SN Computer Science (2026) 7:250
https://doi.org/10.1007/s42979-026-04832-w

ORIGINAL RESEARCH

A Deep Learning Framework for Predicting Business Process Violations

Ghalia Tello[1] · Gabriele Gianini[2] · Rabeb Mizouni[1] · Corrado Mio[1] · Ernesto Damiani[3] · Paolo Ceravolo[3]

Received: 11 July 2025 / Accepted: 14 February 2026

Abstract

Extracting knowledge from business process logs to prevent violations of business rules can protect companies from major losses. Most of the existing approaches toward this goal focus on compliance verification with respect to a target business model and are purely reactive: they detect violations *ex post*. The few existing approaches that try to prevent violations beforehand require substantial manual intervention, don't consider fine-grained logs, ordinarily found in real-world business scenarios, and are based on memoryless techniques. To fill these gaps, we propose an integrated end-to-end framework to predict business model violations from a stream of low-level event logs. We use a Bidirectional Long-Short-Term Memory (BiLSTM) model, integrated with an attention mechanism to capture discriminating features and enable training on long sequences. This framework, whose setup requires minimal human intervention, can forecast not only the type but also the relative location of the upcoming violations in the event sequence. This information is useful in determining the type of countermeasures that need to be taken. We demonstrate the applicability of the framework using a real-life event log and achieve a prediction accuracy of 99.74%.

Keywords Process mining · Predictive models · Attention mechanism · Log-lifting · Business process rule violations · BiLSTM networks

Introduction

Predictive Process Monitoring (PPM) is an area of increasing interest within the field of process mining [1]. PPM techniques aim to forecast the future characteristics of ongoing process executions by exploiting events recorded in information systems to anticipate process behavior. Typically, these techniques map the prediction problem into sequence classification by first training a classifier using prefixes of historical traces labeled with the prediction target, then using the trained model in an online inference phase to predict and assess upcoming instances.

Various predictive approaches address different aspects of future process states, including the next activity to be executed [2–4], the remaining time to completion [5–7], the compliance with rules [8, 9], and indicator values such as the remaining cost [10–12]. Although the remaining time prediction has received substantial attention, compliance prediction remains understudied [13].

However, existing PPM approaches suffer from three critical limitations that hinder their applicability in real-world scenarios:

✉ Gabriele Gianini
gabriele.gianini@unimib.it

Ghalia Tello
ghaliatello@icloud.com

Rabeb Mizouni
rabeb.mizouni@ku.ac.ae

Corrado Mio
corrado.mio@ku.ac.ae

Ernesto Damiani
ernesto.damiani@unimi.it

Paolo Ceravolo
paolo.ceravolo@unim.it

[1] Khalifa University of Science and Technology, Abu Dhabi Campus, Hadbat Al Zaafran, Abu Dhabi 127788, UAE

[2] Department of Informatics, Systems and Communication, University of Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy

[3] Department of Computer Science, University of Milan, Via Celoria 18, 20133 Milano, Italy

Published online: 04 March 2026

(L1) Granularity mismatch between models and execution logs. Current approaches assume a one-to-one mapping between process-model activities and recorded events, failing to address the fine-grained, low-level events encountered in practice. For example, while a process model might record a high-level activity such as `Approve Invoice`, real-world systems generate numerous granular events (e.g., {`Open File`, `Validate Data`, `Request Supervisor Authorisation`}) that traditional process mining techniques do not explicitly capture [14].

(L2) Limited temporal context modeling. Most existing methods rely on memoryless learners such as Decision Trees, Random Forests, or simple LSTM architectures, which disregard information about process dynamics accumulated from the recent past. For instance, a Random Forest may classify an event based on static features (e.g. timestamp or user role), but fails to consider temporal dependencies, such as whether the previous three steps followed the expected sequence—critical information for anomaly detection. Moreover, these approaches require considerable manual effort for feature extraction and labeling. Practitioners must manually design and test numerous feature combinations, such as temporal aggregations (e.g., event frequency over sliding windows) or encoded event sequences (e.g., one-hot versus embeddings), through time-consuming trial and error that risks omitting critical patterns [15, 16].

(L3) Insufficient violation prediction granularity. Current compliance prediction approaches use binary classifiers that indicate whether a process will violate or comply with the model rules based on the current prefix. However, effective countermeasures require more detailed information, specifically the *type* and *timing* of upcoming violations (i.e., their location in the event sequence). For example, a "skipping activity" violation could indicate resource allocation problems, as excessive load on system resources often results in skipped activities. This prediction capability is increasingly important given the constrained resource allocation in modern business process management systems [17–20]. Furthermore, knowing the temporal proximity of violations (whether imminent or distant in the process) enables more timely and targeted interventions.

As to limitation L1, on the one hand prediction models cannot operate unless the low-level stream is first segmented and lifted to the activity level, on the other hand the usefulness of segmentation and log lifting is best assessed through their impact on downstream predictive tasks, rather than via intrinsic accuracy alone. For this reason, we address the problem as a single end-to-end pipeline, in which each stage enables and supports the next.

To address these limitations, we propose an end-to-end framework for predicting the location and type of upcoming violations in streams of low-level event logs, thereby supporting better organizational decision-making. The core prediction component is a Bidirectional Long-Short-Term Memory (BiLSTM) artificial neural network integrated with an attention mechanism. This architecture enables automatic learning of long-term dependencies and activity patterns in process executions. The BiLSTM processes sequences both forward and backward, combining outputs to leverage full sequence context, while the attention mechanism automatically captures features with decisive effects on violation prediction. This approach achieves superior prediction accuracy, scalability, and generalization capability across diverse application scenarios.

The remainder of the paper is structured as follows. "Related Work" section summarizes related work. "Background and Definitions" section provides preliminary definitions and formalizes the problem statement. "The Proposed Framework" section presents the proposed framework. "Case Study" section describes the framework implementation and demonstrates its applicability using a real-life event log case study. "Framework Limitations" section discusses the limitations of this work. "Conclusion" section concludes the paper with final remarks and future research directions.

Related Work

Process mining enables organizations to extract insights from event data recorded in information systems, helping them to analyze and optimize business processes [21]. However, raw data often require significant preprocessing before they can be mined effectively. This includes cleaning the data, adjusting the granularity, and abstracting the events, since event logs from different systems (e.g., CRM and ERP) often vary in detail and structure. A systematic review of the literature has identified 15 key pre-analysis activities, including data extraction, generation, enrichment, cleaning, formatting, and abstraction [22].

A significant challenge arises with user interaction data, which captures detailed actions (e.g., clicks and keystrokes) but lacks explicit links to activities at the process level or to specific process executions. Transforming these data into a format compatible with process mining requires tasks to be segmented, categorized, and linked via object instances [23]. This is a problem that existing methods often attempt to solve using ad hoc approaches that are difficult to generalize. An unsupervised approach that combines control flow, data, and semantic information has shown promise in converting raw interaction logs into structured event data [24]. Additionally, events can be modeled as networks or graphs in which the causal relationships between different types of event (e.g., sequences or compound interactions) reveal systemic patterns. Advanced embedding techniques

encode these events as low-dimensional vectors, enabling the supervised or unsupervised detection of meaningful processes, an approach that has been validated using real-world and synthetic datasets [25].

Event abstraction techniques are crucial when dealing with multigranular event logs, since process mining assumes uniform data granularity. A taxonomy of existing approaches reveals gaps and emerging trends, highlighting the need for automated and adaptive preprocessing methods to improve the applicability of process mining in real-world scenarios [26]. Although research on pre-analysis is evolving, developments in unsupervised learning methods for granularity adjustment and event abstraction are fundamental to improving the accuracy and applicability of process mining [27]. Our paper addresses these challenges by proposing a method for automatically detecting the granularity of events using a small set of ground truth examples ("Log Segmentation" section), which is then exploited by our violation prediction model.

Several research works have tackled the challenge of monitoring event logs against different model constraints [28–32]. However, these works predominantly adopt a *reactive* approach—detecting violations *ex-post*, i.e., after their occurrence—rather than predicting them *ex-ante* in order to prevent them. Although conformance checking provides valuable information on past deviations, it lacks the ability to anticipate potential violations before they materialize. The problem of predicting such violations has received comparatively limited attention until recent years.

Early work on predictive monitoring has mainly employed traditional machine learning methods, such as decision trees and random forests. For example, Maggi et al. proposed a framework that uses decision tree learning to predict violations of business rules formalized in linear temporal logic (LTL) [8]. Based on this, Di Francescomarino et al. [33] introduced a hyperparameter optimization procedure and used clustering techniques to group traces based on control flow features. Classification models (Decision Tree or Random Forest) are trained per cluster, and during execution, an ongoing trace is matched to a cluster for prediction. Conforti et al. [34] also adopt a decision-tree-based approach to support the reduction of process violations, while Suriadi et al. [35] integrate additional contextual information, such as delays and resource workloads, to perform root cause analysis and predict overtime faults.

Although these contributions laid the groundwork for predictive monitoring, significant advances in process mining and machine learning have transformed the landscape. Deep learning models, particularly those based on long-short-term memory (LSTM) and other recurrent architectures, have become commonplace due to their ability to identify long-term dependencies in event sequences and process detailed high-dimensional event logs [36]. These techniques have demonstrated high predictive accuracy in various fields, including healthcare, logistics, and manufacturing. Some applications have reported accuracy rates of up to 99.8% [37]. For this reason, we decided to rely on a BiLSTM architecture in our work ("Predictive Model Learning" section).

