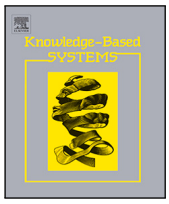
Knowledge-Based Systems 319 (2025) 113544

Contents lists available at ScienceDirect

Knowledge-Based Systems

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



Positional trace encoding for next activity prediction in event logs

Antonio Pellicani [a,b,*], Michelangelo Ceci [a,b,c]

[a] Dept. of Computer Science, University of Bari, Via Orabona, 4, 70125 Bari, Italy
[b] Data Science Lab, National Interuniversity Consortium for Informatics (CINI), Via Volturno, 58, 00185 Roma, Italy
[c] Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

ARTICLE INFO

Keywords:
Process mining
Next activity prediction
Positional encoding

ABSTRACT

The analysis of log data, generated by running processes in many application domains, enables organizations to identify opportunities for operational improvements. For instance, in healthcare, analyzing patient treatment logs can optimize care pathways; in manufacturing, production line logs can reveal bottlenecks; and in customer service, ticket resolution logs can streamline response protocols. One key analytical task is predicting the next activity in a process, which supports operational decision-making through better resource allocation and proactive response to customer needs. In this paper, we solve the next activity prediction task by exploiting a novel positional encoding approach that is based on sliding windows. This approach allows us to consider both a way to adapt to changes in the data distribution, and exploit positional information of the activities in the traces. The method proposed in this paper, called OREO, takes into account these aspects through a positional encoding tightly coupled with specific types of deep neural network architectures. The results on eight real-world process logs show the superiority of the models exploiting OREO encoding over state-of-the-art approaches, confirming our initial intuition of benefits gained by combining a time-window based model with positional information.

1. Introduction

Nowadays, more and more companies are becoming aware of the benefits of analyzing their internal process data to optimize their procedural processes. As a consequence, they continuously collect data in the form of *event logs* through information systems [1]. A log may contain a variety of *processes*, for instance: a user who is withdrawing cash from an ATM, a call center operator answering a call and solving a problem raised by a customer, or a user browsing the Internet. For each of these processes, the specific activities performed are tracked. The aim is to exploit the saved data in a meaningful way and shape the processes themselves [2]. In this context *process mining* has emerged as a research area between business process management and data mining, attracting attention from research and practice.

Process mining includes three main activities, namely: Process Discovery, Conformance Checking, and Process Enhancement. Following an unsupervised approach, Process Discovery aims to mimic the process model by analyzing the given event log. The idea behind this task is to generalize the log data to create a model that can explain and simulate the executed processes. On the other hand, Conformance Checking is performed in a supervised setting. Here the inputs are an already known process model (the reference process model) and the observed event log. The goal is to identify discrepancies between the analyzed model and the underlying event log. Finally, Process Enhancement tries to improve an existing process model using information about the actual processes recorded in an event log.

In addition to the aforementioned activities, *Operational Support* is becoming increasingly important, due to its ability to automatically make decisions while the processes are running, in an *online* fashion. Thus, given a *partial trace* representing the current process execution, it automatically detects deviations (**Detect**), predicts the remaining processing time for the running activity (**Predict**) or recommends the resource to be allocated, in order to complete an activity (**Recommend**) [3]. Researchers are paying particular attention to the prediction tasks (a.k.a. Predictive Process Monitoring — PPM), like the prediction of the following activity to be executed, which may assist in resolving long-standing problems such as the allocation of human resources or the decision on the specific actions to take [4]. Nevertheless, Predictive Process Monitoring is not limited only to those tasks; indeed, it aims to precisely forecast a process performance measure in the future. Fig. 1 shows a graphical representation of a PPM task. We can distinguish two fundamental moments in the task: the prediction moment and the predicted moment. The prediction moment is the instant in the present time when we decide to make a prediction. The predicted moment refers to the subject of the prediction. A prediction

* Corresponding author at: Dept. of Computer Science, University of Bari, Via Orabona, 4, 70125 Bari, Italy.
E-mail addresses: antonio.pellicani@uniba.it (A. Pellicani), michelangelo.ceci@uniba.it (M. Ceci).

https://doi.org/10.1016/j.knosys.2025.113544
Received 22 October 2024; Received in revised form 25 March 2025; Accepted 9 April 2025
Available online 25 April 2025

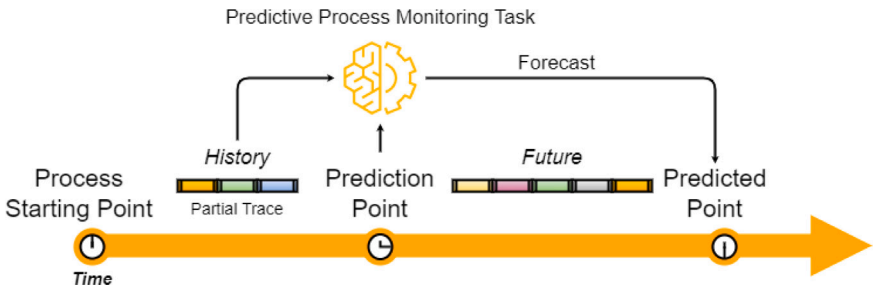


Fig. 1. Predictive process monitoring task pipeline.

is thus dependent on the predictor's memory, which is derived from the knowledge of the process execution history up to the prediction point.

In order to properly take into account the execution history of the process to which the prediction should apply, the predictor's memory should be properly represented. This representation should, in addition, be coherent with the representation used in the learning phase, in order to make the predictor actionable. In this perspective, raw traces cannot be exploited to directly train PPM models since they need a preliminary encoding phase that is able to encapsulate the predictor's memory. This encoding should be able to (i) transform the trace into a feature-vector representation, which can be directly used by common machine learning algorithms, and (ii) extract hidden but precious information about the activities in the trace, including temporal patterns. The trace encoding method, typically used in the literature to deal with the sequence of activities inside a trace, is the one hot encoding. It is usually applied on increasing sequences of activity called prefix traces [5,6]. However, the obtained feature vectors may fail to capture temporal patterns, which may exist between activities inside a trace or among traces [5,7]. These kinds of patterns are essential since they model the temporal sequence, according to which activities are performed, and cannot be ignored when training models that should be able to consider how activities influence the execution of a running trace.

This paper proposes OREO (pOsitional tRace EncOding), a novel positional-based trace encoding method capable of grasping *local temporal patterns* inside a partial trace, by exploiting a sliding window model. The use of sliding windows, typically adopted in stream data mining, limits the analysis to the last significant activities, allowing OREO to properly represent *only* the full sliding window and possible local temporal patterns it should contain. Thus, OREO overcomes the limitation of simple prefix trace encoding methods, creating a robust representation that intrinsically selects the most relevant features that could be effectively exploited to train a predictive model for a running process and for considering the control-flow perspective. More specifically, we adopt a positional encoding so it is possible to represent which activity took place in which position of the trace, but only focusing on the sliding window. It is well-known that trace encoding is a critical step in process mining. Usually trace encoding approaches consider multiple perspectives (i.e. control flow, elapsed time, allocated resources) and summarize them in a single significant representation, creating a new feature space [8]. Thus, choosing the correct views to include in an encoding could significantly influence the successfulness of the process mining task, boosting the performance of the trained models and avoiding the creation of complex separation boundaries, which could prevent the correct classification of traces and activities [9,10]. Moreover, it is well recognized that the time window model is very robust to capture the concept drift phenomenon, according to which the data distribution may change over time [11]. This aspect, translated in process mining, means that in a trace the next activities are more related to the most recent activities than activities carried out in the more distant past.

The main contributions of this paper are therefore the following:

- We present a detailed state-of-the-art analysis of the trace encoding methods and next activity prediction challenge;
- We propose a novel positional trace encoding method;
- We employ the trace encoding method for next activity prediction, using five different deep learning models;
- We provide an extensive empirical evaluation of our method, including a thorough comparison with several state-of-the-art methods.

The rest of the paper is structured as follows: Section 2 summarizes the state of the art related to the next activity prediction, focusing on the chosen representation for the traces. Section 3 introduces the preliminary notions to examine the problem. Then Section 4 describes the proposed method, while Section 5 illustrates the experimental setup and the obtained results. Finally, in Section 6 we draw some conclusions and define some future goals.

2. Background and motivation

Predictive Process Monitoring (PPM) has emerged as a crucial technology in both academic research and industry applications, demonstrating significant impact across various business domains [12–14]. Its economic implications are substantial, with documented benefits in operational efficiency, cost reduction, and competitive advantage [15]. Indeed, monitoring business processes and predicting their future behavior can allow managers to behave proactively before events occur [16].

Among the various PPM tasks, this paper focuses on next activity prediction, which aims to predict the next activity to be executed in a running process. This is one of the most researched classification tasks in PPM [17], with several important use cases: (1) Resource allocation, in which resources have to be distributed in advance based on anticipated future activities [18]; (2) Best action recommendation, in which based on the critical performance metrics that the business wants to maximize, the next best actions have to be identified [19]; 3) Early warning, that monitors the next most probable activity to detect wastes, threats, errors, or difficulties in advance [20]; 4) Anomaly detection, in which abnormal process instances are detected by looking at the probability distribution of the next activity predictions [21].

However, PPM encompasses a broader range of predictive tasks. Indeed, classification approaches can also address risk prediction [22,23], agreement violation prediction [24,25], and outcome prediction [26,27]. Furthermore, PPM techniques can target continuous domains through regression-based approaches [17]. For example, in [28], the authors use and expand existing process mining approaches to provide a framework to analyze and forecast manufacturing costs. Several regression-based approaches aim to predict the time taken to complete a case, which is essential when dealing with service level agreement constraints [29–31]. In [32] the remaining time prediction for running cases is generated after a *transition system* is created and annotated. Similarly, Polato et al. [33] exploit both the control flow perspective and the data flow perspective to improve the prediction quality. These authors use the event log to create a transition system, which they subsequently annotate using *Na"ive Bayes* and *Support Vector Regression* models. These models are exploited to forecast the remaining time from

the next state and the probability of transition from one state to the next, respectively. Ceci et al. [34] rely on *sequence trees* for predicting the completion time. This approach naturally exploits the control-flow perspective and intrinsically allows the clustering of traces showing similar activities. Then a regressor, useful to predict the remaining time, is trained for each node in the built sequence trees. Other examples of work aiming to predict the remaining cycle time of a process instance can be seen in [5,35–37].

Another crucial aspect for PPM algorithms is how to represent traces (sequences of activities) in a format suitable for machine learning algorithms. A trace encoding strategy must be able to capture both the sequential nature of the process and the relationships between activities, while transforming categorical activity information into numerical features that can be processed by predictive models [38]. In the literature, several methods for encoding traces are available, ranging from classic one-hot encoding [5] to word embeddings [39], representational learning [40], and encoding approaches that take into account specific trace information, such as the experience of the resource performing an activity [41].

Apart from the classical offline scenario, where a model is trained on historical batches of logs that usually capture the complete course of cases, PPM (and in general, process mining) algorithms have also been applied in the online scenario, where a stream of events is analyzed [42]. In this context, models can be retrained from scratch or fine-tuned as new instances from the stream are acquired and analyzed [43]. However, this online setting introduces specific challenges, most notably the presence of concept drift, where the statistical properties of the target variable change over time in unexpected ways. This phenomenon can affect the performance of predictive models, as the patterns learned from historical data can become less relevant or entirely obsolete to make predictions on current cases [44]. An example of handling these challenges is the work of Pasquadibisceglie et al. [45], which takes advantage of the fine-tuning of deep learning models along with adaptive windowing mechanisms to detect and handle concept drifts in a stream of events.

Given the main focus of this paper on the next activity prediction task, we now review the main approaches proposed in the literature to address this task.

2.1. Next activity prediction

In the literature, multiple approaches have been proposed to solve this complex but practical PPM task, ranging from traditional methods to neural network-based solutions. Among traditional approaches, automata and state-based models have played a foundational role. Breuker et al. [46] introduced RegPFA, a predictive model based on grammatical inference theory and Probabilistic Finite Automatons (PFA). When tested on the BPI2012 and BPI2013 datasets, RegPFA achieved accuracies between 60%–80%. Following a similar approach, Becker et al. [47] also leveraged PFA, developing a model where states are determined by their previous state and the occurred events. Le et al. [48] extended this line of research using Hidden Markov Models, proposing a Hybrid Markov Model that combines higher-order Markov Models with sequence alignment to match unseen traces with similar known sequences.

Another stream of research focuses on tree-based approaches. Lakshmanan et al. [49] used decision trees to predict next events based on context data from semi-structured business processes, while Rozinat et al. [50] applied them to trace-level features. Taking a different perspective on tree structures, Ceci et al. [51] combined pattern mining with nested model learning, arranging prediction models in a tree structure for frequent activity sequences. Their framework remains algorithm-agnostic, allowing the use of any classification method. In a subsequent work, Ceci et al. [52] extended their research to parallel activities through Parallact, a distributed framework using density-based clustering and multi-target classification. Departing from these approaches, Ferilli et al. [53] explored declarative process mining, using first-order logic reasoning for next activity prediction.

The aforementioned approaches exhibit three significant limitations in their methodological frameworks. First, they do not incorporate temporal window modeling, thus failing to account for the differential impact of recent versus historical activities on subsequent process behavior. Second, these approaches generally lack proper implementation of positional encoding mechanisms. This limitation is particularly notable as positional encoding enables the representation of *total ordering* of actions within a trace, capturing the precise sequence of activities, rather than merely representing *partial ordering* (that is, precedes/follows relationships) or the binary presence or absence of activities. Third, these methods typically do not account for the temporal dimension encoded in activity timestamps, despite its demonstrated significance in process behavior prediction [54].

Following their remarkable success in various predictive tasks across different domains, neural networks have emerged as a predominant methodology in next activity prediction, becoming one of the most widely used approaches [55]. Among neural architectures, Recurrent Neural Networks (RNNs), and particularly Long Short-Term Memory (LSTM) networks, have been widely adopted due to their natural ability to process sequential data and capture temporal dependencies [56,57]. One of the pioneering works in this direction was proposed by Tax et al. [5], who developed a three-layer LSTM architecture predicting both next activities and their timing. Their model used one-hot encoding with additional temporal features, though this encoding approach showed limitations with increasing numbers of activities [54]. Subsequently, Evermann et al. [58] enhanced LSTM-based prediction by introducing an embedding layer, similar to word embeddings, which mapped each input to an n-dimensional vector and incorporated additional event attributes. Building on this, Lin et al. [59] proposed MM-Pred, an encoder–decoder architecture using LSTM for direct attribute transformation. However, both approaches overlooked temporal windows and treated all activities equally regardless of their timeline position. Addressing these limitations, Camargo et al. [54], advanced LSTM application by incorporating timestamps and pre-trained embeddings combining activity and resource encoding. Pasquadibisceglie et al. [60] further developed this direction through multi-view learning, using separate embedding layers for different categorical attributes before concatenation with non-categorical features.

Recent LSTM applications have addressed the inherent *black-box* nature of these models by incorporating interpretable methods. Notably, Wickramanayake et al. [61] proposed two different types of attention mechanisms coupled with a classical LSTM model, demonstrating that high accuracy can be achieved while maintaining model interpretability.

Beyond LSTM architectures, Mehdiyev et al. [62] introduced a sophisticated embedding strategy encompassing multiple process perspectives. Their approach combines n-gram representation with feature hashing across various event attributes, utilizing a deep-stacked autoencoder for hierarchical feature representation before final prediction through a feedforward neural network.

A parallel line of research explored Convolutional Neural Networks (CNNs). Pasquadibisceglie et al. [63] exploited accumulated one-hot encodings obtained from the executed activities, along with the number of days that have passed from the current event and the first event in a process instance, to generate artificial *two-channel images*. A multi-level CNN then analyzes these images and generates the predictions. Their subsequent work [64] arranged features linearly in RGB images, while Di Mauro et al. [65] combined embedding layers with stacked CNN inception modules.

Finally, several other deep neural architectures have been successfully applied to next activity prediction, including Generative Adversarial Networks (GANs) [66], transformers [67], and Graph Neural Networks (GNNs) [68].

Table 1
Example of two traces taken from the event log BPI 2017 Offer. (See Section 5.2 for details of this event log).

