Next-Activity Prediction for Non-stationary Processes with Unseen Data Variability

Amolkirat Singh Mangat[1][0000−0002−7536−8833]
Stefanie Rinderle-Ma[2][0000−0001−5656−6108]

[1] Research Group Workflow Systems and Technology, Faculty of Computer Science, University of Vienna, Austria
amolkirat.singh.mangat@univie.ac.at
[2] Chair of Information Systems and Business Process Management, Department of Informatics, Technical University of Munich, Germany
stefanie.rinderle-ma@tum.de

Abstract. Predictive Process Monitoring (PPM) enables organizations to predict future states of ongoing process instances such as the remaining time, the outcome, or the next activity. A process in this context represents a coordinated set of activities that are enacted by a process engine in a specific order. The underlying source of data for PPM are event logs (ex post) or event streams (runtime) emitted for each activity. Although plenty of methods have been proposed to leverage event logs/streams to build prediction models, most works focus on stationary processes, i.e., the methods assume the range of data variability encountered in the event log/stream to remain the same over time. Unfortunately, this is not always the case as deviations from the expected process behaviour might occur quite frequently and updating prediction models becomes inevitable eventually. In this paper we investigate non-stationary processes, i.e., the impact of unseen data variability in event streams on prediction models from a structural and behavioural point of view. Strategies and methods are proposed to incorporate unknown data variability and to update recurrent neural network based models continuously in order to accommodate changing process behaviour. The approach is prototypically implemented and evaluated based on real-world data sets.

Keywords: Event Streams, Predictive Process Monitoring, Data Variability, Non-Stationary Prediction

1 Introduction

Predicting future states of processes is a highly desirable ability as it empowers organizations to transition from reactive to proactive measures [4]. Predictive Process Monitoring (PPM) exploits traces of event data emitted during the executions of processes to enable evidence based forecasting of the remaining time, the outcome, or the next activity of ongoing processes [7].

However, most of the proposed PPM methods have one critical aspect in common: they are heavily based on the assumption that processes are stationary [9,11]. Stationary processes refer to processes that remain unchanged over time. This enables one to make safe assumptions about the underlying data variability and permits to ignore potential issues when unseen data variability is encountered.

In practice, processes evolve over time, especially when humans are involved. Just as organizations undergo changes and require an overhaul of their processes, so can human behavior change the way to carry out tasks [10]. In both cases, this can lead to inevitable changes in the data variability. These changes can be observed in the event traces which can be encountered in two forms, i.e., as event logs (ex-post) or as event streams emitted and stored during runtime (online). Each event is expected to carry information such as the identity of a particular process instance, an activity label, and potentially context data collected and/or generated by an activity. Changes that could be observed in the event traces include the emergence of new activities, rearranged orders of activities, and new values of context data.

Moreover, in an online setting, we do not have the privilege of foresight; we cannot know all possible forms of data variability upfront that our model may encounter in future. This holds particularly true when we assume non-stationary processes; change is expected, but *when* a change will occur, *what* changes may occur, and *which* impact the changes may have remains unknown. This applies at least until the point in time unseen data variability is encountered. Reliable predictions require complete data. Incomplete data may involuntarily introduce faulty behavior to a prediction model leading to imprecise prediction results. Under these premises and the restriction that event data acquired over time is the only available source of evidence for the processe behavior, predicting the next-activity is a challenging task. The goal is to enable robust, evidence-based next-activity predictions when we can solely rely on (incomplete) event data.

This paper addresses these challenges through the overarching question of

How to predict the next activity for an observed event assuming non-stationary processes in an online setting, i.e., during runtime?

We tackle this question based on the following contributions:

- We discuss potential challenges and pitfalls when dealing with unseen future data that will be utilized as input to predict the next activity.
- We discuss the requirements to build evidence-based robust prediction models under an online setting with non-stationary processes
- We propose and analyze greedy strategies for updating the prediction model and handling unseen event attribute values that serve as input for the prediction model.
- The strategies are prototypically implemented and applied to several real-world data sets. In addition to the feasibility of the strategies, the evaluation results show that considering more historic data does not necessarily lead to

significantly better prediction results. This insight is useful for finding the sweet spot between effort and output quality.

Section 2 discusses related work. Section 3 sets out challenges and requirements and Sect. 4 presents the approach for next activity prediction in an online setting. Section 5 provides an experimental evaluation of the approach. Section 6 discusses the approach and concludes the paper.