Despite progress in predictive process monitoring, several critical limitations persist. Many models emphasize predictive accuracy, often at the cost of interpretability, which constrains their practical value for decision-making [38]. While most approaches focus on predicting *whether* violations will occur, relatively few address the *location* of violations within process executions—an aspect that has been recognized as crucial for effective intervention [39]. Furthermore, although several systems are capable of detecting deviations from expected behavior, they typically provide limited information about violation types or their underlying causes. Recent research increasingly emphasizes the importance of explainability in predictive process monitoring, as interpretable predictions are essential for building trust and enabling actionable insights [40, 41]. Additionally, studies underscore the necessity of memory-aware architectures and hybrid approaches that integrate conformance checking with predictive analytics [42]; however, fully integrated solutions that combine these capabilities remain scarce.

Unlike other methods that rely heavily on manual event abstraction stages and struggle to provide explanations, our approach automatically learns complex patterns while maintaining interpretability through attention mechanisms. By combining violation prediction (type and location) with explainability, we enable real-time decision-making in dynamic business environments. Although deep learning is well-established in PPM, our key advancement lies in unifying process awareness, fine-grained violation prediction, explainability, and proactive capabilities within a single end-to-end framework ("Predictive Model Results" section).

Background and Definitions

In this section we summarize the background needed to understand the components of our framework. We first introduce standard notions from process mining (event logs, traces, events, and activities). We then provide a brief overview of LSTM and BiLSTM networks, whose internal mechanisms (e.g., memory cell operations and gating functions) are used later in "Predictive Model Learning" section.

Process Mining Concepts

Definition 1 *(Logs, Traces, Events)*. In process mining, the execution of processes is reflected by event *logs*. Typically, an event log L contains multiple *traces*: $L = (trace_1, trace_2, trace_3, ...trace_{|L|})$.Eachtracecontains a sequence of low-level *events*: $trace = (e_1, e_2, ..., e_{|trace|})$

Definition 2 *(Low-level event)*. The eXtensible Event Stream (XES) [23] defines a standard for recording information system's events. An event in the log is typically defined by the tuple $e = (n, c, r, s, t)$, consisting of an event *n*, in a case *c*, using the resource *r*, having a status $s \in \{start, complete\}$, at time-stamp *t*. Additional attributes for events can be included, such as location, originator, price, etc.

Definition 3 *(Prefix and suffix)*. The prefix of a trace consists of the events of the sequence up to some special point. Most often, it represents the sequence of events executed so far, or those known. The corresponding *suffix* represents the remainder of the sequence. Most often, it represents the sequence of events not yet executed, those that have to be predicted, or whose global features have to be predicted. A prefix of length *l* can be written as: $\text{prefix} = (e_1, e_2, ..., e_l)$, where $l < |trace|$, and the corresponding suffix as $\text{suffix} = (e_{l+1}, \ldots, e_{|trace|})$.

Definition 4 *(High-level activity)*. A high-level activity *A* is an abstract representation of a task as it appears in the process model. It usually has a human understandable meaning.

In real business scenarios, event logs record low-level events that do not directly map to the high-level activities seen in the process model. In fact, events in the log are finer-grained than the activities of process models and their types are related to activities through an *n* to *m* relationship [43]: a high-level activity typically consists of multiple types of low-level events; whereas, one type of low-level event might appear into multiple high-level activities (different instances of the same type). For example, a low-level event `send an email` can be part of two high-level activities: `notify the request's outcome` and `file a complaint`. At the same time, high-level activity `notifies the outcome of the request` might contain multiple low-level events, such as `send an email` and `make a phone call`.

LSTM, BiLSTM, and Attention

Recurrent Neural Networks (RNN) are widely used for modeling sequential data, as they process input sequences one element at a time while maintaining a hidden state that summarizes the past. Standard RNNs, however, suffer from vanishing or exploding gradients, which limits their ability to capture long-term dependencies.

Long Short-Term Memory (LSTM)

LSTM networks [44] introduce a memory cell regulated by input, forget, and output gates, enabling the model to preserve information over long time horizons.

Bidirectional LSTM (BiLSTM)

In many applications, the interpretation of an element in the sequence depends not only on its past but also on its future context. BiLSTM layers address this by combining a forward LSTM that processes the sequence from $1 \rightarrow T$ with a backward LSTM that processes it from $T \rightarrow 1$. The hidden representation at position *t* is: $h_t = [\overrightarrow{h_t} \oplus \overleftarrow{h_t}]$,, which results in richer contextual encoding.

Attention Mechanisms

Attention mechanisms [45] allow the network to focus on the most informative parts of a sequence. Given the BiLSTM hidden states $\{h_t\}$, attention computes:

$$u_t = \tanh(W_h h_t + b_h), \qquad \alpha_t = \text{softmax}(w_a^\top u_t),$$

where α_t are normalised importance weights. The sequence representation is then produced as a weighted sum:

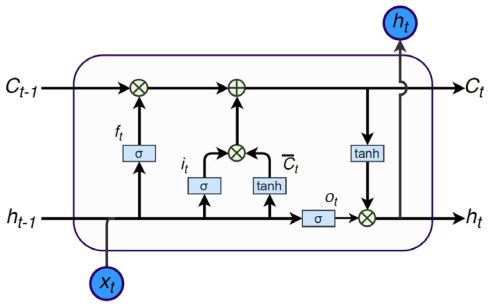
$$c = \sum_t \alpha_t h_t.$$

Together, BiLSTM and attention provide strong modelling capacity for complex temporal patterns, making them well-suited for predictive process monitoring. The operation of the overall cell is schematized in Fig. 1.

The Proposed Framework

Figure 2 provides an overview of the proposed framework. The process consists of three main phases: *log-segmentation*, *log-lifting*, and *predictive model learning*, followed by an inferential phase, in which the trained model is used for *online prediction prediction*.

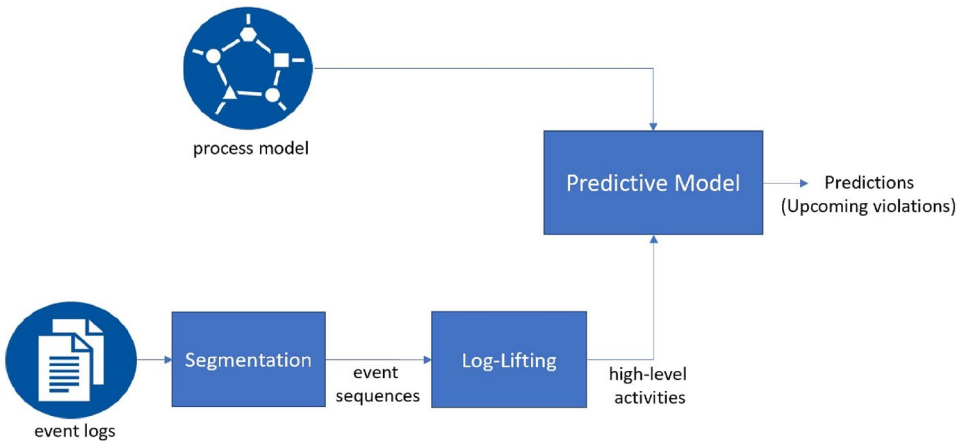
In summary, (A) a segmentation algorithm takes in input the event stream and locates the segment separators in a real-time flow of low-level events, where each segment



$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
C_t &= f_t * C_{t-1} + i_t * \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Fig. 1 Schema of an LSTM memory cell (left) and corresponding recurrence relations (right). In the equations, $\sigma(\cdot)$ denotes the logistic sigmoid activation function, and $\tanh(\cdot)$ is the hyperbolic tangent. The symbol " " indicates matrix–vector multiplication after concatenating the previous hidden state h_{t-1} and the current input x_t. The symbol "$*$ " denotes the element-wise (Hadamard) product between vectors of the same dimension. The input (i) and output (o) gates are used to control the extent to which the information flows into the memory cell and the information used to compute the output, respectively. The forget gate (f) can modify the self-recurrent connection to allow the memory cell to remember or forget its previous state

Fig. 2 The proposed framework



corresponds to an unknown high-level activity that is yet to be discovered; (B) a supervised or semi-supervised Machine Learning algorithm performs the log-lifting, i.e. classifies/maps the event segments into their corresponding high-level activities (C) then a deep-learning model is trained to predict violations in the process execution: the output of the predictive model is the relative location and type of the upcoming violations; (D) eventually, the learned model is used on previously unseen data for prediction. Hereafter we describe in detail the first three phases.

Although each component of the pipeline—event segmentation, log lifting, and violation prediction—could in principle be studied in isolation, their practical value emerges only when combined. The segmentation and lifting phases are designed not only to produce high-level logs but to ensure that the predictive model receives input that retains the temporal and semantic structure necessary for accurate violation forecasting. Thus, the overall contribution lies in the integrated pipeline rather than in the individual components.

Log Segmentation

In the beginning, the flow of low-level events undergoes a segmentation process [46]. Each output segment represents an event sequence that contains a number of low-level events and represents an estimated high-level activity.

Rationale of the Approach

The segmentation algorithm is based on a maximum likelihood approach. This segmentation task is similar to a text decoding problem, in which one has to guess where to add a *space symbol* to break a sequence of *non-space symbols* so as to obtain a sequence of words separated from one another. In the case of process logs, the break could correspond to a pause within a working session, marking the end of an activity (this case could also be identified based on pauses between time-stamped events).

The most representative sequences can be captured, using a technique described in [47], from the *primary path* of each high-level activity automaton: a prime path is a simple path that is not a subpath of any other simple path.

Since organizations' activities are often repetitive, the segmented list is typically relatively small. The likelihood of the different candidate segmentations is computed based on a *segmented list*: this is a relatively small set of ground truth examples annotated by the data owner with the correct separation between one activity and the next. In our case study, this set consists of 58 manually validated prime-path examples for a process with 29 high-level activities and approximately 250,000 total segments, i.e., less than 0.03% of the entire log. Empirically, we observe that 1–3 prime paths per activity are typically sufficient, amounting to well below 0.1% of the data.

