Decay Replay Mining to Predict Next Process Events

JULIAN THEIS [ID], **(Graduate Student Member, IEEE), AND**
HOUSHANG DARABI [ID], **(Senior Member, IEEE)**
Mechanical and Industrial Engineering Department, University of Illinois at Chicago, Chicago, IL 60607, USA

Corresponding author: Houshang Darabi (hdarabi@uic.edu)

ABSTRACT In complex processes, various events can happen in different sequences. The prediction of the next event given an a-priori process state is of importance in such processes. Recent methods have proposed deep learning techniques such as recurrent neural networks, developed on raw event logs, to predict the next event from a process state. However, such deep learning models by themselves lack a clear representation of the process states. At the same time, recent methods have neglected the time feature of event instances. In this paper, we take advantage of Petri nets as a powerful tool in modeling complex process behaviors considering time as an elemental variable. We propose an approach which starts from a Petri net process model constructed by a process mining algorithm. We enhance the Petri net model with time decay functions to create continuous process state samples. Finally, we use these samples in combination with discrete token movement counters and Petri net markings to train a deep learning model that predicts the next event. We demonstrate significant performance improvements and outperform the state-of-the-art methods on nine real-world benchmark event logs.

INDEX TERMS Business process intelligence, decay functions, deep learning, petri nets, neural networks, operational runtime support, predictive process management, process mining.

I. INTRODUCTION

With the ongoing development of digitizing and automatizing industries along with the steady increment of interconnected devices, we can project more interactions onto processes [1], [2]. These processes can represent procedures in different industries such as retail [3], software development [4], healthcare [5], network management [6], project management [7], or manufacturing [8]. One illustrative example is the process of a customer loan application in financial institutes [9]. An applicant can request money for specific purposes. The application then undergoes several process steps such as *negotiation*, *request validation*, *fraud assessment*, *offer creation* and/or *application rejection*. Each step of the process utilizes different institutional resources such as employees, customer records, IT systems, or third-party resources to check the creditworthiness of applicants. Though trivial, the process gets complex with an increasing number of applications and requirements of the institute.

While traditional process mining is primarily concerned with the discovery, analysis, and monitoring of processes, predictive process management gains momentum by enhancing process models. Predictive process management plays an important role in the areas mentioned earlier. Knowing when specific situations occur, or in which state a process will be next, is important to meet qualitative and/or quantitative requirements of businesses and organizations.

Many businesses deploy Process-Aware Information Systems such as workflow management systems, case-handling systems, enterprise information systems, enterprise resource planning, and customer relationship management systems. These are software tools which manage and execute operational processes involving people, applications, and/or information sources based on process models [10]. Such systems record events associated to different process steps along with time and other related information which can be utilized for predictive process management. Typical use cases comprise the prediction of the next event, forecasting of a process' final state, or time interval prediction of future events [11]. Predicting the next event elicits special attention since it gives

---

The associate editor coordinating the review of this article and approving it for publication was Shouguang Wang.

organizations the ability to forecast process deviations. This type of early detection is essential for intervenability before a process enters risky states [12]. Moreover, predictive process management assists businesses in resource planning and allocation, providing insights on the condition of a process to fulfill for instance service-level agreements [13]–[15].

With this motivation, a range of different methods have been proposed on predicting the events in process sequences. Most recent advances are made in utilizing different deep learning architectures such as Long Short-Term Memory (LSTM) neural networks and stacked autoencoders [15]–[18]. However, these techniques do not discover process models at first, but perform their predictions on the raw event logs. This makes decision making hard to understand and difficult to explain, which is crucial to discover the weaknesses of a process. Furthermore, since neural networks are not infallible [19], commonsense knowledge and obvious logical policies are suggested to be introduced into a deep learning model from the beginning to reduce potential vulnerability. This knowledge is easy to obtain from process discovery algorithms. Therefore, modeling processes from scratch using neural networks is costly and partially redundant. Thus, one of the research questions is how to retain process models like Petri nets (PN) [20] with its logic, interpretability, and comprehensibility [21]–[24], and combine it with the strengths of deep learning towards more interpretable models to improve performance at the same time.