2 Related Work

The majority of predictive process monitoring approaches operates in an offline manner, i.e., on process execution logs [12]. For online settings, predictive process monitoring literature distinguishes between dynamic and static prediction approaches. A *static approach* is proposed by [9]. In detail, an annotated transition system processes the observed process sequence and selected data attributes to derive a prediction. Activities that are unknown at runtime are dropped from the input before passing them to the prediction model. *Dynamic approaches* are presented in [6,8,12]. [6,12] investigate various prediction model update strategies based on decision trees in order to handle unseen process behavior. [8] investigate incremental learning strategies for neural networks by reusing existing prediction models and adapting them with newly observed training data. Our approach differs from [6,8,12] in several ways: i) we include more if not all available attributes in addition to the activity label and the timestamp; ii) we consider the full (partial) traces as input for our prediction model in contrast to limiting the input to a fixed number of the most recent events; iii) we use recurrent neural networks; iv) we do not assume any knowledge about future data not seen by the prediction model at training time, and v) we utilize different model update strategies. Moreover [12] investigate update strategies over selected time periods and [8] investigate strategies on a monthly basis. By contrast, the approach at hand, investigates update strategies on a daily basis.

3 Challenges and Requirements

Our focus is on process-oriented data. Let P be a process model that comprises a set of activities a_i $(i = 1, \ldots, n)$ which are executed in a particularly coordinated fashion. Each activity can be augmented by any number of attributes which carry information specific to this activity. At runtime, process instances are created and executed based on P. The progress of an ongoing process instance σ_j is reported by emitted events e_i that carry the *case identifier* of a process to which the event belongs, a *timestamp*, and activity specific data that includes the label of an activity and optionally activity attributes together with their values. With the help of the case identifier of incoming events we can distinguish between different process instances and thus determine an event's affiliation to one particular process instance. The timestamps enable us to reconstruct the order in which activities have been carried out. Let c represent a case identifier.

Then an ordered sequence of events e_i^c sharing the same case identifier is referred to as *trace* $\pi^c = \langle e_1^c, e_2^c, \ldots, e_n^c \rangle$. Since an event's purpose is characterized by its activity label, we use the terms *event* and *activity* interchangeably from here on. For the next-activity prediction task, events and traces are the fundamental resources for evidence-based predictions.

3.1 Data Variability

In the following, we discuss at which levels data variability might occur in the context of processes, i.e., through changes on the *meta* and *value* level.

- **Meta level** changes include changes to the structure of an event. The three possible scenarios include *(M1)* the removal of existing attributes, *(M2)* the inclusion of new attributes, and *(M3)* the alteration of the reference name of an existing attribute.
- **Value level:** We distinguish between two data types for attributes, i.e., categorical and continuous values. Categorical values represent values limited to a finite set of numeric or non-numeric values (e.g., the name or identifier of an activity or resource). Continuous values represent numeric values that can potentially take on any value from an infinite set of numeric values (e.g., a temperature measurement obtained from a sensor). Value level changes refer to deviations of attribute values from values observed in the past.

Deviations may include the absence or change in the frequencies of known values, and the appearance of previously unseen values. In literature such deviations are also referred to as *concept drifts* [5], including recurring or seasonal drifts, incremental drifts reflecting small changes, sudden drifts reflecting completely new process behavior, and gradual drifts reflecting gradually deviating process behavior.

The implications of meta/structural and value/attribute level deviations are manifold. Structural ad-hoc changes e.g. by the exclusion of an expected attribute can lead to a system malfunctioning if the system is not designed to dynamically adapt to change. Value level changes may similarly impair a system's behavior when no mechanisms are in place to handle unseen data.

In this paper, we focus on value level data variability and assume no meta level changes occur. We assume that we only have access to event logs or event streams to derive the information required to build a prediction model.

3.2 Challenges of Evidence Based Predictions with Unseen Activities and Sequences

An event in the context of process execution is primarily characterized by its type, i.e., *which* activity has occurred, and *when* it has occurred. Not knowing all types of events and the order in which the events could potentially occur removes the ability to prepare a model beforehand that can handle future unseen data.

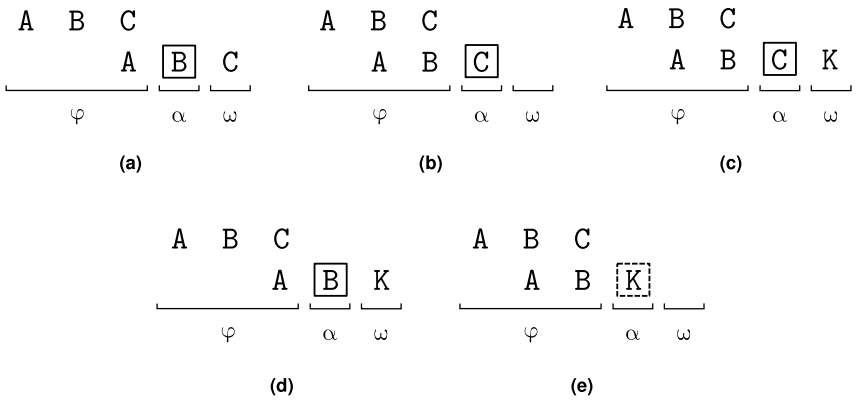

Fig. 1: Challenges when predicting next activities with unseen events. φ represents the set of past known events, α the set of currently observed events for which we want to predict the next activity and ω the set of future unknown events. The box with the solid border around the events under α indicates for which event we can attempt to predict whereas the dashed box indicates we cannot make a prediction as the event has been observed for the first time.