The segmented list allows for running a standard *n*-gram based maximum likelihood decoding. The process can be illustrated using digrams. Counting digrams one creates a prefix-suffix table with one letter prefix and one letter suffix. This provides an estimate of the conditional probabilities of a suffix given a prefix. Thanks to this, one can compute the likelihood of any candidate decoding sequence as a product of the probabilities of each of the involved digrams (or equivalently, as the sum of their logs). Standard optimization techniques allow us to search for the optimal decoding, which maximizes the likelihood. Satisfactory decoding can also be accepted to limit the run time of the procedure. This approach can be used with any *n*-gram model of a language and has been used in biology for DNA segmentation (where the alphabet consists of the four bases) and for protein sequencing (where the alphabet consists of the 20 amino acids) and in computational linguistics (the base units here are in some cases letters, in other cases words) [48].

For setting up an *n*-gram model one has to decide which length *n* to use: using a too short *n*-gram runs the risk of missing some important correlation information, whereas using a too long *n*-gram produces sparse frequency matrices, which estimate the likelihoods poorly. In the absence of prior knowledge, a reasonable empirical choice consists of using the median segment length in the ground truth *segmented list* (also the mode or the value closest to the mean can be adopted).

$\mathcal{C}$	-	A	B	C	D	X	Y	Z
-	0	1	2	0	0	2	1	0
A	1	0	0	0	0	0	0	0
B	0	0	0	2	0	0	0	0
C	1	0	0	0	1	0	0	0
D	1	0	0	0	0	0	0	0
X	0	0	0	0	0	0	2	0
Y	1	0	0	0	0	0	0	2
Z	2	0	0	0	0	0	0	0

Fig. 3 Digram count matrix $\mathcal{C}$ of the example discussed in the text

The output of this segmentation phase is a sequence of words representing high-level activities, which, however, are still unlabeled.

Method Description and an Illustrative Example

We use a simple example to demonstrate the steps of the segmentation algorithm. Assume the ground truth *segmented list* is the following: {{A}, {BC}, {BCD}, {XY}, {XYZ}, {YZ}}, and the sequence of events to be segmented is: {ABCDAXYZ}. The segmentation steps are as follows: 1) construct the digram count matrix *C*; 2) estimate the transition probability matrix *M*; 3) compute the likelihood of the sequence; 4) run a moving window scan over the sequence and insert candidate separators within the window, accepting those that bring a likelihood increase; 5) repeat the previous step until the likelihood convergence.

1. (*Digram-count matrix*) The algorithm loops over the digrams in the segmented list, recording the number of times each event digram occurs in it. Each value in the digram matrix represents the absolute frequency of occurrence of the corresponding event digram.

   - The digram matrix for this illustrative example is shown in Fig. 3.

2. (*Empirical probability matrix*) Calculate the relative frequency of each digram by dividing its observed count by the sum of its row counts. This can be taken as an estimate of the conditional probability $Pr(e_{i+1}|e_i)$ of a symbol following a given one. For convenience in notation, we denote this quantity by $\mathcal{M}(e_i, e_{i+1})$.

   - (*Padding*) A very small value should be added to all the 0-valued cells, corresponding to the unobserved combinations (that a priori are considered possible, but that might have been not observed just due to low statistics), except to the transition from separator to separator (which is 0 by construction). To this end we scaled each observed count by a factor 100 and replace zeros with a small pseudo-count equal to 1, obtaining a scaled and padded matrix $\mathcal{S}_{ij} = \max(100 \cdot \mathcal{C}_{ij}, 1)$ then obtained $\mathcal{M}$ by row-wise normalization: $\mathcal{M}_{ij} = \mathcal{S}_{ij} / \sum_k \mathcal{S}_{ik}$. The resulting empirical transition probability matrix $\mathcal{M}$ for our example is shown in Fig. 4.

3. (*Starting likelihood*) For a given sequence of events, compute its log-likelihood: For a flow with *n* events $\{e_1, e_2, ..., e_n\}$ and a digram matrix *M*, the log-likelihood $\mathcal{L}$ of the flow is:

Fig. 4 Empirical frequency matrix $\mathcal{M}$ obtained from the matrix $\mathcal{C}$ of Fig. 3 (padded with small values in place of the zeroes), acting as estimate of transition probability from one letter to the next. The symbol "-" denotes the separator

$\mathcal{M}$	-	A	B	C	D	X	Y	Z
-	0.0000	0.1658	0.3317	0.0017	0.0017	0.3317	0.1658	0.0017
A	0.9346	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093
B	0.0048	0.0048	0.0048	0.9662	0.0048	0.0048	0.0048	0.0048
C	0.4854	0.0049	0.0049	0.0049	0.4854	0.0049	0.0049	0.0049
D	0.9346	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093	0.0093
X	0.0048	0.0048	0.0048	0.0048	0.0048	0.0048	0.9662	0.0048
Y	0.3268	0.0033	0.0033	0.0033	0.0033	0.0033	0.0033	0.6536
Z	0.9662	0.0048	0.0048	0.0048	0.0048	0.0048	0.0048	0.0048

Transition	$\mathcal{M}(e_i, e_{i+1})$	$\log_2 \mathcal{M}(e_i, e_{i+1})$
$- \rightarrow A$	0.1658	-2.5922
$A \rightarrow B$	0.0093	-6.7486
$B \rightarrow C$	0.9662	-0.0496
$C \rightarrow D$	0.4854	-1.0428
$D \rightarrow A$	0.0093	-6.7486
$A \rightarrow X$	0.0093	-6.7486
$X \rightarrow Y$	0.9662	-0.0496
$Y \rightarrow Z$	0.6536	-0.6135
$log_2 \mathcal{L}$		-24.5933

Fig. 5 Computation of the log-likelihood for the sequence "$-ABCDAXYZ$"

$$\log \mathcal{L} = \sum \log \mathcal{M}(e_i, e_{i+1}) \tag{1}$$

- The computation for our example is illustrated in Fig. 5, the result is

$$\log_2 \mathcal{L}(-ABCDAXYZ) = \sum_t \log_2 M(e_t, e_{t+1}) \approx -24.6 \text{ bits.}$$

4. (*Separators insertion*) Scan the flow with a small-sized moving window and for each position of the window insert a separator in a candidate location within the window, chosen at random. Then, assess whether this insertion determines an improvement in segmentation by recalculating the flow log-likelihood $\mathcal{L}$. If the improvement is confirmed, fix the separator at that candidate position of the flow.

   - For our example and a window of length $\Delta = 4$ covering the subsequence *BCDA*, there are 3 candidate separator positions: after B, after C and after D. In this setting we can replace random separator positioning by exhaustive examination; the value of the overall log-likelihoods in the three options are -41.3361, -33.8374, and -20.5690; thus we choose to set the separator after D, because it is the highest, and because it improves over the initial log-likelihood value.

After this operation one has to look for the placement of another separator. If an improvement is not observed, one can shift the window and redo this step. The step has to be repeated until the end of the flow is reached.

5. (*Iteration until convergence*) After the first pass repeat the procedure, scanning the flow, but instead of inserting new separators, randomly change the location of existing separators until (complete or approximate) convergence of the log-likelihood. The output of the segmentation is the final location of the separators resulting.

- The outoput of our example is: {A - B C D - A - X Y Z }.

Log-Lifting

Each segment issued by segmentation corresponds to an activity: which is not known at this point of the process. The purpose of the log-lifting phase is to map each segment to an activity among those possible in the process.

For this purpose, we propose the two-phase approach illustrated in Fig. 6, consisting of a clustering-based labeling, which provides the data for training a classifier, then used for log-lifting. This log-lifting component builds upon our previous work presented in [46], where the method was introduced and evaluated as a standalone technique. In the present article, we reuse and adapt that approach as a module of our end-to-end framework, and we assess its effectiveness through its impact on the downstream violation-prediction task.

Manual labeling is time-consuming and sometimes even unfeasible. However, by using clustering, one can create groups of segments, then choose a few representatives from each cluster and manually label those representatives, an operation which requires only a minimal effort from the side of a human domain expert. After these steps, all samples can be automatically labeled using the same label as their corresponding cluster. It is important to note that the ground-truth activity labels used in our case study are available only for experimental evaluation. In real-world settings, such labels are not provided, which is precisely why a clustering-based labeling phase is needed to support log lifting.