Trace	Event	CaseID	Activity	Timestamp
1	e_1	Offer_247135719	Create Offer	2016/01/02 10:17:05
	e_2	Offer_247135719	Created	2016/01/02 10:17:08
	e_3	Offer_247135719	Sent (online only)	2016/01/02 10:19:21
	e_4	Offer_247135719	Cancelled	2016/01/02 10:21:26
2	e_5	Offer_1365106765	Create Offer	2016/01/02 10:55:46
	e_6	Offer_1365106765	Created	2016/01/02 10:55:47
	e_7	Offer_1365106765	Sent (mail and online)	2016/01/02 10:59:50
	e_8	Offer_1365106765	Returned	2016/01/08 11:00:24
	e_9	Offer_1365106765	Accepted	2016/01/11 10:42:07

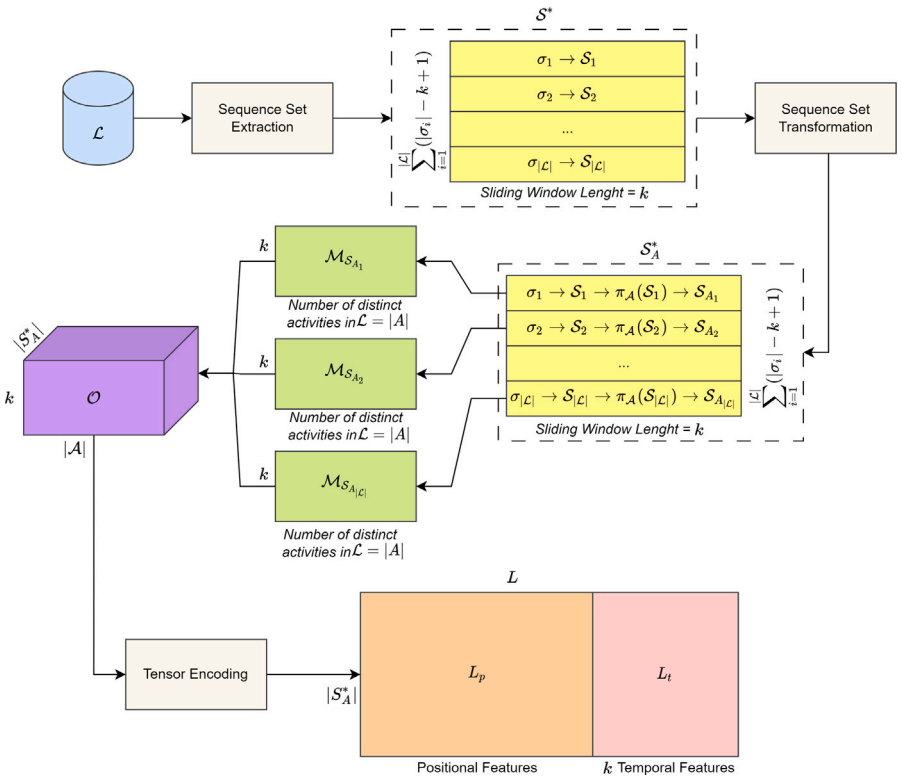


Fig. 2. The steps of OREO encoding.

Despite incorporating temporal information, these methods fail to fully capture the dynamic nature of process execution, as they do not exploit the time-window approach. This approach overcomes the fundamental limitations of simple prefix trace encoding by intelligently identifying and leveraging the most relevant features for prediction in running processes. Moreover, as detailed in Section 4, OREO provides an additional key advantage: its encoding strategy is completely architecture-agnostic, enabling seamless integration with various neural networks including CNNs, LSTMs, and transformer-based models.

3. Preliminary notions

In this section we adapt the definitions provided by van der Aalst [2] for Predictive Process Monitoring to formally define the *next activity prediction* problem.

Given the set of activities $\mathcal{A}$ and the set of case identifiers $\mathcal{C}$ we define the concept of *event*, *trace* and *event log*.

Definition 1 (*Event*)**.** An event $e_i = (a_i, c_i, t_i) \in \mathcal{E}$ is a *triple*, where $a_i \in \mathcal{A}$, $c_i \in C$, t_i is the timestamp and $\mathcal{E}$ is the set of all possible events.

Therefore, given the event e_i, it is possible to define three *utility functions*: $\pi_{\mathcal{A}}(e_i) = a_i$, $\pi_{\mathcal{C}}(e_i) = c_i$, and $\pi_{\mathcal{T}}(e_i) = t_i$, which return the activity, the case identifier and the timestamp associated to an event, respectively.

Definition 2 (*Trace*)**.** A trace is the finite sequence of events $\sigma = \langle e_1, e_2, \dots, e_n \rangle$, having $e_i \in \mathcal{E}$, $\pi_{\mathcal{T}}(e_i) \leq \pi_{\mathcal{T}}(e_{i+1})$ and $\pi_{\mathcal{C}}(e_i) = \pi_{\mathcal{C}}(e_{i+1})$ for $1 \leq i \leq n-1$.

A trace is therefore the abstraction of a completed process, where we can identify a total ordering among the composing events. Moreover, we can further abstract the previously defined *utility functions* to work on traces. Thus, given a trace σ_i composed of n events, $\pi_{\mathcal{A}}(\sigma_i) = \langle a_{i1}, \dots, a_{in} \rangle$ and $\pi_{\mathcal{T}}(\sigma_i) = \langle t_{i1}, \dots, t_{in} \rangle$ are the functions returning sequences of activities and timestamps related to the trace σ_i. Finally, a collection of traces defines an event log.

Definition 3 (*Event Log*)**.** An event log $\mathcal{L} = (\sigma_1, \dots, \sigma_{|\mathcal{L}|})$ is a set of traces, where $\forall\, (\sigma_i, \sigma_j) \in \mathcal{L},\ i \neq j$. We have that $\sigma_i \cap \sigma_j = \emptyset$.

An example of an event log taken from the BPI 2017 Offer Dataset can be found in Table 1. This paper aims to extract valuable knowledge from event logs, in order to perform the next activity prediction task, defined as:

Table 2
Example of traces from BPI 2017 Offer log after the sequence set extraction phase using a sliding window of size $k = 3$.

$S^* =$	σ_1	$S_1 \rightarrow$	$\langle e_1, e_2, e_3 \rangle$
			$\langle e_2, e_3, e_4 \rangle$
	σ_2	$S_2 \rightarrow$	$\langle e_5, e_6, e_7 \rangle$
			$\langle e_6, e_7, e_8 \rangle$
			$\langle e_7, e_8, e_9 \rangle$

Definition 4 (*Next Activity*)**.** Given a time instant t and a (partial) trace σ of length k, that is $\sigma = (e_1, \dots, e_k)$, the next activity of σ is $\pi_{\mathcal{A}}(next(\sigma)) = e_{k+1}$.

4. Method

In the following subsections we describe the event log representation, which leads to the proposed encoding method. Then we give some insights into the deep neural network models that we use for the encoded logs and to solve the next activity prediction task.

4.1. Data representation

The event logs that indicate the execution of activities in a business process are the starting point for predictive process monitoring. As stated in Section 3, events from the same process can be chronologically (totally) ordered into traces, allowing us to analyze the process execution while it is still running. However, current machine learning and deep learning state-of-the-art techniques cannot directly grasp useful information from the raw traces, like that obtained by directly considering the activity position in the trace or the relation among the execution times of the single activities. Thus, encoding techniques able to extract feature vectors from traces are needed before training a predictive process monitoring model [69].

Formally, a *trace encoder* is a function $f : \mathcal{L} \rightarrow \mathcal{F}_1 \times \cdots \times \mathcal{F}_r$ that maps any (complete/partial) trace σ to an r-dimensional feature vector $\mathcal{F}_1 \times \cdots \times \mathcal{F}_r$, with $\mathcal{F}_i \subseteq \mathbb{R}, 1 \leq i \leq r$. A variety of viewpoints could be synthesized by the r extracted features. For example, traditional encoding usually relies on the control-flow perspective, focusing on the activities performed during the trace execution and their order. Other encodings further analyze the categorical and numerical attributes that come with a trace, i.e., deepening the time perspective or the resource perspective [9].

On the contrary, this work proposes a representation that focuses on the time and control-flow perspective, creating a positional encoding that could be exploited in many predictive process monitoring tasks (not only for next activity prediction). To obtain the encoding function, three phases are performed in OREO: (i) sequence set extraction, (ii) sequence set transformation, (iii) tensor encoding.

Fig. 2 provides an abstract representation of the three phases and their operations, while Algorithm 43 presents the corresponding pseudocode. Each phase transforms the original log, encoding the temporal and sequential relationships among the activities in the traces.

In particular, in the *sequence set extraction* phase, each trace σ_i from the log $\mathcal{L}$ is analyzed to generate a sequence set S_i. This set contains the sequences of events obtained by applying a sliding window of dimension k to the original trace.

$$S_i = \{\langle e^i_j, \dots, e^i_{j+k-1} \rangle \mid 0 < j \leq |\sigma_i| - k + 1, 0 \leq i < |\mathcal{L}|\}, \tag{1}$$

where e^i_j is the jth event belonging to the ith trace in the event log. Note that the number of sequences in each set S_i depends on the length of the original trace σ_i: a trace of length n will generate $n-k+1$ sequences. Each of these sequences can be seen as a *partial traces*. Hence, all the utility functions defined in Section 3 can be applied. In addition, we can define the superset S^*, containing all the sequence sets extracted from the event log $\mathcal{L}$.

$$S^* = \bigcup_{i=1}^{|\mathcal{L}|} Si \tag{2}$$

In this phase, the choice of the sliding window dimension k determines the algorithm's *forgetting level*. Indeed, a low value of k will emphasize recently executed events, forgetting the older ones. Otherwise, a higher k value will allow the algorithm to reason on far-in-time events, without considering a possible concept drift in the log. Table 2 shows an example of sequence set extraction starting from the partial log shown in Table 1 (in the example we use a sliding window of size $k = 3$). In the example S^* contains five sequences, two of which have been extracted

Algorithm 1: OREO Encoding

Input: Event log $\mathcal{L}$, window size k
Output: Encoded log L

```
/* Phase 1: Sequence Set Extraction */
S* ← ∅;
foreach trace σ_i ∈ L do
    S_i ← ∅;
    for j ← 1 to |σ_i| − k + 1 do
        sequence ← ⟨e^i_j, ..., e^i_{j+k−1}⟩;
        S_i ← S_i ∪ {sequence};
    end
    S* ← S* ∪ S_i;
end
/* Phase 2: Sequence Set Transformation */
S*_A ← ∅;
A ← get_unique_activities(L);
foreach S ∈ S* do
    S_A ← π_A(S);
    S*_A ← S*_A ∪ {S_A};
end
O ← zeros(k, |A|, |S*_A|);
for i ← 1 to |S*_A| do
    M_{S_A} ← zeros(k, |A|);
    S_A ← S*_A[i];
    for p ← 1 to k do
        M_{S_A}[p] ← one_hot_encode(S_A[p], A);
    end
    O[:, :, i] ← M_{S_A};
end
/* Phase 3: Tensor Encoding */
L ← zeros(|S*_A|, |L_p| + k);
/* Set positional features (first |L_p| columns) */
col ← 1;
for w_p ← 1 to k do
    foreach a_j ∈ A do
        for i ← 1 to |S*_A| do
            L[i, col] ← O_{(w_p, a_j, i)};
        end
        col ← col + 1;
    end
end
/* Set temporal features (last k columns) */
for i ← 1 to |S*_A| do
    for w_p ← 1 to k do
        σ_orig ← get_original_trace(S*_A[i]);
        t_start ← get_first_activity_time(σ_orig);
        t_curr ← get_activity_time_at_position(σ_orig, w_p);
        L[i, |L_p| + w_p] ← t_curr − t_start;
    end
end
return L
```

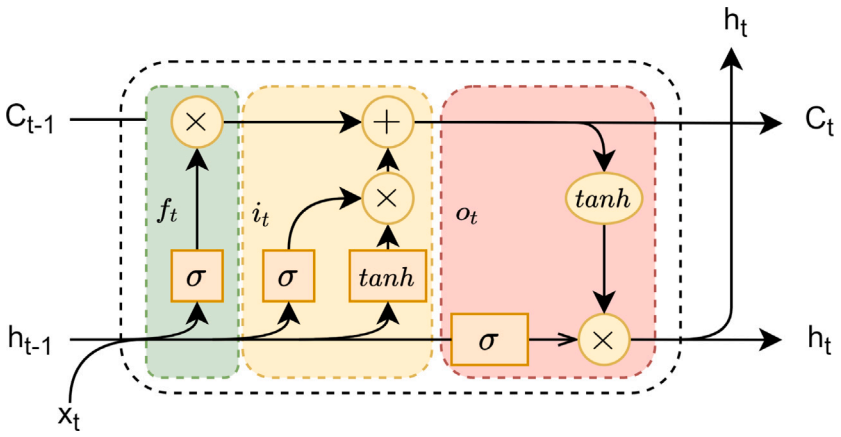


Fig. 3. An example of LSTM cell.

from S_1 (containing events from σ_1), while the other three come from S_2 (with the events extracted from σ_2). This difference in the number of sequences is due to the lengths of the traces: σ_1 has length 4, thus generating $|S_1| = |\sigma_1| - k + 1 = 4 - 3 + 1 = 2$ sequences, while σ_2 has length 5, resulting in $|S_2| = |\sigma_2| - k + 1 = 5 - 3 + 1 = 3$ sequences. It is noteworthy that the term *forgetting level* does not mean that old activities in the trace are discarded and not used in the training phase: they are still used but the models cannot use them as direct descriptive properties for the next activity prediction task.

The second phase, *sequence set transformation*, aims to transform the superset S^* into a three-dimensional tensor. Thanks to this tensor, the partial traces extracted in the first phase are aligned based on their position in the window. Consequently, inter-trace relationships among the activities could be captured and exploited during the learning phase.

In order to build the tensor, several steps have to be performed. First, the executed activities are extracted for each sequence in S^*, exploiting the utility function $\pi_{\mathcal{A}}$ and generating a new superset S^*_A.

$$S^*_A = \{\pi_{\mathcal{A}}(S) \mid \forall S \in S^*\} \tag{3}$$

Then each activity sequence $S_A \subseteq S^*_A$ is further processed by creating a matrix $M_{S_A} \in \mathbb{N}^{k \times |A|}$, where k is the sliding window dimension, A is the set of distinct activities in the event log $\mathcal{L}$ and $|A|$ is the cardinality of this set. Each row of the M_{S_A} matrix will contain the one-hot encoding of each activity in S_A. Finally, the three-dimensional OREO tensor $\mathcal{O} \in \mathbb{N}^{k \times |A| \times |S^*_A|}$ is obtained by combining all the matrices created from the activity sequences in S^*_A.

$\mathcal{O}$ now contains information on all partial traces, each aligned on the basis of its position in the sliding window. However, since the construction of $\mathcal{O}$ includes transformation steps such as one-hot encoding, the tensor is very likely to be sparse, which can cause hampering of the learning effectiveness.

In the third phase, we perform *tensor encoding* to create a representation that reduces data sparsity, captures both positional and temporal patterns, and enables effective model learning. We define our feature set L as the union of positional features L_p and temporal features L_T:

$$L = L_p \cup L_T, \tag{4}$$

The positional features L_p capture the occurrence of activities at specific positions within the sliding window:

$$L_p = \{\mathcal{X}^p_{\langle w_p, a_j \rangle} \mid w_p \leq k \wedge a_j \in A \wedge \exists i = (1, \dots, |S^*_A|) : \mathcal{O}_{(w_p, a_j, i)} = 1\} \tag{5}$$

where $\mathcal{X}^p_{\langle w_p, a_j \rangle}$ is a positional binary feature that, intuitively, represents the fact that activity a_j is executed at position w_p in a sliding window of a trace (sequence). More specifically, each positional feature $\mathcal{X}^p_{\langle w_p, a_j \rangle}$ is characterized by two indices: w_p representing the specific position within the sliding window (ranging from 1 to window size k), and a_j representing a specific activity from the set of possible activities A. For example, feature $\mathcal{X}^p_{\langle 2, a_5 \rangle}$ would indicate whether activity a_5 appears in position 2 of the sliding window.

To complement the positional information, we introduce temporal features L_T, which capture the timing aspects of activities:

$$L_T = \mathcal{X}^T_{\langle a_j \rangle} \mid a_j \in A, \tag{6}$$

where $\mathcal{X}^T_{\langle a_j \rangle}$ represents the normalized time distance between the start of the trace and the moment activity a_j is executed.

The final encoding creates a matrix with $|S^*_A|$ rows and $|L_p| + k$ columns, where each row x_i represents a sliding window of a trace. The values in this matrix are determined as follows:

$$x_{i,l} = \begin{cases} 1, & \text{if } l \leq |L_p|, l = \mathcal{X}^p_{\langle w_p, a_j \rangle} \text{ and } \mathcal{O}_{(w_p, j, i)} = 1 \\ 0, & \text{if } l \leq |L_p|, l = \mathcal{X}^p_{\langle w_p, a_j \rangle} \text{ and } \mathcal{O}_{(w_p, j, i)} = 0 \\ t_{i,a_j}, & \text{if } |L_p| < l \leq |L_p| + k,\ l = \mathcal{X}^T_{\langle a_j \rangle} \end{cases} \tag{7}$$

where t_{i,a_j} represents the time distance between the start of the trace and the execution of activity a_j.

