DARWIN: An Online Deep Learning Approach to handle Concept Drifts in Predictive Process Monitoring

Vincenzo Pasquadibisceglie[a,b,*], Annalisa Appice[a,b], Giovanna Castellano[a,b], Donato Malerba[a,b]

[a] *Department of Computer Science, University of Bari Aldo Moro, Bari, Italy*
[b] *Consorzio Interuniversitario Nazionale per l'Informatica - CINI, Bari, Italy*

Abstract

[1]Predictive process monitoring (PPM) is a specific task under the umbrella of Process Mining that aims to predict several factors of a business process (e.g., next activity prediction) based on the knowledge learned from historical event logs. Despite recent PPM algorithms have gained predictive accuracy using deep learning, they commonly perform an offline analysis of event data assuming that logged processes remain in a steady state over time. However, this is often not the real-world case due to concept drifts. The main goal of this work is to solve the next-activity prediction problem under dynamic conditions of business data streams. To this aim, we propose DARWIN as a novel PPM method that detects concept drifts and adapts a deep neural model to concept drifts. A deep empirical analysis of different factors that may influence the performance of DARWIN in streaming scenarios is provided. Experiments with various benchmark event streams show the effectiveness of the proposed approach.

Keywords: Predictive Process Monitoring, Next-activity Prediction, Online Learning, Event Stream Mining, Concept Drift Detection, Deep Learning, Fine

*Corresponding author
Email addresses: vincenzo.pasquadibisceglie@uniba.it (Vincenzo Pasquadibisceglie), annalisa.appice@uniba.it (Annalisa Appice), giovanna,castellano@uniba.it (Giovanna Castellano), donato.malerba@uniba.it (Donato Malerba)

Tuning.

1. Introduction

It is becoming increasingly evident that companies generate huge amounts of event data that need to be properly processed by efficient data analytics methods, in order to support predictions and decisions. Event data produced by company business processes typically come in the form of business data streams that can be highly non-stationary since they very often evolve, gradually or abruptly, due to the changing of markets, products, and behavior of customers. Handling business data streams is the goal of data stream mining and presents several challenges related to their non-stationary and big data nature along with the peculiar properties of volume, velocity and volatility [19, 44, 65].

In this paper we cast data stream mining in the realm of Predictive Process Monitoring (PPM). This is a specific task under the umbrella of Process Mining that employs machine learning methods to predict future factors of an ongoing (uncompleted) business process execution (e.g., next activity prediction) based on the knowledge learned from historical event logs. Hence the goal of this work is to face the above-mentioned challenges of data stream mining in the context of PPM scenarios.

Since the volume of a business data stream can increase continuously to potentially an infinite amount of data, data stream mining approaches commonly assume that samples are used a few times (or just once) to create the predictive model, which should not require storing the whole data stream in memory. This demands for PPM algorithms that are capable of performing online learning with limited memory constraints [17].

Velocity impacts the mining process, preventing the use of any off-line or time-consuming procedure due to the fact that data arrive quickly and continuously, and latency of the response is a critical factor influencing the performance of data stream mining methods. Despite the high demand for real-time performance in a variety of data stream scenarios, including PPM, few existing models

show convincing fast performance in real-time. Conversely, most PPM methods imply a high latency in learning and adapting the predictive model, hence they need to use an obsolete model until the new one is ready.

Volatility is related to the evolving nature of business data streams that inevitably results in the change of patterns and in the appearance of the so-called concept drift problem [34, 47, 72], meaning that the statistical properties of the target variable change over time in unexpected ways. Under concept drift, the conventional two phases of machine learning methods, i.e., training/learning and prediction, need to be extended so as to cope with the following challenges:

- drift detection: establishing when, where and how a concept begins to drift by identifying change points or change time intervals [8];
- drift adaptation: updating the existing model so as to account for the detected drift. This can be done by simple retraining or model adjusting.

As an attempt to give answers to all the above challenges in the realm of PPM in this paper we propose DARWIN (DynAmic pRedictive process monitoring with ADaptive WINdows) that adapts, in near-real time, a deep neural model learned for next-activity prediction to the evolving environment. The use of a deep learning approach is motivated by the ability of deep neural networks to cope with business process data through the automatic extraction of hierarchical multilevel features via representational learning [61]. However, deep neural models are traditionally trained in a batch setting, requiring the entire training data to be made available prior to the learning task. Scaling deep learning to online PPM scenarios, where new event data arrive sequentially and continuously in a stream [63], is not an easy task. Fine tuning of a deep neural model is a simple and effective few-shot learning method that can be used to adapt a deep neural model to the new data distribution without the need to restart training from the whole set of accumulated traces [18].

The main goal of this work is to investigate the effectiveness of deep learning models to solve the next-activity prediction problem under the dynamic conditions of business data streams that require adaptation of the predictive model.

To this aim, we propose DARWIN as a novel PPM method for this task. The proposed method uses the fine tuning mechanism to adapt the weights of a previously learned deep neural model to fit the drifting conditions of a business process. The updates of the model are facilitated through the ADaptive WINdow (**ADWIN**) mechanism [10]. **ADWIN** is a concept drift detection strategy broadly used in stream classification to monitor the error rate of a classification model [2]. In particular, we use **ADWIN** to monitor the conformance of the upcoming activities to their predictions. **ADWIN** maintains an adaptive window of the newest streamed events with the window size that is dynamically increased when no change occurs and shrunk when a change happens in context. In this way, the deep neural model is periodically updated via a **Fine-tuning** process based on the windowed event data and alerted by the concept drift.

In summary, the following are the key contributions of this work. First, we design and develop a novel method for online next activity prediction capable to adapt the predictive model to concept drifts. Second, we perform an extensive study of different factors that may influence the performance of the proposed adaptive deep learning strategy in streaming scenarios. Specifically, the study is focused on investigating the effectiveness of deep neural models for next activity prediction in accordance to the learning strategy (**Fine-tuning**) and the encoding strategy (**Word2Vec**). As a further side contribution, an extensive evaluation is provided to compare the performance of DARWIN to well-established data stream methods. We conducted thorough experiments by resorting to the Prequential evaluation schema [33] and present an analysis of the performance of DARWIN, in terms of predictive accuracy and computational time. The empirical study shows the ability of our approach to improve performances compared to learning baselines taken from the recent literature on online PPM.

The paper is organized as follows. Section 2 overviews the related work. Section 3 reports preliminary concepts. Section 4 describes the proposed DARWIN method, while Section 5 provides a motivating example of this method. Section 6 discusses the benchmark data collections considered for the empirical

evaluation, the experimental setting and the relevant results. Section 7 discusses limitations of the presented research. Finally, Section 8 draws conclusions and sketches the future work.

2. Related work

PPM covers a range of techniques capable of analyzing events produced by a business process to predict the future state of new process instances (traces) [27]. In recent years, researchers have developed several PPM algorithms for various domains (e.g. healthcare [40], smart home environments [29]), varying for the type of target variable to be predicted. For example, the output is continuous in regression problems such as predicting the remaining cycle time [70], while the output is categorical in classification problems such as predicting the outcome [43, 56, 69] or the next-activity [12, 28, 68] of a running trace. This paper focuses on a classification approach as it investigates the task of the next-activity prediction.

A significant amount of the past PPM literature on next-activity prediction uses explicit representations of process models (state-transition models, hidden Markov models or probabilistic finite automatons) to predict the next-activity (e.g., [12, 45]). Several PPM studies also leverage machine learning to annotate explicit representations of process models and enhance the predictive accuracy of the selected model representations (e.g., [6, 28, 31, 60]). However, the recent PPM literature has mainly focused on training various deep neural models (e.g., [16, 23, 54, 55, 68]) that gain significant accuracy when learning next-activity predictive functions in an implicit manner dispensing with prior assumptions about the models.

Despite deep learning has greatly contributed to achieve recent amazing results in terms of accurate next-activity prediction, a major limit of most deep learning-based PPM approaches, is that they perform an offline analysis of event logs assuming that logged processes remain in a steady state over time. However, due to changing circumstances (concept drifts) process executions commonly

evolve over time and PPM models need to adapt online accordingly.

The importance of dealing with concept drifts in process management was first asserted in the Process Mining manifesto [1]. Thenceforth the topic of concept drift is investigated in various studies of the process mining literature [14]. In fact, several studies explore the use of hypothesis tests for concept drift detection in event logs. For example, both [13] and [52] use hypothesis tests on sliding windows with user-defined sizes, in order to identify if a significant difference occurred between a feature set extracted on events recorded on consecutive windows. [7] explores achievements of cluster analysis in detecting concept drifts occurring in streamed traces. [41] uses **ADWIN** method to detect concept drifts in streams of control-flow frequencies for frequent item mining. Notably all the previous studies focus on detecting and visualizing concept drift points as a means to perform conformance checking.

On the other hand, [15] investigates several stream data mining approaches (e.g., Sliding Window, Lossy Counting and Lossy Counting with Budget) to cope with concept drifts and realize online the discovery of a declarative process model. Instead, [71] explores the online discovery of procedural process models dealing with large amounts of event data using finite memory. In this study, the online procedural process discovery is done continuously updating a procedural process model as new events are recorded without explicitly alerting concept drift events. We note that our study works in a different perspective as it explicitly detects concept drifts in a PPM task.

In the field of PPM, that is the main field of this study, a few recent studies have started the investigation of online learning of predictive models over time. The study of [58] is the closest to our work in the PPM literature. It compares various online learning strategies to update predictive process models trained for next-activity prediction. Such models, like simple neural networks or dynamic Bayesian networks, are updated with **Re-training** or **Fine-tuning** on full data or windowed data. In [58] concept drifts are detected with the approach based on hypothesis tests described in [13]. This approach assumes that the size of windows, that convey event data for detecting concept drifts and performing on-

line learning, is user-defined and does not change over time. Differently, we use **ADWIN** to perform concept drift detection and adaptation from event data streams recorded in dynamic windows whose size may be adapted automatically during the stream. Another difference is that [58] uses the **One-Hot-Encoding** technique, to encode the categorical information ignoring the trace context of the information to encode and demanding for knowing in advance the domain of all distinct categories that may up-come over the stream. Specifically, **One-Hot-Encoding** requires knowing apriori all the activities that will be observed in the stream, in order to set the size of the encoding vector equal to the number of distinct activities. In fact, this assumption is actually done in [58] to set the **One-Hot-Encoding** size. However, this means that **One-Hot-Encoding** can be actually used in event streams with no novelties (new activities) appearing during the stream, since all the distinct activities must be known apriori to set the **One-Hot-Encoding** size. This is not reasonable in real streams. Instead, we use **Word2Vec** in place of **One-Hot-Encoding**, which is a context-aware deep encoding technique that, in the version integrated in Gensim 4.3.0 (see Section 6.1), can be updated as new activities appear over time (or out-of date activities disappear definitely). Final considerations concern the evaluation study described in [58]. This was conducted ignoring the latency of the online learning stage and assuming that a new predictive model is always ready in the stream as a new event is recorded in a running trace and the new next-activity must be predicted. We safely overcome this issue, keeping online the old predictive model until the new model, learned after a concept drift, is ready for the next prediction.

Further related studies investigate issues of online learning for the outcome prediction task. [49] explores the performance of existing online learners that explicitly handle concept drift (e.g., adaptive Hoeffding tree and adaptive Hoeffding option tree). [62] analyses the performance of various online learning strategies adopted for online learning of Random Forests. Both these studies do not consider deep neural models. Both [49] and [62] address the outcome prediction problem. In addition, [62] does not integrate any concept drift detection

Table 1: List of symbols

Symbol	Meaning
$\mathcal{E}$	Event universe
σ	Trace
e	Event $e = (\sigma, A, t)$
A	Activity
$\perp$	Completion activity
Σ	event stream
$\pi_{\mathcal{S}}(e)$	Trace σ recorded in event e
$\pi_{\mathcal{A}}(e)$	Activity A recorded in event e
$\pi_{\mathcal{T}}(e)$	Timestamp t recorded in e
$observed(\sigma, \Sigma)$	Sequence of activities of σ already observed in Σ
$observe(\Sigma)$	Last event observed in Σ
$\Sigma(t)$	t-th event observed in Σ
$\mathbf{D}$	Data synopsis to record events observed in Σ
$\mathbf{A}$	Data synopsis to record next activities predicted for incomplete traces recorded in $\mathbf{D}$
$\mathbf{D}[\sigma]$	Prefix trace of σ recorded in $\mathbf{D}$
$\mathcal{L}$	Multiset of all labeled prefix sequences extracted from traces recorded in $\mathbf{D}$
$H_{F,\Theta}$	Next activity classificatio hypothesis function
$\mathbf{A}[\sigma]$	Prediction produced with $H_{F,\Theta}(\mathbf{D}[\sigma])$ and recorded in $\mathbf{A}$
$CE()$	Classification error function
$\mathbf{W}$	Adaptive sliding window of **ADWIN**
$\Pi_{LP}(\mathbf{W})$	Multiset of labelled prefix trace sequences recorded in $\mathbf{W}$
$\Pi_{CE}(\mathbf{W})$	Multiset of classification errors recorded in $\mathbf{W}$

mechanism causing a burden of computation that, in absence of concept drifts, may lead to repeatedly learn roughly similar predictive functions.

3. Background and terminology

In this section we introduce terminology referred to events, traces, prefix traces, event streams, next-activity prediction functions, window model and concept drift.

3.1. Event stream

Given a business process, a trace describes the life-cycle of a particular case (i.e., a process instance) in terms of the sequence of activities executed. According to this definition of a trace, an event refers to just an activity executed within a trace at a specific time. Let $\mathcal{A}$ be the set of all activity names, $\mathcal{S}$ be

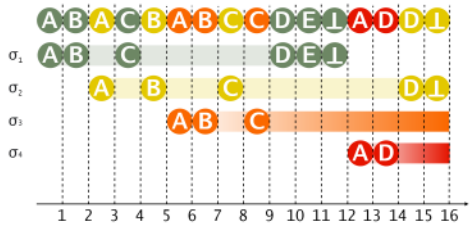


Figure 1: An example of an event stream

the set of all trace identifiers and $\mathcal{T}$ be the set of all timestamps. Let $\perp$ be the name of the completion activity (i.e., the activity that denotes the completion of a case execution). Let us consider that the completion of a trace is commonly declared in a multitude of processes, e.g., ticket resolution in helpdesk processes.

Definition 1 (Event). *Given the event universe $\mathcal{E} = \mathcal{S} \times \mathcal{A} \times \mathcal{T}$, an event $e \in \mathcal{E}$ is a triple $e = (\sigma, A, t)$ that represents the occurrence of activity A in trace σ at timestamp t.*

Let us introduce the functions: 1) $\pi_{\mathcal{S}} \colon \mathcal{E} \mapsto \mathcal{S}$ such that $\pi_{\mathcal{S}}(e) = \sigma$; 2) $\pi_{\mathcal{A}} \colon \mathcal{E} \mapsto \mathcal{A}$ such that $\pi_{\mathcal{A}}(e) = A$; 3) $\pi_{\mathcal{T}} \colon \mathcal{E} \mapsto \mathcal{T}$ such that $\pi_{\mathcal{T}}(e) = t$.

Definition 2 (Trace). *Let $\mathcal{A}^*$ denote the set of all possible sequences on $\mathcal{A}$. A trace σ is a sequence $\sigma = \langle A_1, A_2, \ldots, A_n \rangle$ so that: (1) $\forall i = 1, \ldots, n, \exists e_i \in \mathcal{E}$ such that $\pi_{\mathcal{S}}(e_i) = \sigma$ and $\pi_{\mathcal{A}}(e_i) = A_i$ and (2) $\forall i = 1, \ldots, n-1, \pi_{\mathcal{T}}(e_i) \leq \pi_{\mathcal{T}}(e_{i+1})$.*

$\sigma_1 \cdot \sigma_2$ is the concatenation of two sequences. $\langle\rangle$ is the empty sequence. Given a trace $\sigma = \langle A_1, A_2, \ldots, A_n \rangle$, $|\sigma| = n$ denotes the length of σ. Trace σ is complete if $A_n = \perp$. $prefix(\sigma, h) = \langle A_1, A_2, \ldots, A_h \rangle$ with $1 \leq h < n$ is the prefix of the sequence consisting of the first h elements. The selection $\sigma(h+1) = A_{h+1}$ corresponds to the next activity of $prefix(\sigma, h)$, i.e., $next(\sigma, h) = \sigma(h+1)$. For example, let us consider $\sigma = \langle A, B, C, D, E, F, \perp \rangle$, σ is complete with $|\sigma| = 7$. $prefix(\sigma, 2) = \langle A, B \rangle$. The next activity of $prefix(\sigma, 2)$ is C, while the next activity of $prefix(\sigma, 6)$ is $\perp$.