Figure 1 illustrates scenarios where not all event types that could occur and the order in which those event types could occur are known upfront.

For scenarios $(a) - (e)$, we assume that the sequence of activities $A > B > C$ has been observed, where $>$ denotes the order in which the activities have been observed. In the above case activity B follows A and activity C follows B. The time frame which includes all observed events is marked by φ. We assume that all events in φ have been completed. The time frame α represents the currently observed events. For them the aim is to predict the next activity, while also taking into consideration predecessors of the events. Furthermore the prediction process considers all observed events in φ. Lastly the time frame ω represents the future and thus events that will follow the events observed in the present, but which are unknown during α.

Next we will address each of the scenarios (a) – (e) and their potential pitfalls. In scenario (a), we can predict the next activity for the observed sub sequence $A > B$. Based on the past observations, C would be the only logical candidate and correct choice with regards to ω. In scenario (b), the logical candidate would be to predict the termination of the process, which would coincide with the past as no follow-up event after C was observed. Scenario (c) depicts a similar starting setup as in Scenario (b) except that in ω event K will follow. Unaware of the future and not having observed this sequence pattern in the past, the logical candidate for the next activity prediction would be the termination of the sequence, which is incorrect. Scenario (d) depicts a similar initial setup as in scenario (c), where again based on the past observation the most likely candidate to follow B be is C. However, this again would be an incorrect prediction, since

a pattern that includes K was not observed in φ. Finally Scenario (e) illustrates the case where an event with an unseen activity is encountered. Having no prior knowledge about the unseen activity and thus consequently about the observed partial trace, making an prediction with incomplete adds a certain risk to the validity of the prediction result.

3.3 Uncertainty of the Final Activity of a Trace

Another challenging task is to predict the end of a sequence. Besides predicting the next activity, predicting the end of a process is vital and necessary in order to establish a proper termination point for a sequence of activities. We can only predict the end of an ongoing process if we know what the possible termination states are. For our scenario, where only event logs and streams are available, we differentiate between the following three cases, ordered from high to low confidence with regards to the knowledge of the possible termination states. Let O represent a set of observed activities and T a set of activities observed at the end of sequences to which we refer as *termination activities*.

C1 There exists a subset of termination activities that only appears at the end of sequences and is never preceded by any other activity including the termination activities.

$$a_1 \to \cdots \to a_n \to t \quad a_i \in O, t \in T, O \neq T \tag{1}$$

C2 There exists a subset of termination activities that can be preceded and superseded by any activity including termination activities. The termination of a sequence however only occurs after a termination candidate.

$$a_1 \to \cdots \to a_n \to t \quad a_i \in O, t \in T, T \subset O \tag{2}$$

C3 Every activity can occur at any place in the sequence and can be a candidate for the sequence termination.

$$a_1 \to \cdots \to a_n \to t \quad a_i \in O, t \in T, T \subseteq O \tag{3}$$

As mentioned above each event carries a case identifier and a timestamp. With this information we can determine to which trace an event belongs and in which order the events have been observed and thus presumably in which order activities of a trace have been carried out. Thus the event with the most recent timestamp of a trace is considered as the final event that occurred before the trace terminated.

Under this premise, *C1* is the simplest case as we are ensured that certain activities only appear at the end of a sequence. This should lead to prediction models with the most accurate prediction results as there is a clearly distinguishable set of activities which solely appear at the end of sequences. *C2* faces a more complicated challenge: after we have reached a termination candidate we need to answer the question: *Do we predict the end or do we predict another*

follow-up activity? While Case *C2* has the confidence that predicting the end is a valid option, *C3* cannot provide any assurances in this regard as any activity can be a candidate for the last activity of a sequence.

Case C3 corresponds to total randomness where over time constantly new activities are added or even dropped and new sequences appear such that no patterns can be established. This makes it difficult or impossible to make any reliable predictions. C2 is a more likely scenario where occasionally changes might emerge in addition to established patterns which gradually may change over time. In this paper we assume Case C2.

4 Approach

This section presents data encoding strategies for unseen data variability and an approach for updating the prediction model for next activity prediction with unseen data variability.

4.1 Data Encoding with Unseen Data Variability

Data encoding is the bridge that enables prediction models to utilize raw event logs and stream data. In other words, the occurring event data must be transformed into the structure required by the model. Typically a prediction method has certain requirements for the data encoding whereby certain limitations can be imposed on how we can represent input data for models.

As mentioned in Section 3.1, we differentiate between categorical and continuous values. The traditional and widely used approach is to encode categorical features using dummy variables, where each variable for a binary feature can either take the value 0 or 1 to encode the absence or presence of a category respectively. For features with $K > 2$ categories this technique is extended to *1 of* K or *one-hot* feature encoding [2]. Here a feature is represented by a binary vector with K variables, where all the variables take the value 0, while the variable that represents the category we intend to encode takes the value 1. Assuming a set of three categories $\{Q, R, S\}$ we then can uniquely represent each category by the vectors $(1, 0, 0), (0, 1, 0)$ and $(0, 0, 1)$ respectively.