4.2. The OREO models

To demonstrate the versatility and effectiveness of our encoding, we evaluate it across multiple predictive architectures, which are directly inspired from existing state-of-the-art next activity prediction approaches. This allows us to directly evaluate the effectiveness of our encoding, when compared with such methods. Specifically: (i) OREO-LSTM builds upon the predictive architecture proposed by [60], incorporating their LSTM architecture; (ii) OREO-Image adopts the predictive architecture proposed by [63]; (iii) OREO-Inception adopts the predictive architecture proposed by [65]; and (iv) OREO-Transformer adopts the predictive architecture proposed by [67]. The goal is to learn a function f which given an encoded partial trace σ_{enc} at time t returns the next activity a_{t+1} to be executed in that trace, more formally:

$$f(\sigma_{enc}) \rightarrow a_{t+1} \tag{8}$$

As its name suggests, the OREO-LSTM model is based on the well-known *Long Short-Term Memory* (LSTM) [70], a particular *Recurrent Neural Network* (RNN) capable of grasping strong temporal dependences when dealing with sequence data. Unlike classical Multi-Layer Perceptrons, where a layer is fully connected to the next without any cycle, LSTMs allow feedback loops to share parameters across the model and maintain a *memory*. Our model exploits two LSTM layers

entangled by two Batch Normalization layers to settle the learning process. The number of *hidden units* (LSTM cells) in the two LSTM layers is an adequately optimized hyperparameter (see Section 5.1 for further details). Basically, an LSTM cell incorporates three types of gates: input gate, forget gate and output gate, and can be described by the following equations:

$$f_t = sigmoid\left(W_f x_t + V_f h_{t-1} + b_f\right), \tag{9}$$

$$i_t = sigmoid\left(W_i x_t + V_i h_{t-1} + b_i\right), \tag{10}$$

$$C_t = f_t C_{t-1} + i_t \times tanh\left(W_c x_t + V_c h_{t-1} + b_c\right), \tag{11}$$

$$o_t = sigmoid\left(W_o x_t + V_o h_{t-1} + b_o\right), \tag{12}$$

$$h_t = o_t \times tanh\left(C_t\right), \tag{13}$$

where x_t and h_t are the input and the hidden state at time step t, while W, V, and b are trainable network parameters representing the weight matrices and the biases. The forget gate is described in Eq. (9). Specifically, the forget gate exploits the sigmoid function to combine the information from the previous hidden state and the current input, obtaining a value between 0 and 1. The closer this score is to 0, the less knowledge of the earlier steps is kept. Then the input gate (Eq. (10)) takes into account both the current input and the earlier hidden states, exploiting a sigmoid function. In this case, the output score between 0 and 1 indicates the importance of the new information held by the input. At this point the network has sufficient information to calculate the new cell state (Eq. (11)), weighting the last state by the forget score and adding a scaled input value. Subsequently, the new hidden state h_t (Eq. (13)) is calculated by multiplying the output gate score (Eq. (12)) with the current cell state C_t, solely after applying the tanh activation function. Fig. 3 shows an example of LSTM cell, with all the interactions among the gates highlighted. In OREO-LSTM two LSTM layers are used (coherently with [5]), followed by a dense layer to compute the probabilities of each predictable class, i.e., the next activities which could be executed. Specifically, our dense layer exploits the softmax activation function to predict a class y_i:

$$y_i = softmax(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{|C|} \exp(z_j)}, \tag{14}$$

where z_i and z_j are the values assumed by the ith and the jth output neuron in the final dense layer, respectively. The architecture of the OREO-LSTM model is shown in Fig. 4

Contrary to OREO-LSTM, OREO-Inception and OREO-Image cores are based on the Convolutional Neural Network architecture [71], a popular family of DNNs commonly used in computer vision, speech recognition [72] and time series analysis [73], which are specialized in processing data with a grid-like topology. Nevertheless, Convolutional Neural Networks have also proved their potentiality in the process mining field, showing stimulating results when employed for predicting the next activity that will be executed in a running trace [64,65] or the trace outcome [74]. The building block of a CNN is the convolutional layer, which comprises a group of filters having their own set of parameters that must be learned. The dimensions of the filters are smaller than those of the input. Each filter is convolved with the input to produce an activation map of neurons. Usually convolutional layers are coupled with pooling layers to reduce the output dimensions, while preserving the most critical information. This makes the network robust to slight differences in pattern positions in the input data, as happens in image processing tasks. Our CNN architectures rely on the global max pool operation, which provides the best results in the literature [75], combined with a dense layer. This dense layer is in charge of performing the classification through the softmax activation function (see Eq. (14)).

Nonetheless, traditional deep convolutional networks may suffer from overfitting and high computational costs. Attempts to reduce these problems rely on sparsely connected architectures. Moreover, classic hardware is better suited for the computation of dense matrices, making

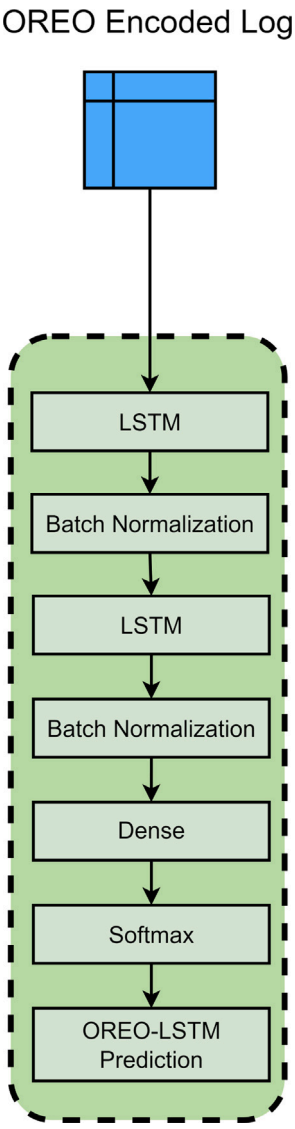


Fig. 4. The OREO-LSTM model and its components.

it necessary to approximate the sparse structure of the networks. For the design of the architecture of OREO-Inception, we consider the well-known naïve inception module. Developed by Google for its state-of-the-art CNN-classifier GoogleNet [76], the inception module creates wider neural networks instead of deeper ones, to avoid overfitting and decrease the computational cost. This result is achieved by combining one max pooling layer with three parallel convolution layers, having distinct filter sizes that simultaneously apply to the same input. Then a concatenation layer is used to juxtapose the outputs of all the former layers. OREO-Inception stacks two inception modules before performing a global max pooling. As usual, a final dense layer outputs the prediction. The architecture of the OREO-Inception model is shown in Fig. 5a.

On the other hand, OREO-Image is still based on classical CNNs, but relies on a 2D Convolution. Thus, an additional transformation phase is required to convert the encoded traces into an image-like matrix. This can easily be achieved by applying some well-established trace-to-image conversion techniques [64,74]. In particular, we use the DeepInsight method [77], which projects each encoded feature into a Cartesian plan, by applying a non-linear dimensionality reduction technique. The convex hull algorithm is then used to find the smallest rectangle containing all the features which define the transformed

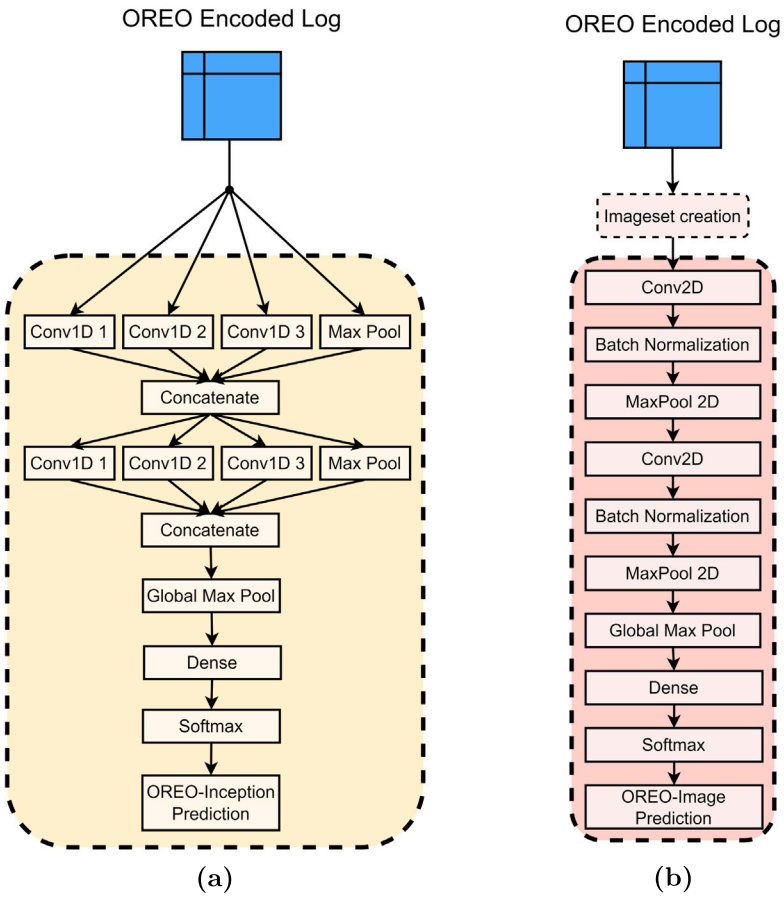


Fig. 5. The CNN-based OREO models and their components. (a) OREO-Inception. (b) OREO-Image.

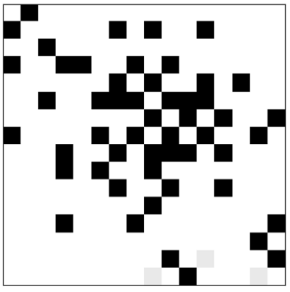


Fig. 6. An example of a bitmap obtained by applying DeepInsight to an OREO-encoded sequence, taken from the BPI Challenge 2020 — Request for Payment event log.

image's border. Finally, each row of the encoded log is converted into an image (bitmap) by assigning feature values to the corresponding features' Cartesian coordinates. Fig. 6 shows an example of a bitmap obtained by applying this technique when the t-SNE dimensionality reduction is used to generate the feature coordinates. To the best of the authors' knowledge, this is the first time the DeepInsight encoding has been applied to the next activity prediction task, allowing the use of 2d-CNNs, which are well known for their capability of extract complex patterns and relationships from image data. The OREO-Image neural network architecture is shown in Fig. 5b.

While CNN architectures have proven effective in capturing spatial patterns in the encoded traces, recent advancements in sequence modeling have highlighted the potential of attention-based mechanisms for process mining tasks [78]. The OREO-Transformer model leverages the Transformer architecture [79], which has revolutionized natural language processing and sequence modeling tasks by introducing a mechanism that can effectively capture long-range dependencies in sequential data without recurring connections. Unlike LSTM and CNN-based approaches, Transformers rely entirely on the self-attention mechanism to model relationships between elements in a sequence. This mechanism allows the model to weigh the importance of different positions in the input sequence when computing a representation for each position, enabling parallel processing and better handling of long-term dependencies. The core component of our OREO-Transformer is the encoder block, which consists of a multi-head self-attention layer followed by a position-wise feed-forward network. The multi-head attention mechanism splits the input into multiple attention heads, allowing the model to jointly attend to information from different representation subspaces at different positions. Each head computes attention scores using scaled dot-product attention, where the input sequence is transformed into queries, keys, and values through learned linear projections. Position encoding is added to the input embeddings to provide the model with information about the relative or absolute position of the events in the sequence, which is crucial since the Transformer architecture is inherently permutation-invariant. The model's architecture concludes with a dense layer using softmax activation for predicting the next activity. This approach has shown promising results in process mining tasks, as it can effectively capture both local and global patterns in event sequences while being computationally efficient due to its parallel nature [67]. The OREO-Transformer neural network architecture is shown in Fig. 7a.

While the Transformer architecture demonstrates the power of pure attention-based approaches, we further investigated how attention mechanisms could enhance traditional sequential models. The last model we developed, OREO-SelfAttention combines LSTM and self-attention mechanisms, leveraging both the temporal modeling

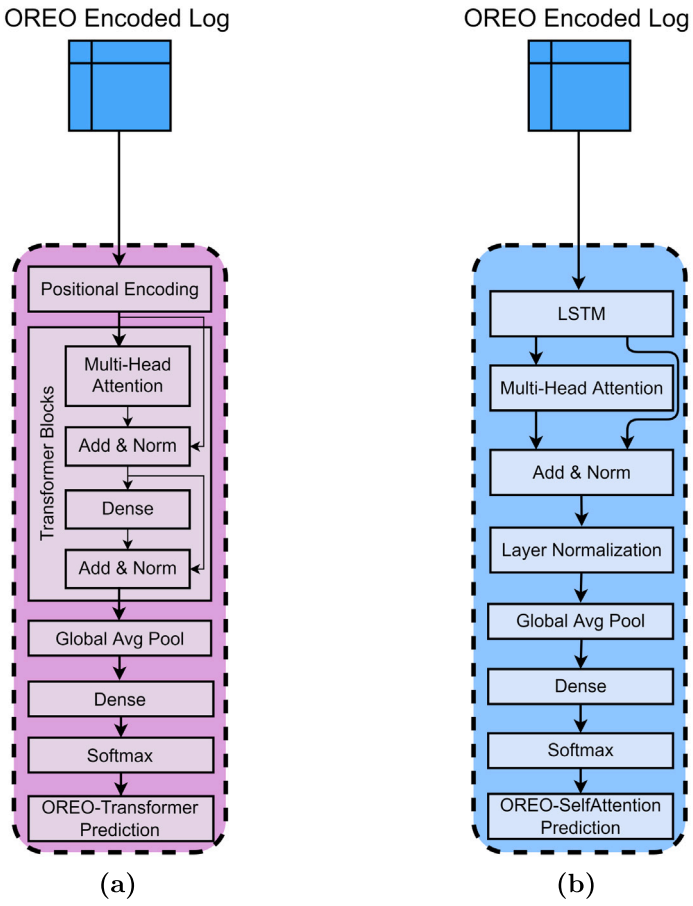


Fig. 7. The attention-based OREO models and their components. (a) OREO-Transformer. (b) OREO-SelfAttention.

Table 3
Optimized hyperparameters.

Network	Parameters	Value
OREO-LSTM	*Batch Size*	{128, 256, 512}
	Learning Rate	[0.0001, 0.01]
	LSTM Unit Size	{50, 100, 150, 200}
OREO-Inception	*Batch Size*	{128, 256, 512}
	Learning Rate	[0.0001, 0.01]
OREO-Image	*Batch Size*	{128, 256, 512}
	Learning Rate	[0.0001, 0.01]
	Number of Filters	{32, 64}
	Kernel Size	{2 × 2, 4 × 4}
	Pool Size	{2 × 2, 4 × 4}
OREO-Transformer	*Batch Size*	{128, 256, 512}
	Learning Rate	[0.0001, 0.01]
	Number of Attention Heads	{4, 8, 16}
	Number of Transformer Blocks	{2, 4, 8}
OREO-SelfAttention	*Batch Size*	{128, 256, 512}
	Learning Rate	[0.0001, 0.01]
	LSTM Unit Size	{50, 100, 150, 200}
	Number of Attention Heads	{4, 8, 16}

Table 4
Quantitative characteristics of the analyzed event logs.

Event Log	Traces	Events	Activities
BPI12Complete	13 087	164 506	23
BPI12 W	9658	170 107	19
BPI12WComplete	9658	72 413	6
Receipt	1434	8577	27
BPI13Incident	7554	65 533	13
BPI13Problem	1841	9011	7
BPI17Offer	42 995	193 849	8
BPI20Request	6886	36 796	19

capabilities of recurrent networks and the flexible focus provided by attention. Through a multi-head self-attention layer applied to the LSTM representations, the model can identify and weigh relevant events in the sequence regardless of their position. The combination of LSTM-generated representations and attention-weighted features is achieved through residual connections and normalization layers, ensuring effective information flow during training. This hybrid architecture provides insights into how attention mechanisms can enhance sequential models for next activity prediction in business processes. The OREO-SelfAttention neural network architecture is shown in Fig. 7b.

5. Experiments

In the following subsections, we first describe the implementation and optimization details of OREO. Subsequently, we describe the datasets (event logs) considered in the evaluation of the performance achieved by the five OREO models. Then we outline the experimental setting that aims to answer the following research questions: **RQ1)** How does the sliding window dimension k affect the accuracy of the predictive model?, and **RQ2)** How do our models compare with recent state-of-the-art deep learning approaches? Finally, we show and discuss the obtained results.

5.1. Models implementation and optimization

OREO has been implemented in Python 3.8.8, using TensorFlow 2.4.1 as the neural network backend. The hyperparameter optimization

Table 5
Summary of evaluated competitor methods.

Approach	Data preprocessing	Architecture	Perspectives used	Strengths	Limitations
[5]	One-hot encoding with temporal features	LSTM	Activities, Time	Clear interpretability; Straightforward implementation	Performance decreases with increasing number of activities
[54]	Pre-trained embedding network	LSTM	Activities, Time, Resources	Rich feature representation; Effective resource integration	Complex pre-training phase required
[58]	Embedding layer for activity encoding	LSTM	Activities, Resources	Efficient representation of categorical data	Omits temporal aspects
[60]	Multi-view encoding	LSTM	All available features	Comprehensive feature utilization	Increased computational demands
[61]	Sequence encoding	LSTM with shared attention	Activities, Time, Resources	Weighs importance of past activities	Requires significant computational resources
[63]	Image-like transformation of sequences	CNN	Activities, Time	Excels at pattern recognition	Fixed input size requirement
[64]	RGB image encoding	CNN	Activities, Time, Resources	Multi-dimensional feature representation	May struggle with long-term dependencies
[67]	Sequence encoding	Transformer	Activities, Time	Models complex sequential dependencies	Highest computational requirements

Table 6
Results on the benchmark datasets with the OREO-LSTM model.

	k	Accuracy			Precision			Recall			Fscore		
BPI12Complete	*2*	0.766	±	0.003	0.692	±	0.027	0.627	±	0.001	0.636	±	0.003
	3	0.806	±	0.008	**0.700**	±	**0.015**	**0.640**	±	**0.004**	**0.665**	±	**0.007**
	4	**0.837**	±	**0.007**	0.688	±	0.020	0.638	±	0.006	0.651	±	0.009
BPI12W	*2*	0.900	±	0.000	0.775	±	0.014	0.723	±	0.009	0.723	±	0.006
	3	**0.906**	±	**0.000**	**0.776**	±	**0.012**	**0.725**	±	**0.013**	**0.749**	±	**0.012**
	4	0.905	±	0.002	0.753	±	0.014	0.715	±	0.010	0.719	±	0.007
BPI12WComplete	*2*	0.811	±	0.004	0.736	±	0.005	0.687	±	0.007	0.699	±	0.005
	3	**0.852**	±	**0.010**	**0.750**	±	**0.030**	**0.708**	±	**0.017**	**0.728**	±	**0.020**
	4	0.849	±	0.004	0.738	±	0.024	0.707	±	0.018	0.704	±	0.009
Receipt	*2*	0.882	±	0.011	0.560	±	0.010	0.511	±	0.015	0.521	±	0.007
	3	0.918	±	0.014	0.559	±	0.008	0.511	±	0.008	0.524	±	0.009
	4	**0.947**	±	**0.013**	**0.575**	±	**0.012**	**0.520**	±	**0.017**	**0.533**	±	**0.013**
BPI13Incident	*2*	0.683	±	0.005	**0.459**	±	**0.063**	0.368	±	0.014	0.374	±	0.014
	3	0.739	±	0.004	0.448	±	0.021	0.393	±	0.017	0.380	±	0.016
	4	**0.760**	±	**0.004**	0.445	±	0.019	**0.396**	±	**0.030**	**0.390**	±	**0.024**
BPI13Problem	*2*	0.678	±	0.004	0.462	±	0.066	0.438	±	0.062	0.435	±	0.065
	3	0.748	±	0.013	**0.466**	±	**0.044**	**0.446**	±	**0.041**	**0.456**	±	**0.043**
	4	**0.826**	±	**0.014**	0.422	±	0.041	0.427	±	0.037	0.422	±	0.040
BPI17Offer	*2*	0.817	±	0.002	0.553	±	0.010	0.572	±	0.000	0.508	±	0.001
	3	0.839	±	0.002	**0.583**	±	**0.004**	**0.601**	±	**0.000**	**0.582**	±	**0.001**
	4	**0.960**	±	**0.001**	0.435	±	0.017	0.500	±	0.000	0.462	±	0.001
BPI20Request	*2*	0.917	±	0.004	0.565	±	0.066	0.527	±	0.062	0.522	±	0.065
	3	0.985	±	0.001	**0.622**	±	**0.019**	**0.599**	±	**0.003**	**0.608**	±	**0.006**
	4	**0.988**	±	**0.001**	0.617	±	0.008	0.594	±	0.032	0.581	±	0.021

phase has been conducted by exploiting the Hyperopt library [80]. For this purpose, we used 20% of the training set as the validation set. Table 3 reports the hyperparameters which have been tuned in the five models. Furthermore, the training phase was stopped if there was no improvement 0n the validation loss for 40 epochs (also known as "early stopping" principle). Finally, the loss function was optimized through the Adam Optimizer, with a maximum number of epochs set to 300.