Definition 3 (Event stream). *An event stream* Σ *is an infinite sequence of observable events* $\mathbb{N}^+ \mapsto \mathcal{E}$*, i.e.,* $\Sigma = e_1, e_2, \ldots, e_t, \ldots$ *so that* $\forall t \in \mathbb{N}^+$*,* $e_t \in \mathcal{E}$ *and* $\pi_{\mathcal{T}}(e_t) \leq \pi_{\mathcal{T}}(e_{t+1})$.

Given a stream Σ, the selection $\Sigma(t) = e_t$ denotes the t-event recorded in Σ, the operator $observe(\Sigma) \in \mathcal{E}$ returns the latest observable event recorded in Σ, and the operator $size(\Sigma) \in \mathbb{N}^+$ returns the number of events available on the stream. Hence, $observe(\Sigma)$ returns the selection $\Sigma(size(\Sigma))$. For example, in Figure 1, $\Sigma(1) = (\sigma_1, A, 1)$, while $observe(\Sigma) = (\sigma_2, \perp, 16)$ and $size(\Sigma) = 16$.

Definition 4 (Observed trace). *Let us consider an event stream* Σ *and a trace* $\sigma = \langle A_1, A_2, \ldots, A_n \rangle$*.* $observed(\sigma, \Sigma)$ *denotes all the first activities of* σ *already recorded in* Σ*, that is,* $observed(\sigma, \Sigma) = \langle A_1, A_2, \ldots, A_h \rangle$ *so that* $\exists t_1, t_2, \ldots, t_h \in \mathbb{N}^+$ *with* $1 \leq t_1 < t_2 < \ldots < t_h \leq size(\Sigma)$ *and* $\forall i = 1, \ldots, h, \pi_{\mathcal{S}}(\Sigma(t_i))) = \sigma$ *and* $\pi_{\mathcal{A}}(\Sigma(t_i)) = A_i$.

We note that $observed(\sigma, \Sigma)$ is complete (started and ended in Σ) iff the completion activity $\perp$ is recorded as last activity in $observed(\sigma, \Sigma)$; incomplete (running trace still in execution in the stream) otherwise. Once a trace is completed in a stream, no further event can be executed in this trace and recorded in Σ. Assuming that each event is recorded in the stream when it happens in the real world,[2] the observed traces will be incomplete most of the time.

Traces observed in an event stream Σ can be recorded into a data synopsis $\mathbf{D}$ composed of an header table $\mathcal{H}$ and a prefix tree $\mathcal{T}$. This data synopsis is introduced in [53] to reduce the amount of memory spent to record multiple incomplete traces observed for the same variant in an event stream. $\mathcal{H}$ is a Hash table, where each cell represents the identifier σ of a running trace. $\mathcal{H}[\sigma]$ contains the link to a node of $\mathcal{T}$ so that the tree path from the root to the pointed node records the activity trace of σ. $\mathcal{T}$ is a tree that consists of one root labeled

[2] In this paper, we neglect the case there is a delay between the event execution and the event recording in the event stream.

as *null* (denoted as *root*) and a set of prefix activity sub-trees as the children of *root*. Details of the tree structure definition, data storage and memory usage are reported in [53]. In the follow-up, we assume that $\mathbf{D}[\sigma]$ denotes the current sequence $observed(\sigma, \Sigma)$ actually recorded in $\mathbf{D}$.

At the beginning of a stream, $\mathbf{D}$ is empty. Let $observe(\Sigma)$ consume an event e with $\pi_{\mathcal{S}}(e) = \sigma$ and $\pi_{\mathcal{A}}(e) = A$, the entry $\mathbf{D}[\sigma]$ is updated according to the rules described in [53] to record the current execution of $observed(\sigma, \Sigma)$, that is here denoted as $\mathbf{D}[\sigma] \cdot \langle A \rangle$. Figures 3a-3b show the data synopsis $\mathbf{D}$ recording the event stream Σ shown in Figure 1. It records σ_1 and σ_2 that have been already completed in Σ, and σ_3 and σ_4 that are still incomplete in Σ. Traces that have been completed in the stream may be removed from $\mathbf{D}$ once they have been processed. This allows us to account for limited memory when recording an event stream by maintaining $\mathbf{D}$ updated with current running traces only.

Definition 5 (Labeled prefix sequence multiset). *Let Σ be an event stream whose observed traces are recorded in $\mathbf{D}$, $\mathcal{L} \in \mathcal{B}(A^* \times A)$ is the multiset of all the prefix sequences (samples) extracted from the observed traces recorded in $\mathbf{D}$. Each prefix sequence is labeled with the next activity (labels) associated to each prefix sequence in the corresponding observed trace so that $L = [prefix(\mathbf{D}[\sigma], h), \mathbf{D}[\sigma](h+1) | \sigma \in \mathcal{S} \wedge \exists \mathbf{D}[\sigma] \neq \langle\rangle \wedge 1 < h \leq |\mathbf{D}[\sigma]|]$.*

Table 2 reports the example of the labeled prefix sequence multiset extracted from the data synopsis $\mathbf{D}$ reported in Figure 3a and associated to the event stream Σ shown in Figure 1.

3.2. Next-activity prediction

The next-activity prediction is a PPM task often addressed as a multi-class classification problem by resorting to machine learning techniques.

Definition 6 (Next-activity classification hypothesis function). *Let F be a model with m real-valued parameters and $\Theta \in \mathbb{R}^m$ be a m-sized vector of real-valued parameters. A hypothesis $H_{F,\Theta}$ of the model F is a function: $H_{F,\Theta} \colon \mathcal{A}^* \mapsto \mathcal{A}$ such that $H_{F,\Theta}(x) \approx F(x, \Theta)$.*

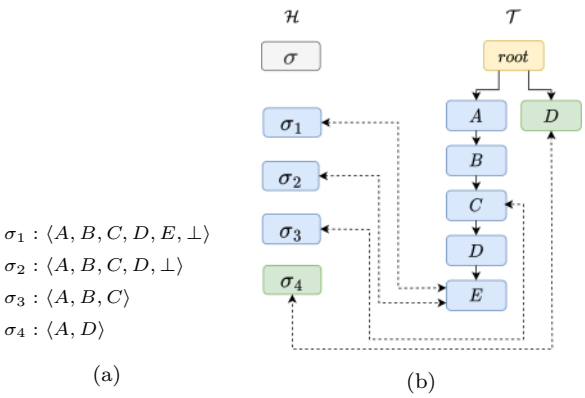


Figure 2: Data synopsis to record the event stream

Definition 7 (Next-activity classification model). *A next-activity classification model F is defined as $F\colon \mathcal{A}^* \times \mathbb{R}^m \mapsto \mathcal{A}$ with m the number of real-valued parameters that are learned in F to train a hypothesis function $H_{F,\Theta}$.*

Definition 8 (Next activity prediction). *Let us consider $H_{F,\Theta}\colon \mathcal{A}^* \mapsto \mathcal{A}$, and $observed(\sigma, \Sigma)$ an incomplete trace observed in Σ and recorded in $\mathbf{D}$. $H_{F,\Theta}$ $(observed(\sigma, \Sigma))$ predicts the expected next activity of $observed(\sigma, \Sigma)$.*

Let us consider the hash table $\mathbf{A}$ to record the next activities predicted for the incomplete observed traces recorded in $\mathbf{D}$. Each tabular entry $\mathbf{A}[\sigma]$ records $H_{F,\Theta}(\mathbf{D}[\sigma])$, where $\mathbf{D}[\sigma]$ records $observed(\sigma, \Sigma)$ that is currently incomplete in Σ. Note that the check of the correctness of the prediction $H_{F,\Theta}(observed(\sigma, \Sigma))$ must be delayed to the next update of $\mathbf{D}[\sigma]$ in Σ. Figure 3 shows an example of delay occurring in the examination of the correctness of a next-activity pre-

Table 2: Multiset of prefix sequences labeled with the next activity and extracted from the observed traces recorded in the event stream shown in Figure 1 . Each superscript number denotes the number of times the distinct variant-labeled prefix trace appears in the log stream.

$(prefix\ trace, next\ activity\ label)^{frequency}$
$(\langle A\rangle, B)^3$
$(\langle A\rangle, D)^1$
$(\langle A, B\rangle, C)^3$
$(\langle A, B, C\rangle, D)^2$
$(\langle A, B, C, D\rangle, E)^1$
$(\langle A, B, C, D\rangle, \perp)^1$
$(\langle A, B, C, D, E\rangle, \perp)^1$

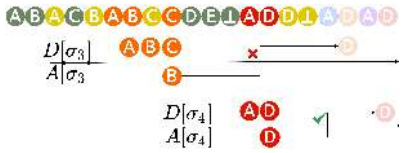


Figure 3: An example of next-activity predictions recorded in **A** and delay to examine the correctness of such predictions. Shadowed nodes represent events to be observed in the future.

diction recorded in **A**.

3.3. Concept drift analysis

The definition of concept drift is not strict in literature. Informally the concept drift is defined as "the event when a set of examples has legitimate class labels at one time and has different legitimate labels at another time" [22]. A few studies distinguish between concept drifts and novelties arguing that concept drifts occur in existing classes, while novelties concern new classes that can appear over time [35]. In this study, we define concept drifts within the next activity prediction context by including the novelties under the concept drift umbrella. Specifically, we define concept drifts according the data stream classification theory reported in [32].

Definition 9 (Concept drift). *A concept drift is a change in the joint probability $P(\mathbf{x}, y)$ where $\mathbf{x}$ is the prefix trace description of an example and y its next activity class.*

According to Definition 9, the concept drift phenomenon is the most commonly reflected in its consequences - a deteriorating overall prediction performance in $H_{F,\Theta}$. The learned knowledge of the classification hypothesis becomes obsolete in the face of the prefix traces with a changed next activity concept. Finding the exact points of change can be very challenging, as the change between two distributions can be gradual and has to be differentiated from transient noise that can affect the stream. In general, data stream theory differentiates between sudden and gradual drifts based on the speed of change [32]. In addition, a concept drift may also be recurrent (when behavior changes repeat with recurring incidence over time). In this study, our main goal is to keep updated the classification hypothesis $H_{F,\Theta}$ by avoiding a significant decrease in its performance. We do not handle the task of recognizing the drift type, which would require specific techniques for each drift type [48].

State-of-the-art data stream approaches to handle concept drifts mainly distinguish between (1) adaptive and evolving learners and (2) adapting training sets [22]. The former family covers the most prominent group of algorithms (commonly tree-based algorithms) that are designed for adaptive incremental learning. These algorithms handle drifts by adapting, removing, or re-combining model individual components. The latter family is used to adapting to changes by applying window strategies that can either monitor performance indicators or blindly maintain the window size. Notably, window strategies are commonly model-agnostic. So they can be easily adapted to any base learner (comprising deep neural models). In this study, we use an adaptive window strategy [14] that copes with infinite streams storing only data related to the most recent events and, periodically, analyzing them. The size of the window is adapted over time to the characteristics of the data. We note that adaptive windows are commonly used in evolving ensemble learners, which represent one of the major mainstream of recent research in data streams [37]. Ensembles handle drifts by replacing some members (or part of members) of the ensemble with new models trained on different windows. Data stream studies generally use Hoeffding trees as base learners of ensembles, while this study explores performances of

strategies to fit deep learning for next activity prediction to event streams. The study of ensemble learning in combination with deep learning in event streams is postponed to future work.

Let $\mathbf{W}$ be a sliding window populated with data triples (prefix trace, executed activity, and classification error) related to the latest events recorded in Σ. Let us consider $observe(\Sigma) = e$ with $\pi_{\mathcal{S}}(e) = \sigma$ and $\pi_{\mathcal{A}}(e) = A$. Before updating both $\mathbf{D}$ and $\mathbf{A}$ with information enclosed in e (i.e., before performing, in the order, the update operations $\mathbf{D}[\sigma] = \mathbf{D}[\sigma] \cdot \langle A \rangle$ and $\mathbf{A}[\sigma] = H_{F,\Theta}(\mathbf{D}[\sigma])$), the classification error $CE(A, \mathbf{A}[\sigma])$ is measured by comparing the observed activity A with its prediction already recorded in $\mathbf{A}[\sigma]$:

$$CE(A, \mathbf{A}[\sigma]) = \begin{cases} 0 & if\ A = \mathbf{A}[\sigma] \\ 1 & otherwise \end{cases}. \tag{1}$$

The triple $(\mathbf{D}[\sigma], A, CE)$ is recorded in $\mathbf{W}$. Note, that this operation can be performed if both $\mathbf{D}[\sigma]$ and $\mathbf{A}[\sigma]$ exist, i.e., e is at least the second event of σ is observed in Σ. Given $\mathbf{W}$, let us introduce the functions: $\Pi_{CE} \colon \mathcal{B}(\mathcal{A}^* \times \mathcal{A} \times \mathbb{R}^+) \mapsto \mathbb{R}^{+^*}$ so that $\Pi_{CE}(\mathbf{W})$ is the sequence of classification errors recorded in $\mathbf{W}$, and $\Pi_{LP} \colon \mathcal{B}(\mathcal{A}^* \times \mathcal{A} \times \mathbb{R}^+) \mapsto \mathcal{B}(\mathcal{A}^* \times \mathcal{A})$ so that $\Pi_{LP}(\mathbf{W})$ is the multiset of labelled prefix trace sequences recorded in $\mathbf{W}$.

Definition 10 (Concept drift detector). *Let* $drift \colon \mathcal{B}(\mathcal{A}^* \times \mathcal{A} \times \mathbb{R}^+) \mapsto \{0, 1\}$ *be a concept drift detection function,* $drift(\mathbf{W}) = 0$ *if no degradation is observed in the performance of* $\Pi_{CE}(\mathbf{W})$*, 1 otherwise.*

If $drift(\mathbf{W}) = 1$ then $H_{F,\Theta}$ is updated to $\Pi_{LP}(\mathbf{W})$.

4. The DARWIN method

In this section we present DARWIN, a PPM method that combines deep learning, stream learning and transfer learning to solve the next-activity prediction problem in the online scenario. In DARWIN the prediction of the next activity of a running prefix trace is formulated as a classification problem, while

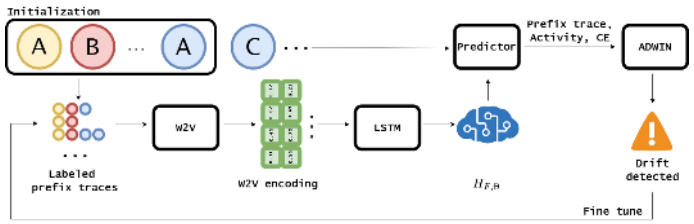


Figure 4: DARWIN pipeline

the learning process is divided into initialization step and online learning step. Let us consider an event stream Σ and an index $\eta \in \mathbb{N}^+$. In the initialization step, DARWIN learns a classification hypothesis from the multi-set of prefix running traces that have been recorded with their next activity labels during the initial η events of Σ. This classification hypothesis is learned by training a deep neural network architecture with LSTM layers. In the online learning step, DARWIN observes one event at a time and updates the classification hypothesis consequently. To do this, DARWIN continues using the same classification hypothesis to predict the next activity of any new running prefix trace recorded in Σ until a concept drift is detected. DARWIN monitors the accuracy of the current classification hypothesis to detect concept drifts, and applies a transfer learning strategy as soon as a change in accuracy is detected. Using transfer learning, DARWIN updates the classification hypothesis on drifted event data and fits the appearance of new activity sequences in the event stream. In the following we describe in detail each step of the proposed method, whose block diagram is reported in Figure 4. The pseudo-code of DARWIN is illustrated in Algorithm 1.