A further research motivation arises due to weaknesses of recent predictive methods. Some of the state-of-the-art algorithms do not consider event timestamps as features at all [12], [16]. However, the duration between two events and/or sequences of events might be correlated with a future process outcome. Therefore, we suggest taking event times into account for predictive modeling.

In the current work, we propose an innovative method to predict the next event of a running process case which engages with the issues mentioned above. We first leverage a state-of-the-art process mining algorithm to discover a PN based process model from an event log. Then, we enhance the process model with time decay functions. In this way, we can create continuous and timed state samples which we finally couple with process resources to train a neural network for the prediction of the next event. We call this approach *Decay Replay Mining - Next TrAnsition Prediction* (DREAM-NAP). By taking this approach, we demonstrate significant performance improvements. Our method outperforms the state-of-the-art techniques on all of the popular benchmark datasets.

This paper is structured as follows. Section II discusses related work and most recent advances in the next event prediction of business processes. We introduce preliminaries in Section III. Section IV focuses on the proposed approach, especially on the decay function modeling in PNs and the deep learning architecture. Section V evaluates the approach against different existing methods. Finally, we conclude the paper and discuss future work in Section VI.

II. RELATED WORK

The application of deep learning on predictive business process mining has grown enormously during recent years. Researchers have shown the applicability of machine and deep learning on several target variables such as the remaining time of running cases [25], forecasting time of events [26], and predicting upcoming events in running processes while utilizing a-priori knowledge [27]. The prediction of events can be considered as a classification problem in which the probability of a next event a given the state s of the process at time τ, $P(a|s(\tau))$, is to be found.

Early predictive models focused on analytical approaches. Le *et al.* [28] introduced a hybrid approach consisting of a sequence alignment technique to extract similar patterns and to predict upcoming events based on a combination of Markov models. The next event of a running process case is therefore determined by the transition probabilities of the Markov models.

Becker *et al.* [29] faced this problem with a similar approach in which historical event data is used to create a Probabilistic Finite Automaton. In comparison, Ceci *et al.* [30] proposed an approach which can handle incomplete traces which is robust to noise and deals with overfitting. This approach leverages sequence mining. Efficient frequent pattern mining is applied to create a tree where prediction models are associated to each node (also called *nested model learning*). These prediction models can be any traditional machine learning algorithms for classification.

Lakshmanan *et al.* [31] developed a method which models a process in a probabilistic and instance-specific way. This model can predict next events and can be translated into a Markov chain. Their approach has been implemented on a simulated automobile insurance claim process.

Similarly, Unuvar *et al.* [32] proposed a method to predict the likelihood of future process tasks by modeling parallel paths which can be either dependent or independent. The authors applied their methodology to a simulated marketing campaign business process model.

More recently, Breuker *et al.* [12] introduced a predictive model based on the theory of grammatical inference. They have modeled business processes probabilistically with a method called RegPFA which is based on Probabilistic Finite Automaton. Grammatical inference is applied on top of the finite automaton. One of the advantages is that the methodology is based on weaker biases while maintaining comprehensibility. This is important because users without deep technical knowledge can interpret and understand the models. Breuker et al. evaluated their approach against two public available real-world logs demonstrating significant performance improvements. Breuker et al. are able to predict the next event given a running process case with accuracies between 69% and 81% according to their reports.

Most recent research studies have shown the applicability of deep learning to predict process states and events. Evermann *et al.* [16] have shown in 2017 that recurrent neural networks can be applied to predict next events in

processes and improve state-of-the-art prediction accuracies. They create word embeddings from each event instance of the event log to train an LSTM neural network. Therefore, the process is modeled implicitly by the neural network itself. Evermann et al. used the same datasets as Breuker *et al.* [12] for comparison.