5.2. Event logs

We use eight event logs extracted from five well-known benchmark datasets, specifically: *BPI12Complete*, *BPI12W*, *BPI12WComplete*, *Receipt*, *BPI13Incident*, *BPI13Problem*, *BPI17Offer* and *BPI20Request*. Table 4 summarizes the quantitative characteristics of the considered event logs.

Table 7
Results on the benchmark datasets with the OREO-Inception model.

	k	Accuracy			Precision			Recall			Fscore		
BPI12Complete	*2*	0.770	±	0.002	0.718	±	0.009	0.623	±	0.003	0.633	±	0.005
	3	0.811	±	0.002	0.728	±	0.012	**0.655**	±	**0.006**	**0.668**	±	**0.005**
	4	**0.842**	±	**0.002**	**0.730**	±	**0.003**	0.645	±	0.005	0.662	±	0.006
BPI12W	*2*	0.889	±	0.002	0.774	±	0.016	0.725	±	0.013	0.727	±	0.012
	3	0.903	±	0.001	0.772	±	0.016	**0.735**	±	**0.006**	**0.753**	±	**0.004**
	4	**0.904**	±	**0.001**	**0.775**	±	**0.017**	0.713	±	0.010	0.719	±	0.007
BPI12WComplete	*2*	0.808	±	0.004	0.730	±	0.001	0.674	±	0.010	0.687	±	0.009
	3	0.830	±	0.003	**0.739**	±	**0.005**	**0.693**	±	**0.005**	**0.701**	±	**0.007**
	4	**0.847**	±	**0.004**	0.737	±	0.026	0.689	±	0.028	0.689	±	0.017
Receipt	*2*	0.882	±	0.010	0.534	±	0.033	0.485	±	0.030	0.497	±	0.031
	3	0.917	±	0.013	**0.539**	±	**0.029**	**0.489**	±	**0.047**	**0.503**	±	**0.042**
	4	**0.946**	±	**0.014**	0.518	±	0.028	0.472	±	0.033	0.486	±	0.030
BPI13Incident	*2*	0.688	±	0.003	**0.456**	±	**0.033**	0.378	±	0.018	0.387	±	0.020
	3	0.743	±	0.005	0.442	±	0.042	0.395	±	0.015	**0.407**	±	**0.009**
	4	**0.758**	±	**0.008**	0.421	±	0.016	**0.397**	±	**0.030**	0.392	±	0.026
BPI13Problem	*2*	0.683	±	0.014	0.450	±	0.076	0.431	±	0.064	0.426	±	0.069
	3	0.762	±	0.014	**0.476**	±	**0.042**	**0.461**	±	**0.057**	**0.465**	±	**0.060**
	4	**0.832**	±	**0.007**	0.467	±	0.069	0.460	±	0.047	0.455	±	0.051
BPI17Offer	*2*	0.817	±	0.002	0.556	±	0.013	0.572	±	0.000	0.507	±	0.001
	3	0.839	±	0.002	**0.596**	±	**0.019**	**0.601**	±	**0.000**	**0.518**	±	**0.001**
	4	**0.960**	±	**0.001**	0.447	±	0.017	0.500	±	0.000	0.462	±	0.001
BPI20Request	*2*	0.917	±	0.014	0.573	±	0.076	0.530	±	0.064	0.525	±	0.069
	3	0.985	±	0.001	0.629	±	0.014	0.591	±	0.013	0.603	±	0.015
	4	**0.988**	±	**0.001**	**0.650**	±	**0.033**	**0.624**	±	**0.078**	**0.612**	±	**0.062**

Table 8
Results on the benchmark datasets with the OREO-Image model.

	k	Accuracy			Precision			Recall			Fscore		
BPI12Complete	*2*	0.744	±	0.001	0.681	±	0.012	0.581	±	0.001	0.589	±	0.001
	3	0.799	±	0.008	0.729	±	0.015	**0.636**	±	**0.004**	0.656	±	0.007
	4	**0.845**	±	**0.001**	**0.730**	±	**0.007**	0.634	±	0.009	**0.658**	±	**0.006**
BPI12W	*2*	0.899	±	0.001	0.778	±	0.035	0.720	±	0.010	0.717	±	0.012
	3	**0.916**	±	**0.003**	**0.779**	±	**0.010**	**0.721**	±	**0.008**	**0.719**	±	**0.020**
	4	0.913	±	0.005	0.694	±	0.084	0.651	±	0.083	0.653	±	0.083
BPI12WComplete	*2*	0.830	±	0.003	0.768	±	0.001	0.707	±	0.001	0.724	±	0.001
	3	0.860	±	0.002	**0.781**	±	**0.023**	**0.763**	±	**0.020**	**0.763**	±	**0.021**
	4	**0.881**	±	**0.002**	0.778	±	0.012	0.756	±	0.014	0.747	±	0.013
Receipt	*2*	0.827	±	0.004	0.494	±	0.040	0.473	±	0.035	0.460	±	0.034
	3	0.905	±	0.011	0.569	±	0.045	**0.531**	±	**0.041**	**0.535**	±	**0.037**
	4	**0.947**	±	**0.022**	**0.576**	±	**0.074**	0.520	±	0.069	0.533	±	0.069
BPI13Incident	*2*	0.680	±	0.005	0.368	±	0.021	0.359	±	0.015	0.354	±	0.015
	3	0.739	±	0.006	0.405	±	0.014	0.363	±	0.027	0.358	±	0.015
	4	**0.758**	±	**0.004**	**0.441**	±	**0.058**	**0.388**	±	**0.034**	**0.380**	±	**0.029**
BPI13Problem	*2*	0.683	±	0.015	0.455	±	0.056	0.437	±	0.054	0.434	±	0.055
	3	0.775	±	0.005	0.468	±	0.066	0.457	±	0.052	0.446	±	0.052
	4	**0.847**	±	**0.009**	**0.480**	±	**0.063**	**0.461**	±	**0.052**	**0.458**	±	**0.052**
BPI17Offer	*2*	0.817	±	0.002	0.550	±	0.003	0.572	±	0.000	0.506	±	0.001
	3	0.838	±	0.002	**0.589**	±	**0.003**	**0.601**	±	**0.000**	**0.517**	±	**0.002**
	4	**0.960**	±	**0.001**	0.435	±	0.001	0.500	±	0.000	0.462	±	0.001
BPI20Request	*2*	0.916	±	0.015	0.512	±	0.056	0.474	±	0.054	0.463	±	0.055
	3	**0.984**	±	**0.002**	**0.611**	±	**0.021**	**0.554**	±	**0.013**	**0.548**	±	**0.010**
	4	0.981	±	0.011	0.572	±	0.185	0.544	±	0.174	0.541	±	0.177

BPI12 [81] is a real-world event log that contains processes extracted from loan applications at a Dutch financial institute, collected through an online system from 2011/10/01 to 2012/03/14. The logged activities are grouped into three specific sub-processes, each tracking a different state of the process i.e., state of the application – denoted by letter A, state of work items related to the application – denoted by letter W, and state of the offer – denoted by letter O. Starting from BPI12, we extracted three different event logs with three different complexity levels, to compare our method with the existing approaches. Specifically, BPI12Complete contains all the activity completion events from all the traces, BPI12 W contains all the activities for the subprocess W, and BPI12WComplete contains all the activity completion events for the subprocess W.

The *Receipt* log [82] derives from the CoSeLoG project. Within the CoSeLoG project, the (dis)similarities between several municipality processes in the Netherlands have been investigated. Specifically, the dataset records logs of building permit application processes in different anonymous municipalities.

The *BPI13* [83] log collects traces recorded by Volvo IT Belgium. The log contains events from an incident and problem management system called VINST. The primary goal of the Handle Incidents Process is to restore regular service operation as quickly as possible, therefore ensuring that the best possible levels of service quality and availability are maintained. If the action owner suspects that the incident might reoccur, a problem record is registered in the system. Thus, the log

Table 9
Results on the benchmark datasets with the OREO-Transformer model.

	k	Accuracy			Precision			Recall			Fscore		
BPI12Complete	*2*	0.773	±	0.001	0.725	±	0.006	0.612	±	0.006	0.627	±	0.006
	3	0.813	±	0.001	**0.730**	±	**0.015**	**0.632**	±	**0.004**	**0.652**	±	**0.003**
	4	**0.847**	±	**0.002**	0.729	±	0.014	0.614	±	0.011	0.638	±	0.010
BPI12W	*2*	0.891	±	0.003	0.759	±	0.026	0.712	±	0.010	0.715	±	0.004
	3	**0.903**	±	**0.003**	**0.768**	±	**0.027**	**0.715**	±	**0.004**	**0.716**	±	**0.011**
	4	0.903	±	0.001	0.753	±	0.015	0.704	±	0.011	0.707	±	0.005
BPI12WComplete	*2*	0.802	±	0.003	0.712	±	0.019	0.663	±	0.025	0.667	±	0.022
	3	0.840	±	0.003	**0.744**	±	**0.013**	**0.726**	±	**0.006**	**0.721**	±	**0.013**
	4	**0.852**	±	**0.005**	0.731	±	0.034	0.709	±	0.026	0.697	±	0.023
Receipt	*2*	0.884	±	0.009	0.499	±	0.035	0.472	±	0.022	0.467	±	0.033
	3	0.919	±	0.011	**0.523**	±	**0.019**	**0.492**	±	**0.031**	**0.494**	±	**0.032**
	4	**0.942**	±	**0.014**	0.516	±	0.045	0.491	±	0.032	0.489	±	0.041
BPI13Incident	*2*	0.688	±	0.004	**0.426**	±	**0.015**	0.372	±	0.014	0.374	±	0.014
	3	0.742	±	0.005	0.398	±	0.026	0.378	±	0.017	0.366	±	0.012
	4	**0.761**	±	**0.005**	0.409	±	0.028	**0.395**	±	**0.031**	**0.380**	±	**0.027**
BPI13Problem	*2*	0.587	±	0.077	0.265	±	0.045	0.309	±	0.059	0.270	±	0.060
	3	0.717	±	0.090	0.344	±	0.124	0.366	±	0.076	0.344	±	0.099
	4	**0.846**	±	**0.010**	**0.474**	±	**0.063**	**0.456**	±	**0.055**	**0.447**	±	**0.059**
BPI17Offer	*2*	0.817	±	0.002	0.558	±	0.001	0.572	±	0.000	0.506	±	0.001
	3	0.838	±	0.002	**0.599**	±	**0.003**	**0.600**	±	**0.000**	**0.516**	±	**0.002**
	4	**0.960**	±	**0.001**	0.435	±	0.001	0.500	±	0.000	0.462	±	0.001
BPI20Request	*2*	0.917	±	0.003	0.523	±	0.038	0.525	±	0.021	0.509	±	0.017
	3	0.984	±	0.001	**0.579**	±	**0.066**	**0.556**	±	**0.030**	**0.550**	±	**0.034**
	4	**0.987**	±	**0.001**	0.575	±	0.026	0.550	±	0.036	0.545	±	0.033

can be naturally split into two sets of traces: those related to incidents (BPI13Incident) and those related to problems (BPI13Problem).

The *BPI17Offer* [84] is also a real event log containing sequences of a loan application process at the same Dutch financial institute as BPI12, but which started in 2016 and handled before 02-02-2017.

The *BPI2020* [85] log collects data from the reimbursement process at TU/e University in the Netherlands. In particular, we exploited the request log, containing all the events involved in requests for payment not related to trips, from 2017 (two departments only) to 2018 (the full TU/e).

We choose these datasets since they come from different application domains, with a different number of traces and events, thus they can be effectively considered to understand if OREO encoding helps the model to generalize well.

5.3. Experimental setting

We compared our method to eight state-of-the-art next activity prediction frameworks ([5,54,58,60], [61], [63,64] and [67]).

Among the LSTM-based approaches, Tax et al. [5] introduced a straightforward yet effective method using one-hot encoding combined with temporal features. While this approach offers clear interpretability and straightforward implementation, its performance tends to decrease as the number of activities grows. Camargo et al. [54] addressed this limitation by developing a more sophisticated approach that pre-trains an embedding network to combine activity and resource information. This results in richer feature representation, though at the cost of a more complex pre-training phase. Evermann et al. [58] took a different approach by implementing an embedding layer for activity encoding, achieving efficient representation of categorical data, but notably omitting temporal aspects in their analysis. Pasquadibisceglie et al. [60] proposed a comprehensive multi-view learning approach that effectively utilizes all available features, though this comes with increased computational demands.

Moving to CNN-based approaches, [63] introduced an innovative approach by transforming activity sequences into image-like representations. This method excels at pattern recognition but is constrained by its fixed input size requirement. Similarly, [64] developed a method using RGB image encoding, offering multi-dimensional feature representation while potentially struggling with long-term dependencies.

The most recent developments in the field include attention-based approaches by Wickramanayake et al. [61] and Bukhsh et al. [67]. In particular, [61] proposes an LSTM with shared attention mechanism that weighs the importance of past activities, while [67] leverages transformer architecture to model complex sequential dependencies. Both approaches demonstrate strong capabilities in capturing complex relationships between activities, though requiring significant computational resources.

Beyond architectural differences, these methods also vary in their considered log perspectives. Indeed, [5,63] and [67] base their prediction on the activities and time perspectives, while [58] neglects time and focuses only on activities and resources. The works presented in [54], [61] and [64] try to reconcile the previous points of view by exploiting activities, time and resource perspectives. Finally, [60] considers all the available features to predict the next activity. Similarly to [5,63] and [67], our work uses both the activity and time perspectives when predicting the next activity in a trace. Table 5 provides a comprehensive comparison of these methods, highlighting their data preprocessing approaches, model architectures, and respective strengths and limitations.

All methods were trained using the best parameters reported in their respective papers when available [5,54,63], through optimization when the published code included it [60,64,67], or with default parameters otherwise [58,61].

To systematically evaluate and compare these different approaches, we performed a 3-fold cross-validation, training each model on two folds and evaluating the next activity prediction using the traces of the remaining fold. In the experiments we record the common macro performance measures in multi-class classification, that is, Accuracy, Precision, Recall, F-score, AUC and AUPRC.