4.1. Initialization step (Algorithm 1, lines 2-11)

DARWIN starts observing Σ and learns a next activity classification hypothesis function $H_{F,\Theta}$ as η events have been observed on Σ and recorded in $\mathbf{D}$ (Algorithm 1, lines 2-7). During this step all observed traces, comprising complete traces, are maintained in $\mathbf{D}$. Differently, from the completion of this step

Algorithm 1: DARWIN algorithm

```
   Data: Σ (event stream), η(number of events for the initialization)
   Result: Eval (evaluation metrics)
1  begin
       /* Initialization step                                              */
2      i = 0
3      D = (H = {}, T = {root})
4      for i < η do
5          e=observe(Σ)
6          Record e in D
7          i = i + 1
8      L = labeledPrefixSequenceMultiset(D)
9      H_{F,Θ} = trainModel(L) /* Word2Vec + LSTM                          */
10     Delete complete traces from D
11     Use H_{F,Θ} to predict next activities of incomplete traces kept in D and record these
         predictions in A
       /* Online learning step                                             */
12     initialize ADWIN with an empty adaptive window W
13     Eval = []
14     for e=observe(Σ) do
15         σ = π_S(e), A = π_A(e)
           /* Check if a prediction was produced for σ in the past and recorded in A
              */
16         if there exists A[σ] then
17             add(Eval, CE(A, A[σ]))
18             drift = updateADWIN(W, ⟨D[σ], A, CE(A, A[σ])⟩)
19             Delete A[σ]
20             if drift = 1 then
21                 H_{F,Θ} = finetune(H_{F,Θ}, Π_LP(W))
22         if A = ⊥ then
23             Delete σ from D
24         else
25             Record e in D
26             Record H_{F,Θ}(D[σ]) in A
27     return Eval
```

and during the online learning step, only the incomplete observed traces will be maintained in $\mathbf{D}$.

Let $\mathcal{L} \in \mathcal{B}(\mathcal{A}^* \times \mathcal{A})$ be the labeled prefix sequence multiset extracted from $\mathbf{D}$ as η events have been observed in Σ (Algorithm 1, line 8). $H_{F,\Theta}$ is learned from $\mathcal{L}$

(Algorithm 1, line 9). Specifically, $H_{F,\Theta}$ is learned training a Long-Short-Term Memory (LSTM) neural network on a **Word2Vec** encoding of prefix traces in $\mathcal{L}$. To train the LSTM neural network, the padding technique is combined with the windowing mechanism to standardize different prefix lengths and obtain fixed sized prefixes (with padding length w). Specifically, dummy events are added to prefix traces with length less than w, while the most recent w events are kept into prefix traces with length greater than w. In addition, to make activity information suitable to be processed by the neural network, we apply the **Word2Vec** scheme that transforms categorical activity values in a 2-dimensional, real-valued, encoded representation with encoding size set equal to d. A brief description of the adopted **Word2Vec** architecture is reported in Appendix A. The derived **Word2Vec** representation of samples in $\mathcal{L}$ is then used to feed a supervised LSTM neural network that is used to learn the classification hypothesis $H_{F,\Theta}$. The output of the LSTM module is fed into a softmax layer to compute the final output (i.e., the next activity) from probabilities of different classes (activities). The cross-entropy loss function, that measures the error between the expected label and the probability predicted by the neural network, is used as cost function to be optimized. The number of LSTM units on each layer is a hyper-parameter that is fixed according to some optimization scheme (see details in Section 6.2).

At the end of the initialization step, DARWIN starts maintaining in $\mathbf{D}$ only the trace entries incompletely observed in Σ and records $H_{F,\Theta}(observed(\sigma, \Sigma))$ in $\mathbf{A}[\sigma]$ for each $observed(\sigma, \Sigma)$ recorded in $\mathbf{D}$. Hence, complete traces are deleted from $\mathbf{D}$ (Algorithm 1, line 10). Note that maintaining in memory only incomplete traces (also during the online learning step) aids in addressing memory constraints to record an event stream.

4.2. Online learning step (Algorithm 1, lines 12-27)

After the initialization step, DARWIN continues observing Σ using the classification hypothesis $H_{F,\Theta}$ to predict the next activity of any new observed trace. As a concept drift deteriorates the classification performance, $H_{F,\Theta}$ is updated

through the **Fine-tuning** strategy [67]. This is a strategy to transfer learning, where deep neural model parameters are changed to fit the new task. Notably recent studies in both cybersecurity [5] and remote sensing [3] have shown that **Fine-tuning** may contribute to reduce the impact of the catastrophic forgetting often observed when re-training (**Re-training**) a new deep neural model from scratch on new data. We use the state of the art concept drift detector **ADWIN** [10] to detect concept drifts in the error performance of $H_{F,\Theta}$, and **Fine-tuning** to adapt $H_{F,\Theta}$ to drifted data. **ADWIN** is often used also in adaptive and evolving ensemble learners, which represent one of the major mainstream of recent research in data streams [37]. Specifically, let e be the latest event observed on Σ, i.e., $observe(\Sigma) = e$, so that $\pi_{\mathcal{S}}(observe(\Sigma)) = \sigma$ and $\pi_{\mathcal{A}}(observe(\Sigma)) = A$ (Algorithm 1, line 15). DARWIN performs the following three steps on e: (1) concept drift detection (Algorithm 1, lines 16-18), (2) concept drift adaptation (Algorithm 1, lines 20-21) and (3) data structure update (Algorithm 1, lines 19, 22-26).

Concept drift detection. DARWIN verifies that a next activity has been predicted for σ in a previous step, i.e., it checks the existence of a table entry $\mathbf{A}[\sigma]$ (Algorithm 1, line 16). If this condition is verified, then the classification error $CE(A, \mathbf{A}[\sigma])$ is measured according to Eq. 1 (Algorithm 1, line 17) and the triple $(\mathbf{D}[\sigma], A, CE(A, \mathbf{A}[\sigma]))$ is passed to **ADWIN** that performs the concept drift detection on the sequence $\pi_{CE}(\mathbf{W})$ inside $\mathbf{W}$ (Algorithm 1, line 18). A short description of the concept drift detection mechanism realized by **ADWIN** is reported in Appendix B. Notice that, according to the data stream theory [32], detecting the change is harder in gradual changes than in abrupt changes. However, as shown in [10], **ADWIN** can take advantage from the ability of automating the window setting to its optimal size also in case of gradual changes. Finally, the processed entry $\mathbf{A}[\sigma]$ is deleted (Algorithm 1, line 19).

Concept drift adaptation. If **ADWIN** detects a concept drift, then $H_{F,\Theta}$ is adapted to cope with the drift (Algorithm 1, lines 20-21). In particular, to cope with a drift alerted by **ADWIN**, $H_{F,\Theta}$ is adapted to the labeled prefix

sequence multiset $\Pi_{LP}(\mathbf{W})$ through the **Fine-tuning** strategy (Algorithm 1, lines 20-21). The **Fine-tuning** strategy starts with the weights of the pre-trained $H_{F,\Theta}$ (both **Word2Vec** network and LSTM network). This is the source model. Subsequently, the **Fine-tuning** strategy updates these model weights to minimise the cross-entropy loss function both in the **Word2Vec** network and the LSTM network right now evaluated on $\Pi_{LP}(\mathbf{W})$. In this way the pre-trained **Word2Vec** and LSTM networks are adapted to the latest event changes that caused the decrease of performance of $H_{F,\Theta}$ without retraining from scratch with window label estimates only, which would incur in significant overhead and cause artefacts due to the catastrophic forgetting phenomenon. The fine-tuned model represents the target model.

Data structure update. Finally, data structures $\mathbf{D}$ and $\mathbf{A}$ are updated. If $A = \perp$, then trace σ has been completed in Σ. In this case, $\mathbf{D}[\sigma]$ is deleted (Algorithm 1, lines 19). Otherwise, the updates $\mathbf{D}[\sigma] = \mathbf{D}[\sigma] \cdot \langle A \rangle$ and $\mathbf{A}[\sigma] = H_{F,\Theta}(\mathbf{D}[\sigma])$ are performed in this order (Algorithm 1, lines 25-26).

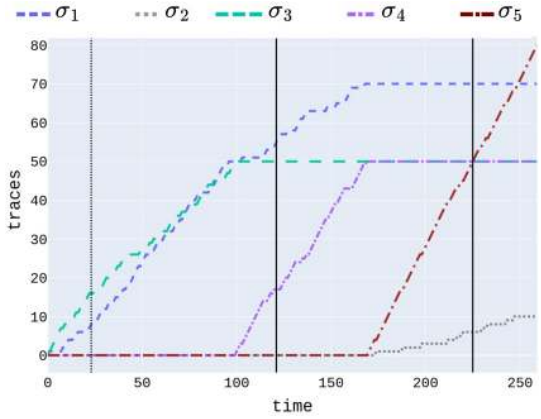
Latency concerns. Note that the **Fine-tuning** operation may introduce a learning latency until the new LSTM is ready. Therefore, in the interim, the old LSTM model (i.e., the classification hypothesis being updated) must be used to predict the next activity of the incoming observed traces, which may induce a temporary performance decay relative to the amount of the drift.

5. Motivating example

To illustrate the potential of relying on a mechanism for concept drift detection and adaptation in PPM, let us consider as an example the event stream reported in Figure 5. In particular, Figure 5a reports five distinct variant-traces (sequence of activities) recorded with their frequencies in the event stream, while Figure 5b shows how these traces are collected during the stream. We call running traces prefixes of complete traces. We select the first 10% of this event stream to perform the initialization step, while we consider the remaining 90%

$\sigma_1 : \langle A, B, C, D, E, F, \perp \rangle^{70}$
$\sigma_2 : \langle A, B, C, D, E, F, G, \perp \rangle^{10}$
$\sigma_3 : \langle A, B, C, D, F, \perp \rangle^{50}$
$\sigma_4 : \langle A, B, D, F, \perp \rangle^{50}$
$\sigma_5 : \langle A, B, D, F, G, \perp \rangle^{80}$

(a)



(b)

Figure 5: Event stream: multi-set of variant-traces recorded in the example event stream. Each superscript number indicates the number of times the distinct variant-trace appears in the stream. $\perp$ denotes the completion activity. (a) Number of traces per variant recorded in the event stream over time. (b) The vertical dotted line corresponds to the beginning of the online learning step. The vertical straight lines correspond to two drifts detected during the online learning step.

of this event stream to perform the online learning step. In the initialization step, a deep neural model with LSTM layers is trained as a predictive model for the next-activity of prefix traces recorded in the event stream. Subsequently, this predictive model is used in the online learning step, in order to predict the next-activity of new prefix traces recorded in the event stream.

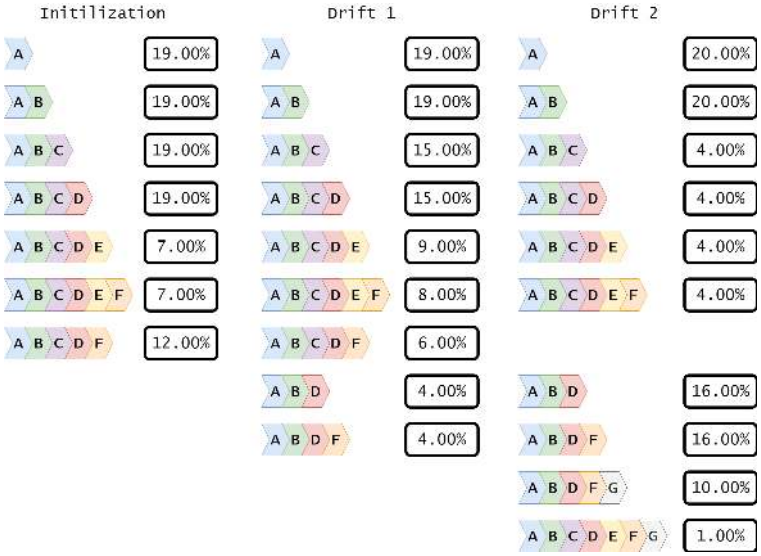


Figure 6: Distribution of prefix traces recorded in the initialization step and in the adaptive windows of **ADWIN** in correspondence of drifts detected in the online learning step

We note that the example event stream starts recording prefix traces of executions of variant-traces σ_1 ($\langle A, B, C, D, E, F, \perp \rangle$) and σ_3 ($\langle A, B, C, D, F, \perp \rangle$), respectively. Hence, prefix traces of σ_1 and σ_2 are processed for the initialization of the predictive model. This predictive model is used for the next-activity prediction of any prefix trace recorded in the online learning step until any concept drift is detected triggering the predictive model adaptation.

In this example, DARWIN is able to detect two concept drifts in the event stream (Figure 5b). Drift 1 is detected as prefix traces of σ_4 ($\langle A, B, D, F, \perp \rangle$) start being recorded in the event stream in place of prefix traces of σ_3 ($\langle A, B, C, D, F, \perp \rangle$). We note that σ_4 changes σ_3 by dropping the execution of C. On the other hand, drift 2 is detected as prefix traces of both σ_2 ($\langle A, B, C, D, E, F, G, \perp \rangle$) and σ_5 ($\langle A, B, D, F, G, \perp \rangle$) start being recorded in the event stream in place of the prefix traces of σ_1 ($\langle A, B, C, D, E, F, \perp \rangle$) and σ_4 ($\langle A, B, D, F, \perp \rangle$), respectively. Both σ_2 and σ_5 change σ_1 and σ_4 by executing the new activity G before the completion of each trace.

Table 3: Fscore metrics for the predictive model learned in the initialization step (**Static**) and adapted with DARWIN. $\perp$ is the activity denoting the completion of a trace. support is the number of occurrences of each activity.

Activity	**Static**	DARWIN	support
B	**1.000**	**1.000**	234
C	0.618	**0.660**	105
D	0.447	**0.691**	235
E	0.000	**0.521**	71
F	0.145	**0.732**	235
G	0.000	**0.512**	90
$\perp$	0.739	**0.781**	235

Figure 6 shows the distribution of prefix traces processed during the initialization step, and prefix traces recorded in the adaptive windows maintained by the **ADWIN** mechanism of DARWIN in corresponding of the two concept drifts. The adaptive window of drift 1 shows how the new prefix traces $\langle A, B, D\rangle$ and $\langle A, B, D, F\rangle$ bring out the new behaviour emerged in the event stream due to the drop of activity C. On the other hand, in the adaptive window of drift 2, it can be seen that the behaviour identified in the first drift is now established and that the new activity G has emerged in the process model. With regard to prefix trace $\langle A, B, C, D, F\rangle$, it goes from a frequency of 12% in the initialization to a frequency of 6% in the adaptive window of drift until it disappears in drift 2. Finally, in drift 2 it can be seen that the behaviour identified in the first drift is now established and that the new activity G has emerged in the process model.

Table 3 shows the Fscore values for each activity recorded in the event stream. The results compare the performance of DARWIN to that of the baseline **Static** that uses always the same predictive model, regardless of any concept drift. These results show that DARWIN can successfully gain accuracy for all activities, especially for predicting activities E and G. This gain in accuracy is due to the ability of DARWIN to detect concept drifts through **ADWIN** and to adapt the predictive model to the drifted prefix traces through fine tuning.

Finally, in order to perform the route cause analysis of the concept drifts,

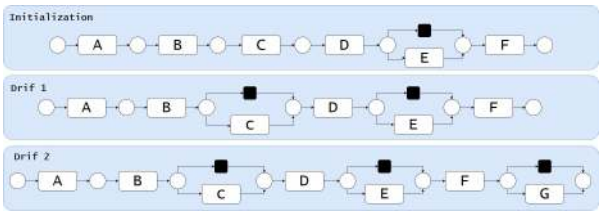


Figure 7: Petri net representation of the process models generated by Inductive Miner after the initialization step and at the concept drift points during the online learning of the stream

we use Inductive Miner [46] to generate the process models underlining the traces recorded in correspondence of the initialisation step and the two concept drifts identified by DARWIN. Figure 7 shows the Petri net representation of process models discovered from the traces whose prefix traces are recorded in the initialization step, as well as in the adaptive windows of drift 1 and drift 2 maintained by **ADWIN**. These Petri nets disclose useful information for the route cause analysis of the detected concept drifts. In fact, the Petri net of drift 1 shows an activity path, unobserved in the Petri net of the initialization, that drops C and allows us to execute D after B. This is the route cause of drift 1. On the other hand, the Petri net of drift 2 allows us to identify the appearance of a new activity G executed before the trace completion as the route cause of drift 2.

6. Empirical evaluation

We conducted a range of experiments on several benchmark event logs handled as event streams. The goal of this experimentation is to investigate the performance of DARWIN, evaluated in terms of accuracy, computation time and memory usage on twelve, real-life event logs dealt as event streams. The evaluation mainly aims at exploring the performance of DARWIN with respect to the learning strategy (**Fine-tuning**) and the encoding strategy (**Word2Vec**). In addition, the evaluation compares DARWIN to several state of the art data stream methods.