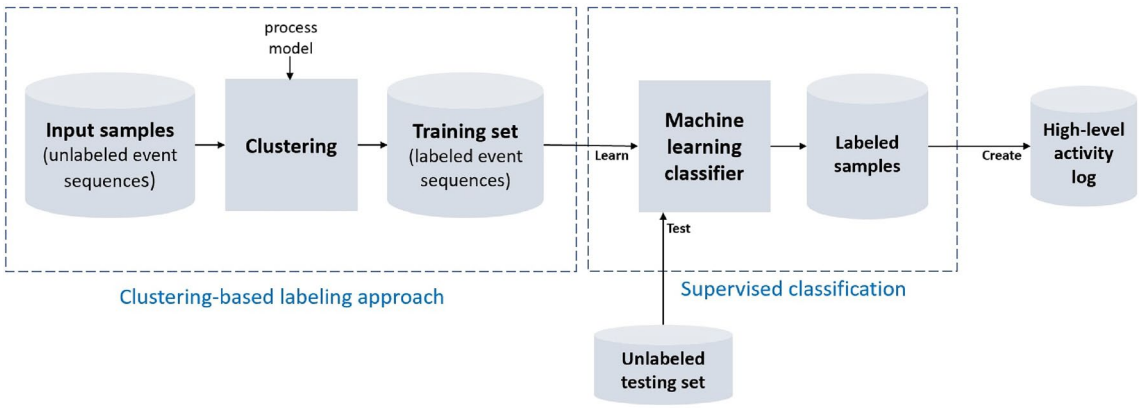


Fig. 6 Log-lifting model

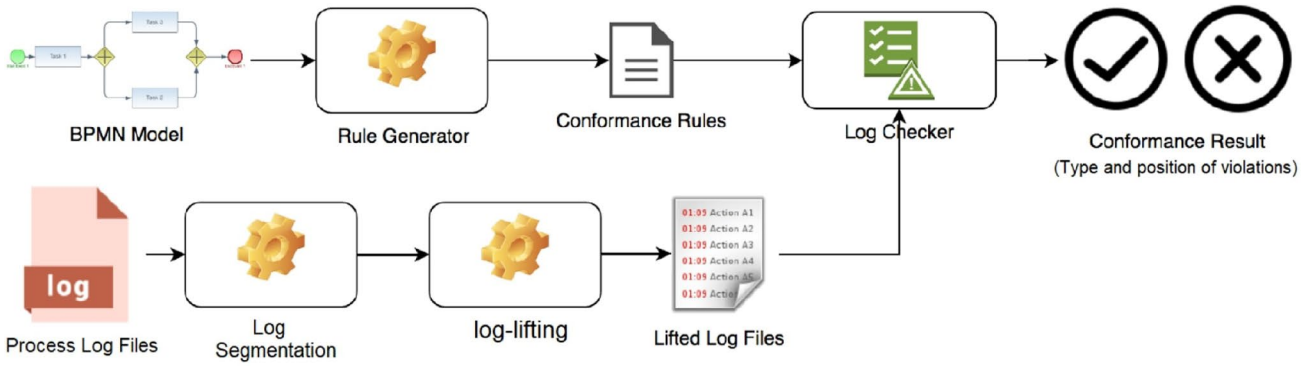


Fig. 7 Conformance checker

Subsequently, the labeled examples are used to train a machine learning classifier. The classifier learns the mapping between low-level event sequences and high-level activities. Later, it can be used to map a new set of unlabeled low-level sequences of events to the corresponding high-level activities.

The output of this two-phase log-lifting procedure is a high-level activity log, which can be used to analyze the business process under several perspectives.

Predictive Model Learning

The high-level activity sequence, generated in the log-lifting phase, provides the training set for the predictive model that aims to predict future violations.

Label Generation

As mentioned above, our predictive model employs a BiLSTM with an attention mechanism, configured as a multiclass learner. Each output class represents the relative position of upcoming violations with respect to the current prefix.

Training such a model requires a substantial number of labeled examples with and without violations, where the positions of the violations are precisely identified. Figure 7 shows an overview of the conformance checking technique integrated with our log segmentation and log-lifting approaches. Our approach employs a *Declare-based conformance checking* technique [28, 49], which enables the precise identification of violation points within process executions. First, the conformance rules are extracted from the underlying business process model expressed in the Business Process Modeling and Notation (BPMN) language and formalized as Declare constraints. These declarative rules specify requirements, prohibitions, and temporal dependencies that must be satisfied during process execution [50]. The generated declare constraints are then used to check the compliance of the lifted high-level activity sequences against the specified rules. The conformance checker's output provides the exact types and positions, i.e., indices, of violations in each trace. This enables fine-grained labeling of training data. The Declare-based approach has several advantages. It supports multiple constraint types, including existence, choice, response, precedence, and temporal constraints. It allows for flexible rule specification without requiring complete process models. It can pinpoint the

precise event at which each violation occurs, which is a critical capability for training location-aware predictive models.

We analyze the traces using a fixed-size prefix window. To encode the ground-truth labels, we use the offset index of the next violation relative to a given prefix, normalized by the prefix window size. Specifically, for a given prefix window size, the label indicates how many windows after the prefix window the violation occurs. There are several cases: traces with no violations in the remainder of the sequence (label 0); violations in the first window after the prefix window (label 1); violations in the second window after the prefix window (label 2); and so on. When multiple violations occur in a trace, we classify based on the window containing the first violation.

Regarding the possible types of violation, the classes we identified are the following:

S *Skipped*: an activity was performed even though a precondition was not satisfied (i.e., the precondition was skipped).
P *Prohibited*: an activity was carried out even though a prohibiting condition had occurred.
T *Temporal violation*: an activity violated its predefined time constraints.
U *User violation*: an action was not performed by the authorized user.
D *Duplicate activity*: an activity intended to be executed once was performed multiple times.
K *Unknown task*: an activity is present in the execution log but not defined in the process model.

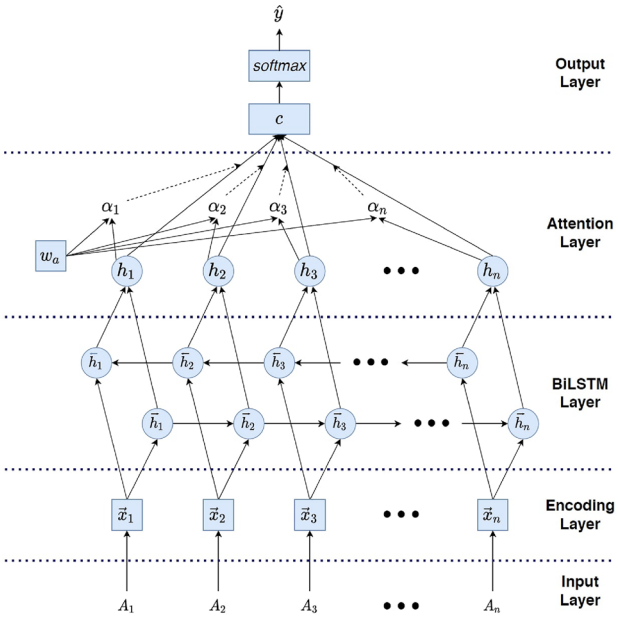


Fig. 8 Structure of the Att-BiLSTM based predictive model

Attention-Based BiLSTM Model

After log segmentation and log lifting, each trace prefix is represented as a sequence of high-level activities. These sequences form the input to our predictive model, whose goal is to estimate the relative location and the type of the next violation.

Input representation. Each activity is encoded using one-hot vectors over the alphabet of high-level activities extracted during log lifting. A fixed-size prefix window of length w is applied to each trace, and the model receives one activity per time step.

Model architecture. The predictive component, schematized in Fig. 8, consists of:

- an embedding layer that projects one-hot activity vectors into a dense representation of dimension d;
- a Bidirectional LSTM layer with h hidden units per direction;
- an attention layer that computes a weighted combination of hidden states, emphasizing the most informative parts of the prefix;
- a fully connected output layer with softmax activation, predicting either the relative position of the next violation or its type.*Training.* The model is trained using cross-entropy loss and the Adam optimiser (learning rate 10^{-3}). Ground-truth labels are automatically generated by the Declare-based conformance checker ("Predictive Model Learning" section), which identifies both the violation type and its position relative to the current prefix window.

This architecture is specifically tailored to the predictive task defined in our pipeline: it leverages the temporal dynamics of lifted traces while offering interpretability through attention weights, which highlight the segments of the prefix most influential for predicting upcoming violations.

Case Study

In this section, we validate the performance of the proposed framework using a case study. The experimental results are compared with the state-of-the-art machine learning algorithms, including different variants of LSTM.

Evaluation Metrics

In order to validate our proposed framework, several performance metrics have been used. In this paper, we focus on three main evaluation metrics: accuracy, earliness, and execution time. The *accuracy* of the model is the ratio of

correctly classified instances divided by the total number of instances. *Earliness* indicates how early the approach can predict the upcoming violation. It is defined by the ratio between the prefix length in which the classifier can accurately predict violation and the size of the longest trace in the test set [51]. Earliness ranges between 0 and 1, and low values of earliness indicate early predictions. *Execution time* is the time it takes to run the model. Measures the CPU time in seconds needed to predict a violation in an online scenario; it is obviously machine-dependent, but it is indicative of the model complexity.

Dataset Preparation

As a case study, we used a real-life log[1] from a conformance check challenge (CCC.2019) [52] and related to a medical training process. Log activities have the following features: `User`, `Time`, `Date`, `Stage`, and `Case_Id`. The log contains 29 different types of high-level activities. The dataset, which initially consisted only of high-level activities, was expanded by simulation and augmented by the addition of low-level events.

Dataset Expansion

Both the BPMN model (Fig. 9) and the Petri nets related to the use case were provided by the data owner. To obtain better testing and evaluation of the proposed framework, the data set was extended by simulating the Petri net provided using a specific plug-in of the ProM application [53] (the plug-in "*Perform a simple simulation of a (stochastic) Petri net*"). Violations were also added to the simulated part using the *"Add noise to log filter"* ProM plug-in. The size of the extended log is 250,000 samples.

Dataset Augmentation

Since the medical training log used is already expressed in terms of high-level activities, we artificially generated low-level event sequences associated with each activity, in order to test the end-to-end framework. To do this, we follow the steps of the medical low-level events that are performed in each of the underlying high-level activities following the model indicated in [54]: we associated a probabilistic automaton with each high-level activity, then we generated a low-level sequence by running the automaton with respect to its transition probabilities.

One way to form the needed segmented list is to use the concept of prime paths for each high-level activity automaton. A prime path is a simple path that is not a sub-path of any other simple path. As such, prime paths represent the most representative sequences for each high-level activity.