However, one-hot encoded features can lead to sparse representations, especially if the number of categories increases. A more compact variation for binary feature encoding is achieved by applying an *ordinal encoding* scheme [2]. In this case, each category of a feature is mapped to a cardinal number. Then the mapped number of the category is represented in the base-2 numeral system to encode a binary vector. Assuming that the categories $\{Q, R, S\}$ are indexed from 1 to 3 in ascending order, we then can represent the categories with a vector of size two as $(0, 1), (1, 0)$ and $(1, 1)$ respectively.

Continuous values can technically be encoded as is. However this typically depends on the prediction technique applied. The prediction technique might be subject to, e.g., only permitting integers or requiring normalization in order to achieve an overall homogeneous representation with respect to other attributes

for the input data. Alternatively, one can encode a continuous value similarly to the approach for categorical values by organizing the values into bins. The time elapse since the start of a process case, for example, can be modeled into bins of a day, a week or a month and so on.

For unseen data the challenge is that we do not have a mapping based on previously observed data that translates it into a representation a prediction model understands. To tackle this challenge, we propose two strategies, namely using a *void category* and introducing additional *reserve capacity* in the prediction model, that can be utilized with existing encoding techniques to handle unseen data variability and to enable prediction models to operate without immediate interruption and to facilitate faster updates for prediction models.

Void Category The purpose of the *void category* strategy is to serve as a placeholder for arbitrary input unknown to us. Let V^a represent a collection of known data values for an attribute $a \in A$ and M the set of transformed and encoded values m and $m^* \notin M$ representing the void category. Then $v_i^a \rightarrow m^*$ represents the void mapping function with $v_i^a \notin V^a$. This allows to keep the structure of a prediction model as is and enable the model to keep operating with unseen data.

We define the void category as the zero vector and in addition to the known set of categories $\{Q, R, S\}$ we observe $\{T, U\}$ not known to our prediction model. Then T and U are mapped to $(0, 0, 0)$ when using the one-hot encoding or to $(0, 0)$ when using the ordinal encoding scheme.

Reserve Capacity Another strategy is to integrate additional *reserve capacity* Ψ for every attribute of the input. Additional slots are added to the binary vector for future unseen data categories. Since the additional reserve capacity is embedded into the input structure, the prediction model must not undergo major structural changes. This enables to incrementally update existing models. For minimal alteration it is necessary to keep the order of the encoded attributes and their categories in the vector representation.

The challenge here is how to pick the additional capacity size and to decide when to further increase the capacity at runtime. Assuming the same set of known categories $\{Q, R, S\}$ for an attribute and the reserve capacity to be $\Psi = 2$ the resulting vector length for the one-hot encoding scheme is 5 (3 known categories + 2 reserve slots) and in the case of the ordinal encoding scheme the vector length is 3 (as 3 binary digits can encode 8 numbers from 0-7 which covers the overall required capacity of 5 binary numbers.) The corresponding one-hot encoded representation for Q, R, S would be $(1, 0, 0, 0, 0), (0, 1, 0, 0, 0)$ and $(0, 0, 1, 0, 0)$. A new category T could then be encoded at the next unoccupied slot $(0, 0, 0, 1, 0)$. For the ordinal encoding scheme this results in the vectors $(0, 0, 1), (0, 1, 0)$ and $(0, 1, 1)$ for Q, R, S respectively, whereas T would be represented by $(1, 0, 0)$ assuming T is indexed by 4.

For the void category strategy, which has its practical use in an online setting, it is necessary that a prediction model can continue to operate despite unseen

data values. By contrast, the reserve capacity strategy aims to enable faster prediction model update by preserving parts of the model learned from the past. Through the introduction of the additional capacity for newly observed value types, this should result in additive changes to the model and thus to faster convergence during the model training. In this paper, we focus on the void category strategy, leaving the reserve capacity strategy for future work.

4.2 Determining the Termination of a Sequence

To determine the final event of an observed trace that leads to termination, we introduce an additional attribute for each event that keeps track of the lifecycle of the trace. Existing approaches such as [1,14], by contrast, require an additional event/activity that indicates the end of a case. The trace lifecyle attribute keeps track of the current progress of an instantiated process and is set by the underlying process execution engine. The first activity's event will carry the value 'START' whereas the final activity's event will hold the value 'END' for the trace life cycle attribute.