Table 4: Explored configurations of deep neural model hyper-parameters

Parameter	Considered values
N. of LSTM layers	{1, 2, 3}
Layer size	[10, 150]
d (**Word2Vec** size)	$\{2^4 - 2^{10}\}$
Batch size	$\{2^5 - 2^{10}\}$
Learning rate	[0.00001, 0.1]
Dropout	[0, 1]

In the following, we describe the adopted implementation of DARWIN in Section 6.1, the event logs in Section 6.2, the compared learning and encoding strategies in Section 6.3, the validation procedure and the metrics measured to evaluate the effectiveness of the described strategies in Section 6.4, and the research questions in Section 6.5. We illustrate the results in Section 6.6.

6.1. Implementation details

We implemented DARWIN in Python 3.9 - 64 bit version using Keras 2.4 with TensorFlow as the backend. The source code is made publicly available at the github repository.[3] We conducted the optimization phase of the hyper-parameters in the initialization phase by using the 20% of the training set as validation set, selected with stratified sampling according to the Pareto Principle. The hyper-parameter (see Table 4) optimization was automatically performed with the tree-structured Parzen estimator (TPE) [9]. We integrated **ADWIN** algorithm implemented in River[4] – Machine learning library for data streams in Python – with the default parameter configuration ($\delta = 0.002$). We adopted **Word2Vec** algorithm implemented in Gensim Version 4.3.0[5] library. This version supports online training with a resuming strategy that was also used for **Fine-tuning** strategy. Specifically, we used build_vocab with the option update=True, in order to add the new activities appearing in an **ADWIN**

[3] `https://github.com/vinspdb/DARWIN`

[4] `https://riverml.xyz/dev/api/drift/ADWIN/`

[5] `https://radimrehurek.com/gensim/models/word2vec.html`

Table 5: Event stream description: number of activities (**A**), resources (**R**), traces (**T**), events (**E**), trace variants (**V**), ratio **T**/**V**, average trace length ($\mathbf{T_{avgL}}$), average trace duration in days ($\mathbf{T_{avgD}}$) and entire stream duration in days ($\mathbf{\Sigma_{totD}}$)

Event stream	**A**	**R**	**E**	**T**	**V**	**T/V**	$\mathbf{T_{avgL}}$	$\mathbf{T_{avgD}}$ (days)	$\mathbf{\Sigma_{totD}}$ (days)
BPI20D [25]	17	2	56437	10500	99	106.06	5	11.5	889
BPI20T [25]	51	2	86581	7065	1478	4.78	12	87.4	1792
BPI20R [25]	19	2	36796	6886	89	77.37	5	8.2	941
BPI20I [25]	34	2	72151	6449	753	8.56	11	86.5	1313
BPI20P [25]	29	2	18246	2099	202	10.39	12	36.8	772
Helpdesk [59]	14	22	21348	4580	226	20.26	5	40.9	1451
BPI13C [66]	7	585	6660	1487	327	4.54	4	179.2	2332
BPI13I [66]	13	1440	65533	7554	2278	3.31	9	12.1	784
BPI12L [24]	36	69	262200	13087	4366	2.99	20	8.6	165
BPIC18G [26]	23	117	569209	29059	9372	19.58	20	143.5	576
BPIC18P [26]	10	111	132963	14750	3615	9.01	9	196	976
BPIC18C [26]	7	113	161296	43808	59	742.51	4	57.3	1051

window to the vocabulary the **Word2Vec** model and continue training the **Word2Vec** model with the new prefix traces and the new activities recorded in the **ADWIN** window. We integrated **Word2Vec** with default parameters of Gensim implementation. In particular, we set the neighbourhood size $j = 5$, while we optimised the embedding size d with TPE (see Table 4). For the LSTM network, we set the size $w = 4$ for the padding of sequences. The number of LSTM layers, the layer size, the batch size, the learning rate and the dropout were optimized with TPE (see Table 4). The Backpropagation training was used with an early stopping to avoid overfitting. The training phase was stopped when the loss on the validation set did not improve for 20 consecutive epochs (early stopping criterion). The Nadam optimizer was used to minimise the loss function. The maximum number of epochs was equal to 200.

6.2. Event streams

We processed trace streams generated from twelve real-life event logs available on the 4TU Centre for Research.[6] These event logs record executions of

[6]https://data.4tu.nl/portal

business processes in tourism, assistance and management.

In particular, the event logs *BPI20D*, *BPI20T*, *BPI20R*, *BPI20I* and *BPI20P* were provided for the Business Process Intelligence (BPI) Challenge, in 2020 [25]. These event logs contain events pertaining to two years of anonymised travel expense claims at Eindhoven University of Technology. In 2017, events were collected for two departments, while in 2018 for the entire university. *BPI20T* was extracted referred to travel permits. *BPI20D*, *BPI20I*, *BPI20P* and *BPI20R* were extracted referred to four request types, namely domestic declarations, international declarations, prepaid travel costs and requests for payment, respectively. The event log *Helpdesk* was provided by a private Italian software company and concerns the ticketing management process of the company [59]. The event logs *BPI13C* and *BPI13I* were provided for the Business Process Intelligence (BPI) Challenge, in 2013 [66]. Specifically, *BPI13C* contains events collected from the closed problem management system of Volvo IT in Belgium. *BPI13I* contains events collected from the incident management system. The event log *BPI12L* was provided for the Business Process Intelligence (BPI) Challenge, in 2012 [24]. It contains events collected monitoring the loan application process of a Dutch financial institute. Finally, the event logs *BPIC18G*, *BPIC18P*, and *BPIC18C* were provided for the Business Process Intelligence (BPI) Challenge, in 2018 [26]. *BPIC18G* contains events recorded after 2007 from the European Agricultural Guarantee Fund, in order to handle geo-parcel document applications for EU direct payments of German farmers. *BPIC18P* contains events concerning parcels for which subsidies were requested before 2007. *BPIC18C* contains events regards documents containing summarized results of various checks (reference alignment, department control, inspections). Notice that events of *BPIC18G* were produced according to three sub-processes, while events of *BPIC18P* and *BPIC18C* were produced according to one sub-process. Table 5 reports a summary of the characteristics of these event logs. We note that event logs selected for this study exhibit different characteristics in terms of number of activities, resources, events, traces and variants, percentage of traces per variants, average trace length and duration,

stream duration.

6.3. Evaluated strategies

The **Fine-tuning** learning strategy is compared to the **Static** and **Re-training** learning strategies defined as follows:

- In the **Static** strategy the deep neural model trained during the initialization step was never updated during the online learning step.
- In the **Re-training** strategy the deep neural model was re-trained from scratch with the streamed event data recorded in the adaptive window of **ADWIN** each time a concept drift is alerted.

Static acts as the offline baseline against which to compare any dynamic strategy for online learning, while both **Re-training** and **Fine-tuning** are dynamic strategies for online learning. **Fine-tuning** can conceivably keep some historical information as it starts with weights of a previously trained deep neural model to adapt these weights to the potentially drifting distribution of new data. Instead **Re-training** can be susceptible to catastrophic forgetting [39] over long time restarting the training of a deep neural model on new data.

Both **Static**, **Re-training** and **Fine-tuning** strategies were tested with both simple deep neural networks and dynamic Bayesian networks for next-activity prediction in [58] by using **One-Hot-Encoding** as data encoding strategy. Based upon this previous work, we compare the encoding strategy **Word2Vec** to **One-Hot-Encoding**. The **One-Hot-Encoding** strategy applies a well known encoding technique that neglects context during the encoding stage. It was commonly used in predictive process mining literature to encode categorical attributes (such as activity information) as vectors of binary values (one value for each category) before training a deep neural model [42, 51, 68]. However, we note that **One-Hot-Encoding** learns a static representation of categorical information that must be re-trained from scratch as new activity types are recorded in the stream. This turns to be a major issue when **One-Hot-Encoding** must be used with **Fine-tuning**. This issue was solved in [58] by

performing the **One-Hot-Encoding** of the activity information before starting the analysis of the event stream and assuming that the domain of all the distinct activity types, which may appear along the stream, was known apriori at the encoding time. We adopted this solution with **Fine-tuning**, although we were aware that this strategy did not adjust to the stream scenarios where new activity types might not be known apriori. Instead, **One-Hot-Encoding** was safely used in combination with **Re-training**.

6.4. Experimental setting and metrics

We adopted the Prequential Evaluation schema [33] that is the most used evaluation schema to evaluate online classification models in data streams. According to the Prequential Evaluation, we used prefix traces of new events recorded in the stream to test the online predictive model (i.e., to predict the next-activity of the prefix trace) and, then, to update the predictive model as the ground truth labels of the predicted activities were finally recorded in the stream. This schema was used to measure the accuracy of the predictive model since the start of the online learning step as the ground truth of each predicted activity label was recorded in the stream. In particular, we measured the accuracy performance of the compared strategies by computing the OA (Overall Accuracy) and Fscore on predictions made in the online learning step as the first event was observed in the traces. In the computation of the Fscore, we gave equal weights to each activity type (computing the macro-Fscore). In this way, we avoid that our evaluation offsets the possible impact of imbalanced data learning. In addition, we measured N.Drifts, that is the number of drifts detected in the online learning step and TIME that is the computation time (spent in seconds) processing the entire event stream. The computation times were collected running the experiments on Intel(R) Core(TM) i7-9700 CPU, GeForce RTX 2080 GPU, 32GB Ram Memory, Windows 10 Home. Notice that, in the measurement of TIME, we do not account for the time spent waiting for new events acquired in the stream without performing any operation.

6.5. Research questions

The empirical validation was done to answer the following questions

Q1 How does the two dynamic strategies – **Re-training** and **Fine-tuning**– compare to the strategy **Static** when evaluated in next-activity prediction problems?

Q2 Is the use of context information through **Word2Vec** more powerful than the traditional **One-Hot-Encoding**?

Q3 How does each individual perspective (activity, resource, timestamp) affect the performance of the next-activity predictive model trained in the stream setting?

Q4 Can the route-cause analysis of concept drifts allow us to disclose useful information on how a process model changed over time?

Q5 How does the proposed approach scale in memory usage by varying the event stream characteristics?

Q6 How does the proposed approach compare to well-established stream mining approaches (e.g., Adaptive Hoeffding Tree [11], Adaptive Random Forest [36], Streaming Random Patches [38])?

Q7 What is the sensitivity of the proposed approach to the confidence δ of **ADWIN**, the size η of the initialization step, the size j of the neighbourhood of **Word2Vec** and the size w of the padding window?

The results of the learning strategy study and the encoding strategy study to answer Q1 and Q2 are reported in Sections 6.6.1 and 6.6.2, respectively. The results of the perspective study to answer Q3 are illustrated in Section 6.6.3. The route-cause analysis of some examples of detected concept drifts to answer Q4 is analysed in Section 6.6.4. The results of the memory usage study to answer Q5 is illustrated in Section 6.6.5. The results of the comparative analysis performed to answer Q6 are shown in Section 6.6.6. Finally, the results of the parameter sensitivity study to answer Q7 is illustrated in Section 6.6.7.

6.6. Results and discussion

Results presented in this Section were collected by processing 10% of events in each stream for the initialization phase and the remaining events of the stream for the online learning step according to the Prequential Evaluation strategy.

6.6.1. Learning strategy analysis (Q1)

We compared the performance of **Static**, **Re-training** and **Fine-tuning** by running DARWIN on the activity perspective of each event stream and using **Word2Vec** to encode activity information.

Table 6 collects the N.Drifts, OA, Fscore and TIME of the compared strategies. These results show that the dynamic learning strategy adopted may have an effect on the number of drifts detected. On the other hand, these results also provide empirical evidence that the two dynamic learning strategies **Re-training** and **Fine-tuning** are systematically more accurate than the baseline **Static**. In addition, **Fine-tuning** outperforms (or performs equally to) **Re-training** in eleven out of twelve streams for OA and in ten out of twelve streams for Fscore. In general, **Fine-tuning** achieves the highest accuracy performance with **Re-training** the runner-up of this comparative evaluation except for a few cases (*BPI20D* for OA, *BPI20R*, *BPI20P* and *BPIC18P* for Fscore), where **Re-training** achieves the highest accuracy performance with **Fine-tuning** as runner-up. As in the majority of the experimented event streams, **Fine-tuning** achieves the highest Fscore coupled to the highest OA, we can conclude that **Fine-tuning** can commonly learn a more robust neural classification hypothesis by typically gaining accuracy predicting correctly minority activities, in addition to majority activities. This is a notable outcome considering that predicting minority activities is commonly recognized as a major issue to address in next-activity prediction problems [55]. In addition, by examining in depth results on *BPI20D*, we note that the difference in the OA values of **Fine-tuning** and **Re-training** is negligible (0.891 vs 0.892), while Fscore of **Fine-tuning** is slightly better than Fscore of **Re-training** (0.425 vs 0.417) also in this dataset. Hence, we conclude that even this dataset supports the

Table 6: Comparison between **Static**, **Re-training** and **Fine-tuning** of DARWIN (run with activity perspective and **Word2Vec**) in terms of N.Drifts, OA, Fscore and TIME (in seconds). The highest OA and Fscore and the lowest TIME are in bold.

Stream	Strategy	N.Drifts	OA	Fscore	TIME
	Static	-	0.568	0.238	**2085.89**
BPI20D	**Re-training**	51	**0.892**	0.417	5210.11
	Fine-tuning	63	0.891	**0.425**	5572.26
	Static	-	0.585	0.212	**2900.73**
BPI20T	**Re-training**	166	0.608	0.249	7260.02
	Fine-tuning	153	**0.650**	**0.296**	8036.72
	Static	-	0.636	0.238	**1263.88**
BPI20R	**Re-training**	52	0.836	**0.339**	2850.67
	Fine-tuning	36	**0.872**	0.334	2709.17
	Static	-	0.653	0.288	**2424.78**
BPI20I	**Re-training**	119	0.743	0.366	6113.73
	Fine-tuning	76	**0.793**	**0.387**	5977.03
	Static	-	0.557	0.273	**603.63**
BPI20P	**Re-training**	16	0.786	**0.437**	1547.88
	Fine-tuning	7	**0.827**	0.411	1439.05
	Static	-	0.801	0.261	**702.66**
Helpdesk	**Re-training**	9	**0.849**	0.290	1788.23
	Fine-tuning	10	**0.849**	**0.308**	1819.2
	Static	-	0.259	0.145	**222.33**
BPI13C	**Re-training**	17	0.613	0.397	729.97
	Fine-tuning	16	**0.646**	**0.424**	762.55
	Static	-	0.554	0.281	**2357.94**
BPI13I	**Re-training**	50	0.713	0.506	4295.21
	Fine-tuning	51	**0.722**	**0.520**	4374.57
	Static	-	0.838	0.659	**18231.02**
BPI12L	**Re-training**	4	**0.841**	0.674	27041.11
	Fine-tuning	4	**0.841**	**0.675**	22952.66
	Static	-	0.080	0.017	**40066.74**
BPIC18G	**Re-training**	293	0.703	0.347	47630.84
	Fine-tuning	349	**0.727**	**0.362**	53934.63
	Static	-	0.407	0.249	**5735.35**
BPIC18P	**Re-training**	50	**0.648**	**0.349**	7850.70
	Fine-tuning	36	**0.648**	0.348	6230.80
	Static	-	0.820	0.395	**8728.71**
BPIC18C	**Re-training**	68	0.852	0.475	9689.04
	Fine-tuning	80	**0.861**	**0.484**	10117.42

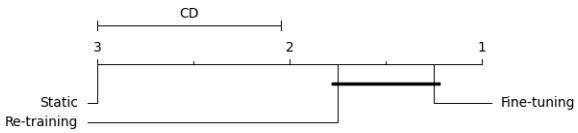


Figure 8: Comparison of Fscore of the learning strategies **Static**, **Re-training** and **Fine-tuning** of DARWIN (run with activity perspective and **Word2Vec**) with the Nemenyi test. Groups of strategies that are not significantly different (at $p - value \leq 0.05$) are connected.

general conclusion that **Fine-tuning** can be preferred to **Re-training** considering the better performance on minority activities. Instead, this conclusion on the minority classes performance is not supported by the Fscore performance of *BPI20R*, *BPI20P*, and *BPIC18P*. In these datasets, **Re-training** achieves higher Fscore than **Fine-tuning**, despite **Fine-tuning** still achieves higher OA than (or equal OA to) **Re-training**. This result suggests that despite accounting for model parameters estimated on past data commonly helps **Fine-tuning** in coping with catastrophic forgetting, **Fine-tuning** strategy may sometime fail in appropriately adapting the classification hypothesis to drifts of minority classes for which insufficient data have been windowed. Hence, in a few cases, a dynamic selection between **Re-training** and **Fine-tuning** may be explored.