For example, Fig. 10 shows the probabilistic automaton for the identification of high-level activity `compression`. This automaton has two prime paths: {`choose compression option`, `scan body area`, `check probe orientation`, `compress vein`, `confirm vein patency`, `confirm identification`}, and {`choose compression option`, `scan body area`, `check probe orientation`, `compress vein`, `verify thrombus`, `report thrombus`, `confirm identification`}.

We generate the prime paths from each activity automaton to form the segmented list. The resulting segmented list in this experiment contains 58 prime paths/sequences, where each sequence contains a series of low-level events.

Seen as a whole these form an un-segmented log, to be used as input to the segmentation, clustering, labeling procedure, whose output provides the training set for the BiLSTM model.

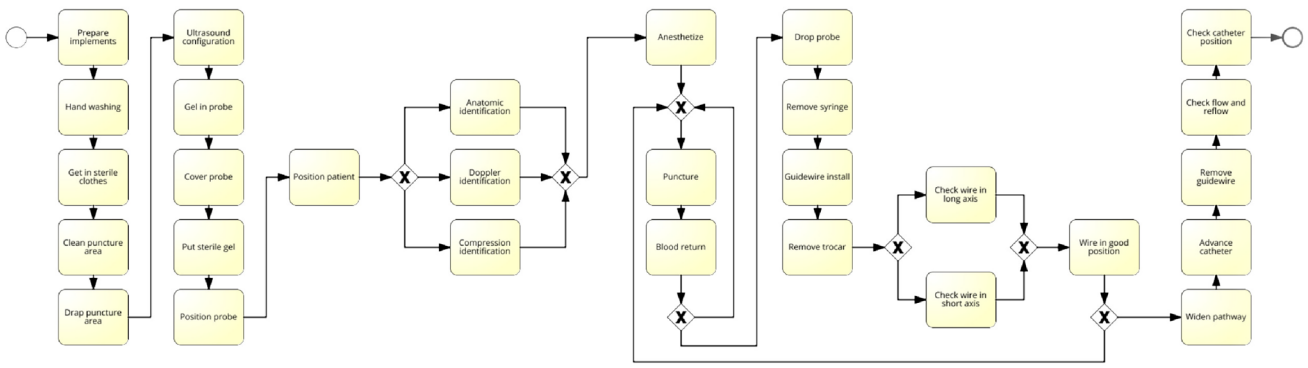


Fig. 9 CCC.2019 BPMN model

[1] Dataset DOI:10.4121/uuid:c923af09-ce93-44c3-ace0-c5508cf103ad.

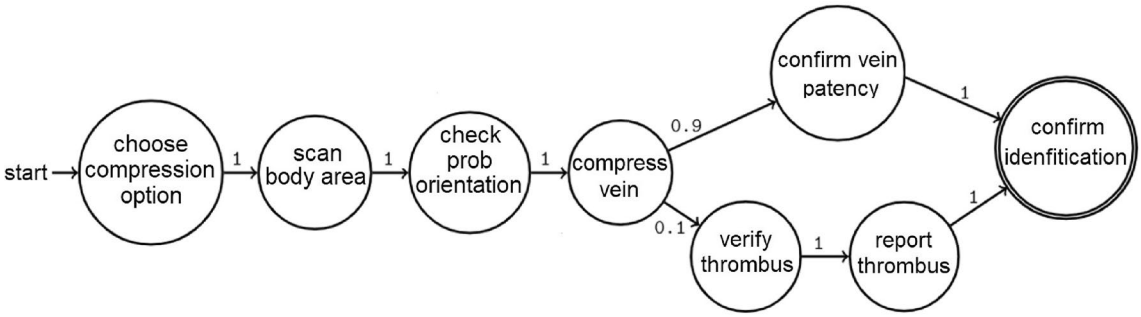


Fig. 10 Probabilistic automaton for compression identification

Log Segmentation Results

An excerpt of the flow of low-level events is the following: { {choose anatomy option} {scan body area}, {check probe orientation}, {identify anatomy}, {locate artery}, {locate vein}, {confirm identification}, {choose compression option}, {scan body area}, {check probe orientation}, {locate vein}, {compress vein}, {confirm vein patency}, {confirm identification} }.

We ran on the whole flow the segmentation algorithm described in the previous section and evaluated its efficiency. The resulting number of mis-segmented sequences turned out to be 1,622 out of 250,000 (total number of flow segments). Thus, the percent error of the estimated segments in relation to the actual amount is 0.65%, which corresponds to a segmentation accuracy of 99.35%

Log Lifting Results

After the segmentation phase, each segment contains a number of low-level events, which correspond to an unknown activity. These sequences must be classified in order to map each sequence to the corresponding high-level activity. The (practically unfeasible) complete manual labeling is replaced in our approach by a clustering sub-phase, followed by the manual labeling of a small number of *seed* segments (one per cluster) and by a Machine Learning phase where one learns a classifier able to perform the mapping. In terms of clustering and classification algorithms, we chose, respectively, k-medoids and Random Forest. K-medoids are known for their remarkable simplicity, effectiveness, and speed; Random Forest is known for its simplicity, effectiveness, and robustness with respect to label noise [55–59] and is also one of the most widely used classifiers in existing studies to predict violations.

To this end, the dataset of 250, 000 sequences/segments has been split into training and testing sets: 60% of the data, (150, 000 sequences) have been used for training, and the remaining (100, 000 sequences) for testing. In the following, we describe the results of each stage.

Clustering-Based Labeling

In this case study, we deal with discrete features with a finite set of values. Therefore, we used the clustering algorithm k-medoids, a member of the k-means clustering algorithms family, whose aim is to minimize the distance of the points within a group from the centroid of that cluster [60]. *k*-medoids doesn't require the points to be represented in an Euclidean space and it can be used to cluster data with categorical attributes. The output of k-medoids on a collection of sequences U is a labeling of the sequences by the cluster identifier (i.e., a labeled dataset L). Using k-medoids, similar low-level events were grouped into 29 different clusters (k=29), where each cluster represents a high-level activity.

To evaluate the accuracy of k-medoids, we compared the labels obtained from clustering with the ground-truth labels. The number of incorrectly clustered samples is only 1, 000 of 150, 000 samples. Hence, the resulted clustering accuracy is 99.33%.

After clustering, we reduced each cluster based on the similarity of objects to the centroids, using a variant of the silhouette method (SBBS), illustrated in [46], aimed at removing potentially ambiguous segments (the silouette method drops the elements that are as close to a cluster centroid as to another cluster centroid), thus producing a reduced data set L^*, suitable for use by a supervised algorithm. The number of the removed ambiguous sequences was 7, 485; hence, the size of the filtered data set was 142, 515 sequences. The resulting number of incorrectly clustered sequences after applying the SBBS method was 540. Thus, the accuracy was increased to 99.62%.

Machine Learning-Based Classification

After preparing the training set via the previous clustering-based labeling algorithm, a Random Forest classifier has been trained to classify event segments into the

corresponding high-level activities. To do this, we constructed a random forest, which includes *n* trees. Each tree is trained using random event sequences from the training set with replacement. Once the forest is built, each tree votes for an activity class: the final activity decision is the class with the highest number of votes.

The highest classification accuracy achieved with the random forest is 99.67% with $n = 100$ trees. It is worth mentioning that, as expected from the literature, Radom Forest was able to maintain a good classification performance even in the presence of noisy labels in the training set that might be generated by the previous clustering-based labeling phase.

Regarding execution time: the time taken to train the Random Forest model was 104.96 seconds; the time taken to run the model on test data was 53.16 seconds. This was achieved using a Java-based program running on a 64-bit operating system, a 2.5 GHz processor, Intelcore-i7, and 16GB RAM.

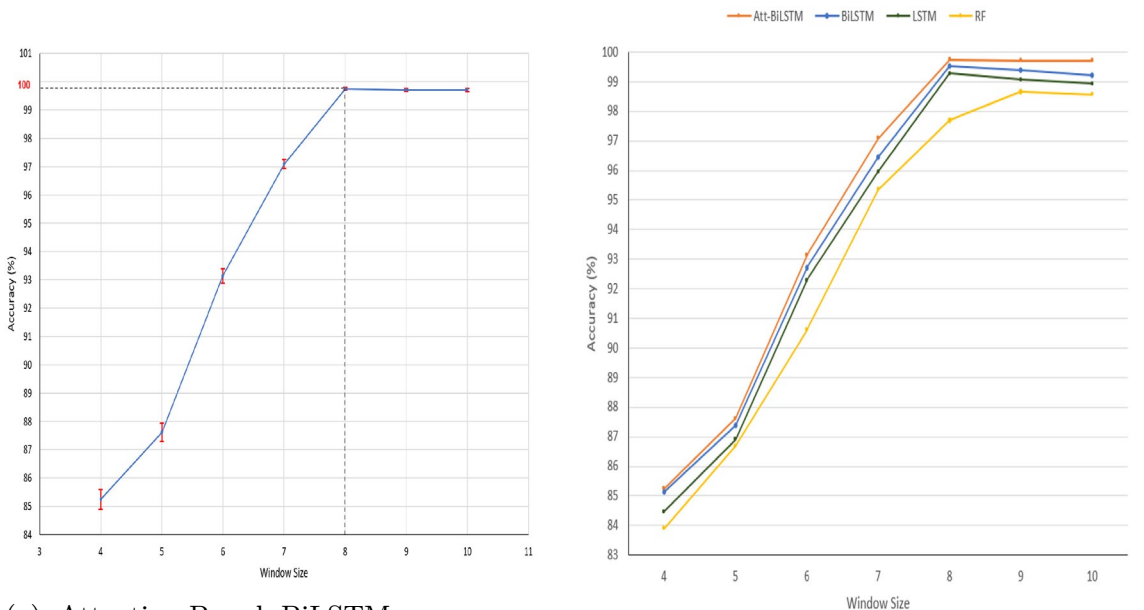
Predictive Model Results

The output of the log-lifting model provided high-level activities to be fed to the predictive model. The predictive model predicts upcoming violations (with respect to the process model) in high-level activity sequence. To build the predictive model, we split the data into a 60% (encompassing 60,000 activities) used for training and 40% (40,000 activities) for testing. A Bidirectional LSTM network with an attention mechanism has been trained using historical prefixes labeled by the conformance checker. We have set the number of training epochs at 100 and the learning rate at 0.001.