A comparable approach has been elaborated by Tax *et al.* [15] who predict next events including their timestamps and remaining case times using LSTMs. This approach is similar to the one Evermann et al. demonstrated before. However, according to Khan *et al.* [17], a major drawback of LSTMs in this context is their limited memory due to the predefined size of the memory state representation which is used to predict next events. They claim that distant event instances in long-running cases vanish over time from the memory state vector. Therefore, Khan *et al.* [17] adapted to overcome the memory limitations of LSTMs by applying memory-augmented neural networks. This technique leverages external memory modules for long-term retention to model complex event processes. The authors demonstrate the applicability and report slight performance improvements compared to Tax *et al.* [15].

A further approach has been elaborated by Mehdiyev *et al.* [18]. The authors encode events into n-gram features using a sliding window approach and leverage feature hashing on top. These features, in turn, are used to train a deep learning model consisting of unsupervised stacked autoencoders and supervised fine-tuning. This architecture has shown significant performance improvements across most of the datasets, yet it is more complex compared to the methods described earlier. Mehdiyev et al. can predict the next event given a running process case with accuracies between 66% and 83% according to their reports.

Since deep learning techniques are difficult to interpret, Lee *et al.* [33] developed a method based on matrix factorization and knowledge from business process management to create predictive models which are easier to understand. The authors claim to require fewer parameters than neural networks while maintaining good performance.

In this work, we have three major contributions. First, we propose an approach to represent process model states in a continuous rather than a discrete format by enhancing PNs with time decay functions. Second, we show that we can use this approach to incorporate time as a continuous feature to predict the next process event since the duration between two events might be correlated with the type of subsequent occurring events in real-world processes. At the same time, we retain a comprehensible process model with its advantages [21]–[24]. With these advancements, we demonstrate that our next event prediction algorithm performs significantly better than the previously introduced methods. Third, we contribute a comprehensive evaluation of recent next event prediction algorithms across nine benchmark datasets reporting five different classification evaluation metrics.

III. PRELIMINARIES

In this section, we introduce the preliminaries which are required throughout the paper. We introduce event logs, followed by PNs. We provide a general introduction to Process Mining in III-C and introduce a state-of-the-art process discovery algorithm in Section III-D. Finally, we define neural networks in Section III-E.

A. EVENT LOGS

The definitions in this subsections are partially based on the work of van der Aalst *et al.* [21] and Guo *et al.* [34].

An *event* $a \in \mathcal{A}$ describes an instantaneous change of a process' state. In this definition, $\mathcal{A}$ is the finite set of all possible events. Example events based on the process steps described in Section I are *start negotiation*, *end fraud assessment*, and *start offer creation*. A specific event a may happen more than once in a given process. An event instance E is a vector with at least two attributes: the name of the associated event a and the corresponding occurrence timestamp. An instance vector may contain further non-mandatory attributes like costs, people, and resources associated with that event occurrence. Based on the definition of an event instance, two event instances cannot have the same timestamp, i.e. cannot occur simultaneously. This is because of the continuous nature of time, and the fact that point probabilities in continuous probability distributions are zero.

We define $\mathcal{N}$ as the set of all possible event instances and $\mathcal{D}$ as the set of all possible attributes. Then for any event instance $E \in \mathcal{N}$ and any attribute $d \in \mathcal{D}$: $\upsilon_d(E)$ is the value of the attribute d for the event instance E. If an event instance E does not contain an attribute d, then $\upsilon_d(E) = \varnothing$ (empty set). We denote the attribute *timestamp* by d_{ts}.

A *case* g is a finite and chronological sequence of event instances. In literature, the term *trace* is also used to describe a *case*, thus we use both terms synonymously. We define $\mathcal{G}$ as the finite set of all possible traces and $\gamma(g)$ as a function that returns the number of event instances of a trace $g \in \mathcal{G}$, i.e. the length of g.

An *event log* is a set of traces $\mathcal{L} \subseteq \mathcal{G}$. Moreover, $\mathcal{L}_{i,j}$ refers to the jth event instance in the ith trace of an event log $\mathcal{L}$. $|\mathcal{L}|$ denotes the cardinality of $\mathcal{L}$ corresponding to its number of traces. Similarly, $\gamma(\mathcal{L}_i)$ expresses the number of event instances of the ith trace of the event log $\mathcal{L}$.