To statistically test whether the improvement in accuracy performance of the dynamic learning strategies is significant, we used the Friedman's test [21] with the post-hoc Nemenyi test for pairwise comparisons [21]. We performed this test on the Fscore of the compared strategies on each event stream and reject the null hypothesis with $p\text{-}value \leq 0.05$. The critical difference diagram, obtained with this using a 0.05 significance level, is reported in Figure 8. This shows that both **Fine-tuning** and **Re-training** are statistically better than **Static**. In addition, despite **Fine-tuning** and **Re-training** are connected in the Nemenyi test, the Nemenyi test ranking also shows that **Fine-tuning** appears the strategy to prefer in the majority of cases with **Re-training** as runner-up.

In any case, the higher accuracy of both **Fine-tuning** and **Re-training** is achieved at the cost of the more time spent detecting drifts and, possibly,

retraining or fine-tuning the predictive neural model in correspondence of the detected drifts. No stable difference is observed in the TIME spent by both **Re-training** and **Fine-tuning** completing the event stream analysis (as shown in Table 6). In general, the higher the number of concept drifts detected, the more the time spent completing the stream processing. The only exceptions are observed with *BPI20T* and *BPI13C*. However, in *BPI13C*, the difference in the number of drifts, as well in the TIME, of both **Fine-tuning** and **Re-training** is negligible (16 drifts detected and 762,55 seconds spent with **Fine-tuning** versus 17 drifts detected and 729,97 seconds spent with **Re-training**). Instead, in *BPI20T*, **Fine-tuning** with 153 drifts is unexpectedly slower than **Re-training** with 166 drifts (8036,72 vs 7260 seconds). To explain this result, we analysed the average number of epochs completed in *BPI20T* with **Re-training** and **Fine-tuning**, respectively.[7] **Re-training** completed 89 epochs in average, while **Fine-tuning** completed 126 epochs in average. This explains the observed behavior in the TIME of *BPI20T*. In this regard, we highlight that **Fine-tuning** starts from pre-trained networks, in order to mitigate the catastrophic forgetting by accounting for some knowledge on past data enclosed in the pre-trained networks. However, this does not necessarily lead to speed-up the back-propagation computation.

To complete this analysis we explore the Fscore of **Static**, **Re-training** and **Fine-tuning** measured on each activity type. Table 7 reports the Fscore values of **Static**, **Re-training** and **Fine-tuning**, which were measured on each activity type recorded in *BPI20I*, at the completion of the online learning step. These results show that the dynamic learning strategies – **Re-training** and **Fine-tuning**– gain accuracy compared to **Static** on the majority of the activity types (and, in particular, on both the new activities and the minority activities). A few exceptions occur on six out of thirty-five activity types (i.e.

[7]The number of epochs was dynamically decided with the early stopping criterion that stopped back-propagation training when the loss on the validation set did not improve for 20 consecutive epochs.

Table 7: Fscore per activity of **Static**, **Re-training** and **Fine-tuning** of DARWIN (run with activity perspective and **Word2Vec**) on *BPI20I*. Support is the number of events executing the activity in the initialization (**Init**) and evaluation (**Eval**) phases. (*) marks new activities appearing in the online learning step. The best Fscore values are in bold.

Activity	**Support**		Fscore		
	Init	**Eval**	**Static**	**Re-training**	**Fine-tuning**
Declaration approved by admin (*)	0	5037	0	0.766	**0.837**
Declaration approved by budget owner(*)	0	1834	0	0.237	**0.317**
Declaration approved by pre-approver	329	283	0.142	**0.446**	**0.446**
Declaration approved by supervisor	49	206	0.01	**0.009**	**0.009**
Final declaration approved by director	44	205	**0.981**	0.767	0.913
Final declaration approved by supervisor	733	5301	0.499	0.753	**0.784**
Declaration rejected by admin(*)	0	1549	0	**0.091**	0.043
Declaration rejected by budget owner(*)	0	40	0	0	0
Declaration rejected by director	2	2	0	0	0
Declaration rejected by employee	64	1716	0.156	0.830	**0.964**
Declaration rejected by missing	50	51	0	0	0
Declaration rejected by pre-approver	43	41	0	0	0
Declaration rejected by supervisor	20	105	0	0	0
Declaration saved by employee	7	59	0	**0.018**	0
Declaration submitted by employee	694	6912	0.830	0.804	**0.874**
End trip	677	5754	0.909	0.797	0.852
Payment Handled	696	5487	**0.996**	0.830	0.984
Permit approved by admin(*)	0	4839	0	0.885	**0.961**
Permit approved by budget owner(*)	0	1763	0	0.151	**0.163**
Permit approved by pre-approver	350	183	0.059	**0.09**	**0.09**
Permit approved by supervisor	141	500	0.003	**0.036**	0.033
Final permit approved by director	139	499	**0.966**	0.865	0.934
Final permit approved by supervisor	721	4658	0.511	0.682	**0.727**
Permit rejected by admin(*)	0	83	0	0	**0.022**
Permit rejected by budget owner(*)	0	31	0	0	0
Permit rejected by director(*)	0	1	0	0	0
Permit rejected by employee	33	198	0.562	0.729	**0.776**
Permit rejected by missing	20	23	0	0	0
Permit rejected by pre-approver	18	7	0	0	0
Permit rejected by supervisor	15	77	0	0	0
Permit submitted by employee	121	826	**0.689**	0.657	0.617
Request Payment	700	5459	**0.991**	0.875	0.986
Send Reminder(*)	0	434	0	0	**0.020**
Start trip	615	4775	**0.859**	0.723	0.821
END	548	5901	0.910	0.768	**0.934**

"*Final declaration approved by director*", "*Payment Handled*", "*Final permit approved by director*", "*Permit submitted by employee*", "*Request Payment*", and "*Start trip*"), where **Static** outperforms its runner-up **Fine-tuning**. However, in these few cases, the differences between the measured Fscore of both **Static** and **Fine-tuning** are small. This result may depend on the fact that the concept drift detection is performed monitoring the overall error done on the multi-class formulation of the next-activity prediction problem. The analysis of per-class results reported in Table 7 shows that this approach can occasionally lead to alert concept drifts that may concern the majority of activities, but not necessarily all activities. **Re-training** or **Fine-tuning** the multi-class neural model in these cases may reduce the accuracy performance on the ability of predicting these activities. Notably, this behaviour is more evident with **Re-training** than with **Fine-tuning**, since **Re-training** is more prone than **Fine-tuning** to be subjected to the catastrophic forgetting phenomenon. In any case, we note that the dynamic learning strategies, and in particular **Fine-tuning**, commonly outperform **Static** being able to predict occurrences of several new activity types (e.g., "*Declaration approved by admin*") that appeared in the online learning step, despite they were never observed in the initialization phase. In addition, the dynamic learning strategies generally outperform **Static** being able to predict several activity types (e.g., "*Final declaration approved by supervisor*") that were already observed during the initialization phase, but subsequently appeared in prefix traces that presumably changed their activity behaviour path along the evaluation time.

Finally, Figure 9 shows how the Fscore of **Static**, **Re-training** and **Fine-tuning**, measured on both "*Declaration approved by admin*" and "*Final declaration approved by supervisor*" of *BPI20I*, varied over the time as any prediction was evaluated in the online learning step. The Fscore of **Static** measured on "*Declaration approved by admin*" is constantly equal to zero due to the absence of any mechanism to adapt the predictive model of **Static** to the appearance of any new activity type, while the Fscore measured on the same activity for both **Re-training** and **Fine-tuning** grows over time achieving 0.766 and 0.837,

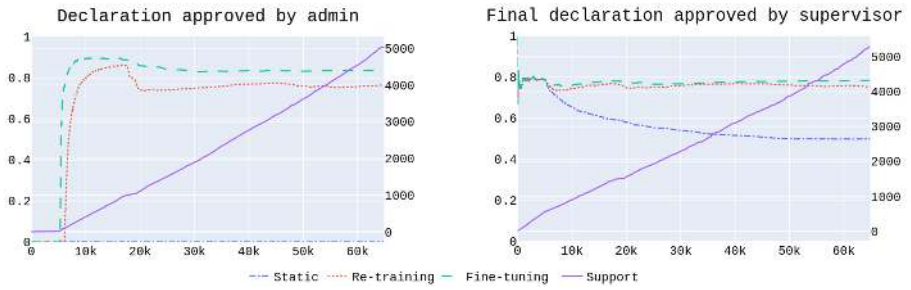


Figure 9: Comparison of how Fscore (axis Y) measured on "*Declaration approved by admin*" and "*Final declaration approved by supervisor*" for **Static**, **Re-training** and **Fine-tuning** of DARWIN (run with activity perspective and **Word2Vec**) varies over time during the online learning step

respectively, at the end of the online learning step. These results provide the evidence of the ability of how both **Re-training** and **Fine-tuning** are actually able to adapt the predictive neural model to the appearance of occurrences of a new activity type in the stream.

On the other hand, we note that the Fscore of **Static**, **Re-training** and **Fine-tuning** measured on "*Final declaration approved by supervisor*" evolves with a similar, roughly constant, trend in the beginning of the online learning step. This behaviour suggests that no concept drift presumably concerned the prefix traces recorded for this specific activity in the beginning of the online learning step and, consequently, the predictive neural model trained in the initialization phase keeps its initial performance on this activity. In addition, the Fscore of **Static** goes down to 0.446 over time due to the absence of any mechanism to adapt the predictive neural model of **Static** to drifts presumably upcoming over time on the recorded prefix traces of '*Final declaration approved by supervisor*". Instead the Fscore of both **Re-training** and **Fine-tuning** measured on the same activity remains stable over time being equal to 0.753 and 0.784, respectively, at the completion of online learning step. This is thanks to the adaptation mechanisms adopted in both **Re-training** and **Fine-tuning**. Similar conclusions can be drawn analysing the Fscore per activity type achieved

processing the remaining event streams.

In short, results illustrated in this Section show that the dynamic learning strategies **Fine-tuning** and **Re-training** are systematically more accurate than the static baseline **Static**, despite this gain in accuracy is achieved at the cost of the more time spent detecting and handling drifts. In addition, **Fine-tuning** is commonly more accurate than **Re-training** that is generally more prone than **Fine-tuning** to be subjected to the catastrophic forgetting phenomenon.

6.6.2. Encoding strategy analysis (Q2)

We compared the performance of **Word2Vec** to that of **One-Hot-Encoding** (already used in [58]) by considering **Re-training** and **Fine-tuning** as learning strategies and taking into account the information enclosed in the activity perspective of each event stream.

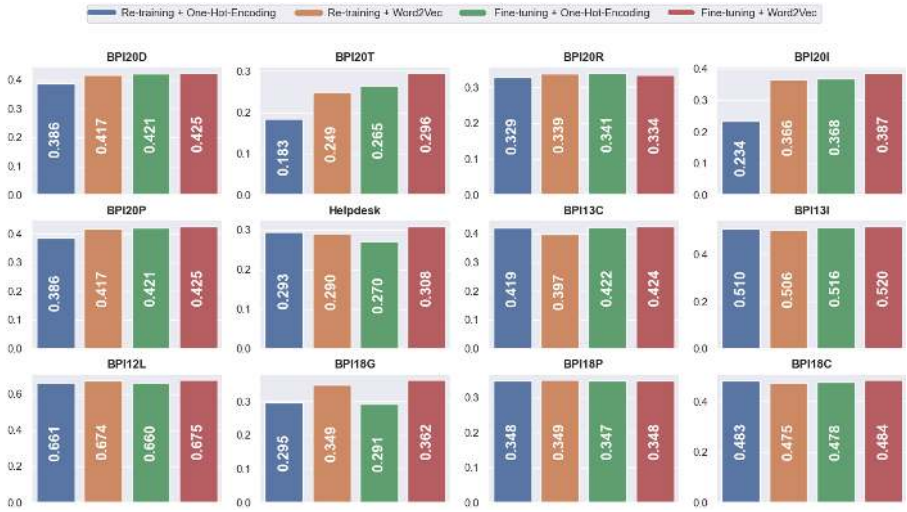


Figure 10: Fscore of deep neural models trained with **Word2Vec** and **One-Hot-Encoding**. Results were collected in DARWIN run with activity perspective, **Re-training** and **Fine-tuning** as learning strategies.

Figures 10 and 11 show the Fscore and the Nemenyi test on Fscore of **Re-training** and **Fine-tuning**, respectively, with both **One-Hot-Encoding** and **Word2Vec** as encoding techniques. The Fscore values show that

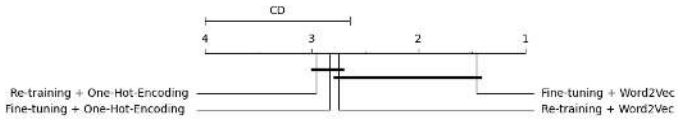


Figure 11: Comparison of Fscore of the **Re-training + Word2Vec**, **Re-training + One-Hot-Encoding**, **Fine-tuning + Word2Vec** and **Fine-tuning + One-Hot-Encoding** (run in DARWIN with activity perspective) with the Nemenyi test. Groups of strategies that are not significantly different (at $p - value \leq 0.05$) are connected.

Fine-tuning + Word2Vec achieves the highest Fscore in ten out of twelve data streams. This result is also confirmed by the Nemenyi test shown in Figure 11, where **Fine-tuning + Word2Vec** appears as the best configuration with **Re-training + Word2Vec** as runner up. In addition, Nemenyi test highlights that **Fine-tuning + Word2Vec** is statistically better than the two configurations with **One-Hot-Encoding**.

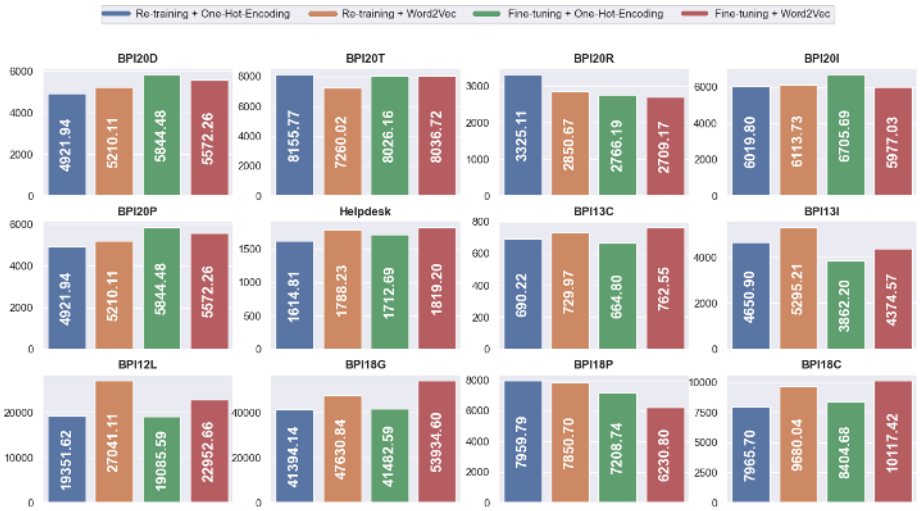


Figure 12: TIME spent in seconds processing event streams with **Word2Vec** and **One-Hot-Encoding**. Results were collected in DARWIN run with activity perspective, **Re-training** and **Fine-tuning** as learning strategies.

Finally, Figure 12 shows the TIME spent processing event streams by **Fine-tuning + Word2Vec**, **Fine-tuning + One-Hot-Encoding**, **Re-training +**

Word2Vec and **Re-training + One-Hot-Encoding**. These results show that no compared method appears more efficient than the related methods, although differences in TIME among the compared methods are commonly small in all the event streams.

So, this analysis highlights that **Word2Vec** outperforms **One-Hot-Encoding** with the additional advantage of going beyond the **One-Hot-Encoding** requirement of assuming that all activities that will be acquired along the stream should be known apriori. Notably, this outcome is achieved without any remarkable burden of computation.