5.4. Effect of the sliding window size

In Tables 6–10 we show the results obtained by our models OREO-LSTM, OREO-Inception, OREO-Image, OREO-Transformer and OREO–SelfAttention, respectively, when varying the dimension of the sliding

Table 10
Results on the benchmark datasets with the OREO-SelfAttention model.

	k	Accuracy			Precision			Recall			Fscore		
BPI12Complete	*2*	0.767	±	0.002	0.734	±	0.009	0.603	±	0.003	0.618	±	0.005
	3	0.807	±	0.001	**0.740**	±	**0.014**	0.616	±	0.008	0.641	±	0.009
	4	**0.842**	±	**0.002**	0.730	±	0.012	**0.620**	±	**0.014**	**0.645**	±	**0.016**
BPI12W	*2*	0.872	±	0.002	0.691	±	0.018	**0.664**	±	**0.005**	**0.648**	±	**0.010**
	3	0.883	±	0.003	**0.697**	±	**0.016**	0.651	±	0.017	0.634	±	0.012
	4	**0.890**	±	**0.003**	0.679	±	0.017	0.653	±	0.011	0.640	±	0.016
BPI12WComplete	*2*	0.750	±	0.007	0.656	±	0.004	0.564	±	0.014	0.581	±	0.011
	3	0.786	±	0.004	0.690	±	0.025	0.655	±	0.020	0.652	±	0.005
	4	**0.817**	±	**0.009**	**0.701**	±	**0.011**	**0.672**	±	**0.029**	**0.658**	±	**0.011**
Receipt	*2*	0.882	±	0.010	0.523	±	0.039	0.476	±	0.029	0.483	±	0.030
	3	0.919	±	0.012	0.519	±	0.024	0.493	±	0.027	0.497	±	0.028
	4	**0.945**	±	**0.014**	**0.538**	±	**0.023**	**0.511**	±	**0.047**	**0.512**	±	**0.043**
BPI13Incident	*2*	0.689	±	0.006	**0.442**	±	**0.027**	0.373	±	0.018	**0.375**	±	**0.019**
	3	0.740	±	0.005	0.426	±	0.039	0.370	±	0.023	0.359	±	0.016
	4	**0.760**	±	**0.004**	0.426	±	0.017	**0.385**	±	**0.038**	0.374	±	0.028
BPI13Problem	*2*	0.687	±	0.014	0.467	±	0.071	0.437	±	0.060	0.433	±	0.063
	3	0.776	±	0.011	**0.479**	±	**0.069**	**0.459**	±	**0.052**	0.451	±	0.049
	4	**0.840**	±	**0.009**	0.467	±	0.041	0.456	±	0.042	**0.452**	±	**0.043**
BPI17Offer	*2*	0.817	±	0.002	0.563	±	0.007	0.572	±	0.000	0.506	±	0.001
	3	0.838	±	0.001	**0.593**	±	**0.018**	**0.601**	±	**0.000**	**0.516**	±	**0.001**
	4	**0.960**	±	**0.001**	0.435	±	0.001	0.500	±	0.000	0.462	±	0.001
BPI20Request	*2*	0.917	±	0.003	0.544	±	0.034	0.519	±	0.023	0.506	±	0.024
	3	0.983	±	0.002	0.558	±	0.014	0.552	±	0.030	0.545	±	0.024
	4	**0.987**	±	**0.001**	**0.595**	±	**0.050**	**0.600**	±	**0.069**	**0.579**	±	**0.047**

Table 11
Comparison of the AUC metric on the benchmark datasets for the OREO models.

	k	OREO LSTM			OREO Inception			OREO Image			OREO Transformer			OREO SelfAttention		
BPI12 Complete	*2*	0.808	±	0.001	0.806	±	0.002	0.783	±	0.000	0.800	±	0.003	0.795	±	0.002
	3	**0.817**	±	**0.002**	0.813	±	0.003	0.802	±	0.002	**0.811**	±	**0.002**	0.803	±	0.004
	4	0.815	±	0.003	**0.819**	±	**0.002**	**0.813**	±	**0.005**	0.803	±	0.006	**0.806**	±	**0.007**
BPI12W	*2*	0.858	±	0.005	0.859	±	0.007	0.857	±	0.005	0.853	±	0.005	**0.828**	±	**0.002**
	3	**0.860**	±	**0.006**	**0.860**	±	**0.003**	**0.858**	±	**0.004**	**0.855**	±	**0.002**	0.822	±	0.009
	4	0.855	±	0.005	0.854	±	0.005	0.823	±	0.042	0.849	±	0.006	0.823	±	0.005
BPI12W Complete	*2*	0.826	±	0.003	0.820	±	0.005	0.838	±	0.001	0.813	±	0.012	0.759	±	0.007
	3	**0.841**	±	**0.009**	0.827	±	0.003	**0.869**	±	**0.010**	**0.849**	±	**0.003**	0.808	±	0.010
	4	0.840	±	0.009	**0.831**	±	**0.013**	0.868	±	0.007	0.841	±	0.013	**0.819**	±	**0.015**
Receipt	*2*	0.767	±	0.005	**0.750**	±	**0.017**	0.739	±	0.017	0.740	±	0.013	0.746	±	0.020
	3	0.766	±	0.003	0.748	±	0.025	0.757	±	0.025	**0.754**	±	**0.016**	0.758	±	0.012
	4	**0.774**	±	**0.010**	0.748	±	0.023	**0.774**	±	**0.040**	0.752	±	0.018	**0.759**	±	**0.029**
BPI13 Incident	*2*	0.670	±	0.007	0.675	±	0.009	0.665	±	0.007	0.672	±	0.007	0.672	±	0.009
	3	0.677	±	0.008	0.680	±	0.007	0.670	±	0.008	0.678	±	0.008	0.673	±	0.012
	4	**0.687**	±	**0.014**	**0.688**	±	**0.015**	**0.683**	±	**0.016**	**0.687**	±	**0.015**	**0.682**	±	**0.018**
BPI13 Problem	*2*	0.691	±	0.027	0.688	±	0.028	0.692	±	0.023	0.615	±	0.037	0.691	±	0.026
	3	0.703	±	0.017	0.706	±	0.025	0.710	±	0.023	0.655	±	0.056	0.711	±	0.023
	4	**0.708**	±	**0.012**	**0.716**	±	**0.022**	**0.718**	±	**0.024**	**0.715**	±	**0.026**	**0.714**	±	**0.019**
BPI17 Offer	*2*	0.770	±	0.000	0.770	±	0.000	0.770	±	0.000	0.770	±	0.000	0.770	±	0.000
	3	**0.782**	±	**0.000**	**0.782**	±	**0.000**	**0.781**	±	**0.000**	**0.781**	±	**0.000**	**0.781**	±	**0.000**
	4	0.744	±	0.000	0.744	±	0.000	0.744	±	0.000	0.744	±	0.000	0.744	±	0.000
BPI20 Request	*2*	0.761	±	0.027	0.762	±	0.028	0.734	±	0.023	0.760	±	0.010	0.756	±	0.011
	3	0.789	±	0.002	0.785	±	0.006	**0.776**	±	**0.007**	**0.778**	±	**0.015**	0.781	±	0.015
	4	**0.819**	±	**0.034**	**0.812**	±	**0.039**	0.772	±	0.087	0.774	±	0.018	**0.799**	±	**0.034**

window k. We emphasize (in bold) the best result obtained for a given evaluation measure (column of the table). In the tables, we show the results for three different sliding window dimensions, i.e., $k \in \{2, 3, 4\}$. This set of candidate values has been selected after a preliminary test to show the sensitivity of our encoding to the sliding window size. We do not show the results for $k \geq 5$, since we noticed a general performance decrease when training models on traces obtained with such a value. Additionally, in Tables 11 and 12 we show the results obtained for the AUC and AUPRC metrics by varying the OREO models.

We start our examination by looking at the accuracy metric (first column). Generally, we can immediately notice that using $k = 4$ our models reach the best accuracy 35 times over 40 attempts. The remaining five best accuracies are reached when $k = 3$. Therefore, we should avoid smaller sliding window dimensions (i.e., $k = 2$) to maximize the accuracy. These results are expected, since small time windows (i.e., $k \leq 2$) ignore positional encoding, while large time windows (i.e., $k \geq 5$) are prone to overfitting.

Nonetheless, accuracy may not be the preferred metric when dealing with highly unbalanced logs, showing rare activities appearing only a few times over the entire log. Since in the performed experiments, we included logs having such characteristics, namely Receipt, BPI13Problem, BPI13Incident and BPI20Request, other measures have to be considered, such as macro precision, macro recall and their harmonic mean indicated by the macro f-measure. From the results on

Table 12
Comparison of the AUPRC metric on the benchmark datasets for the OREO models.

	k	OREO LSTM	OREO Inception	OREO Image	OREO Transformer	OREO SelfAttention
BPI12 Complete	*2*	0.485 ± 0.005	0.487 ± 0.004	0.451 ± 0.002	0.499 ± 0.004	0.491 ± 0.005
	3	0.496 ± 0.012	0.501 ± 0.005	0.493 ± 0.012	0.512 ± 0.002	0.505 ± 0.009
	4	**0.499 ± 0.012**	**0.514 ± 0.009**	**0.522 ± 0.005**	**0.515 ± 0.008**	**0.518 ± 0.009**
BPI12W	*2*	0.634 ± 0.004	0.635 ± 0.010	**0.649 ± 0.007**	0.621 ± 0.005	**0.578 ± 0.005**
	3	**0.638 ± 0.012**	**0.636 ± 0.005**	0.645 ± 0.005	**0.627 ± 0.004**	0.572 ± 0.010
	4	0.622 ± 0.007	0.622 ± 0.007	0.576 ± 0.082	0.615 ± 0.009	0.568 ± 0.005
BPI12W Complete	*2*	0.559 ± 0.004	0.549 ± 0.008	0.595 ± 0.001	0.534 ± 0.016	0.445 ± 0.004
	3	0.585 ± 0.022	0.565 ± 0.003	0.629 ± 0.020	**0.594 ± 0.011**	0.522 ± 0.009
	4	**0.587 ± 0.017**	**0.575 ± 0.013**	**0.636 ± 0.009**	0.586 ± 0.018	**0.542 ± 0.014**
Receipt	*2*	0.478 ± 0.010	**0.457 ± 0.033**	0.412 ± 0.039	0.429 ± 0.043	0.446 ± 0.045
	3	0.481 ± 0.012	0.449 ± 0.051	0.461 ± 0.049	**0.461 ± 0.046**	**0.466 ± 0.040**
	4	**0.496 ± 0.018**	0.452 ± 0.049	**0.496 ± 0.072**	0.449 ± 0.049	0.463 ± 0.060
BPI13 Incident	*2*	**0.308 ± 0.013**	**0.315 ± 0.014**	0.302 ± 0.013	**0.313 ± 0.013**	**0.312 ± 0.016**
	3	0.299 ± 0.009	0.303 ± 0.009	0.291 ± 0.009	0.298 ± 0.009	0.292 ± 0.014
	4	**0.308 ± 0.017**	0.310 ± 0.021	**0.305 ± 0.022**	0.310 ± 0.019	0.300 ± 0.026
BPI13 Problem	*2*	**0.332 ± 0.048**	0.331 ± 0.047	0.333 ± 0.042	0.244 ± 0.046	0.335 ± 0.046
	3	0.322 ± 0.035	**0.333 ± 0.043**	0.339 ± 0.043	0.279 ± 0.046	**0.342 ± 0.042**
	4	0.320 ± 0.024	0.332 ± 0.040	**0.342 ± 0.040**	**0.339 ± 0.041**	0.333 ± 0.032
BPI17 Offer	*2*	0.487 ± 0.001	0.487 ± 0.001	0.487 ± 0.001	0.487 ± 0.001	0.487 ± 0.001
	3	**0.493 ± 0.001**	**0.493 ± 0.001**	**0.493 ± 0.001**	**0.493 ± 0.001**	**0.493 ± 0.001**
	4	0.445 ± 0.001	0.445 ± 0.001	0.445 ± 0.001	0.445 ± 0.001	0.445 ± 0.001
BPI20 Request	*2*	0.492 ± 0.048	0.494 ± 0.047	0.441 ± 0.042	0.485 ± 0.022	0.484 ± 0.023
	3	0.534 ± 0.009	0.532 ± 0.013	**0.511 ± 0.009**	0.516 ± 0.027	0.518 ± 0.019
	4	**0.567 ± 0.043**	**0.564 ± 0.054**	0.499 ± 0.176	**0.531 ± 0.040**	**0.525 ± 0.045**

Table 13
p-values of the signed Wilcoxon rank tests for different analyzed metrics.

		p-Value	Winner
F1 Score	k = 3 vs k = 2	**6.00E−09**	k = 3
	k = 3 vs k = 4	**0.009**	k = 3
Accuracy	k = 3 vs k = 2	**9.09E−13**	k = 3
	k = 3 vs k = 4	0.999	k = 4
AUC	k = 3 vs k = 2	**1.13E−06**	k = 3
	k = 3 vs k = 4	0.804	k = 4

In bold statistically significant values (confidence=0.01).

these measures it appears that for all our models, the sliding window dimension leading to the best F-score is generally $k = 3$.

These results can also be confirmed by looking at the AUPRC and AUC measures (Tables 11 and 12), where the models trained on a sliding window dimension of $k = 3$ are consistently ranked in the top 2 positions.

Additionally, in order to further confirm our evaluation, we performed the one-tail Wilcoxon signed-rank test [86]. The *null hypothesis* for this test is that the two samples are equal, so the tested approaches are equivalent. To accept or reject the null hypothesis, a *p-value* is calculated and compared with a threshold $\alpha = 0.001$. Specifically, we compare the F-score obtained when our models are trained with a sliding window dimension $k = 2$ versus $k = 3$ and $k = 3$ versus $k = 4$, reporting the result in Table 13. The obtained p-values generally confirm our hypothesis, showing that the sliding window dimension that maximizes the F-score is $k = 3$.

5.5. Comparison with state-of-the-art approaches

We compared the performance of our framework against those of eight state-of-the-art predictive process mining techniques based on deep learning. We emphasize that four of our predictive models, OREO-LSTM, OREO-Image, OREO-Inception, and OREO-Transformer, adopt the same predictive architectures from existing state-of-the-art next activity prediction methods, i.e., [60,63,65], and [67], respectively. This approach allows us to directly evaluate how our encoding framework enhances the performance of these existing architectures. The results are shown in Tables 14–17, highlighting the best score for each considered metric in bold. Regarding the OREO models, we only provide the results obtained with $k = 3$, according to the results provided in Section 5.4.

From the results obtained by our competitor systems, we observe that [67] always shows the best result in terms of F-score, while [60] achieves the best accuracy for seven logs over eight. Therefore, we consider [67] and [60] to be the leading competitors. The remaining competitors provide very similar results, depending on the dataset. We can also notice a general performance degradation when unbalanced datasets (like Receipt, BPI13Problem, BPI13Incident and BPI20Request) are considered.

Tables 14 and 15 show that OREO obtains the best performance in term of F-Score on six out of eight tested event logs. On BPI13Incident and BPI17Offer, while OREO achieves the third-highest F-scores behind [60,67], it does so using only activity and time perspectives, resulting in much lighter models. Moreover, looking at the accuracy results, OREO outperforms competitors on seven logs over eight. Another consideration comes from the results on unbalanced datasets where there is a clear indication that the use of sliding windows can mitigate the impact of rare classes on the overall classification. This is probably because the sliding window model, combined with the positional encoding, is able to catch local patterns. The AUC and AUPRC results (Tables 16 and 17) confirm the same considerations discussed before. Indeed, for these measures, the OREO models consistently achieve the best results for five out of eight tested datasets.

Comparing the five OREO variants, the Image-based model achieves the highest average accuracy (~ 0.852), followed closely by LSTM-based and inception-based models (~ 0.850). OREO-Transformer and OREO-SelfAttention show marginally lower average accuracies (~ 0.845 and ~ 0.842 respectively). Looking at average F-scores, OREO-LSTM shows clearer superiority with ~ 0.587, compared to OREO-Inception (~ 0.577) and OREO-Image (~ 0.568). Also in this case, OREO-Transformer and OREO-SelfAttention show lower performance with ~ 0.545 and

Table 14
Comparison between OREO models and state-of-the-art methods described in [5], [54], [58], [60], [61], [63], [64] and [67] for BPI12Complete, BPI12 W, BPI12WComplete and Receipt datasets. The best results are in bold.

	Approach	Accuracy			Precision			Recall			Fscore		
BPI12Complete	OREO-LSTM	0.806	±	0.008	0.700	±	0.015	0.640	±	0.004	0.665	±	0.007
	OREO-Inception	0.811	±	0.002	0.728	±	0.012	**0.655**	±	**0.006**	**0.668**	±	**0.005**
	OREO-Image	0.799	±	0.008	0.729	±	0.015	0.636	±	0.004	0.656	±	0.007
	OREO-Transformer	**0.813**	±	**0.001**	0.730	±	0.015	0.632	±	0.004	0.652	±	0.003
	OREO-SelfAttention	0.807	±	0.001	0.740	±	0.014	0.616	±	0.008	0.641	±	0.009
	[5]	0.791	±	0.001	0.765	±	0.011	0.628	±	0.003	0.645	±	0.006
	[54]	0.768	±	0.002	0.708	±	0.030	0.580	±	0.002	0.578	±	0.003
	[58]	0.720	±	0.002	0.649	±	0.003	0.539	±	0.001	0.549	±	0.000
	[60]	0.795	±	0.003	**0.780**	±	**0.011**	0.629	±	0.005	0.646	±	0.004
	[61]	0.791	±	0.001	0.751	±	0.012	0.631	±	0.008	0.644	±	0.007
	[63]	0.763	±	0.007	0.692	±	0.009	0.602	±	0.008	0.612	±	0.009
	[64]	0.789	±	0.001	0.761	±	0.032	0.625	±	0.004	0.640	±	0.005
	[67]	0.774	±	0.011	0.760	±	0.015	0.652	±	0.010	0.654	±	0.012
BPI12W	OREO-LSTM	0.906	±	0.000	0.776	±	0.012	0.725	±	0.013	0.749	±	0.012
	OREO-Inception	0.903	±	0.001	0.772	±	0.016	**0.735**	±	**0.006**	**0.753**	±	**0.004**
	OREO-Image	**0.916**	±	**0.003**	**0.779**	±	**0.010**	0.721	±	0.008	0.719	±	0.020
	OREO-Transformer	0.903	±	0.003	0.768	±	0.027	0.715	±	0.004	0.716	±	0.011
	OREO-SelfAttention	0.883	±	0.003	0.697	±	0.016	0.651	±	0.017	0.634	±	0.012
	[5]	0.881	±	0.002	0.767	±	0.032	0.686	±	0.009	0.677	±	0.011
	[54]	0.869	±	0.020	0.628	±	0.015	0.610	±	0.017	0.589	±	0.017
	[58]	0.818	±	0.002	0.626	±	0.040	0.602	±	0.002	0.566	±	0.001
	[60]	0.909	±	0.001	0.775	±	0.032	0.693	±	0.015	0.693	±	0.018
	[61]	0.905	±	0.001	0.751	±	0.012	0.702	±	0.006	0.700	±	0.008
	[63]	0.879	±	0.006	0.697	±	0.011	0.657	±	0.010	0.646	±	0.017
	[64]	0.904	±	0.003	0.746	±	0.012	0.684	±	0.006	0.680	±	0.019
	[67]	0.902	±	0.932	0.772	±	0.003	0.732	±	0.003	0.708	±	0.003
BPI12WComplete	OREO-LSTM	0.852	±	0.010	0.750	±	0.030	0.708	±	0.017	0.728	±	0.020
	OREO-Inception	0.830	±	0.003	0.739	±	0.005	0.693	±	0.005	0.701	±	0.007
	OREO-Image	**0.860**	±	**0.002**	0.781	±	0.023	**0.763**	±	**0.020**	**0.763**	±	**0.021**
	OREO-Transformer	0.840	±	0.003	0.744	±	0.013	0.726	±	0.006	0.721	±	0.013
	OREO-SelfAttention	0.786	±	0.004	0.690	±	0.025	0.655	±	0.020	0.652	±	0.005
	[5]	0.764	±	0.001	0.737	±	0.010	0.643	±	0.015	0.653	±	0.010
	[54]	0.780	±	0.006	0.652	±	0.037	0.594	±	0.015	0.583	±	0.008
	[58]	0.711	±	0.005	0.589	±	0.015	0.535	±	0.003	0.528	±	0.001
	[60]	0.842	±	0.004	**0.790**	±	**0.006**	0.684	±	0.011	0.699	±	0.009
	[61]	0.835	±	0.004	0.771	±	0.022	0.690	±	0.007	0.707	±	0.009
	[63]	0.776	±	0.009	0.717	±	0.005	0.603	±	0.013	0.609	±	0.024
	[64]	0.822	±	0.013	0.782	±	0.032	0.653	±	0.018	0.679	±	0.015
	[67]	0.848	±	0.002	0.742	±	0.007	0.746	±	0.002	0.741	±	0.004
Receipt	OREO-LSTM	0.918	±	0.014	0.559	±	0.008	0.511	±	0.008	0.524	±	0.009
	OREO-Inception	0.917	±	0.013	0.539	±	0.029	0.489	±	0.047	0.503	±	0.042
	OREO-Image	0.905	±	0.011	**0.569**	±	**0.045**	**0.531**	±	**0.041**	**0.535**	±	**0.037**
	OREO-Transformer	**0.919**	±	**0.011**	0.523	±	0.019	0.492	±	0.031	0.494	±	0.032
	OREO-SelfAttention	0.919	±	0.012	0.519	±	0.024	0.493	±	0.027	0.497	±	0.028
	[5]	0.854	±	0.009	0.479	±	0.034	0.482	±	0.021	0.467	±	0.037
	[54]	0.841	±	0.013	0.467	±	0.013	0.470	±	0.006	0.460	±	0.009
	[58]	0.812	±	0.009	0.446	±	0.079	0.438	±	0.049	0.429	±	0.062
	[60]	0.864	±	0.013	0.523	±	0.034	0.498	±	0.045	0.490	±	0.044
	[61]	0.858	±	0.010	0.497	±	0.047	0.497	±	0.037	0.491	±	0.044
	[63]	0.782	±	0.082	0.472	±	0.089	0.418	±	0.095	0.424	±	0.103
	[64]	0.848	±	0.021	0.491	±	0.061	0.468	±	0.070	0.471	±	0.065
	[67]	0.576	±	0.576	0.565	±	0.121	0.503	±	0.122	0.521	±	0.123