Violation Relative-Location Prediction

In this work we evaluate the predictive model using accuracy as the primary metric, and we also report per-class precision and recall through the confusion matrices. Although further metrics such as macro-averaged F1-score or balanced accuracy may be appropriate in highly imbalanced scenarios, the class frequencies generated by our prefix-window construction are relatively balanced, making accuracy, precision and recall suitable and informative measures in this context.

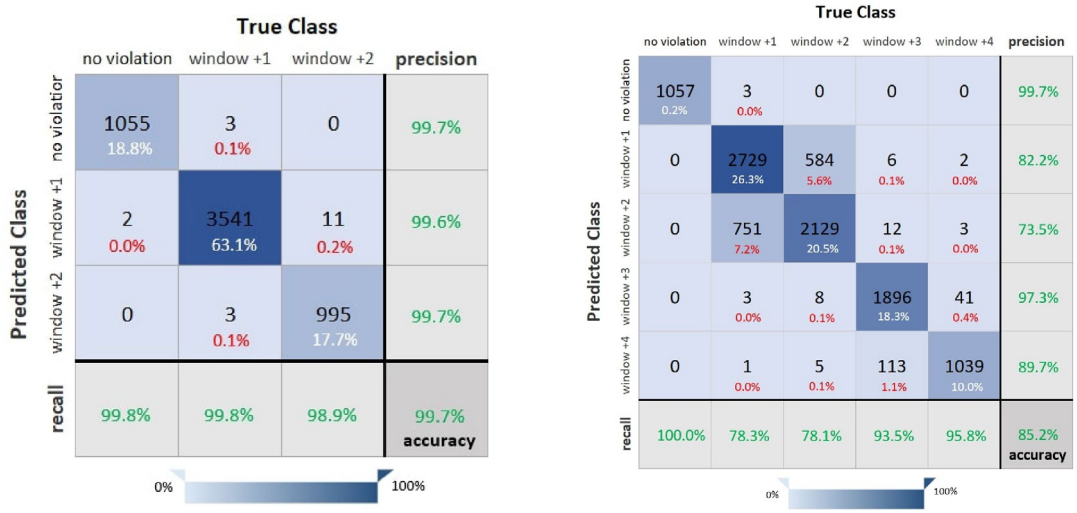
The network has been tested with different window/prefix sizes. Figure 11a shows the performance of the proposed framework in terms of accuracy against different window sizes. Initially, the prediction accuracy increases as the length of the prefix increases. After some point, increasing the prefix length does not significantly affect the performance. It is worth mentioning that an important increase in precision (from 97.1% to 99.7%) has been noticed when changing the window size from window 7 to window 8, suggesting that some vital information usually appears in this window / prefix length for this process model. The highest accuracy achieved in the testing set is 99.7% with window size 8. This accuracy is achieved with 30 hidden units in the hidden layer. The time taken to train the predictive model was 33 seconds; the time taken to test the model was 0.3 seconds.



(a) Attention-Based BiLSTM accuracy vs window size

(b) Comparison between different classifiers

Fig. 11 Model accuracy comparisons

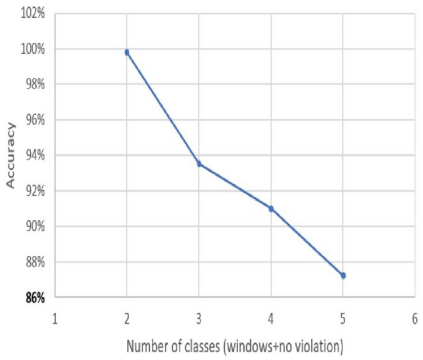


(a) Attention-Based BiLSTM confusion matrix for window size 8. Average precision= 99.6%, average recall= 99.6%

(b) Attention-Based BiLSTM confusion matrix for window size 4. Average precision= 85.2%, average recall= 85.3%

Fig. 12 Comparison of confusion matrices for different window sizes

Fig. 13 Comparison of framework accuracy with number of output classes. Left: graphical representation. Right: numerical summary



#classes	Accuracy	Cardinality
2	99.80%	27467
3	93.51%	39778
4	91.00%	30908
5	87.20%	10382

Figure 12a shows the confusion matrix for window size 8, with different output classes (with respect to the relative location of the violation). The gray column shows the percentage of correctly classified samples for each corresponding row, whereas the gray row shows the percentage of correctly classified samples for each corresponding column. The darker gray box at the bottom right shows the total percentage (with respect to the whole confusion matrix) and represents the total accuracy.

The prediction horizon (i.e., after how many windows the predicted event will appear) depends on the window size. For example, considering a shorter window size (e.g. size 4), the possible number of classes amounts to 5 possible output classes, as in Fig. 12b: this happens because, with a smaller window size, the trace can be segmented into more windows before reaching the end of the trace, and hence the number of output classes increases (there are more possible locations for the upcoming violation). However, less information is encoded with a smaller prefix size, leading to a lower model accuracy than with a larger prefix (that is, window size 8).

Figure 13 shows the performance of the framework in terms of accuracy when categorizing the examples based on the length of the prediction horizon. With a lower number of prediction classes, the accuracy is higher. For example, with only two prediction classes (that is, violation in the next window or no violation), the prediction accuracy is very high, reaching 99.8%, as indicated in the numerical summary in Fig. 13. With a higher number of classes, the accuracy decreases. However, even with a high number of alternatives (for example, classes 5, the framework can still achieve a considerable accuracy of 87.2%.

We compare the predictive performance of the BiLSTM model with the attention mechanism against Random Forest and against other LSTM-based models, namely the original LSTM and BiLSTM.

As seen in Fig. 11b, Att-BiLSTM (BiLSTM with attention) outperforms other classifiers in terms of accuracy. As mentioned above, Att-BiLSTM achieved its best accuracy of 99.74% with window 8. Random Forest has the lowest accuracy and achieved its best accuracy with window 9. Thus, Att-BiLSTM is better than the traditional RF classifier in terms of earliness.

It is worth mentioning that for short window sizes (i.e. window 4 and window 5) there is a small difference between the accuracy of LSTM-based classifiers and Random Forest, but after that LSTM-based approaches start to leverage their advantages (i.e. better in learning long-term dependencies) and outperform Random Forest significantly with longer prefix sizes.

Violation Type Prediction

Our approach can also predict the type of violation. As mentioned in "The Proposed Framework" section, the conformance checker that we utilized for ground truth labeling can also indicate the violation types. In this case study, the conformance checker has detected three different types of violations: *duplicated*, *prohibited*, and *skipped*. Consequently, we trained our attention-based BiLSTM framework with these labels. The results obtained show that our framework can predict the type of upcoming violation with an accuracy of 97.1%. Figure 14a shows the confusion matrix in which the output classes include possible types of violations.

Since some systems might see a specific type of violation as more critical than others, below we show the performance of the framework when focusing on one type of violation at a time. Figure 14b–d show the confusion matrices when predicting the first location of violation of type *duplicated*, type *prohibited*, and type *skipped*, respectively. The framework still maintained good performance with accuracy between 94.1% to 96.7% for type *skipped* and type *duplicated*, respectively.

Framework Limitations

Although our framework addresses several critical gaps in PPM, the following limitations warrant consideration:

Ground truth availability for segmentation. Our segmentation algorithm requires a "segmented list", or a small set of manually annotated examples of ground truth, to properly separate low-level events into high-level activities. While we have shown that this list can be relatively small for repetitive organizational tasks, creating it still necessitates domain knowledge and manual effort. In real-world scenarios, process mining practitioners typically do not have access to such ground truth data, necessitating an initial investment in manual annotation. However, we emphasize that this effort is substantially smaller than manually labeling entire event logs because the segmented list serves as seed data for the automatic learning process.

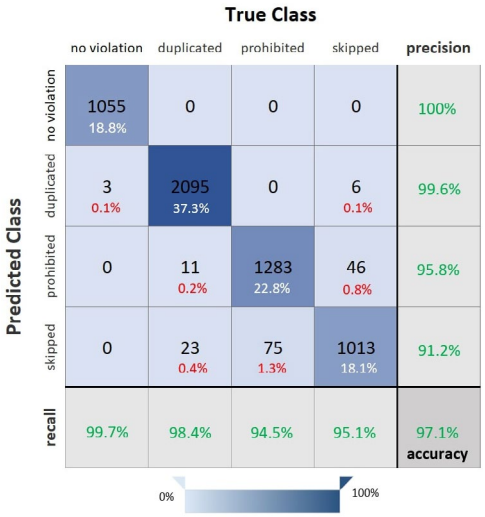
Tradeoff between prediction granularity and accuracy. Our experiments reveal an inherent trade-off between the granularity of violation location prediction and model accuracy. With binary classification (i.e., violation in the next window versus no violation), the framework achieves 99.8% accuracy. However, as the number of prediction classes increases to provide finer-grained location information, such as five classes that distinguish violations across multiple future windows, accuracy decreases to 87.2%. These results suggest that practitioners must balance the need for precise violation timing with prediction confidence and select an appropriate level of granularity based on their specific use case requirements.