4.3 Online Prediction Process

The online next activity prediction approach proposed in this work is depicted in Figure 2 and comprises three repeating sub processes. The first sub process includes activities *Read Event Stream* followed by *Aggregate/Map Events into/to Traces* π^c which processes incoming events and organizes the events into traces with the help of an event's case identifier and timestamp and persists them into a database denoted as Π. The second sub process consists of activities *Detect Data Variability* and *Trigger Prediction Model Update* which actively monitors deviations in Π and initiates the process to update a prediction model $\mathcal{M}_s$, both based on a model update strategy $\mathcal{S}$. A new model $\mathcal{M}_{s+1}$ is built using the update function $\mathbf{h}(\Pi, \theta)$ where θ prescribes a set of rules regarding which traces of Π to consider. Examples for θ include applying a sliding window or expanding window approach. In the sliding window approach one only includes traces that are within a time frame (e.g., traces observed within the last month from today). An expanding window approach includes all future traces since a particular point in time in the past. The third sub process is composed of the activities *Query Running Traces* and *Predict Next Activities with* $\mathcal{M}_{s+1}$. It predicts a subsequent activity with regards to the last activity registered in a trace for all incomplete (running) traces with the most recent prediction model $\mathcal{M}_{s+t}$ available.

4.4 Prediction Model Update Strategies

In this paper, we consider and analyze three greedy prediction model update strategies $\mathcal{S}_1$–$\mathcal{S}_3$ illustrated in Algorithms 1-3 respectively. These strategies enable us to address changes in activities and sequences and facilitate investigating the impact of unseen activities and sequences on the prediction results.

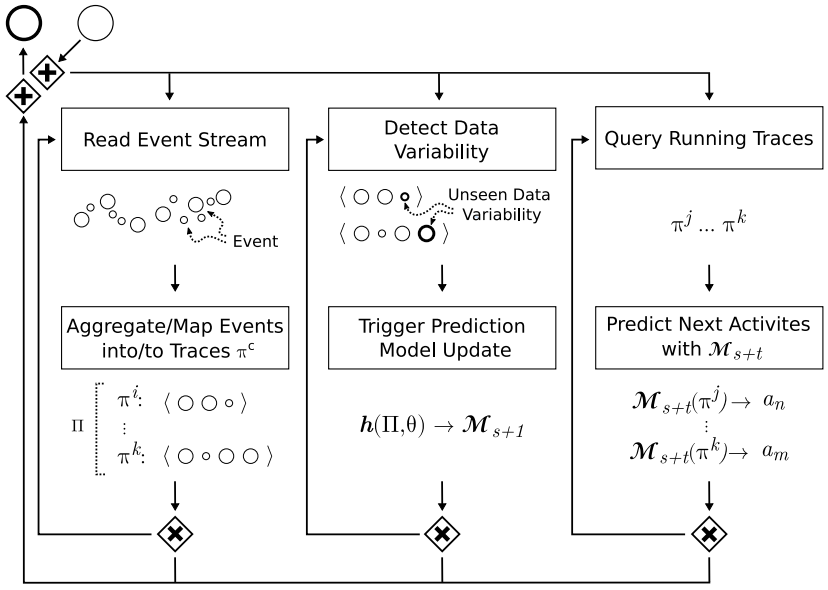

Fig. 2: Online next activity prediction approach.

The first strategy $\mathcal{S}_1$ described by Algorithm 1 employs a periodic update strategy where the time span between updates is controlled by δ. This strategy does not consider deviations in the observed collection of traces Π. For all strategies θ determines which part of the historical data in Π is considered for the prediction model update. $\mathcal{S}_1$ is suitable for stationary processes where change in the process behavior and data is not expected.

Algorithm 1 *$\mathcal{S}_1$: Periodic Updates*

```
1: Input: Π, θ, δ ← time until next update
2: t ← now()
3: M ← h(Π, θ)
4: while true do
5:   if t + δ ≤ now() then
6:     M ← h(Π, θ)
7:     t ← now()
8:   end if
9: end while
```

In contrast to $\mathcal{S}_1$, strategies $\mathcal{S}_2$ and $\mathcal{S}_3$ base their decision to trigger a model update when unseen variability in Π is detected. Both $\mathcal{S}_2$ and $\mathcal{S}_3$ ensure unseen activities and sequences respectively are incorporated into the prediction model which exposes the model to new patterns and improves the ability to make evidence based predictions. Thus these strategies are suited for non-stationary processes. $\mathcal{S}_2$ actively monitors for activity labels in Π that are unknown to

Algorithm 2 *$\mathcal{S}_2$: Update on new Activity*

```
Input: Π, θ, Ω
M ← h(Π, θ)
t ← ∞
while true do
  if Π has unseen activities then
    t ← schedule next update with policy Ω
  end if
  if t ≤ now() then
    M ← h(Π, θ)
    t ← ∞
  end if
end while
```

Algorithm 3 *$\mathcal{S}_3$: Update on new Sequences*

```
Input: Π, θ, Ω
M ← h(Π, θ)
t ← ∞
while true do
  if Π has unseen sequences then
    t ← schedule next update with policy Ω
  end if
  if t ≤ now() then
    M ← h(Π, θ)
    t ← ∞
  end if
end while
```

the prediction model $\mathcal{M}$ while $\mathcal{S}_3$ actively monitors for new sequences, i.e. new arrangement of activity sequences. $\mathcal{S}_2$ and $\mathcal{S}_3$ require an update policy Ω which determines *when* to update a model if new activities or activity sequences are encountered. Such a policy could, for example, dictate to update a model immediately, after at least K number of new occurrences have been observed since the last update, or, at the end of a day.