~ 0.537 respectively. This lower performance can be attributed to their inherent focus on learning complex long-range dependencies. In our case, where the encoding already captures temporal patterns, these attention mechanisms might introduce unnecessary complexity without adding substantial benefits. Moreover, the sliding window approach we adopted naturally limits the sequence length, making simpler architectures like LSTM and CNN more effective at learning the local patterns within these bounded contexts.

Figs. 8 and 9 provide a statistical analysis of the F-score and accuracy results. We conducted a Friedman test to detect statistical differences among the compared approaches, followed by a Nemenyi post-hoc test for pair-wise performance comparisons. The final rankings are shown in these figures. Among the five variants of OREO, LSTM achieves the best ranking for both F-score and accuracy, although no statistically significant differences emerge among the variants. For F-score, while OREO-LSTM shows no statistical difference from [60,61], and [67], it consistently achieves better rankings. This superiority becomes even more evident in accuracy measurements, where all five OREO variants consistently rank in the top positions. It is worth noting that in both scenarios, the remaining competitors [5,54,58,61,63,64] show no statistically significant differences among themselves.

Furthermore, Table 18 presents the average F-score and accuracy rankings of all evaluated approaches, categorized by their underlying architectures. The advantages of using OREO are readily apparent, as models incorporating our proposed encoding outperform those with the same architecture but alternative encodings in 3 out of 4 cases. The performance improvements can be attributed to the core innovation of OREO encoding, which emphasizes recent activities through sliding windows to better manage process dynamics, employs total ordering to

Table 15
Comparison between OREO models and state-of-the-art methods described in [5], [54], [58], [60], [61], [63], [64] and [67] for BPI13Incident, BPI13Problem, BPI17Offer and BPI20Request datasets. The best results are in bold.

	Approach	Accuracy			Precision			Recall			Fscore		
BPI13Incident	OREO-LSTM	0.739	±	0.004	0.448	±	0.021	0.393	±	0.017	0.380	±	0.016
	OREO-Inception	**0.743**	±	**0.005**	0.442	±	0.042	0.395	±	0.015	0.407	±	0.009
	OREO-Image	0.739	±	0.006	0.405	±	0.014	0.363	±	0.027	0.358	±	0.015
	OREO-Transformer	0.742	±	0.005	0.398	±	0.026	0.378	±	0.017	0.366	±	0.012
	OREO-SelfAttention	0.740	±	0.005	0.426	±	0.039	0.370	±	0.023	0.359	±	0.016
	[5]	0.665	±	0.006	0.345	±	0.019	0.270	±	0.001	0.270	±	0.001
	[54]	0.652	±	0.009	0.295	±	0.018	0.260	±	0.013	0.249	±	0.014
	[58]	0.647	±	0.001	0.296	±	0.001	0.249	±	0.001	0.240	±	0.002
	[60]	0.724	±	0.003	0.455	±	0.027	0.423	±	0.007	0.435	±	0.012
	[61]	0.682	±	0.016	0.405	±	0.004	0.383	±	0.012	0.389	±	0.007
	[63]	0.628	±	0.005	0.304	±	0.021	0.291	±	0.047	0.258	±	0.027
	[64]	0.661	±	0.004	0.421	±	0.018	0.332	±	0.017	0.334	±	0.016
	[67]	0.567	±	0.073	**0.553**	±	**0.070**	**0.566**	±	**0.069**	**0.549**	±	**0.069**
BPI13Problem	OREO-LSTM	0.748	±	0.013	0.466	±	0.044	0.446	±	0.041	0.456	±	0.043
	OREO-Inception	0.762	±	0.014	0.476	±	0.042	**0.461**	±	**0.057**	**0.465**	±	**0.060**
	OREO-Image	0.775	±	0.005	0.468	±	0.066	0.457	±	0.052	0.446	±	0.052
	OREO-Transformer	0.717	±	0.090	0.344	±	0.124	0.366	±	0.076	0.344	±	0.099
	OREO-SelfAttention	**0.776**	±	**0.011**	**0.479**	±	**0.069**	0.459	±	0.052	0.451	±	0.049
	[5]	0.529	±	0.013	0.342	±	0.016	0.293	±	0.006	0.286	±	0.003
	[54]	0.505	±	0.007	0.283	±	0.055	0.266	±	0.007	0.238	±	0.014
	[58]	0.582	±	0.004	0.291	±	0.003	0.256	±	0.019	0.235	±	0.001
	[60]	0.621	±	0.011	0.411	±	0.013	0.402	±	0.008	0.405	±	0.009
	[61]	0.609	±	0.007	0.404	±	0.006	0.386	±	0.009	0.392	±	0.008
	[63]	0.502	±	0.005	0.295	±	0.046	0.266	±	0.018	0.246	±	0.021
	[64]	0.595	±	0.006	0.404	±	0.008	0.374	±	0.010	0.381	±	0.011
	[67]	0.428	±	0.022	0.434	±	0.050	0.444	±	0.023	0.415	±	0.030
BPI17Offer	OREO-LSTM	0.839	±	0.002	0.583	±	0.004	0.601	±	0.000	0.582	±	0.001
	OREO-Inception	0.839	±	0.002	0.596	±	0.019	0.601	±	0.000	0.518	±	0.001
	OREO-Image	0.838	±	0.002	0.589	±	0.003	0.601	±	0.000	0.517	±	0.002
	OREO-Transformer	0.838	±	0.002	0.599	±	0.003	0.600	±	0.000	0.516	±	0.002
	OREO-SelfAttention	0.838	±	0.001	0.593	±	0.018	0.601	±	0.000	0.516	±	0.001
	[5]	0.804	±	0.024	0.454	±	0.013	0.563	±	0.014	0.496	±	0.014
	[54]	0.817	±	0.003	0.461	±	0.001	0.571	±	0.000	0.504	±	0.001
	[58]	0.857	±	0.002	0.602	±	0.044	0.624	±	0.001	0.566	±	0.001
	[60]	**0.894**	±	**0.001**	**0.820**	±	**0.010**	0.715	±	0.002	0.721	±	0.002
	[61]	0.814	±	0.002	0.666	±	0.025	0.576	±	0.000	0.530	±	0.008
	[63]	0.817	±	0.002	0.485	±	0.024	0.571	±	0.000	0.505	±	0.001
	[64]	0.819	±	0.003	0.701	±	0.028	0.578	±	0.003	0.529	±	0.011
	[67]	0.793	±	0.002	0.679	±	0.003	**0.806**	±	**0.002**	**0.730**	±	**0.002**
BPI20Request	OREO-LSTM	**0.985**	±	**0.001**	0.622	±	0.019	0.599	±	0.003	**0.608**	±	**0.006**
	OREO-Inception	**0.985**	±	**0.001**	**0.629**	±	**0.014**	0.591	±	0.013	0.603	±	0.015
	OREO-Image	0.984	±	0.002	0.611	±	0.021	0.554	±	0.013	0.548	±	0.010
	OREO-Transformer	0.984	±	0.001	0.579	±	0.066	0.556	±	0.030	0.550	±	0.034
	OREO-SelfAttention	0.983	±	0.002	0.558	±	0.014	0.552	±	0.030	0.545	±	0.024
	[5]	0.856	±	0.004	0.527	±	0.020	0.433	±	0.003	0.428	±	0.002
	[54]	0.857	±	0.004	0.499	±	0.014	0.423	±	0.017	0.425	±	0.017
	[58]	0.879	±	0.003	0.425	±	0.049	0.405	±	0.027	0.390	±	0.026
	[60]	0.882	±	0.004	0.586	±	0.006	0.472	±	0.036	0.491	±	0.040
	[61]	0.858	±	0.002	0.503	±	0.020	0.425	±	0.015	0.418	±	0.015
	[63]	0.855	±	0.003	0.507	±	0.009	0.424	±	0.016	0.418	±	0.014
	[64]	0.856	±	0.004	0.500	±	0.023	0.420	±	0.016	0.411	±	0.013
	[67]	0.914	±	0.004	0.612	±	0.004	**0.604**	±	**0.004**	**0.608**	±	**0.003**

accurately capture activity sequences, and integrates time lag information. Together, these elements create a robust process representation that effectively captures only the relevant information needed for the next activity prediction task.

To further validate our approach, we analyzed the computational aspects of all methods, considering preprocessing time, training time, and model size (number of learnable weights), as shown in Tables 19 and 20. This analysis provides insights into the practical applicability and efficiency of each method. The analysis reveals interesting patterns across different architectures and datasets. In terms of preprocessing time, OREO variants generally show consistent and relatively low preprocessing times (1–90 s), with OREO-Image requiring slightly more preprocessing time than other variants due to an additional step needed to create the imageset using the DeepInsight method [77]. In contrast, some competitors like [64,67] require significantly longer preprocessing (up to 1400 s) to prepare the logs to be effectively exploited by their neural networks for the learning phase.

Regarding training time, OREO variants demonstrate efficient performance. OREO-Inception consistently shows the fastest training times among our variants (5–172 s), followed by OREO-Image (10–148 s) and OREO-LSTM (51–306 s). OREO-Transformer typically requires the longest training time among our variants (81–829 s). Compared to competitors, our methods show substantially faster training times, with some competitors requiring up to 9300 s (e.g., [60,67] on BPI12 W). Thus, from the analysis of preprocessing and training time, it emerges that some of the closest competitors (in terms of F-score and accuracy),

Table 16
Comparison of AUC and AUPRC achieved by OREO models and the state-of-the-art methods described in [5], [54], [58], [60], [61], [63], [64] and [67] for BPI12Complete, BPI12 W, BPI12WComplete and Receipt datasets. The best results are in bold.

	Approach	AUC			AUPRC		
BPI12Complete	OREO-LSTM	**0.817**	±	**0.002**	0.496	±	0.012
	OREO-Inception	0.813	±	0.003	0.501	±	0.005
	OREO-Image	0.802	±	0.002	0.493	±	0.012
	OREO-Transformer	0.811	±	0.002	0.512	±	0.002
	OREO-SelfAttention	0.803	±	0.004	0.505	±	0.009
	[5]	0.809	±	0.001	0.522	±	0.003
	[54]	0.785	±	0.001	0.462	±	0.003
	[58]	0.762	±	0.002	0.410	±	0.004
	[60]	0.810	±	0.002	**0.526**	±	**0.005**
	[61]	0.810	±	0.004	0.518	±	0.006
	[63]	0.795	±	0.004	0.477	±	0.005
	[64]	0.807	±	0.002	0.516	±	0.003
	[67]	0.810	±	0.008	0.491	±	0.003
BPI12W	OREO-LSTM	**0.860**	±	**0.006**	0.638	±	0.012
	OREO-Inception	**0.860**	±	**0.003**	0.636	±	0.005
	OREO-Image	0.858	±	0.004	**0.645**	±	**0.005**
	OREO-Transformer	0.855	±	0.002	0.627	±	0.004
	OREO-SelfAttention	0.822	±	0.009	0.572	±	0.010
	[5]	0.840	±	0.004	0.601	±	0.005
	[54]	0.801	±	0.009	0.533	±	0.027
	[58]	0.794	±	0.001	0.492	±	0.003
	[60]	0.844	±	0.007	0.623	±	0.008
	[61]	0.848	±	0.003	0.622	±	0.003
	[63]	0.825	±	0.005	0.576	±	0.007
	[64]	0.838	±	0.006	0.611	±	0.004
	[67]	0.834	±	0.044	0.639	±	0.008
BPI12WComplete	OREO-LSTM	0.841	±	0.009	0.585	±	0.022
	OREO-Inception	0.827	±	0.003	0.565	±	0.003
	OREO-Image	**0.869**	±	**0.010**	**0.629**	±	**0.020**
	OREO-Transformer	0.849	±	0.003	0.594	±	0.011
	OREO-SelfAttention	0.808	±	0.010	0.522	±	0.009
	[5]	0.799	±	0.005	0.534	±	0.008
	[54]	0.771	±	0.008	0.488	±	0.008
	[58]	0.740	±	0.001	0.425	±	0.002
	[60]	0.824	±	0.005	0.586	±	0.002
	[61]	0.826	±	0.004	0.582	±	0.013
	[63]	0.775	±	0.007	0.498	±	0.013
	[64]	0.812	±	0.003	0.570	±	0.010
	[67]	0.861	±	0.010	0.619	±	0.003
Receipt	OREO-LSTM	**0.766**	±	**0.003**	**0.481**	±	**0.012**
	OREO-Inception	0.748	±	0.025	0.449	±	0.051
	OREO-Image	0.757	±	0.025	0.461	±	0.049
	OREO-Transformer	0.754	±	0.016	0.461	±	0.046
	OREO-SelfAttention	0.758	±	0.012	0.466	±	0.040
	[5]	0.748	±	0.012	0.425	±	0.056
	[54]	0.738	±	0.007	0.414	±	0.028
	[58]	0.720	±	0.015	0.375	±	0.044
	[60]	0.754	±	0.029	0.444	±	0.075
	[61]	0.756	±	0.020	0.452	±	0.047
	[63]	0.703	±	0.050	0.367	±	0.112
	[64]	0.730	±	0.036	0.431	±	0.083
	[67]	0.756	±	0.035	0.435	±	0.014

Table 17
Comparison of AUC and AUPRC achieved by OREO models and state-of-the-art methods described in [5], [54], [58], [60], [61], [63], [64] and [67] for BPI13Incident, BPI13Problem, BPI17Offer and BPI20Request datasets. The best results are in bold.