Algorithmic choices in preprocessing stages. The log-lifting phase uses specific algorithms—K-Medoids for clustering and Random Forest for classification—that were chosen for their simplicity, effectiveness, and ability to withstand label noise. While these choices were effective in our case study, the performance of the overall framework may vary with different algorithmic selections. Alternative approaches, such as deep clustering methods or other ensemble classifiers, could improve performance in certain domains, but a systematic comparison would be necessary.

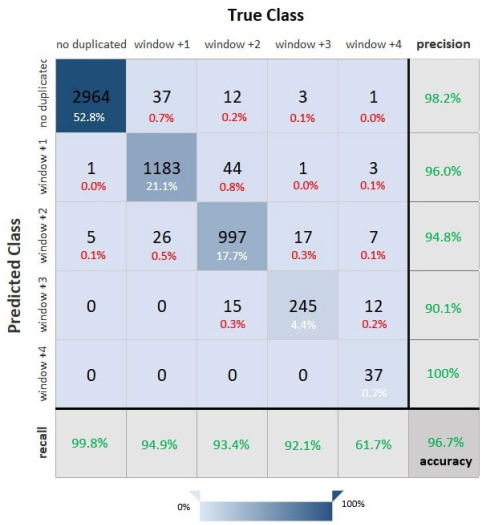
Focus on first violation only. When multiple violations occur in a single trace, our framework classifies cases based on the window containing the first violation. Although this design enables effective early warning systems, it may not capture the full complexity of traces involving cascading or multiple independent violations. A promising direction for future work is addressing this limitation by predicting all violations and their interdependencies.

Domain-specific validation. We validated our framework using a real-world event log from a medical training process. Although we applied data augmentation techniques to expand the dataset and enhance generalizability, the framework's performance in other business domains with substantially different structural characteristics remains to be empirically validated. Examples of such characteristics include highly unstructured processes, extreme variations in event frequency, and complex inter-case dependencies. Cross-domain evaluation would bolster confidence in the framework's broad applicability.

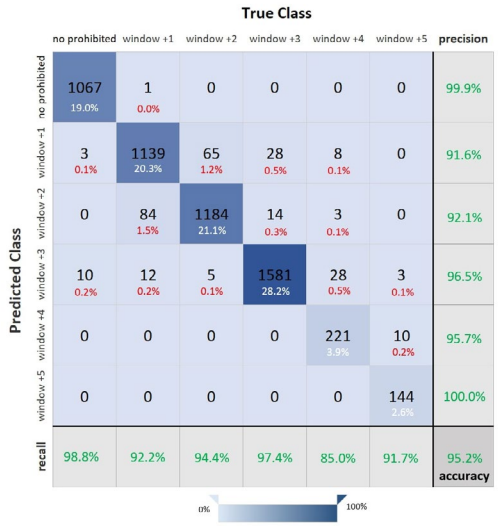
Parameter sensitivity. The accuracy of the predictive model depends on the size of the prefix window, which must be optimized for each specific process. In our case study, a window size of 8 yielded optimal results, increasing accuracy from 97.1% to 99.7%. However, determining the appropriate window size for new processes may require



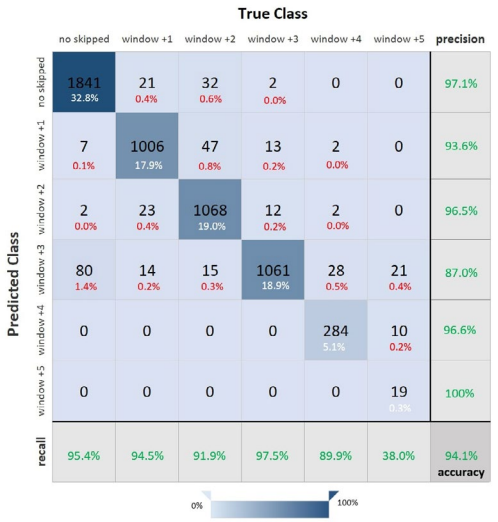
(a) Type prediction. Average precision 97.1%, average recall 97.1%



(b) Type *duplicated*. Average precision 96.7%, average recall 96.7%



(c) Type *prohibited*. Average precision 95.1%, average recall 95.1%



(d) Type *skipped*. Average precision 94.1%, average recall 94.1%

Fig. 14 Confusion matrices in prediction of location of first violation (different types)

experimentation and could benefit from automated hyperparameter optimization techniques. The optimal window size likely depends on process characteristics, such as average trace length, temporal dynamics, and violation patterns.

Finally, we note that the effectiveness of the segmentation and log-lifting phases is intrinsically linked to the quality of downstream prediction. Evaluating the full pipeline therefore provides a more realistic measure of their utility than isolated accuracy metrics. This interdependence further motivates the unified treatment of the framework as a single, end-to-end contribution.

Despite its limitations, our framework is a significant advance in predictive process monitoring. It addresses the fundamental challenges of granularity mismatch, temporal context modeling, and fine-grained violation prediction. The identified limitations provide directions for future research and practical implementation considerations.

Conclusion

This paper proposes an end-to-end framework for predicting business model violations from streams of fine-grained event logs to systematically address limitations L1–L3. The framework comprises three interconnected phases:

(1) *Event Segmentation (addressing L1):* This phase takes a stream of low-level events as input and produces meaningful sequences of low-level events. Each segment corresponds to a well-defined model activity that has not yet been discovered in the execution log. This phase automatically groups fine-grained events into coherent, activity-level segments, resolving the granularity mismatch between execution logs and process models.
(2) *Log-Lifting (addressing L1):* Building upon the segmentation output, this phase uses a log-lifting model to map low-level event sequences to their corresponding high-level model activities. This abstraction layer allows the framework to work with real-world event logs while maintaining alignment with process model semantics. Thus, it bridges the gap between fine-grained execution data and abstract process representations.
(3) *Violation Prediction (addressing L2 and L3):* This phase implements a predictive model based on a bidirectional long short-term memory (BiLSTM) with an integrated attention mechanism that predicts upcoming business model violations. The BiLSTM architecture addresses L2 by automatically capturing long-term temporal dependencies and process dynamics without manual feature engineering. The attention mechanism identifies features critical for violation detection. Furthermore, the model addresses L3 by predicting not only *whether* violations will occur but also their *type* and *relative location* in the event sequence. This enables the implementation of timely and proportionate countermeasures. For example, knowing that a resource-related violation will occur within the next three activities enables immediate intervention, whereas a violation predicted to occur later in the sequence allows for proactive planning.

Experimental results on real-world event logs demonstrate that the framework achieves an overall accuracy of 99.74% and 97.1% in predicting the relative location and type of violations, respectively, confirming its effectiveness in overcoming the limitations of existing PPM approaches.

Author Contributions All the authors contributed to Methodology and to Investigation, GT, GG, RM, ED and PC contributed to Conceptualization and Writing the original draft, GT, RM, and CM contributed to Data Curation, and to Software, RM and PC contributed to Formal analysis, GG, RM, ED, and PC contributed to Funding Acquisition and to Supervision, GG and PC reviewed and edited the manuscript.

Funding Open access funding provided by Università degli Studi di Milano - Bicocca within the CRUI-CARE Agreement. This work was partially supported by: the European Union's Horizon 2020 Research and Innovation Programme under the CISC project (Marie Skłodowska-Curie grant agreement no. 955901), and the FineTETHER project (Marie Skłodowska-Curie grant agreement no. 847402). This work was partially supported by the European Union-NextGenerationEU under the MUSA-Multilayered Urban Sustainability Action project, Mission 4 Component 2 Investment Line of the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.5 (CUP G43C22001370007, Code ECS00000037); FAIR-Future Artificial Intelligence Research-Spoke 4-PE00000013-D53C22002380006, funded by the European Union-Next Generation EU within the project NRPP M4C2, Investment 1.,3 DD. 341. We also acknowledge the support of the Program "Piano sostegno alla ricerca" PSR and the PSR-GSA-Linea 6; Project ReGAInS (code 2023-NAZ-0207/DIP-ECC-DISCO23), funded by the Italian University and Research Ministry, within the Excellence Departments program 2023–2027 (law 232/2016). We also acknowledge the REPA project founded under the PRIN 2022 - D.D. n. 104.

Data Availability Data sets generated during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest On behalf of all authors, the corresponding author declares that there is no conflict of interest.

Human and Animal Rights Not applicable.

Informed Consent Not applicable.

References

1. Ceravolo P, Comuzzi M, De Weerdt J, Di Francescomarino C, Maggi FM. Predictive process monitoring: concepts, challenges, and future research directions. Process Sci. 2024;1(1):2.
2. Tax N, Verenich I, La Rosa M, Dumas M. Predictive business process monitoring with lstm neural networks. In: International conference on advanced information systems engineering. 2017;477–92. Springer.
3. Aversano L, Iammarino M, Madau A, Pirlo G, Semeraro G. What time is it? Finding which temporal features is more useful for next activity prediction. IEEE Open J Comput Soc. 2024.