6.6.3. Perspective analysis (Q3)

Various PPM algorithms process activities, timestamps and resources. So, we measured the accuracy and efficiency of the following variants of both **Re-training** and **Fine-tuning**: (1) A that accounts for activities (baseline), (2) AT that accounts for activities and timestamps, (3) AR that accounts for activities and resources and (4) ATR that accounts for activities, timestamps and resources. Both activities and resources are encoded using **Word2Vec** neural networks.

Figure 13 shows the Fscore and TIME, respectively, of the compared methods. Again, the Fscore values show that the highest accuracy is commonly achieved by a configuration that used **Fine-tuning** as dynamic learning strategy. The only exceptions are observed with *BPI12L* where the highest Fscore is achieved by **Re-training** (AR) and *BPIC18P* with **Re-training** (AT), respectively. These results also highlight that accounting for timestamps and/or resources, in addition to activities, may allow us to gain accuracy in some event streams. In fact, the highest Fscore is achieved using A in five out of twelve streams, AT in two out of twelve streams, AR in four out of twelve streams, ATR in one out of twelve streams. Hence, either resources or timestamps may sometime provide valuable information for gaining accuracy in next-activity prediction problems. Notably, this experimentation reveals that processing both resource and timestamps is helpful in *BPI20R* only. In general, the relevance of a perspective

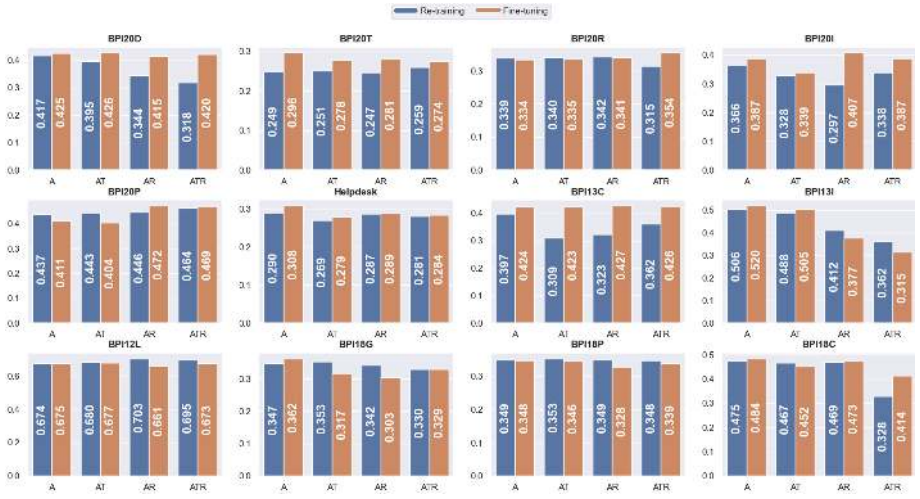


Figure 13: Fscore of deep neural models trained with DARWIN (**Re-training** or **Fine-tuning** as learning strategy and **Word2Vec** as encoding strategy) with information enclosed in A, AT, AR and ATR

on the accuracy performance of the next-activity prediction function changes with data streams. The current analysis of several event log characteristics collected in Table 5 does not provide any clear insight into potential stream characteristics that may have an effect on the importance of perspectives for next activity prediction. On the other hand, our feeling is that the importance of a perspective may change over time in a dynamic business process scenario. This consideration highlights a limit of performing multi-view learning without monitoring possible drifts in the perspective importance.

Finally Figure 14 shows the TIME of the compared methods. The collected results show that training a predictive neural model with multiple perspectives may not require more computation time to process the entire stream. In several cases, processing a greater amount of input information may allow us to speed up the loss minimization during the training of a deep neural model.

We note that this analysis shows that supporting multi-view deep learning in event streams poses specific challenges that are unaddressed in this study. Future investigations, that are out of the scope of this study, are required,

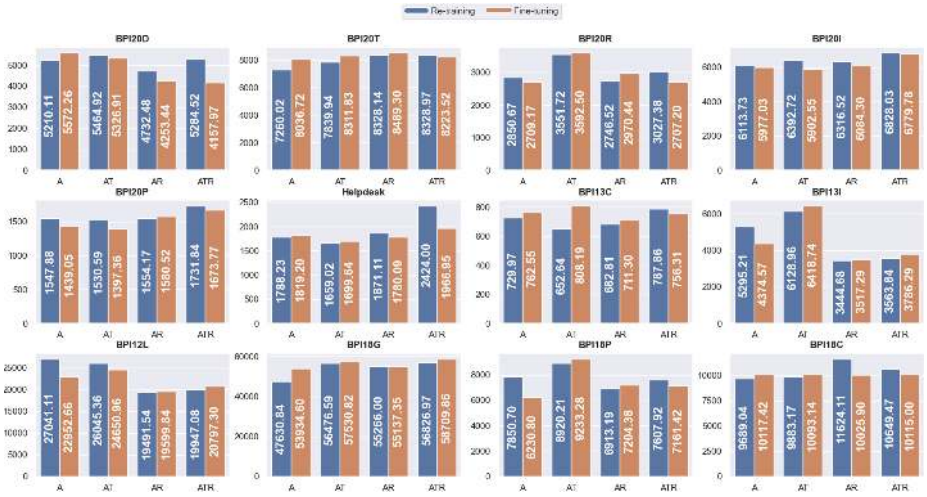


Figure 14: TIME of deep neural models trained with DARWIN (**Re-training** or **Fine-tuning** as learning strategy and **Word2Vec** as encoding strategy) with information enclosed in A, AT, AR and ATR

in order to achieve a solution able to support both the dynamic perspective selection and the adaptation in online neural models for PPM applications.

6.6.4. Route cause analysis (Q4)

We continue this study by examining Petri nets generated in correspondence of concept drifts 1 and 2 of *Helpdesk*. These Petri nets disclose information that may help in the route cause analysis of drifts. Figure 15 shows the Petri nets of the initialization step, drift 1 and drift 2. The Petri net of the initialization was generated from the traces recorded during the initialization step. The Petri nets of drift 1 and drift 2 were generated from the traces of the prefix traces selected by **ADWIN** for the concept drift adaptation of drift 1 and drift 2, respectively. All Petri nets were generated with Inductive Miner (run with the default filtering mechanism). The Petri net of drift 1 shows the disappearance of "Schedule intervention". On the other hand, the Petri net of drift 2 shows the re-appearance of "Schedule intervention" as an alternative of "Create Sw anomaly" and followed by a new activity path enclosing activities of "Resolved", "Invalid" and of "Verified", which were absent in the previous Petri nets. An

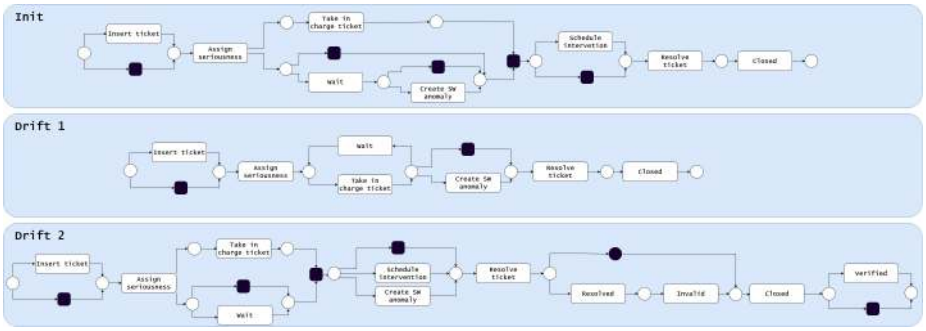


Figure 15: Petri net representation of the process models generated by Inductive Miner (with default filtering) after the initialization step and at the concept drift points 1-2 during the online learning of the *Helpdesk* stream

interpretation of this history may be that the execution of the activity "Schedule intervention" was suspended around drift 1 and re-introduced (as alternative of "Create Sw anomaly") around drift 2 after a revision of the ticket resolution pipeline in the business process. We can also deduce that something changed in process model at level of the ticket resolution pipeline.

Notably this analysis provides an example of how process discovery may be performed orthogonally to PPM as a means to explain concept drifts.

6.6.5. Memory usage study (Q5)

We analyse the average amount of memory consumed by DARWIN to record data during the online step. DARWIN was run in the default configuration with activity as perspective, **Fine-tuning** as learning strategy and **Word2Vec** as encoding strategy. Table 8 reports the mean and standard deviation of the memory (measured in KBytes) spent recording **D**, **A** and the window of **ADWIN** as a new event is acquired in the online step of DARWIN.

The results show that the average memory used is less than 4.5 MBytes in all the event streams. The more memory is consumed in *BPI13I*, *BPI12L*, *BPIC18G* and *BPIC18P*. By examining in depth the characteristics of these event streams, we note that they acquired the highest number of distinct variants (**V** in Table 5). In addition, *BPI13I* and *BPI12L* acquired the lowest

Table 8: Mean and Standard Deviation on the memory usage of DARWIN (in KBytes)

Event stream	**Mean ± Standard Deviation** (KBytes)
BPI20D	39.23 ± 44.03
BPI20T	48.08 ± 22.58
BPI20R	45.68 ± 44.85
BPI20I	53.42 ± 35.90
BPI20P	80.90 ± 59.81
Helpdesk	35.09 ± 19.22
BPI13C	16.23 ± 7.01
BPI13I	178.98 ± 104.56
BPI12L	4105.18 ± 2221.64
BPIC18G	569.03 ± 465.52
BPIC18P	716.64 ± 561.48
BPIC18C	103.98 ± 66.82

average number of distinct traces per variants ($\mathbf{T/V}$ in Table 5). So, as shown in [53], this family of event streams does not take actual advantage of the tree-based data synopsis integrated to record incomplete traces of the same variant. On the other hand, both *BPI12L* and *BPIC18G* acquired traces with the highest length in average ($\mathbf{T_{avgL}}$ in Table 5). This consideration on the trace length is also supported by the fact that the lowest average memory usage (i.e., less than 45 Kbytes in Table 8) is measured in *BPI20D*, *BPI20R*, *Helpdesk* and *BPI13C*, which recorded traces with the lowest average length ($\mathbf{T_{avgL}} \leq 5$ in Table 5). Finally, we note that the average duration of traces ($\mathbf{T_{avgD}}$ in Table 5) coupled to the average length of traces ($\mathbf{T_{avgL}}$ in Table 5) can also have an effect on the memory usage. In fact, despite *BPI13C* registered traces with high average duration ($\mathbf{T_{avgD}} = 179.2$ days in Table 5), keeping incomplete small traces in memory for long time does not lead DARWIN to use a high amount of memory since $\mathbf{T_{avgL}} = 4$ in *BPI13C*. On the other hand, *BPIC18G* and *BPIC18P* also registered traces with high duration in average, i.e., $\mathbf{T_{avgD}} = 143.5$ and $\mathbf{T_{avgD}} = 196$ days, respectively. However, these event streams acquired long traces, i.e., $\mathbf{T_{avgL}} = 20$ and $\mathbf{T_{avgL}} = 9$, respectively. This supports the conclusion that keeping longer incomplete traces in memory for longer time may cause an increase in the memory usage.

In short, this analysis shows that DARWIN is able to process streams with small memory consumption. In addition, it highlights that the highest amount of memory is consumed in presence of a high number of trace variants or traces that stay incomplete for long time.

6.6.6. DARWIN *vs data stream competitors (Q6)*

We compare the performance of DARWIN (run in the default configuration with the activity perspective, the **Fine-tuning** learning strategy and the **Word2Vec** encoding strategy) to that of: (1) Adaptive Hoeffding Tree (AHT) [11], (2) Adaptive Random Forest (ARF) [36], and (3) Streaming Random Patches (SRP) [38].

Both AHT and ARF are well-established stream mining algorithms that have been recently used in [49] to handle concepts drifts in outcome predictive process monitoring [49]. In this comparative study, we used both methods for next activity prediction. In particular, AHT learns incrementally an Hoeffding tree from a data stream by guaranteeing the Hoeffding bound and building alternate trees whenever changes are alerted in the data stream distribution. In addition, similarly to DARWIN, AHT integrates **ADWIN** for concept drift detection. ARF is an adaptation of Random Forest of Hoeffding trees to data streams. Random Forest has been recently tested for outcome prediction in [62]. ARF comprises a drift adaptation strategy that does not simply reset base models whenever a drift is detected. In fact, it starts training a background tree after a warning has been detected and only replaces the primary model if the drift occurs. Finally, SRP is a recent ensemble method designed specially to cope with evolving data streams. It combines random sub-spaces and online bagging to achieve competitive predictive performance in comparison with other methods. It integrates the Hoeffding tree as base model of the ensemble. Both ARF and SRP are not bounded to a specific detector. In this study, we used **ADWIN** as concept drift detector of both ARF and SRP as it was used in DARWIN, as well as in the reference studies [36] and [38], respectively. AHT, ARF and SRP were run with the optimal parameter set-up suggested in the reference papers.

Table 9: Fscore and TIME of deep neural models trained with default configuration of DARWIN, as well as tree-based and ensemble-based data stream models trained with AHT, ARF and SRP, respectively. The best results are in bold.

Stream	Fscore				TIME			
	DARWIN	AHT	ARF	SRP	DARWIN	AHT	ARF	SRP
BPI20D	**0.425**	0.397	0.402	0.415	5572.26	**77.02**	179.76	245.51
BPI20T	0.296	0.291	0.338	**0.341**	8036.72	**229.78**	582.37	667.84
BPI20R	0.334	0.353	0.351	**0.360**	2709.17	**33.86**	81.65	134.01
BPI20I	0.387	0.373	0.401	**0.414**	5977.03	**150.33**	359.09	482.28
BPI20P	0.411	0.403	0.451	**0.457**	1439.05	**11.64**	31.56	63.73
Helpdesk	**0.308**	0.285	0.306	0.299	1819.20	**13.77**	32.93	69.70
BPI13C	**0.424**	0.234	0.371	0.397	762.55	**2.19**	7.72	17.79
BPI13I	**0.520**	0.419	0.453	0.391	4374.57	**153.34**	243.25	392.78
BPI12L	**0.675**	0.452	0.637	0.644	22952.66	**1921.17**	3235.36	2942.48
BPIC18G	0.362	0.361	**0.402**	0.390	53934.60	**9746.56**	14453.67	14567.99
BPIC18P	**0.348**	0.304	0.341	0.336	6230.80	**612.04**	855.52	1043.50
BPIC18C	**0.484**	0.446	0.482	0.469	10117.42	**929.35**	934.17	1254.31

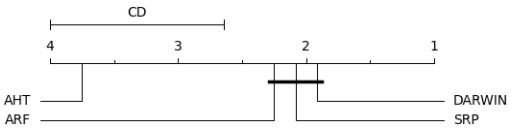


Figure 16: Comparison of Fscore of DARWIN, AHT, ARF and SPR with the Nemenyi test. Groups of strategies that are not significantly different (at $p - value \leq 0.05$) are connected.

Table 9 reports Fscore and TIME of DARWIN, AHT, ARF and SRP. The results show that DARWIN is more accurate than AHT. The only exception is observed in *BPI20R*. We recall that AHF is the related method that, similarly to DARWIN, trains an evolving classification hypothesis composed by a single model (specifically, an adaptive Hoeffding tree in AHF and a deep neural model in DARWIN). So, the higher accuracy of DARWIN compared to AHF provides the evidence of the ability of trainable deep neural models of taking advantage of (non-linear) multiple activation layers to facilitate the discovery of effective patterns that keep their effectiveness also under drifting conditions. On the other hand, DARWIN outperforms both ARF and SRP in seven out of twelve streams. Figure 16, that shows the ranking of the compared methods determined

with the post-hoc Nemenyi test on Fscore values, highlights that DARWIN, ARF and SRP statistically outperform AHT. In particular, DARWIN ranks in the first place with SRP as runner-up. By examining in depth the characteristics of the event streams, we note that ARF and SRP commonly outperform DARWIN in the event streams that record the higher number of distinct activities (**A** in Table 5). This behaviour supports the conclusion that an ensemble learning strategy may aid a classification hypothesis (so also a deep neural model) in gaining accuracy in event streams that exhibit a high variety of activities. This improvement may be achieved thanks to the ensemble ability of better covering the expected diverse data behaviour of a high variety of activities by keeping classification functions trained and adapted on diverse data sub-spaces. We recognize that the lack of an ensemble strategy may be a current limit of DARWIN in some complex scenarios.

