lastly, we acknowledge that the usefulness of the Shapley values is limited since it is based on the features generated in CIBE. Thus, we can only refrain to Shapley values of features like “activity of third event” or “resource of third event” but not of features like “resource performs activity”, which might be more relevant, especially in complex processes. To improve the usefulness of the feature importance of features for action orientation, we want to experiment with joint Shapley values, i.e., Shapley values of a set of features [46]. This way, we could provide the importance of features like “resource X performs activity Y” or “activity Z is performed in September”.

## 9. Conclusion

In this paper, we advance proactive conformance checking and propose an approach for deviation prediction, called BPDP, which can predict deviations that will occur in the future of running process instances. We designed BPDP to explicitly consider external knowledge on the to-be process behavior, cope with multiple deviations and deviation patterns in one trace, incorporate context information in the trace, and prioritize the detection of deviating behavior. BPDP outperforms all baseline approaches and reaches acceptable to excellent prediction



performance. The evaluations also show that it predicts deviations sufficiently early, such that it can be used in practice.

In the future, we want to experiment with other types of conformance violations besides alignment-based deviations, such as rule violations. This could also include conformance violations defined with domain knowledge, in which case the prediction targets might be closer aligned to what process managers actually want to monitor. Related to that, we recently published an approach to detect deviation patterns of the types *insert*, *skip*, *repeat*, *replace*, and *swap* within trace alignments [47]. These pattern types allow for a more concrete assessment of deviating behavior, for example by explicating whether an activity is performed too early or too late in a trace (i.e., the activity is swapped with another activity). We aim to investigate whether it is possible to utilize these patterns for predictive purposes. Additionally, we aim to address action orientation by a heuristic which, dependent on prefix prediction and earliness, triggers the manual trace inspection by managers to reduce costs through false positives and a deviation severity score, which shows users the most urgent problems in their process. In the same vein, we not only want to predict whether a deviation will occur, but also at what point in the future. For example, we could predict the reaction time in form of events to react as defined in Section 7.1, i.e., how many events will occur in the ongoing trace before a deviation occurs. We also want to investigate if the suffix prediction can identify any new deviations which are not known so far but also meaningful. This could be used to warn the manager about unknown deviations that could potentially occur. Furthermore, although BDPD performs well, it can be improved in terms of encoding, network architecture, and learning strategy. For instance, different encoding approaches could be tested instead of CIBE. While IDP with fully connected layers performs better than with LSTM layers, most likely since the LSTM cannot capture trace attributes adequately, a fully connected architecture does not account for the processual characteristic of event data. An architecture that accounts for the processual nature of event data and trace attributes appropriately would be very advantageous. Further, we aim to develop a complete front-end user interface based on the mock-up shown in Fig. 12. In its current form, we have conceptually developed this GUI but not connected it to the predictive model in the back-end, which would be desirable. Finally, the training strategy could be improved. To better differentiate between conforming and non-conforming traces, Contrastive training approaches like a triplet-loss could be tested.

### CRedit authorship contribution statement

**Michael Grohs:** Writing – review & editing, Writing – original draft, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Peter Pfeiffer:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Jana-Rebecca Rehse:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

### Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Jana-Rebecca Rehse reports financial support was provided by German Research Foundation under grant number RE 4805/1-1. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

### Data availability

We have shared the data in a Gitlab repository linked in the paper.