5 Experiments

We evaluate model update strategies $\mathcal{S}_1$–$\mathcal{S}_3$ (cf. Section 4.4) on several data sets using an expanding window approach in combination with the proposed void category encoding for unseen data variability. We predict the next activities on a daily basis in order to simulate an online setting. The underlying assumption is that strategies leading to more frequent updates are expected to perform better as more data is available during the training of the prediction model. However, how much improvement can be expected and at what cost the improvement is acquired is unknown and thus the subject of investigation. In the following, we describe the experimental setup in more detail and present the results of the evaluation. The source code for the experiments is available on GitHub[3].

5.1 Online Prediction Simulation and Model Update

For the periodic update strategy $\mathcal{S}_1$ we use a delay δ of 24 hours and infinity. The latter case equates to training a prediction model once and is utilized as our baseline. We predict the next activity for all events observed on the next day outside the past window. The past window considers all observed events prior to the prediction day. Once all the next-activity predictions have been made for the events in the prediction window, the past window is expanded by including the events in the prediction window. These steps are repeated until all events of the data set have been processed. For the update strategies $\mathcal{S}_2$ and $\mathcal{S}_3$ we use

[3] `https://github.com/auroeur/kronos`

a similar approach, except the past window is expanded only after a criterion (i.e., unseen activities or activity sequences) is met and after all events in the prediction day window have been processed. If the activity of a newly observed event is unknown, no prediction for the next activity is made. This applies to all update strategies. A prediction model update is only done after the window has been expanded.

5.2 Datasets

Helpdesk Dataset The Helpdesk[4] data set comprises of event logs of a ticketing management process of an Italian software firm. Overall the data set includes 20777 events that are part of 4580 distinct process instances. The data was collected between 2010-01-13 and 2014-01-03. The ticketing management process comprises of 14 distinct activities. Each activity includes up to 12 attributes as additional context data such as the resource, customer and severity of the issue.
BPI 2012 The BPI2012[5] data set has been released as part of the Business Processing Intelligence Challenge (BPIC) 2012. This data set is a collection of event logs from a Dutch financial institute concerning processes for loan applications. The log consists of 262220 events distributed among 13087 cases. Each process instance comprises of three sub-processes that include automatic and manual tasks carried out by humans. The data set includes process cases from 2011-10-01 till 2012-03-14 with 24 unique activities and two attributes including the resources handling the applications and the applied loan amount.
Sepsis The Sepsis[6] data set comprises of anonymized real-life event logs collected over a time span of almost two years that tracks the pathway in a hospital of patients with sepsis. The log consists of 15214 events distributed among 1050 cases. Overall the dataset consists of 16 distinct activities which can include up to 30 attributes.

5.3 Data Pre-Processing

For all data sets, we omit the life cycle information by discarding events that track the start and intermediate states of an activity. We only consider events marked as complete. As addressed in Section 4.2, we in addition augment events with information that signals if an event marks the first, an intermediate or the final activity of a trace. Furthermore, we consider all non-redundant trace and event level attributes found in the data sets in addition to the activity label and timestamp attributes as features for the prediction models.

For every data set, we split the event logs by taking the first 10% of the events to build the initial prediction model and we treat the remaining events as the source for future incoming events. We include all events that occur on the same day as the last event of the initial split.

[4] `https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb`
[5] `https://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
[6] `https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`

For the prediction model training, the already observed events are aggregated into traces with regards to matching case identity numbers. From these (in-)complete traces we incrementally generate $n-1$ partial traces/sequences of events where n represents the number of observed events belonging to the same trace. The smallest trace has size 1. The order of the events in a trace is determined by the timestamp of the event. As the prediction label we use the event activity that would naturally follow the incomplete partial trace.

5.4 Event Data Encoding

We encode categorical and continuous values of a trace as a binary vector using the base-2 numeral system. Unknown attribute values are encoded using the *void category* strategy for which we use the zero vector encoding.

In addition we include an attribute 'delay' for every event as a categorical feature which describes the time passed since the first event of the joint trace. The attribute 'delay' is subdivided into the following categories: an hour, four hours, eight hours, a week, 2 weeks, 3 weeks, a month, 2 months, 3 months and beyond 3 months.

5.5 LSTM Neural Networks and Model Training

As the prediction method we use a variation of Long-Short-Term Memory (LSTM) [3] recurrent neural network proposed in [13]. LSTM in general have shown to be well-suited for sequential data such as process events as demonstrated in [1,14].