	Approach	AUC			AUPRC		
BPI13Incident	OREO-LSTM	0.677	±	0.008	0.299	±	0.009
	OREO-Inception	0.680	±	0.007	**0.303**	±	**0.009**
	OREO-Image	0.670	±	0.008	0.291	±	0.009
	OREO-Transformer	0.678	±	0.008	0.298	±	0.009
	OREO-SelfAttention	0.673	±	0.012	0.292	±	0.014
	[5]	0.625	±	0.006	0.221	±	0.012
	[54]	0.612	±	0.007	0.208	±	0.010
	[58]	0.601	±	0.006	0.196	±	0.011
	[60]	0.698	±	0.003	0.297	±	0.019
	[61]	0.676	±	0.006	0.262	±	0.010
	[63]	0.625	±	0.020	0.211	±	0.017
	[64]	0.648	±	0.008	0.238	±	0.011
	[67]	**0.706**	±	**0.019**	0.301	±	0.056
BPI13Problem	OREO-LSTM	0.703	±	0.017	0.322	±	0.035
	OREO-Inception	0.706	±	0.025	0.333	±	0.043
	OREO-Image	0.710	±	0.023	0.339	±	0.043
	OREO-Transformer	0.655	±	0.056	0.279	±	0.046
	OREO-SelfAttention	**0.711**	±	**0.023**	**0.342**	±	**0.042**
	[5]	0.607	±	0.008	0.236	±	0.012
	[54]	0.584	±	0.004	0.210	±	0.004
	[58]	0.592	±	0.004	0.210	±	0.004
	[60]	0.665	±	0.005	0.296	±	0.009
	[61]	0.656	±	0.005	0.286	±	0.007
	[63]	0.584	±	0.011	0.208	±	0.010
	[64]	0.648	±	0.006	0.278	±	0.007
	[67]	0.691	±	0.031	0.338	±	0.036
BPI17Offer	OREO-LSTM	0.782	±	0.000	0.493	±	0.001
	OREO-Inception	0.782	±	0.000	0.493	±	0.001
	OREO-Image	0.781	±	0.000	0.493	±	0.001
	OREO-Transformer	0.781	±	0.000	0.493	±	0.001
	OREO-SelfAttention	0.781	±	0.000	0.493	±	0.001
	[5]	0.765	±	0.009	0.478	±	0.015
	[54]	0.770	±	0.000	0.487	±	0.001
	[58]	0.802	±	0.000	0.545	±	0.001
	[60]	**0.849**	±	**0.002**	**0.632**	±	**0.002**
	[61]	0.772	±	0.000	0.490	±	0.002
	[63]	0.770	±	0.000	0.487	±	0.001
	[64]	0.774	±	0.001	0.493	±	0.004
	[67]	0.504	±	0.003	0.480	±	0.001
BPI20Request	OREO-LSTM	**0.789**	±	**0.002**	**0.534**	±	**0.009**
	OREO-Inception	0.785	±	0.006	0.532	±	0.013
	OREO-Image	0.776	±	0.007	0.511	±	0.009
	OREO-Transformer	0.778	±	0.015	0.516	±	0.027
	OREO-SelfAttention	0.781	±	0.015	0.518	±	0.019
	[5]	0.711	±	0.001	0.404	±	0.003
	[54]	0.706	±	0.017	0.396	±	0.015
	[58]	0.698	±	0.013	0.374	±	0.029
	[60]	0.738	±	0.009	0.434	±	0.014
	[61]	0.707	±	0.007	0.397	±	0.016
	[63]	0.706	±	0.008	0.395	±	0.013
	[64]	0.704	±	0.007	0.393	±	0.013
	[67]	0.724	±	0.011	0.525	±	0.756

like [60,67], require a higher computational complexity to achieve their results. As further confirmed by the statistical analysis shown in Fig. 10, OREO variants consistently ranked among the fastest methods. While some approaches like [58,63] show comparable total times, it is important to note that they achieve significantly lower predictive performance, making our methods more efficient in terms of the accuracy-time trade-off. We motivate this behavior with the core idea of OREO of combining the sliding window mechanism with the positional encoding that allows the method to represent only relevant information for the inference task.

These conclusions are further supported by analyzing the number of learnable weights, which generally indicates model complexity and directly correlates with memory consumption during both training and inference. In this sense, OREO variants maintain relatively compact models. OREO-Image consistently shows the smallest model size (17K–295K parameters), while OREO-SelfAttention typically has the largest (451K–515K parameters). These sizes are generally competitive with or smaller than most baselines, particularly compared to methods like [64] which can reach up to 14M parameters.

6. Conclusions

This paper proposes a novel method for next activity prediction from event logs, called OREO, which performs event log encoding, based on sliding windows. It exploits relative activity position to re-align the partial traces and extract useful information on traces and

Table 18
Comparison of the average F-score and precision rank for the proposed model versus competitors, categorized by model architecture.

Architecture	Approach	Avg Fscore rank	Avg accuracy rank
LSTM	OREO-LSTM	**2.563**	**3.188**
	[5]	9.500	9.875
	[54]	11.500	10.438
	[58]	11.750	10.000
	[60]	5.938	4.750
CNN	OREO-Image	4.125	**3.500**
	OREO-Inception	**3.250**	3.563
	[63]	11.063	11.563
	[64]	8.750	8.438
Transformer	OREO-Transformer	5.625	**3.813**
	[67]	**2.813**	9.875
Attention-based	OREO-SelfAttention	**6.563**	**4.688**
	[61]	7.563	7.313

Table 19
Preprocessing and training times (in seconds) and the number of weights for each evaluated model on the BPI12Complete, BPI12 W, BPI12WComplete, and Receipt datasets.

	Approach	Preprocessing time	Training time	Weights
BPI12Complete	OREO-LSTM	25.44	202.36	171K
	OREO-Inception	25.44	**130.25**	68K
	OREO-Image	89.66	142.07	38K
	OREO-Transformer	25.44	581.72	357K
	OREO-SelfAttention	25.44	182.90	502K
	[5]	25.69	1745.70	216K
	[54]	276.45	4493.54	329K
	[58]	**23.01**	163.83	10K
	[60]	34.55	7715.91	118K
	[61]	32.14	1965.73	132K
	[63]	52.86	181.03	626K
	[64]	827.63	982.56	6M
	[67]	839.41	7177.86	34K
BPI12W	OREO-LSTM	25.60	241.29	37K
	OREO-Inception	25.60	149.00	63K
	OREO-Image	75.28	195.89	17K
	OREO-Transformer	25.60	819.17	276K
	OREO-SelfAttention	25.60	166.05	493K
	[5]	34.61	1898.50	214K
	[54]	282.38	4232.03	329K
	[58]	23.23	235.17	10K
	[60]	36.58	9309.19	72K
	[61]	32.89	2745.14	119K
	[63]	**10.55**	**59.04**	650K
	[64]	654.24	687.48	2,1M
	[67]	930.62	7667.45	36K
BPI12WComplete	OREO-LSTM	10.75	118.61	101K
	OREO-Inception	10.75	79.65	28K
	OREO-Image	16.92	65.16	29K
	OREO-Transformer	10.75	405.33	44K
	OREO-SelfAttention	10.75	77.83	451K
	[5]	**6.64**	719.39	207K
	[54]	183.24	1778.82	327K
	[58]	10.66	393.59	9K
	[60]	14.58	1541.54	83K
	[61]	13.16	356.50	105K
	[63]	11.38	**59.07**	40K
	[64]	85.89	148.76	902K
	[67]	126.50	2757.88	32K
Receipt	OREO-LSTM	1.40	54.88	43K
	OREO-Inception	1.40	**9.58**	93K
	OREO-Image	6.85	11.75	21K
	OREO-Transformer	1.40	103.28	481K
	OREO-SelfAttention	1.40	41.19	515K
	[5]	**0.59**	214.45	217K
	[54]	55.14	538.90	329K
	[58]	1.83	338.30	10K
	[60]	2.68	295.56	99K
	[61]	1.56	134.32	115K
	[63]	1.53	9.61	588K
	[64]	51.02	50.87	7,4M
	[67]	3.89	402.21	32K

Table 20
Preprocessing and training times (in seconds) and the number of weights for each evaluated model on the BPI13Incident, BPI13Problem, BPI17Offer, and BPI20Request datasets.

	Approach	Preprocessing time	Training time	Weights
BPI13Incident	OREO-LSTM	10.31	83.36	137K
	OREO-Inception	10.31	38.57	68K
	OREO-Image	20.55	**35.02**	18K
	OREO-Transformer	10.31	263.02	138K
	OREO-SelfAttention	10.31	86.35	472K
	[5]	**7.02**	1229.75	210K
	[54]	295.01	2461.71	327K
	[58]	15.55	309.02	9K
	[60]	25.74	1414.45	2M
	[61]	12.09	372.87	3.5M
	[63]	11.64	84.89	583K
	[64]	1418.74	1014.59	14M
	[67]	103.72	2790.06	34K
BPI13Problem	OREO-LSTM	1.57	51.47	182K
	OREO-Inception	1.57	**5.63**	71K
	OREO-Image	2.67	10.80	29K
	OREO-Transformer	1.57	81.12	55K
	OREO-SelfAttention	1.57	39.70	454K
	[5]	**0.37**	303.80	208K
	[54]	40.96	473.97	327K
	[58]	1.49	181.34	9K
	[60]	3.01	132.22	535K
	[61]	1.70	71.63	1M
	[63]	1.13	8.73	565K
	[64]	64.21	52.16	12M
	[67]	3.44	372.00	31K
BPI17Offer	OREO-LSTM	32.94	306.56	254K
	OREO-Inception	32.94	172.96	415K
	OREO-Image	46.67	**148.57**	39K
	OREO-Transformer	32.94	829.64	44K
	OREO-SelfAttention	32.94	322.93	451K
	[5]	**8.98**	6443.18	208K
	[54]	427.92	8629.07	327K
	[58]	40.39	220.69	9K
	[60]	65.14	2581.88	117K
	[61]	33.57	718.10	202K
	[63]	32.67	174.74	34K
	[64]	750.05	681.28	1,4M
	[67]	1180.13	8150.20	30K
BPI20Request	OREO-LSTM	6.16	58.02	181K
	OREO-Inception	6.16	**20.57**	456K
	OREO-Image	15.31	33.16	295K
	OREO-Transformer	6.16	182.29	259K
	OREO-SelfAttention	6.16	57.04	491K
	[5]	**2.19**	707.25	213K
	[54]	61.68	1070.26	329K
	[58]	6.84	176.75	10K
	[60]	10.12	640.25	364K
	[61]	6.69	432.02	61K
	[63]	5.92	28.78	586K
	[64]	39.00	266.86	2,1M
	[67]	35.84	1612.15	32K

temporal patterns among the activities. OREO is designed to work with different deep learning architectures, based on LSTMs, CNNs, and attention-based models.

We evaluated the proposed method on several real-world process mining datasets for the next activity prediction of running traces. The (statistical) comparison of OREO performances, in all its five variants, with those achieved by eight other competitors, show the superiority of our method. This confirms our initial intuition that the encoding which (i) adapts to sliding windows and (ii) is able to model the position of the activities within the sliding windows, is beneficial.

The method's versatility makes it particularly suitable for various industrial applications. In manufacturing, OREO can enhance production planning by predicting next operations and potential bottlenecks. In healthcare, it can support patient flow management by anticipating next treatment steps and resource requirements. In financial services, it can improve customer service by predicting next client interactions and preparing appropriate responses. Furthermore, In IoT-enabled environments, it can process real-time sensor data to predict maintenance needs and optimize resource utilization. The sliding window approach is particularly valuable in these contexts as it allows for real-time adaptation to changing process patterns.

Despite these promising results, we acknowledge certain limitations of our approach. The sliding window size needs to be carefully tuned based on the specific process characteristics, as it affects both prediction accuracy and computational efficiency. Additionally, the method's performance might be affected when dealing with highly irregular processes where the temporal patterns are less evident. The current implementation also assumes that all activities within the sliding window have equal importance, which might not always reflect real-world scenarios.

To possibly address these limitations, as future work, we plan to implement an adaptive window size mechanism that automatically

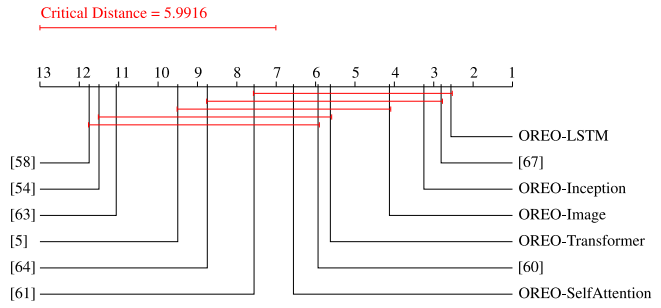


Fig. 8. Comparison of the F-score achieved by the OREO models and their competitors through the Nemenyi test. Approaches that are not significantly different are connected by a red line.

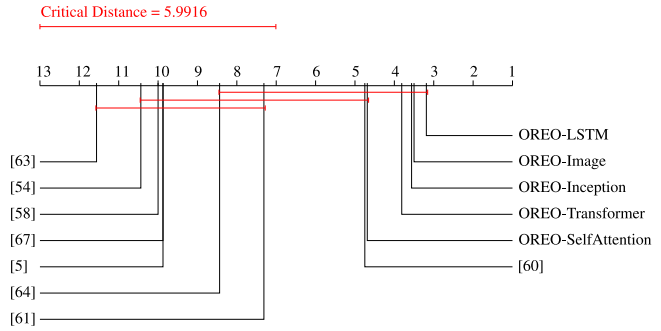


Fig. 9. Comparison of the accuracy achieved by the OREO models and their competitors through the Nemenyi test. Approaches that are not significantly different are connected by a red line.

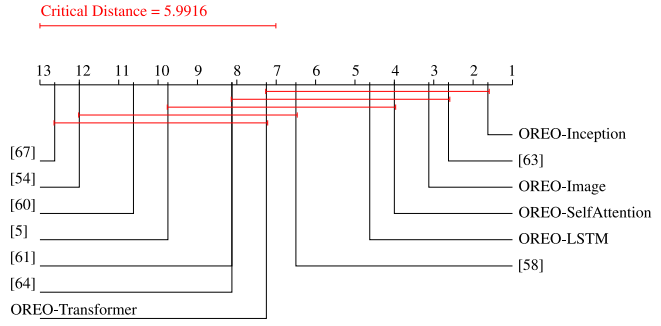


Fig. 10. Comparison of the total running time (preprocessing + training) of the OREO variants and competitors. Models on the right side have lower running times. According to the Nemenyi test, the approaches that are not significantly different are connected by a red line.

determines the size on the basis of process characteristics, possibly combining multiple window sizes for more robust predictions. Additionally, we will investigate enhanced temporal pattern detection techniques to better handle irregular processes, making OREO more effective across various real-world scenarios.

Furthermore, while this paper demonstrates the effectiveness of OREO for next activity prediction, its position-aware encoding approach shows promise for other predictive process monitoring tasks. For trace completion time prediction, the relative positions encoded by OREO could help capture temporal dependencies that influence completion times, particularly in processes with parallel activities or varying execution speeds. For outcome prediction, the method's ability to capture activity patterns within windows could be leveraged to identify sequence signatures that correlate with specific outcomes. The sliding window mechanism could be particularly valuable for early outcome prediction, as it naturally handles partial traces. Additionally, the method could be extended to predict multiple activities ahead or even entire trace suffixes, where the position encoding could help maintain sequential consistency in the predictions. These extensions would further expand OREO's practical applications across different domains while maintaining its core advantages in position-aware pattern recognition.

CRediT authorship contribution statement

Antonio Pellicani: Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Michelangelo Ceci:** Writing – review & editing, Validation, Supervision, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the project FAIR - Future AI Research (PE00000013), Spoke 6 - Symbiotic AI, under the NRRP MUR program funded by the NextGenerationEU. The research presented in this paper was also partially supported by the Italian MUR, through the project "MAD – La metamorfosi Additiva del Design", Grant No. ARS01_00717 (MAD – The Additive Metamorphosis of Design) for the activities of WP3 & WP5.

Data availability

Data will be made available on request.

References

[1] M. Dumas, W.M. van der Aalst, A.H. ter Hofstede, Process-Aware Information Systems: Bridging People and Software Through Process Technology, John Wiley & Sons, Inc., New York, NY, USA, 2005, http://dx.doi.org/10.1002/0471741442.

[2] W.M.P. van der Aalst, Process Mining: Data Science in Action, second ed., Springer, Heidelberg, 2016, http://dx.doi.org/10.1007/978-3-662-49851-4.

[3] W. van der Aalst, A. Adriansyah, A.K.A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J.C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, et al., Process mining manifesto, in: International Conference on Business Process Management, Springer, 2011, pp. 169–194, http://dx.doi.org/10.1007/978-3-642-28108-2_19.

[4] W.M. van der Aalst, Decision support based on process mining, in: Handbook on Decision Support Systems 1: Basic Themes, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 637–657, http://dx.doi.org/10.1007/978-3-540-48713-5_29.

[5] N. Tax, I. Verenich, M. La Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: International Conference on Advanced Information Systems Engineering, Springer, 2017, pp. 477–492, http://dx.doi.org/10.1007/978-3-319-59536-8_30.

[6] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Analyzing business process anomalies using autoencoders, Mach. Learn. 107 (11) (2018) 1875–1893, http://dx.doi.org/10.1007/s10994-018-5702-8.

[7] R.S. Oyamada, G.M. Tavares, S.B. Junior, P. Ceravolo, Enhancing predictive process monitoring with time-related feature engineering, in: International Conference on Advanced Information Systems Engineering, Springer, 2024, pp. 71–86, http://dx.doi.org/10.1007/978-3-031-61057-8_5.

[8] S. Barbon Junior, P. Ceravolo, E. Damiani, G. Marques Tavares, Evaluating trace encoding methods in process mining, in: International Symposium: From Data to Models and Back, Springer, 2020, pp. 174–189, http://dx.doi.org/10.1007/978-3-030-70650-0_11.

[9] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, F.M. Maggi, Complex symbolic sequence encodings for predictive monitoring of business processes, in: International Conference on Business Process Management, Springer, 2016, pp. 297–313, http://dx.doi.org/10.1007/978-3-319-23063-4_21.

[10] T.K. Ho, M. Basu, Complexity measures of supervised classification problems, IEEE Trans. Pattern Anal. Mach. Intell. 24 (3) (2002) 289–300, http://dx.doi.org/10.1109/34.990132.

[11] J. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) http://dx.doi.org/10.1145/2523813.

[12] S. Lund, J. Manyika, S. Nyquist, L. Mendonca, S. Ramaswamy, Game Changers: Five Opportunities for US Growth and Renewal, McKinsey & Company, 2013.

[13] C. Di Francescomarino, C. Ghidini, Predictive process monitoring, in: Process Mining Handbook, Springer International Publishing Cham, 2022, pp. 320–346, http://dx.doi.org/10.1007/978-3-031-08848-3_10.