### References

- [1] R. Zaman, A. Cuzzocrea, M. Hassani, An innovative online process mining framework for supporting incremental GDPR compliance of business processes, in: *IEEE Big Data*, 2019, pp. 2982–2991.
- [2] M. Jans, J.M. van der Werf, N. Lybaert, K. Vanhoof, A business process mining application for internal transaction fraud mitigation, *Expert Syst. Appl.* 38 (2011) 13351–13359.
- [3] J. Liang, M. Yang, J. Yang, Y. Deng, Identification of human operation deviation using a conformance checking technique, in: *POWERCON*, IEEE, 2021, pp. 551–555.
- [4] M. Hashmi, G. Governatori, H.-P. Lam, M.T. Wynn, Are we done with business process compliance: state of the art and challenges ahead, *KAIS* 57 (2018) 79–133.
- [5] H.A. López, S. Debois, T. Slaats, T.T. Hildebrandt, *Business process compliance using reference models of law*, in: *FASE*, Vol. 12076, Springer, 2020, pp. 378–399, [http://dx.doi.org/10.1007/978-3-030-45234-6\\_19](http://dx.doi.org/10.1007/978-3-030-45234-6_19).
- [6] A. Elgammal, O. Turetken, W.-J. van den Heuvel, M. Papazoglou, Formalizing and applying compliance patterns for business process compliance, *Softw. Syst. Model.* 15 (2016) 119–146.
- [7] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, *Conformance Checking - Relating Processes and Models*, Springer, 2018.
- [8] W.M.P. van der Aalst, *Process mining: A 360 degree overview*, in: *Process Mining Handbook*, Springer, 2022, pp. 3–34.
- [9] S. Dunzer, M. Stierle, M. Matzner, S. Baier, Conformance checking: A state-of-the-art literature review, in: *S-BPM ONE*, ACM, 2019, pp. 1–10.
- [10] S. Weinzierl, S. Dunzer, J. Tenschert, S. Zilker, M. Matzner, Predictive business process deviation monitoring, in: *ECIS*, 2021, pp. 1–10.
- [11] L. Genga, C. Di Francescomarino, C. Ghidini, N. Zannone, Predicting critical behaviors in business process executions: When evidence counts, in: *BPM Forum*, Springer, 2019, pp. 72–90.
- [12] S. van Zelst, A. Bolt, M. Hassani, B. Dongen, W. Aalst, Online conformance checking: relating event streams to process models using prefix-alignments, *Int. J. Data Sci. Anal.* 8 (2019) 269–284.
- [13] T. Nolle, A. Seeliger, M. Mühlhäuser, Binet: Multivariate business process anomaly detection using deep learning, in: *BPM*, Springer, 2018, pp. 271–287.
- [14] C. Di Francescomarino, C. Ghidini, Predictive process monitoring, in: *Process Mining Handbook*, Springer, 2022, pp. 320–346.
- [15] J. Evermann, J.-R. Rehse, P. Fettke, Predicting process behaviour using deep learning, *Decis. Support Syst.* 100 (2017) 129–140.
- [16] C.D. Francescomarino, M. Dumas, F.M. Maggi, I. Teinemaa, Clustering-based predictive process monitoring, *IEEE Trans. Serv. Comput.* 12 (2019) 896–909.
- [17] M. Grohs, P. Pfeiffer, J.-R. Rehse, Business process deviation prediction: Predicting non-conforming process behavior, in: *ICPM*, IEEE, 2023, pp. 113–120.
- [18] M. de Leoni, W.M. van der Aalst, M. Dees, A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs, *Inf. Syst.* 56 (2016) 235–257.
- [19] B. Depaire, J. Swinnen, M. Jans, K. Vanhoof, A process deviation analysis framework, in: *BPM Workshops*, Springer, 2013, pp. 701–706.
- [20] L. García-Bañuelos, N. Van Beest, M. Dumas, M. La Rosa, W. Mertens, Complete and interpretable conformance checking of business processes, *Trans. Softw. Eng.* 44 (3) (2017) 262–290.
- [21] L. Genga, M. Alizadeh, D. Potena, C. Diamantini, N. Zannone, Discovering anomalous frequent patterns from partially ordered event logs, *J. Intell. Inf. Syst.* 51 (2018) 257–300.
- [22] A. Adriansyah, B.F. Van Dongen, N. Zannone, Controlling break-the-glass through alignment, in: *ICSC*, IEEE, 2013, pp. 606–611.
- [23] S. Banescu, M. Petković, N. Zannone, Measuring privacy compliance using fitness metrics, in: *BPM*, Springer, 2012, pp. 114–119.
- [24] W. Waal, I. Van de Weerd, I. Beerepoot, X. Lu, T. Kappen, S. Haitjema, H. Reijers, Putting the sword to the test: Finding workarounds with process mining, *Bus. Inf. Syst. Eng.* (2024) 1–20.
- [25] S. Weinzierl, V. Wolf, T. Pauli, D. Beverungen, M. Matzner, Detecting temporal workarounds in business processes – a deep-learning-based method for analysing event log data, *J. Bus. Ana.* 5 (1) (2022) 76–100.
- [26] F. Folino, G. Folino, M. Guarascio, L. Pontieri, Data- & compute-efficient deviance mining via active learning and fast ensembles, *J. Intell. Inf. Syst.* (2024) 1–25.
- [27] T. Nolle, A. Seeliger, N. Thoma, M. Mühlhäuser, Deepalign: Alignment-based process anomaly correction using recurrent neural networks, in: *CAiSE*, Springer, 2020, pp. 319–333.
- [28] F.M. Maggi, C. Di Francescomarino, M. Dumas, C. Ghidini, Predictive monitoring of business processes, in: *CAiSE*, Springer, 2014, pp. 457–472.
- [29] G. Park, W.M. van der Aalst, Action-oriented process mining: bridging the gap between insights and actions, *Prog. Artif. Intell.* (2022) 1–22.
- [30] G. Park, D. Schuster, W.M. van der Aalst, Pattern-based action engine: Generating process management actions using temporal patterns of process-centric problems, *Comput. Ind.* 153 (2023) 104020.
- [31] A. Berti, S.J. van Zelst, W.M.P. van der Aalst, Process mining for python (pm4py): Bridging the gap between process-and data science, in: *ICPM Demo Track*, Vol. 2019, 2019.