For the LSTM we use 16 hidden cells with a single layer. The output nodes of the LSTM are first fed into a batch normalization layer and then are passed onto a fully connected layer. The input and output size of the network is dynamically set during the prediction model training and depends on the data observed. As the loss function we use cross-entropy since we are dealing with a classification problem where we want to predict the label of the next activity based on the partial trace observed out of K classes. As the learning algorithm we use Adam with a step size of $1e^{-3}$ and epsilon set to $1e^{-8}$. The prediction models are implemented with PyTorch and trained on the CPU. All experiments were conducted on a Fedora 32 system with an AMD Ryzen 5 3600 6-Core CPU and 32 GB memory.

5.6 Model Selection

We split the events of the past window by first grouping them into traces with the help of the case identifier and then taking the first 80% for training and the remaining 20% for validating prediction models. The training set is used to train the prediction models and the validation set is used to determine when to stop the training procedure in order to avoid the models over-fitting the underlying data and to ensure the model is capable of generalizing for unseen data variability. Overall we train a model at most for 16 epochs with a batch

size of 4. We stop the training early if the prediction model accuracy does not improve by 1 percentage point over the last 5 consecutive epochs. We adopt the most recent prediction model that surpassed the improvement threshold.

5.7 Results

Table 1 summarizes the results for the experiments. We report the accuracy (ACC), Matthews correlation coefficient (MCC), F1 as the weighted average of the per activity label F1 scores w.r.t to a label's support (i.e., the number of occurrences of an activity label), precision (PR) and recall (RC) scores. These scores include the results of next-activity predictions post the initial past window. The column *#P* represents the number of predictions made, whereas *#NP* the number of times a prediction has not been made due to an unseen activity label. The column *#Model* depicts the number of prediction models trained after an update is triggered and *#Time* the cumulative time spent training models for a strategy over the entire time span of a data set.

The suffix of a data set represents the update strategy applied: $\mathcal{S}_1^{\delta=\infty}$ does not update models, $\mathcal{S}_1^{\delta=24h}$ updates models every 24 hours, $\mathcal{S}_2$ updates models when encountering unseen activities, whereas $\mathcal{S}_3$ triggers model updates when unseen activity sequences are detected. The bold scores represent the best scores obtained for the respective metric.

Data Set-$\mathcal{S}_i$	ACC	MCC	F1	PR	RC	#NP	#P	#Models	Time
Helpdesk-$\mathcal{S}_1^{\delta=\infty}$	0.837	0.794	0.818	0.845	0.837	346	17992	1	0:00:11
Helpdesk-$\mathcal{S}_2$	0.914	0.89	0.908	0.907	0.914	14	18324	6	0:03:14
Helpdesk-$\mathcal{S}_3$	**0.931**	**0.912**	**0.926**	**0.923**	**0.931**	14	18324	334	4:34:59
Helpdesk-$\mathcal{S}_1^{\delta=24h}$	0.928	0.908	0.923	0.919	0.928	13	18325	1253	15:14:58
BPI2012-$\mathcal{S}_1^{\delta=\infty}$	0.736	0.722	0.693	0.716	0.736	286	113691	1	0:01:02
BPI2012-$\mathcal{S}_2$	0.742	0.729	0.699	0.724	0.742	286	113691	1	0:01:21
BPI2012-$\mathcal{S}_3$	**0.778**	**0.765**	**0.756**	**0.782**	**0.778**	286	113691	128	14:17:45
BPI2012-$\mathcal{S}_1^{\delta=24h}$	0.777	0.764	0.754	0.781	0.777	286	113691	144	16:07:17
Sepsis-$\mathcal{S}_1^{\delta=\infty}$	0.588	0.528	0.562	0.574	0.588	8	12734	1	0:00:19
Sepsis-$\mathcal{S}_2$	0.615	0.559	0.593	0.607	0.615	2	12740	2	0:00:41
Sepsis-$\mathcal{S}_3$	0.654	0.602	0.634	**0.636**	0.654	2	12740	425	15:05:40
Sepsis-$\mathcal{S}_1^{\delta=24h}$	**0.656**	**0.604**	**0.636**	**0.636**	**0.656**	2	12740	503	19:57:03

Table 1: Next-activity prediction results for the model update strategies $\mathcal{S}_1 - \mathcal{S}_3$.

All update strategies significantly outperform our baseline strategy $\mathcal{S}_1^{\delta=\infty}$ that does not update the prediction model. This seems to indicate that the presence of unseen data variability is not reflected in the initial starting training data. In terms of the prediction quality the results clearly suggest that more frequent model updates lead to better prediction scores. The only exception in this case applies to the most greedy update strategy $\mathcal{S}_1^{\delta=24h}$ with periodic model updates on a daily basis, which despite significantly higher cumulative computation time at most performs as well as or in some cases even worse than the update strategy $\mathcal{S}_3$ that is triggered when unseen activity sequences are observed. A possible explanation for this observation is that too frequent

updates impede a model from generalizing due to the model overfitting the data, especially considering that the use of an expanding window includes all observed events for training, leading to potentially complex models.