[14] P. Ceravolo, M. Comuzzi, J. De Weerdt, C. Di Francescomarino, F.M. Maggi, Predictive process monitoring: concepts, challenges, and future research directions, Process. Sci. 1 (1) (2024) 1–22, http://dx.doi.org/10.1007/s44311-024-00002-4.

[15] A. Corallo, M. Lazoi, F. Striani, Process mining and industrial applications: A systematic literature review, Knowl. Process. Manag. 27 (3) (2020) 225–233, http://dx.doi.org/10.1002/kpm.1630.

[16] F.M. Maggi, C. Di Francescomarino, M. Dumas, C. Ghidini, Predictive monitoring of business processes, in: International Conference on Advanced Information Systems Engineering, Springer, 2014, pp. 457–472, http://dx.doi.org/10.1007/978-3-319-07881-6_31.

[17] C. Di Francescomarino, C. Ghidini, F.M. Maggi, F. Milani, Predictive process monitoring methods: Which one suits me best? in: M. Weske, M. Montali, I. Weber, J. vom Brocke (Eds.), Business Process Management, Springer International Publishing, Cham, 2018, pp. 462–479, http://dx.doi.org/10.1007/978-3-319-98648-7_27.

[18] G. Park, M. Song, Prediction-based resource allocation using LSTM and minimum cost and maximum flow algorithm, in: 2019 International Conference on Process Mining, ICPM, IEEE, 2019, pp. 121–128, http://dx.doi.org/10.1109/ICPM.2019.00027.

[19] S. Weinzierl, S. Zilker, M. Stierle, M. Matzner, G. Park, From predictive to prescriptive process monitoring: Recommending the next best actions instead of calculating the next most likely events, in: Wirtschaftsinformatik (Zentrale Tracks), 2020, pp. 364–368, http://dx.doi.org/10.30844/wi_2020_c12-weinzierl.

[20] C. Di Francescomarino, C. Ghidini, F.M. Maggi, G. Petrucci, A. Yeshchenko, An eye into the future: leveraging a-priori knowledge in predictive business process monitoring, in: International Conference on Business Process Management, Springer, 2017, pp. 252–268, http://dx.doi.org/10.1007/978-3-319-65000-5_15.

[21] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Binet: Multi-perspective business process anomaly classification, Inf. Syst. 103 (2022) 101458, http://dx.doi.org/10.1016/j.is.2019.101458.

[22] R. Conforti, M. de Leoni, M. La Rosa, W.M. van der Aalst, A.H. ter Hofstede, A recommendation system for predicting risks across multiple business process instances, Decis. Support Syst. 69 (2015) 1–19, http://dx.doi.org/10.1016/j.dss.2014.10.006.

[23] A. Pika, W.M. van der Aalst, C.J. Fidge, A.H. Ter Hofstede, M.T. Wynn, Profiling event logs to configure risk indicators for process delays, in: International Conference on Advanced Information Systems Engineering, Springer, 2013, pp. 465–481, http://dx.doi.org/10.1007/978-3-642-38709-8_30.

[24] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, F. Leymann, Runtime prediction of service level agreement violations for composite services, in: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, Springer, 2009, pp. 176–186, http://dx.doi.org/10.1007/978-3-642-16132-2_17.

[25] L. Zeng, C. Lingenfelder, H. Lei, H. Chang, Event-driven quality of service prediction, in: International Conference on Service-Oriented Computing, Springer, 2008, pp. 147–161, http://dx.doi.org/10.1007/978-3-540-89652-4_14.

[26] A.E. Márquez-Chamorro, M. Resinas, A. Ruiz-Cortés, M. Toro, Run-time prediction of business process indicators using evolutionary decision rules, Expert Syst. Appl. 87 (2017) 1–14, http://dx.doi.org/10.1016/j.eswa.2017.05.069.

[27] H. Nguyen, M. Dumas, M. La Rosa, F.M. Maggi, S. Suriadi, Mining business process deviance: a quest for accuracy, in: OTM Confederated International Conferences" on the Move to Meaningful Internet Systems", Springer, 2014, pp. 436–445, http://dx.doi.org/10.1007/978-3-662-45563-0_25.

[28] T.B. Hong Tu, M. Song, Analysis and prediction cost of manufacturing process based on process mining, in: 2016 International Conference on Industrial Engineering, Management Science and Application, ICIMSA, 2016, pp. 1–5, http://dx.doi.org/10.1109/ICIMSA.2016.7503993.

[29] C. Di Francescomarino, M. Dumas, F.M. Maggi, I. Teinemaa, Clustering-based predictive process monitoring, IEEE Trans. Serv. Comput. 12 (6) (2019) 896–909, http://dx.doi.org/10.1109/TSC.2016.2645153.

[30] M. Weske, Business process management architectures, in: Business Process Management: Concepts, Languages, Architectures, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 305–343, http://dx.doi.org/10.1007/978-3-642-28616-2_7.

[31] A.C. Choueiri, D.M.V. Sato, E.E. Scalabrin, E.A.P. Santos, An extended model for remaining time prediction in manufacturing systems using process mining, J. Manuf. Syst. 56 (2020) 188–201, http://dx.doi.org/10.1016/j.jmsy.2020.06.003.

[32] W.M. van der Aalst, M.H. Schonenberg, M. Song, Time prediction based on process mining, Inf. Syst. 36 (2) (2011) 450–475, http://dx.doi.org/10.1016/j.is.2010.09.001.

[33] M. Polato, A. Sperduti, A. Burattin, M. de Leoni, Data-aware remaining time prediction of business process instances, in: 2014 International Joint Conference on Neural Networks, IJCNN, IEEE, 2014, pp. 816–823, http://dx.doi.org/10.1109/IJCNN.2014.6889360.

[34] M. Ceci, P.F. Lanotte, F. Fumarola, D.P. Cavallo, D. Malerba, Completion time and next activity prediction of processes using sequential pattern mining, in: International Conference on Discovery Science, Springer, 2014, pp. 49–61, http://dx.doi.org/10.1007/978-3-319-11812-3_5.

[35] 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, http://dx.doi.org/10.1016/j.is.2015.07.003.

[36] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, F.M. Maggi, Intra and inter-case features in predictive process monitoring: A tale of two dimensions, in: International Conference on Business Process Management, Springer, 2017, pp. 306–323, http://dx.doi.org/10.1007/978-3-319-65000-5_18.

[37] I. Verenich, H. Nguyen, M. La Rosa, M. Dumas, White-box prediction of process performance indicators via flow analysis, in: Proceedings of the 2017 International Conference on Software and System Process, 2017, pp. 85–94, http://dx.doi.org/10.1145/3084100.3084110.

[38] G.M. Tavares, R.S. Oyamada, S.B. Junior, P. Ceravolo, Trace encoding in process mining: A survey and benchmarking, Eng. Appl. Artif. Intell. 126 (2023) 107028, http://dx.doi.org/10.1016/j.engappai.2023.107028.

[39] J. Evermann, J.-R. Rehse, P. Fettke, A deep learning approach for predicting process behaviour at runtime, in: Business Process Management Workshops: BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers 14, Springer, 2017, pp. 327–338, http://dx.doi.org/10.1007/978-3-319-58457-7_24.

[40] P. De Koninck, S. vanden Broucke, J. De Weerdt, act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes, in: Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16, Springer, 2018, pp. 305–321, http://dx.doi.org/10.1007/978-3-319-98648-7_18.

[41] J. Kim, M. Comuzzi, M. Dumas, F.M. Maggi, I. Teinemaa, Encoding resource experience for predictive process monitoring, Decis. Support Syst. 153 (2022) 113669, http://dx.doi.org/10.1016/j.dss.2021.113669.

[42] W. Rizzi, C. Di Francescomarino, C. Ghidini, F.M. Maggi, How do I update my model? On the resilience of Predictive Process Monitoring models to change, Knowl. Inf. Syst. 64 (5) (2022) 1385–1416, http://dx.doi.org/10.1007/s10115-022-01666-9.

[43] S. Pauwels, T. Calders, Incremental predictive process monitoring: The next activity case, in: International Conference on Business Process Management, Springer, 2021, pp. 123–140, http://dx.doi.org/10.1007/978-3-030-85469-0_10.

[44] P. Ceravolo, G.M. Tavares, S.B. Junior, E. Damiani, Evaluation goals for online process mining: a concept drift perspective, IEEE Trans. Serv. Comput. 15 (4) (2020) 2473–2489, http://dx.doi.org/10.1109/TSC.2020.3004532.

[45] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, Darwin: An online deep learning approach to handle concept drifts in predictive process monitoring, Eng. Appl. Artif. Intell. 123 (2023) 106461, http://dx.doi.org/10.1016/j.engappai.2023.106461.

[46] D. Breuker, M. Matzner, P. Delfmann, J. Becker, Comprehensible predictive models for business processes, MIS Q. 40 (4) (2016) 1009–1034, http://dx.doi.org/10.25300/MISQ/2016/40.4.10.

[47] J. Becker, D. Breuker, P. Delfmann, M. Matzner, Designing and implementing a framework for event-based predictive modelling of business processes, Enterp. Model. Inf. Syst. Architectures- EMISA 2014 (2014).

[48] M. Le, B. Gabrys, D. Nauck, A hybrid model for business process event prediction, in: International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, 2012, pp. 179–192, http://dx.doi.org/10.1007/978-1-4471-4739-8_13.

[49] G.T. Lakshmanan, D. Shamsi, Y.N. Doganata, M. Unuvar, R. Khalaf, A markov prediction model for data-driven semi-structured business processes, Knowl. Inf. Syst. 42 (1) (2015) 97–126, http://dx.doi.org/10.1007/s10115-013-0697-8.

[50] A. Rozinat, W.M. van der Aalst, Decision mining in ProM, in: International Conference on Business Process Management, Springer, 2006, pp. 420–425, http://dx.doi.org/10.1007/11841760_33.

[51] M. Ceci, M. Spagnoletta, P.F. Lanotte, D. Malerba, Distributed learning of process models for next activity prediction, in: Proceedings of the 22nd International Database Engineering & Applications Symposium, IDEAS 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 278–282, http://dx.doi.org/10.1145/3216122.3216125.

[52] M. Ceci, A. Impedovo, A. Pellicani, Leveraging multi-target regression for predicting the next parallel activities in event logs, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2020, pp. 237–248, http://dx.doi.org/10.1007/978-3-030-65965-3_15.

[53] S. Ferilli, S. Angelastro, Activity prediction in process mining using the WoMan framework, J. Intell. Inf. Syst. 53 (1) (2019) 93–112, http://dx.doi.org/10.1007/s10844-019-00543-2.

[54] M. Camargo, M. Dumas, O. González-Rojas, Learning accurate LSTM models of business processes, in: International Conference on Business Process Management, Springer, 2019, pp. 286–302, http://dx.doi.org/10.1007/978-3-030-26619-6_19.

[55] E. Rama-Maneiro, J.C. Vidal, M. Lama, Deep learning for predictive business process monitoring: Review and benchmark, IEEE Trans. Serv. Comput. 16 (1) (2021) 739–756, http://dx.doi.org/10.1109/TSC.2021.3139807.
[56] A. Nguyen, S. Chatterjee, S. Weinzierl, L. Schwinn, M. Matzner, B. Eskofier, Time matters: Time-aware lstms for predictive business process monitoring, in: Process Mining Workshops: ICPM 2020 International Workshops, Padua, Italy, October 5–8, 2020, Revised Selected Papers 2, Springer, 2021, pp. 112–123, http://dx.doi.org/10.1007/978-3-030-72693-5_9.
[57] J. De Smedt, J. De Weerdt, Predictive process model monitoring using long short-term memory networks, Eng. Appl. Artif. Intell. 133 (2024) 108295, http://dx.doi.org/10.1016/j.engappai.2024.108295.
[58] J. Evermann, J.-R. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decis. Support Syst. 100 (2017) 129–140, http://dx.doi.org/10.1016/j.dss.2017.04.003.
[59] L. Lin, L. Wen, J. Wang, Mm-pred: A deep predictive model for multi-attribute event sequence, in: Proceedings of the 2019 SIAM International Conference on Data Mining, SIAM, 2019, pp. 118–126, http://dx.doi.org/10.1137/1.9781611975673.14.
[60] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, A multi-view deep learning approach for predictive business process monitoring, IEEE Trans. Serv. Comput. (2021) http://dx.doi.org/10.1109/TSC.2021.3051771.
[61] B. Wickramanayake, Z. He, C. Ouyang, C. Moreira, Y. Xu, R. Sindhgatta, Building interpretable models for business process prediction using shared and specialised attention mechanisms, Knowl.-Based Syst. 248 (2022) 108773, http://dx.doi.org/10.1016/j.knosys.2022.108773.
[62] N. Mehdiyev, J. Evermann, P. Fettke, A novel business process prediction model using a deep learning method, Bus. Inf. Syst. Eng. 62 (2) (2020) 143–157, http://dx.doi.org/10.1007/s12599-018-0551-3.
[63] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, Using convolutional neural networks for predictive process analytics, in: 2019 International Conference on Process Mining, ICPM, 2019, pp. 129–136, http://dx.doi.org/10.1109/ICPM.2019.00028.
[64] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, Predictive process mining meets computer vision, in: BPM, 2020, http://dx.doi.org/10.1007/978-3-030-58638-6_11.
[65] N. Di Mauro, A. Appice, T.M. Basile, Activity prediction of business process instances with inception cnn models, in: International Conference of the Italian Association for Artificial Intelligence, Springer, 2019, pp. 348–361, http://dx.doi.org/10.1007/978-3-030-35166-3_25.
[66] F. Taymouri, M.L. Rosa, S. Erfani, Z.D. Bozorgi, I. Verenich, Predictive business process monitoring via generative adversarial nets: the case of next event prediction, in: Business Process Management: 18th International Conference, BPM 2020, Seville, Spain, September 13–18, 2020, Proceedings 18, Springer, 2020, pp. 237–256, http://dx.doi.org/10.1007/978-3-030-58666-9_14.
[67] Z.A. Bukhsh, A. Saeed, R.M. Dijkman, Processtransformer: Predictive business process monitoring with transformer network, 2021, http://dx.doi.org/10.48550/arXiv.2104.00721, arXiv preprint arXiv:2104.00721.
[68] A. Chiorrini, C. Diamantini, A. Mircoli, D. Potena, Exploiting instance graphs and graph neural networks for next activity prediction, in: International Conference on Process Mining, Springer, 2021, pp. 115–126, http://dx.doi.org/10.1007/978-3-030-98581-3_9.
[69] N. Harane, S. Rathi, Comprehensive survey on deep learning approaches in predictive business process monitoring, Mod. Approaches Mach. Learn. Cogn. Sci.: Walkthrough (2020) 115–128, http://dx.doi.org/10.1007/978-3-030-38445-6_9.
[70] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Comput. 9 (8) (1997) 1735–1780, http://dx.doi.org/10.1162/neco.1997.9.8.1735.
[71] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, Handb. Brain Theory Neural Netw. 3361 (10) (1995) 1995.
[72] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, D. Yu, Convolutional neural networks for speech recognition, IEEE ACM Trans. Audio Speech Lang. Process. 22 (10) (2014) 1533–1545, http://dx.doi.org/10.1109/TASLP.2014.2339736.
[73] B. Zhao, H. Lu, S. Chen, J. Liu, D. Wu, Convolutional neural networks for time series classification, J. Syst. Eng. Electron. 28 (1) (2017) 162–169, http://dx.doi.org/10.21629/JSEE.2017.01.18.
[74] V. Pasquadibisceglie, A. Appice, G. Castellano, D. Malerba, G. Modugno, ORANGE: outcome-oriented predictive process monitoring based on image encoding and CNNs, IEEE Access 8 (2020) 184073–184086, http://dx.doi.org/10.1109/ACCESS.2020.3029323.
[75] D. Scherer, A. Müller, S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in: International Conference on Artificial Neural Networks, Springer, 2010, pp. 92–101, http://dx.doi.org/10.1007/978-3-642-15825-4_10.
[76] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9, http://dx.doi.org/10.1109/CVPR.2015.7298594.
[77] A. Sharma, E. Vans, D. Shigemizu, K.A. Boroevich, T. Tsunoda, DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture, Sci. Rep. 9 (1) (2019) 1–7, http://dx.doi.org/10.1038/s41598-019-47765-6.
[78] P. Philipp, R. Jacob, S. Robert, J. Beyerer, Predictive analysis of business processes using neural networks with attention mechanism, in: 2020 International Conference on Artificial Intelligence in Information and Communication, ICAIIC, 2020, pp. 225–230, http://dx.doi.org/10.1109/ICAIIC48513.2020.9065057.
[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), in: Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.
[80] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D.D. Cox, Hyperopt: a python library for model selection and hyperparameter optimization, Comput. Sci. Discov. 8 (1) (2015) 014008, http://dx.doi.org/10.1088/1749-4699/8/1/014008.
[81] B. van Dongen, BPI Challenge 2012, Eindhoven University of Technology, 2012, http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
[82] J. Buijs, Environmental Permit Application Process ('WABO'), CoSeLoG Project, Eindhoven University of Technology, 2014, http://dx.doi.org/10.4121/uuid:c399c768-d995-4086-adda-c0bc72ad02bc.
[83] W. Steeman, BPI challenge 2013, 2013, http://dx.doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07.
[84] B. van Dongen, BPI challenge 2017 - offer log, 2017, http://dx.doi.org/10.4121/12705737.v2.
[85] B. van Dongen, BPI challenge 2020, 2020, http://dx.doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.
[86] F. Wilcoxon, Individual comparisons by ranking methods, Biom. Bull. 1 (1945) 80–83, http://dx.doi.org/10.1007/978-1-4612-4380-9_16.