4. Pellicani A, Ceci M. Positional trace encoding for next activity prediction in event logs. Knowl-Based Syst. 2025;319:113544.
5. Polato M, Sperduti A, Burattin A, Leoni M. Data-aware remaining time prediction of business process instances. In: 2014 international joint conference on neural networks (IJCNN). 2014;816–23. IEEE.
6. Huber S, Fietta M, Hof S. Next step recommendation and prediction based on process mining in adaptive case management. In: Proceedings of the 7th international conference on subject-oriented business process management. 2015;1–9.
7. Choueiri AC, Sato DMV, Scalabrin EE, Santos EAP. An extended model for remaining time prediction in manufacturing systems using process mining. J Manuf Syst. 2020;56:188–201.
8. Maggi FM, Di Francescomarino C, Dumas M, Ghidini C. Predictive monitoring of business processes. In: International conference on advanced information systems engineering. 2014;457–72. Springer.
9. Rinderle-Ma S, Winter K, Benzin J-V. Predictive compliance monitoring in process-aware information systems: state of the art, functionalities, research directions. Inf Syst. 2023;115:102210.
10. Wynn MT, Low WZ, Hofstede AH, Nauta W. A framework for cost-aware process management: cost reporting and cost prediction. J Univ Comput Sci. 2014;20(3):406–30.
11. Folino F, Guarascio M, Pontieri L. A prediction framework for proactively monitoring aggregate process-performance indicators. In: IEEE international enterprise distributed object computing conference, 2015;128–33. IEEE.
12. Ceci M, Lanotte PF, Fumarola F, Cavallo DP, Malerba D. Completion time and next activity prediction of processes using sequential pattern mining. In: International conference on discovery science, 2014:49–61. Springer.
13. Márquez-Chamorro AE, Resinas M, Ruiz-Cortes A. Predictive monitoring of business processes: a survey. IEEE Trans Serv Comput. 2017;11(6):962–77.
14. Ceravolo P, Junior SB, Damiani E, Van Der Aalst W. Tuning machine learning to address process mining requirements. IEEE Access. 2024;12:24583–95.
15. Elkhovskaya L, Kovalchuk S. Feature engineering with process mining technique for patient state predictions. In: International conference on computational science. 2021;584–92. Springer.
16. Tavares GM, Oyamada RS, Junior SB, Ceravolo P. Trace encoding in process mining: a survey and benchmarking. Eng Appl Artif Intell. 2023;126:107028.
17. Pika A, Wynn MT. Workforce upskilling: a history-based approach for recommending unfamiliar process activities. In: International conference on advanced information systems engineering. 2020;334–49. Springer.
18. Martin N, Depaire B, Caris A, Schepers D. Retrieving the resource availability calendars of a process from an event log. Inform Syst. 2020;88:101463.
19. Ihde S, Pufahl L, Lin M-B, Goel A, Weske M. Optimized resource allocations in business process models. In: International conference on business process management. 2019;55–71. Springer.
20. Kim J, Comuzzi M, Dumas M, Maggi FM, Teinemaa I. Encoding resource experience for predictive process monitoring. Dec Support Syst. 2022;153:113669.
21. Aalst WM. Process mining: a 360 degree overview. In: Process mining handbook. 2022;3–34. Springer.
22. Pradhan SK, Jans M, Martin N. Getting the data in shape for your process mining analysis: an in-depth analysis of the pre-analysis stage. ACM Comput Surveys. 2025.
23. Ceravolo P, Damiani E, Torabi M, Barbon Jr S. Toward a new generation of log pre-processing methods for process mining. In: International conference on business process management. 2017;55–70. Springer.
24. Rebmann A, Aa H. Recognizing task-level events from user interaction data. Inf Syst. 2024;124:102404.
25. Bellandi V, Ceravolo P, Maghool S, Pindaro M, Siccardi S. Correlation and pattern detection in event networks. In: 2021 IEEE international conference on big data (big data). 2021;4103–12. IEEE.
26. Zelst SJ, Mannhardt F, Leoni M, Koschmider A. Event abstraction in process mining: literature review and taxonomy. Granular Comput. 2021;6:719–36.
27. Seiger R, Zerbato F, Burattin A, García-Bañuelos L, Weber B. Towards iot-driven process event log generation for conformance checking in smart factories. In: 2020 IEEE 24th international enterprise distributed object computing workshop (EDOCW). 2020;20–6. IEEE.
28. Maggi FM, Westergaard M, Montali M, Aalst WM. Runtime verification of ltl-based declarative process models. In: International conference on runtime verification. 2011;131–46. Springer.
29. Maggi FM, Montali M, Aalst WM. An operational decision support framework for monitoring business constraints. In: International conference on fundamental approaches to software engineering. 2012;146–62. Springer.
30. Weidlich M, Ziekow H, Mendling J, Günther O, Weske M, Desai N. Event-based monitoring of process execution violations. In: International conference on business process management. 2011;182–98. Springer.
31. Mulo E, Zdun U, Dustdar S et al. Model-aware monitoring of soas for compliance. In: Service engineering. 2011;117–36. Springer.
32. Santos EA, Francisco R, Vieira AD, FR Loures E, Busetti MA. Modeling business rules for supervisory control of process-aware information systems. In: International conference on business process management. 2011;447–58. Springer.
33. Di Francescomarino C, Dumas M, Maggi FM, Teinemaa I. Clustering-based predictive process monitoring. IEEE Trans Serv Comput. 2017;12(6):896–909.
34. Conforti R, De Leoni M, La Rosa M, Van Der Aalst WM. Supporting risk-informed decisions during business process execution. In: International conference on advanced information systems engineering. 2013;116–32. Springer.
35. Suriadi S, Ouyang C, Aalst WM, Hofstede AH. Root cause analysis with enriched process logs. In: International conference on business process management. 2012;174–86. Springer.
36. Aalikhani R, Fathian M, Rasouli MR. Comparative analysis of classification-based and regression-based predictive process monitoring models for accurate and time-efficient remaining time prediction. IEEE Access. 2024.
37. Irene T, Marlon D, La Rosa Marcello MFM. Outcome-oriented predictive process monitoring: review and benchmark. ACM Trans Knowl Disc Data. 2017;1.
38. Weinzierl S, Dunzer S, Tenschert J, Zilker S, Matzner M. Predictive business process deviation monitoring. In: ECIS. 2021;1–10.
39. Vazifehdoostirani M, Abbaspour Onari M, Grau I, Genga L, Dijkman R. Uncovering the hidden significance of activities location in predictive process monitoring. In: International conference on process mining. 2023;191–203. Springer.
40. Buliga A, Vazifehdoostirani M, Genga L, Lu X, Dijkman R, Di Francescomarino C, Ghidini C, Reijers HA. Uncovering patterns for local explanations in outcome-based predictive process monitoring. In: International conference on business process management. 2024;363–80. Springer.
41. Stevens A, De Smedt J. Explainability in process outcome prediction: guidelines to obtain interpretable and faithful models. Eur J Oper Res. 2024;317(2):317–29.
42. Pasquadibisceglie V, Appice A, Castellano G, Malerba D. Predictive process mining meets computer vision. In: Business process management forum: BPM forum 2020, Seville, Spain, September 13–18, 2020. Proceedings. 2020;18:176–92. Springer.

43. Zelst SJ, Mannhardt F, Leoni M, Koschmider A. Event abstraction in process mining: literature review and taxonomy. Granular Comput. 2020;1–18.
44. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
45. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proceedings of the 3rd international conference on learning representations (ICLR). 2015. arXiv:1409.0473.
46. Tello G, Gianini G, Mizouni R, Damiani E. Machine learning-based framework for log-lifting in business process mining applications. In: International conference on business process management. 2019;232–49. Springer.
47. Offutt J, Ammann P. Introduction to software testing. Cambridge: Cambridge University Press; 2008.
48. Doval Y, Gómez-Rodríguez C. Comparing neural-and n-gram-based language models for word segmentation. J Am Soc Inf Sci. 2019;70(2):187–97.
49. Donadello I, Riva F, Maggi FM, Shikhizada A. Declare4py: a python library for declarative process mining. BPM (PhD/Demos). 2022;3216:117–21.
50. Maggi FM. Declarative process mining with the declare component of prom. In: Proceedings of the BPM demo sessions 2013, co-located with 11th international conference on business process management (BPM2013), Beijing, China, August 26–30, 2013. 2013;1021. CEUR-WS.
51. Di Francescomarino C, Dumas M, Federici M, Ghidini C, Maggi FM, Rizzi W. Predictive business process monitoring framework with hyperparameter optimization. In: International conference on advanced information systems engineering. 2016;361–76. Springer.
52. Munoz-Gama J, Fuente R, Seplveda M, Fuentes R. Conformance checking challenge 2019. 4TU. Centre for research data. https://doi.org/10.4121/uuid:c923af09-ce93-44c3-ace0-c5508cf103ad.
53. Aalst WM, Dongen BF, Günther CW, Rozinat A, Verbeek E, Weijters T. Prom: the process mining toolkit. BPM (Demos). 2009;489(31):2.
54. Saugel B, Scheeren TW, Teboul J-L. Ultrasound-guided central venous catheter placement: a structured review and recommendations for clinical practice. Crit Care. 2017;21(1):225.
55. Altendrof J, Brende P, Lessard L. Fraud detection for online retail using random forests. Technical report. 2005.
56. Boinee P, De Angelis A, Foresti GL. Ensembling classifiers-an application to image data classification from cherenkov telescope experiment. In: IEC (Prague). 2005;394–98.
57. Ma Y, Guo L, Cukic B. A statistical framework for the prediction of fault-proneness. In: Advances in machine learning applications in software engineering. 2007;237–63. IGI Global.
58. Zhang J, Zulkernine M. Network intrusion detection using random forests. In: PST. Citeseer. 2005.
59. Folleco A, Khoshgoftaar TM, Van Hulse J, Bullard L. Software quality modeling: the impact of class noise on the random forest classifier. In: Evolutionary computation, 2008. CEC 2008. (IEEE world congress on computational intelligence). IEEE congress on. 2008;3853–59. IEEE.
60. Park H-S, Jun C-H. A simple and fast algorithm for k-medoids clustering. Exp Syst Appl. 2009;36(2):3336–41.