Final considerations concern results of TIME, i.e., the computation time spent by DARWIN completing the elaboration of the entire stream. This time depends on the total amount of events processed. DARWIN spends more time than all related methods completing the processing of the entire stream. This is an expected outcome as training and adapting an Hoeffding tree, also within an ensemble system, is expected to spend less time than training and adapting a deep neural model. In any case, we note that in all experiments the TIME of DARWIN is significantly lower than the real duration of the processed (real) event streams (Σ_{totD} in Table 5), which varies between 165 days (in *BPI12L*) and 2332 days (in *BPI13C*). On the other hand, the TIME of DARWIN is compatible with a challenging stream scenarios, where events might arrive every second or even more frequently. In fact, the total TIME of DARWIN varies between 762.55 seconds (in *BPI13C*) and 53934.60 seconds (in *BPIC18G*), while the number of acquired events ranges between 6660 (in *BPI13C*) and 569209 (in *BPIC18G*).

In conclusion, despite this analysis shows that the time spent by DARWIN completing the elaboration of the entire stream is close to two orders of magnitude higher than that spent by the data stream-related shallow methods, the computation effort is still acceptable as the ratio of the computation time

spent processing all events in the stream on the number of events in the stream (i.e.TIME/**E**) is less than 0.1 seconds in all the event streams of this study. On the other hand, the higher computation time consumption is counterbalanced by the higher accuracy achieved by DARWIN in several experiments of this comparative study. In general, DARWIN would be preferred to AHT as it adapts a deep neural model that is more accurate than the single Hoeffding tree adapted by AHT. In addition, DARWIN would be preferred to ARF and SRP in presence of streams with a small variety of distinct activities. In any case, we are aware that this study paves the way for further investigations in the direction of neural model ensembles in event streams, in order to study ad-hoc strategies to keep with sub-spaces in deep neural model adaptations and allow deep neural models to gain accuracy also in presence of a high variety of activities.

6.6.7. Sensitivity study (Q7)

Finally, we explore the sensitivity of the accuracy performance of DARWIN (run in the default configuration with the activity perspective, **Fine-tuning** as learning strategy and **Word2Vec** as encoding strategy) to the set-up of δ (i.e., confidence of **ADWIN**), η (i.e., number of events processed in the initialization step), j (i.e., neighbourhood size in **Word2Vec**) and w (i.e., window size of padding). This analysis was performed with *BPI20D* and *BPI13I* that were selected for their different characteristics (e.g., number of activities, number of traces, percentage of traces per variants, average trace length). For each event stream, for each parameter, we ran DARWIN by varying the set-up of the selected parameter and adopting the default set-up for the three left-out parameters. Specifically, we performed experiments by varying δ among 0.001, 0.002 (default), 0.005 and 0.01; η among 5%, 10% (default) and 15% of the number of events in the entire stream; j among 3, 4 and 5 (default); w between 3, 4 (default), 5 and 6. Results of Fscore are reported in Figure 17 for both datasets.

The results achieved by varying δ show that the higher the value of δ, the lower the Fscore of DARWIN. In particular, the highest Fscore is achieved with

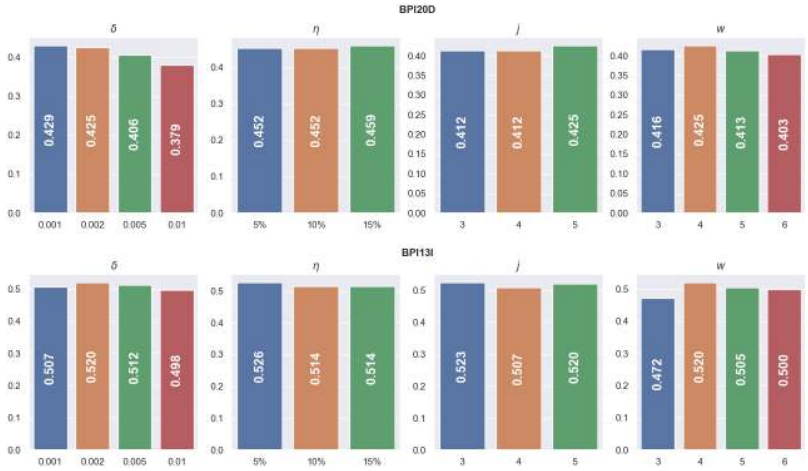


Figure 17: Fscore of DARWIN run in the default configuration in *BPI20D* and *BPI13I* by varying δ among 0.001, 0.002 (default), 0.005 and 0.01, η among 5%, 10% (default) and 15% of the number of events in the entire stream, j among 3, 4 and 5 (default) and w among 3, 4 (default), 5 and 6

the default $\delta = 0.002$ in *BPI20D* and $\delta = 0.001$ in *BPI13I*, respectively. The runner-up Fscore is achieved with default $\delta = 0.002$ also in *BPI13I*. The results achieved by varying η show that the amount of events processed during the initialization step has a small effect on the Fscore registered on predictions performed in the online step.[8] The **Fine-tuning** strategy allows the online process to adapt the model to data without being greatly affected by the amount of data processed in the initialization. The results collected by varying j shows that the highest performance is achieved with the default $j = 5$ in *BPI20D* and $j = 3$ in *BPI13I*, respectively. The runner-up Fscore is achieved with default $j = 5$ also in *BPI13I*. In any case, differences in Fscore by varying j are negligible in both datasets. Finally, the results collected by varying w show that the highest Fscore is achieved with the default padding window size $w = 4$.

[8] In the experiment performed to evaluate the effect of η on the accuracy performance of DARWIN we measured the Fscore of the predictions yielded in the online stage for the latest 85% of events in the stream. This allows us to perform a safe comparison with Fscore values measured on the same samples in the configurations with $\eta = 5\%$, 10% and 15%, respectively.

Notably, the results of this sensitivity study shows that DARWIN achieves the highest accuracy with the smallest values of δ. On the other hand, the accuracy of DARWIN is slightly conditioned by changing the set-up of parameters η, j and w.

7. Discussion and limitations

Experimental results showed the effectiveness of DARWIN to predict the next activity with deep neural models while taking into account eventual concept drifts in event streams. However, the method has some limitations that can be overcome by opening up interesting new avenues of research.

A first limitation of the proposed study is that we considered an insert-only stream model, where events can only be added in a monotonic fashion. As discussed in [14], scenarios where observed events can be changed or removed (i.e., insert-delete models) are yet to be considered. An interesting additional functionality for our proposed method is considering out-of-order events [30]. This would make DARWIN easier to apply in real-world scenarios. Indeed, we have already introduced some kind of delay (during re-training) before processing the newly coming events, therefore DARWIN can be easily extended to handle out-of-order events.

Another limit of the current version of DARWIN is that it can not distinguish among different types of concept drifts (gradual, sudden, reoccurring,...). Adding this capability would require to monitoring the drift's characteristics, such as speed and magnitude, inside each event stream. Moreover, the model learnt by DARWIN is unaware of so-called adversarial examples that may result in mis-classifications and thus evade the prediction. One possible solution to overcome this limitation is to combine ADWIN with an anomaly detection mechanism that post-processes the window in order to remove anomalies.

Another limitation of the proposed method is that it does not exploit additional information regarding changes in the search space. In this way, the performance of the method could be compromised by anomalous samples used

to adapt the current model. One possibility is to use information coming from different perspectives (views), as in [55] where we presented a deep learning method for predictive business process monitoring that processes multiple characteristics of an event as multiple views, taking advantage of the diversity of the feature set in each view. In its current version DARWIN is able to work with multiple views, but it can not adapt the predictive model to concept drifts related to the importance of perspectives. Indeed, evaluating the importance of each view in a data stream scenario is not an easy task and needs a careful analysis since the influence of each perspective could vary during time.

As a final remark, we observe that DARWIN adapts a single deep learning model (learner) to concept drift using adaptive windows. This may represent a limitation of the method when compared to a more effective strategy that adapts an ensemble of learners, which represent a mainstream of recent research in data streams [37]. In this case the adaptation consists in replacing some learners of the ensemble with new learners trained on different windows. We believe that the study of ensemble learning in combination with deep learning in event streams can be useful to empower the current strategy implemented in DARWIN.

8. Conclusion

In this paper, we presented DARWIN, a deep learning-based PPM method for next activity prediction in event streams. The proposed method takes advantage of a concept drift detection and adaptation approach to scale a deep neural model to online PPM scenarios. To this purpose, we use **ADWIN** both to monitor the conformance of upcoming activities to their predictions and to keep a selection of streamed events to update the deep neural model to the drifted data. The main strength of the proposed method is the adoption of the **Fine-tuning** strategy in combination with **Word2Vec** encoding strategy to adapt the deep learning model to the new activity data distribution. This avoids the need to restart training from the whole set of accumulated traces and ensures

that the model is always up-to-date.

The experiments performed using the Prequential Evaluation strategy on several event streams confirm the effectiveness of the proposed approach. In particular, they show that the dynamic learning strategies **Fine-tuning** and **Re-training** are systematically more accurate than the static learning strategy **Static** also thanks to their ability to predict occurrences of new activity types that appeared in the online learning phase, despite they were never observed in the initialization phase. However, this gain in accuracy is achieved at the cost of the more time spent detecting and handling drifts. Results also show that **Fine-tuning** can be commonly preferred to **Re-training** as it appears less subjected than **Re-training** to the catastrophic forgetting phenomenon. **Word2Vec** outperforms **One-Hot-Encoding** without any remarkable burden of computation and without assuming that all activities that will be acquired along the stream should be known apriori. In addition, DARWIN can process event streams with small memory consumption, and its accuracy performances are not significantly conditioned by the set-up of input parameters η, j and w. Instead, DARWIN achieves the highest accuracy with the smallest values of δ. The process discovery performed on traces involved in drift alerts may disclose useful information for the route cause analysis of drifts. Finally, DARWIN is more time consuming than the shallow data stream competitors AHT, ARF and SRP, but the greater computation effort is still acceptable for a streaming scenario. In fact, the ratio of the computation time spent processing all events in the stream on the number of events in the stream is less than 0.1 seconds in all the event streams of this study. In any case, the higher computation effort is often counterbalanced by the gain in accuracy. In fact, DARWIN adapts a deep neural model that is more accurate than the Hoeffding tree adapted by AHT. On the other hand, DARWIN is also more accurate than ARF and SRP, which adapt an ensemble of Hoeffding trees, in streams with a small variety of distinct activities. The study of ensemble learning in combination with deep learning in event streams may be a promising approach to gain accuracy also in presence of a high variety of activity.

Future works comprise exploring ensemble deep learning and multi-view deep learning for PPM in event streams. However, there are additional open points to be investigated for future works. An avenue for future work is the extension of the proposed online method to different PPM problems such as outcome, remaining time and sequence predictions. Another pathway of future research could be to integrate explanations in the process model generated by DARWIN, that can have a major effect on its usability for end-users. On the other hand, a recent study [57] has started the investigation of PPM coupled to process discovery, we plan to continue this investigation in the online scenario. Finally, due to ever-increasing diffusion of stream of event data collected in Internet of Things (IoT) applications [64], we plan to explore the performance of the proposed PPM algorithm in these applications for enabling IoT PPM.

Appendix A. Word2Vec

Word2Vec [50] is a word embedding approach that was originally formulated in the field of natural language processing to be used in text analysis applications (e.g., [4, 20]) to learn an embedding model of words by exploring a text corpus. It adopts two different neural network architectures, that is, Continuous-Bag-of-Words (CBOW) and Skip-gram (SG). Both architectures are able to model complex syntactic and semantic links between words, despite they apply different approaches. Specifically, CBOW uses a feed-forward neural network to predict a target word from the neighbored context, whereas SG predicts the neighbour words of a target word. In this study, we consider activities as words and traces as text sentences. Hence, we adopt the CBOW-based variant of **Word2Vec**, as its learning process is closer to the task handled in this paper. In fact, a feed-forward neural network able to predict a target activity from a context is appropriate for the next-activity prediction task.

Let us consider a sequence $\langle A_{t-j}, \ldots, A_t, \ldots, A_{t+j} \rangle$ that represents the neighbourhood of the target activity A_t with size $2j$, **Word2Vec** maps each context activity A_i to its one-hot-encoding representation $\mathbf{A_i}$ of size v. The value v rep-

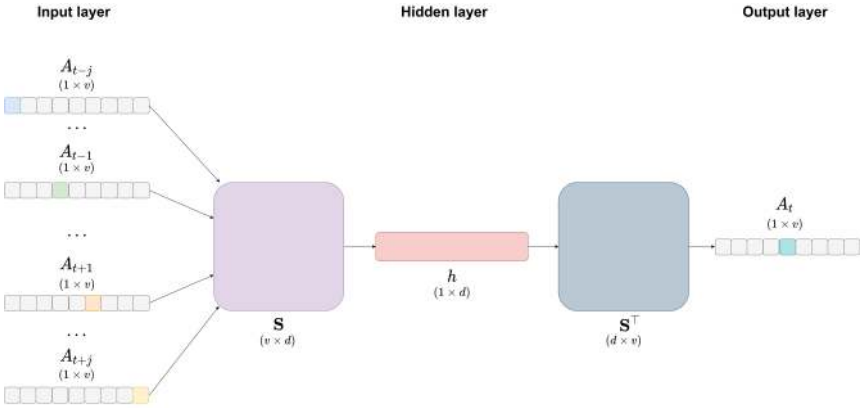


Figure A.18: CBOW model architecture of **Word2Vec**

resents the cardinality of the activity vocabulary. The CBOW neural network (see Figure A.18) learns the matrix $\mathbf{S} \in \mathbb{R}^{v \times d}$ where d is the expected size of the embedding space. The $2j$ one-hot-encoding vectors of the neighbour activities are then multiplied by $\mathbf{S}$. The obtained $2j$ vectors in $\mathbb{R}^d$ are averaged by the hidden layer, in order to compute the embedding of the target activity A_t. The hidden layer is computed as $h = \sum_{A_i \in [A_{t-j},\ldots,A_{t-1},A_{t+1},\ldots,A_{t+j}]} A_i \cdot \mathbf{S}$. The output layer is computed by multiplying the embedding of the target activity A_t by $\mathbf{S}^{\intercal}$. Therefore, the neural network is trained to reconstructs, in an accurate manner, the one-hot-encoding vector of the target activity A_t, given the one-hot-encoding representations of the context activities. Training is completed with the gradient descent to minimize the probability of wrongly reconstructing the one-hot-encoding representation of the target activity A_t, given the one-hot-encoding representation of the neighbour activities (i.e., to minimize the cross-entropy loss over the training set of sequences). The CBOW neural network is trained using sequences of $\mathcal{L}$. The obtained matrix $\mathbf{S}$ is leveraged to embed each activity of $\mathcal{L}$ into a d-dimensional numerical feature space.

Appendix B. ADWIN

ADWIN [10] is an adaptive, model-agnostic, sliding window approach to identify concept drifts in a numeric stream. It maintains summary statistics about the streamed data. The general idea is to keep event statistics (e.g., classification hypothesis performance) from a window **W** of an adaptive, variable size, while detecting concept drifts on the sequence of classifications errors CE recorded in the **W**. The inputs to **ADWIN** are: (1) a confidence value $\delta \in [0, 1]$ and (2) a (potentially infinite) sequence of classification errors. When each classification error value CE is added to the adaptive window **W**, **ADWIN** analyzes the classification error sequence $\pi_{CE}(\mathbf{W})$ inside **W** to detect concept drifts. To this aim, **ADWIN** performs a search on all possible combinations of two sliding sub-windows in **W**. If the distributions of classification errors in the two sub-windows are significantly different, i.e., the observed average of the classification error in both sub-windows is greater than threshold $\epsilon_{cut} = \sqrt{\frac{n_0 n_1}{2(n_0+n_1)} ln \frac{4n}{\delta}}$, being n the size of the window, n_1 and n_2 the size of the sub-windows [10], then **ADWIN** alerts a concept drift and removes the oldest triples from the adaptive window. This cut detection step recognizes as the adaptive window must be cut to remove the oldest triples. The cut detection is repeated by removing the oldest triples until the adaptive window does not alert a new concept drift. In this way, the summary statistics rely on an adaptive window of prefix traces since **ADWIN** shrinks the size of **W** by removing oldest triples as a concept drift is alerted. A detailed description of the **ADWIN** approach is reported in [10].