- [32] A.G. Asuero, A. Sayago, A. González, The correlation coefficient: An overview, *Crit. Rev. Anal. Chem.* 36 (1) (2006) 41–59.
- [33] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, F.M. Maggi, Complex symbolic sequence encodings for predictive monitoring of business processes, in: *BPM*, Springer, 2015, pp. 297–313.
- [34] L. Pufahl, R. Hobeck, P. Binetruy, W. Chada, Digtar Mykola, K.J. Gülle, M. Slarzynska, F. Stiehle, I. Weber, Performance, variant, and conformance analysis of an academic travel reimbursement process, in: *BPI Challenge*, Vol. 2020, 2020.
- [35] M. Kubat, S. Matwin, Addressing the curse of imbalanced training sets: One-sided selection, in: *ICML*, 1997, pp. 179–186.
- [36] X. Dong, Z. Yu, W. Cao, Y. Shi, Q. Ma, A survey on ensemble learning, *Front. Comput. Sci.* 14 (2020) 241–258.
- [37] J.N. Mandrekar, Receiver operating characteristic curve in diagnostic test assessment, *J. Thorac. Oncol.* 5 (2010) 1315–1316.
- [38] J. Peeperkorn, S.v. Broucke, J. De Weerd, Can recurrent neural networks learn process model structure? *J. Intell. Inf. Syst.* 61 (1) (2023) 27–51.
- [39] M. Feurer, F. Hutter, Hyperparameter optimization, automated machine learning: Methods, systems, challenges, 2019, pp. 3–33.
- [40] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive business process monitoring with lstm neural networks, in: *CAiSE*, Springer, 2017, pp. 477–492.
- [41] L. Prokhorenkova, G. Gusev, A. Vorobev, A.V. Dorogush, A. Gulin, Catboost: unbiased boosting with categorical features, *Adv. Neural Inf. Process.* 31 (2018).
- [42] P. Pfeiffer, J. Lahann, P. Fettke, Multivariate business process representation learning utilizing gramian angular fields and convolutional neural networks, in: *BPM*, Springer, 2021, pp. 327–344.
- [43] Z. Wang, T. Oates, Encoding time series as images for visual inspection and classification using tiled convolutional neural networks, in: *Workshops At the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [44] A. Stevens, J. De Smedt, J. Peeperkorn, Quantifying explainability in outcome-oriented predictive process monitoring, in: *ICPM Workshops*, Springer, 2022, pp. 194–206.
- [45] R. Galanti, B. Coma-Puig, M.d. Leoni, J. Carmona, N. Navarin, Explainable predictive process monitoring, in: *ICPM, IEEE*, 2020, pp. 1–8.
- [46] C. Harris, R. Pymar, C. Rowat, Joint shapley values: a measure of joint feature importance, 2021, arXiv preprint arXiv:2107.11357.
- [47] M. Grohs, H. van der Aa, J.-R. Rehse, Beyond log and model moves in conformance checking: Discovering process-level deviation patterns, in: *BPM*, 2024, pp. 381–399.