In light of the higher computational cost incurred by strategies with more frequent updates the question arises, how much additional computational effort is justifiable for acquiring more accurate prediction results. With regards to the obtained results, $\mathcal{S}_3$ provides the optimal trade-off; it achieves on par prediction results and requires less computational effort in contrast to the aggressive daily update strategy $\mathcal{S}_1^{\delta=24h}$.

	Periodic Update		Data Driven Update		
Data Set	[8] (Monthly)	$\mathcal{S}_1^{\delta=24h}$	[8] (Drift)	$\mathcal{S}_2$	$\mathcal{S}_3$
Helpdesk	0.78	**0.928**	0.81	0.914	**0.931**
BPI2012	**0.79**	0.777	**0.79**	0.742	0.778

Table 2: Next-activity prediction accuracy of LSTM models [8] against $\mathcal{S}_1$-$\mathcal{S}_3$.

In Table 2, we compare the prediction quality of our LSTM models with the ACC scores reported by [8] for the common data sets. In [8], the authors use a LSTM architecture presented in [14] and operate on batches of events collected on a monthly basis with an expanding window, whereas we operate on a daily basis (cf. Sect. 2). For Helpdesk, our daily periodic update strategy outperforms the monthly approach of [8] by approx. 14% and our $\mathcal{S}_2$ and $\mathcal{S}_3$ outperform their drift detection based update strategy by 10-12%. For BPIC2012, our $\mathcal{S}_1^{\delta=24h}$ and $\mathcal{S}_3$ achieve similar results in comparison to their periodic and drift based update strategy, where our approach falls short of approx. 1.3% in ACC.

6 Conclusion

We have proposed encoding and update strategies for predicting next process activities in an online setting, in particular dealing with unseen data such as process activities that have not been observed in the process event stream so far. The encodings enable the consideration of such unseen activities by holding "empty spots". The update strategies vary in the frequency the prediction model is updated. This enables us to compare and analyze whether continuous updates result in the best prediction or if even less frequent updates are more beneficiary. Based on a prototypical implementation and three real-world data sets it could be shown that an "update on demand" strategy yields the best results in terms of balancing prediction quality and performance. This work has some limitations. In our experiments we did not consider potential data imbalances, such as infrequent activities. For instance it is possible that if we split the historic data into a training and validation set, that all infrequent activities might land in the validation set and thus are unknown to the prediction model and potentially ignored because they are underrepresented. Future work includes the testing of greedy update strategies based on additional attributes, using a sliding window in order to limit training data to the recent past, conducting additional

experiments with alternative encoding techniques for the data, experimenting with above proposed reserve capacity encoding approach, experimenting with alternative prediction model training approaches such as randomly dropping attribute values during training to further increase a prediction model's ability to further generalize and be able to handle future unseen data.

Acknowledgments This work has been supported by Deutsche Forschungsgemeinschaft (DFG), GRK 2201 and by the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 881843.

References

1. Evermann, J., Rehse, J.R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: International Conference on Business Process Management. pp. 327–338 (2016)
2. Hancock, J.T., Khoshgoftaar, T.M.: Survey on categorical data for neural networks. Journal of Big Data **7**(1), 1–41 (2020)
3. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
4. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
5. Maisenbacher, M., Weidlich, M.: Handling concept drift in predictive process monitoring. In: Services Computing. pp. 1–8 (2017)
6. Márquez-Chamorro, A.E., Nepomuceno-Chamorro, I.A., Resinas, M., Ruiz-Cortés, A.: Updating prediction models for predictive process monitoring. In: Advanced Information Systems Engineering. pp. 304–318 (2022)
7. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: A survey. IEEE Trans. Serv. Comput. **11**(6), 962–977 (2018)
8. Pauwels, S., Calders, T.: Incremental predictive process monitoring: The next activity case. In: Business Process Management. pp. 123–140 (2021)
9. Polato, M., Sperduti, A., Burattin, A., Leoni, M.d.: Time and activity sequence prediction of business process instances. Computing **100**(9), 1005–1031 (2018)
10. Rinderle-Ma, S., Mangler, J.: Process automation and process mining in manufacturing. In: Polyvyanyy, A., Wynn, M.T., Looy, A.V., Reichert, M. (eds.) Business Process Management. pp. 3–14 (2021)
11. Rinderle-Ma, S., Winter, K.: Predictive compliance monitoring in process-aware information systems: State of the art, functionalities, research directions. Tech. Rep. arXiv:2205.05446 (2022). https://doi.org/10.48550/ARXIV.2205.05446
12. Rizzi, W., Di Francescomarino, C., Ghidini, C., Maggi, F.M.: How do i update my model? on the resilience of predictive process monitoring models to change. Knowledge and Information Systems pp. 1–32 (2022)
13. Sak, H., Senior, A., Beaufays, F.: Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv preprint arXiv:1402.1128 (2014)
14. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: Advanced Information Systems Engineering. pp. 477–492 (2017)