Acknowledgments

Vincenzo Pasquadibisceglie, Giovanna Castellano and Donato Malerba are partially supported by the project FAIR - Future AI Research (PE00000013), Spoke 6 - Symbiotic AI (CUP H97G22000210007), under the NRRP MUR program funded by the NextGenerationEU.

CRediT Authorship Contribution Statement

Vincenzo Pasquadibisceglie: Conceptualization, Methodology, Software, Data curation, Investigation, Validation, Visualization, Writing - original draft, Writing - review & editing. **Annalisa Appice**: Conceptualization, Methodology, Investigation, Validation, Supervision, Writing - original draft, Writing - review & editing. **Giovanna Castellano**: Conceptualization, Validation, Writing - original draft, Writing - review & editing. **Donato Malerba**: Conceptualization, Writing - review & editing.

References

[1] Aalst, W. v. d., Adriansyah, A., Medeiros, A. K. A. d., Arcieri, F., Baier, T., Blickle, T., Bose, J. C., Brand, P. v. d., Brandtjen, R., Buijs, J. et al. (2011). Process mining manifesto. In *International Conference on Business Process Management, BPM 2011, Proceedings* (pp. 169–194). Springer.

[2] Agrahari, S., & Singh, A. K. (2021). Concept drift detection in data stream mining : A literature review. *Journal of King Saud University - Computer and Information Sciences*, (pp. 1–18).

[3] Andresini, G., Appice, A., Ienco, D., & Malerba, D. (2023). SENECA: change detection in optical imagery using siamese networks with active-transfer learning. *Expert Syst. Appl.*, *214*, 1–15. doi:10.1016/j.eswa.2022.119123.

[4] Andresini, G., Iovine, A., Gasbarro, R., Lomolino, M., de Gemmis, M., & Appice, A. (2022). EUPHORIA: A neural multi-view approach to combine content and behavioral features in review spam detection. *Journal of Computational Mathematics and Data Science*, *3*, 100036.

[5] Andresini, G., Pendlebury, F., Pierazzi, F., Loglisci, C., Appice, A., & Cavallaro, L. (2021). INSOMNIA: towards concept-drift robustness in network intrusion detection. In N. Carlini, A. Demontis, & Y. Chen (Eds.), *14th*

ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2021, Proceedings (pp. 111–122). ACM.

[6] Appice, A., Di Mauro, N., & Malerba, D. (2019). Leveraging shallow machine learning to predict business process behavior. In E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, & K. Oyama (Eds.), *2019 IEEE International Conference on Services Computing, SCC, 2019 Proceedings* (pp. 184–188). IEEE.

[7] Barbon Junior, S., Tavares, G. M., da Costa, V. G. T., Ceravolo, P., & Damiani, E. (2018). A framework for human-in-the-loop monitoring of concept-drift detection in event log stream. In P. Champin, F. Gandon, M. Lalmas, & P. G. Ipeirotis (Eds.), *Companion of the Web Conference 2018, WWW 2018* (pp. 319–326). ACM.

[8] Basseville, M., Nikiforov, I. V. et al. (1993). *Detection of abrupt changes: theory and application* volume 104. prentice Hall Englewood Cliffs.

[9] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *NIPS* (pp. 2546–2554).

[10] Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *7th SIAM International Conference on Data Mining, Proceedings* (pp. 443–448). SIAM.

[11] Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In N. M. Adams, C. Robardet, A. Siebes, & J. Boulicaut (Eds.), *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Proceedings* (pp. 249–260). Springer volume 5772 of *Lecture Notes in Computer Science*.

[12] Böhmer, K., & Rinderle-Ma, S. (2018). Probability based heuristic for predictive business process monitoring. In *On the Move to Meaningful Internet Systems* (pp. 78–96). Springer.

[13] Bose, R. P. J. C., van der Aalst, W. M. P., Zliobaite, I., & Pechenizkiy, M. (2014). Dealing with concept drifts in process mining. *IEEE Trans. Neural Networks Learn. Syst.*, *25*, 154–171.

[14] Burattin, A. (2022). In W. van der Aalst, & d J. Carmona (Eds.), *Process Mining Handbook* (pp. 349–372). Springer volume 448 of *LNBIP*.

[15] Burattin, A., Cimitile, M., Maggi, F. M., & Sperduti, A. (2015). Online discovery of declarative process models from event streams. *IEEE Transactions on Services Computing*, *8*, 833–846.

[16] Camargo, M., Dumas, M., & Rojas, O. G. (2019). Learning accurate LSTM models of business processes. In T. T. Hildebrandt et al. (Ed.), *17th International Conference on Business Process Management, BPM 2019, Proceedings* (pp. 286–302). Springer volume 11675 of *LNCS*.

[17] Cameron, J. J., Cuzzocrea, A., Jiang, F., & Leung, C. K. (2013). Mining frequent itemsets from sparse data streams in limited memory environments. In *International Conference on Web-Age Information Management* (pp. 51–57). Springer.

[18] Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., & Alahari, K. (2018). End-to-end incremental learning. In *European Conference on Computer Vision, ECCV 2018, Proceedings* Lecture Notes in Computer Science. Springer.

[19] Chandak, M. B. (2016). Role of big-data in classification and novel class detection in data streams. *Journal of Big Data*, *3*, 1–9.

[20] De Martino, G., Pio, G., & Ceci, M. (2021). Prilj: an efficient two-step method based on embedding and clustering for the identification of regularities in legal case judgments. *Artificial Intelligence and Law*, (pp. 1–1).

[21] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, *7*, 1–30.

[22] Demšar, J., & Bosnić, Z. (2018). Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, *92*, 546–559. doi:https://doi.org/10.1016/j.eswa.2017.10.003.

[23] Di Mauro, N., Appice, A., & Basile, T. M. A. (2019). Activity prediction of business process instances with inception CNN models. In M. Alviano, G. Greco, & F. Scarcello (Eds.), *XVIIIth International Conference of the Italian Association for Artificial Intelligence, AI*IA 2019, Proceedings* (pp. 348–361). Springer volume 11946 of *Lecture Notes in Computer Science*.

[24] van Dongen, B. (2012). BPI challenge 2012. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

[25] van Dongen, B. (2020). Bpi challenge 2020. doi:10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.

[26] van Dongen, B., & Borchert, F. F. (2018). BPI challenge 2018. doi:10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972.

[27] Fahrenkrog-Petersen, S. A., Tax, N., Teinemaa, I., Dumas, M., de Leoni, M., Maggi, F. M., & Weidlich, M. (2022). Fire now, fire later: alarm-based systems for prescriptive process monitoring. *Knowl. Inf. Syst.*, *64*, 559–587.

[28] Ferilli, S., Redavid, D., & Angelastro, S. (2017). Activity prediction in process management using the woman framework. In P. Perner (Ed.), *17th Industrial Conference on Advances in Data Mining. Applications and Theoretical Aspects, ICDM 2017, Proceedings* (pp. 194–208). Springer volume 10357 of *Lecture Notes in Computer Science*.

[29] Fortino, G., Guzzo, A., Ianni, M., Leotta, F., & Mecella, M. (2021). Predicting activities of daily living via temporal point processes: Approaches and experimental results. *Comput. Electr. Eng.*, *96*. doi:10.1016/j.compeleceng.2021.107567.

[30] Fragkoulis, M., Carbone, P., Kalavri, V., & Katsifodimos, A. (2020). A survey on the evolution of stream processing systems. *CoRR*, *abs/2008.00842*. URL: https://arxiv.org/abs/2008.00842. arXiv:2008.00842.

[31] Francescomarino, C. D., Dumas, M., Federici, M., Ghidini, C., Maggi, F. M., & Rizzi, W. (2016). Predictive business process monitoring framework with hyperparameter optimization. In S. Nurcan, P. Soffer, M. Bajec, & J. Eder (Eds.), *28th International Conference on Advanced Information Systems Engineering, CAiSE 2016, Proceedings* (pp. 361–376). Springer volume 9694 of *LNCS*.

[32] Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman and Hall / CRC Data Mining and Knowledge Discovery Series. CRC Press.

[33] Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Mach. Learn.*, *90*, 317–346.

[34] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, *46*, 1–37.

[35] Garcia, K. D., Poel, M., Kok, J. N., & de Carvalho, A. C. P. L. F. (2019). Online clustering for novelty detection and concept drift in data streams. In P. Moura Oliveira, P. Novais, & L. P. Reis (Eds.), *Progress in Artificial Intelligence* (pp. 448–459). Cham: Springer International Publishing.

[36] Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfahringer, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Mach. Learn.*, *106*, 1469–1495. doi:10.1007/s10994-017-5642-8.

[37] Gomes, H. M., Montiel, J., Mastelini, S. M., Pfahringer, B., & Bifet, A. (2020). On ensemble techniques for data stream regression. In *2020 International Joint Conference on Neural Networks, IJCNN 2020* (pp. 1–8). IEEE.

[38] Gomes, H. M., Read, J., Bifet, A., & Durrant, R. J. (2021). Learning from evolving data streams through ensembles of random patches. *Knowl. Inf. Syst.*, *63*, 1597–1625. doi:10.1007/s10115-021-01579-z.

[39] Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., & Bengio, Y. (2014). An empirical investigation of catastrophic forgeting in gradient based neural networks.

[40] Guzzo, A., Rullo, A., & Vocaturo, E. (2022). Process mining applications in the healthcare domain: A comprehensive review. *WIREs Data Mining Knowl. Discov.*, *12*. doi:10.1002/widm.1442.

[41] Hassani, M. (2019). Concept drift detection of event streams using an adaptive window. In M. Iacono, F. Palmieri, M. Gribaudo, & M. Ficco (Eds.), *33rd International ECMS Conference on Modelling and Simulation, ECMS 2019, Proceedings* (pp. 230–239). European Council for Modeling and Simulation.

[42] Hinkka, M., Lehto, T., & Heljanko, K. (2020). Exploiting event log event attributes in rnn based prediction. In P. Ceravolo, M. van Keulen, & M. T. Gómez-López (Eds.), *Data-Driven Process Discovery and Analysis* (pp. 67–85). Cham: Springer International Publishing.

[43] Kim, J., Comuzzi, M., Dumas, M., Maggi, F. M., & Teinemaa, I. (2022). Encoding resource experience for predictive process monitoring. *Decision Support Systems*, *153*, 113669.

[44] Krempl, G., Žliobaite, I., Brzeziński, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M. et al. (2014). Open challenges for data stream mining research. *ACM SIGKDD explorations newsletter*, *16*, 1–10.

[45] Le, M., Gabrys, B., & Nauck, D. (2012). A hybrid model for business process event prediction. In *Research and Development in Intelligent Systems* (pp. 179–192). Springer.

[46] Leemans, S. J. J., Fahland, D., & van der Aalst, W. (2013). Discovering block-structured process models from event logs - a constructive approach. In J.-M. Colom, & J. Desel (Eds.), *Application and Theory of Petri Nets and Concurrency* (pp. 311–329). Springer Berlin Heidelberg.

[47] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, *31*, 2346–2363.

[48] Maaradji, A., Dumas, M., Rosa, M. L., & Ostovar, A. (2017). Detecting sudden and gradual drifts in business processes from execution traces. *IEEE Trans. Knowl. Data Eng.*, *29*, 2140–2154. doi:10.1109/TKDE.2017.2720601.

[49] Maisenbacher, M., & Weidlich, M. (2017). Handling concept drift in predictive process monitoring. In *2017 IEEE International Conference on Services Computing (SCC)* (pp. 1–8).

[50] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In Y. Bengio, & Y. LeCun (Eds.), *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*.

[51] Navarin, N., Vincenzi, B., Polato, M., & Sperduti, A. (2017). Lstm networks for data-aware remaining time prediction of business process instances. *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017*, (pp. 1–7).

[52] Ostovar, A., Maaradji, A., Rosa, M., Hofstede, A. T., & Dongen, B. V. (2016). Detecting drift from event streams of unpredictable business processes. In *ER: Conceptual Modeling*. Cham: Springer International Publishing.

[53] Pasquadibisceglie, V., Appice, A., Castellano, G., Fiorentino, N., & Malerba, D. (2022). Stardust: A novel process mining approach to dis-

cover evolving models from trace streams. *IEEE Transactions on Services Computing*, (pp. 1–14).

[54] Pasquadibisceglie, V., Appice, A., Castellano, G., & Malerba, D. (2020). Predictive process mining meets computer vision. In D. Fahland et al. (Ed.), *Business Process Management Forum - BPM Forum 2020, Proceedings* (pp. 176–192). Springer volume 392 of *LNBIP*.

[55] Pasquadibisceglie, V., Appice, A., Castellano, G., & Malerba, D. (2021). A multi-view deep learning approach for predictive business process monitoring. *IEEE Transactions on Services Computing (2021)*, .

[56] Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D., & Modugno, G. (2020). ORANGE: outcome-oriented predictive process monitoring based on image encoding and cnns. *IEEE Access*, *8*, 184073–184086.

[57] Pasquadibisceglie, V., Appice, A., Castellano, G., & van der Aalst, W. (2022). Promise: Coupling predictive process mining to process discovery. *Information Sciences*, *606*, 250–271.

[58] Pauwels, S., & Calders, T. (2021). Incremental predictive process monitoring: The next activity case. In A. Polyvyanyy, M. T. Wynn, A. Van Looy, & M. Reichert (Eds.), *Business Process Management* (pp. 123–140). Cham: Springer International Publishing.

[59] Polato, M. (2017). Dataset belonging to the help desk log of an italian company. doi:10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb.

[60] Pravilovic, S., Appice, A., & Malerba, D. (2014). Process mining to forecast the future of running cases. In A. Appice, M. Ceci, C. Loglisci, G. Manco, E. Masciari, & Z. W. Ras (Eds.), *New Frontiers in Mining Complex Patterns - Second International Workshop, NFMCP 2013, Revised Selected Papers* (pp. 67–81). Springer volume 8399 of *LNCS*.

[61] Rama-Maneiro, E., Vidal, J., & Lama, M. (2021). Deep learning for predictive business process monitoring: Review and benchmark. *IEEE Transactions on Services Computing*, (pp. 1–1).

[62] Rizzi, W., Francescomarino, C. D., Ghidini, C., & Maggi, F. M. (2022). How do I update my model? on the resilience of predictive process monitoring models to change. *Knowl. Inf. Syst.*, *64*, 1385–1416.

[63] Sahoo, D., Pham, Q., Lu, J., & Hoi, S. C. H. (2018). Online deep learning: Learning deep neural networks on the fly. In J. Lang (Ed.), *27th International Joint Conference on Artificial Intelligence, IJCAI 2018, Proceedings* (pp. 2660–2666). ijcai.org.

[64] Savaglio, C., & Fortino, G. (2021). A simulation-driven methodology for iot data mining based on edge computing. *ACM Trans. Internet Technol.*, *21*. doi:10.1145/3402444.

[65] Shahad, P., & Raj, E. D. (2021). Challenges in streaming data analysis for building an adaptive model for handling concept drifts. In *2021 International Conference on System, Computation, Automation and Networking (ICSCAN)* (pp. 1–6). IEEE.

[66] Steeman, W. (2013). BPI Challenge 2013, Ghent University, Dataset. doi:https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07.

[67] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A survey on deep transfer learning. In *International Conference on Artificial Neural Networks and Machine Learning, ICANN 2018, Proceedings*.

[68] Tax, N., Verenich, I., La Rosa, M., & Dumas, M. (2017). Predictive business process monitoring with LSTM neural networks. In E. Dubois, & K. Pohl (Eds.), *International Conference on Advanced Information Systems Engineering* (pp. 477–492). Springer.

[69] Teinemaa, I., Dumas, M., Rosa, M. L., & Maggi, F. M. (2019). Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Trans. Knowl. Discov. Data*, *13*.

[70] Verenich, I., Dumas, M., Rosa, M. L., Maggi, F. M., & Teinemaa, I. (2019). Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *ACM Trans. Intell. Syst. Technol.*, *10*.

[71] van Zelst, S., van Dongen, B., & van der Aalst, W. (2018). Event stream-based process discovery using abstract representations. *Knowl. Inf. Syst.*, *54*, 407–435.

[72] Žliobaitė, I., Pechenizkiy, M., & Gama, J. (2016). An overview of concept drift applications. *Big data analysis: new algorithms for a new society*, (pp. 91–114).