B. PETRI NET

A PN is a mathematical model that can represent a process. It consists of a set of places; these are graphically represented as circles and transitions represented as rectangles. Transitions correspond to events. Transitions and places are also referred to as *nodes*. Additionally, arcs are used to unidirectionally connect places to transitions and vice versa. A labeled PN is defined as

$$PN = \langle \mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{A}, \pi \rangle \qquad (1)$$

where $\mathcal{P}$ is the set of places, $\mathcal{T}$ is the set of transitions, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is the set of directed arcs connecting

places and transitions, and $\mathcal{A}$ is the set of events [21], [34], [35]. The set $\mathcal{P} \cup \mathcal{T}$ is called the set of nodes. The first node of each pair $(x, y) \in \mathcal{F}$ represents always the source whereas the second node represents always the sink of the directed arc. In other words, a node x is the input node to another node y iff $(x, y) \in \mathcal{F}$. Similarly, x is the output node to another node y iff $(y, x) \in \mathcal{F}$. For any $x \in \mathcal{P} \cup \mathcal{T}$, $\bullet x = \{y | (y, x) \in \mathcal{F}\}$ is the set of *input nodes* to x and $x \bullet = \{y | (x, y) \in \mathcal{F}\}$ is the set of *output nodes* of x. The function $\pi : \mathcal{T} \to \mathcal{A} \cup \{\bot\}$ maps each transition $t \in \mathcal{T}$ to either a single event of $\mathcal{A}$ or to the non-observable event $\bot$. A labeled PN is defined such that

$$\forall_{a \in \mathcal{A}} \exists!_{t \in \mathcal{T}} \pi(t) = a. \tag{2}$$

Each place can hold a non-negative integer number of tokens. We define $\sigma(p)$ as the number of tokens in a place p where $p \in \mathcal{P}$.

The state of a PN corresponds to a marking $M \in \mathcal{M}$ where $\mathcal{M}$ is the set of all possible markings. We define $M \in Z^{|\mathcal{P}|}$ as a vector of size $|\mathcal{P}|$ where $\mathcal{Z}$ denotes the set of all non-negative integers and $|\mathcal{P}|$ corresponds to the cardinality of $\mathcal{P}$. Each element $M_i = \sigma(p_i), i = 1, \ldots, |\mathcal{P}|$ where p_i is the ith place of $\mathcal{P}$. The initial state M^{init} is also called *initial marking*, whereas the final state M^{final} is called *final marking* [21]. Usually discovered process models in process mining have a dedicated source and a dedicated sink place that indicate the start and end of the process. All other process nodes are on a path between them. Hence, M^{init} and M^{final} describe the process source and sink states [21].

Moreover, a transition $t \in \mathcal{T}$ is mathematically defined as *enabled* [21], i.e. can only be fired if

$$\forall_{p \in \bullet t} \sigma(p) \geq 1. \tag{3}$$

Hidden transitions, a special type of transition, are associated to the non-observable event $\bot$. Such transitions can always fire independent of observed events as long as the introduced token requirements at incoming places are met. When firing a transition t, a token is removed from each of the input places $\bullet t$, while a token is added to each of the output places $t \bullet$.

Process models do not always behave as desired. For example, PNs may contain unintended deadlocks or transitions that can never become enabled. Different criteria have been specified under the term *soundness* to prevent process models from such behavior [36], [37]. It is defined as follows [21]. A labeled PN with dedicated source and sink places is considered sound iff:

- for any place $p \in \mathcal{P}$, p cannot hold multiple tokens at the same time,
- for any marking $M \in \mathcal{M}$ that indicates a token in the dedicated sink place of the PN, $M = M^{final}$ which implies that there are no remaining tokens in other places than the dedicated sink one when the final marking is reached,
- for any marking $M \in \mathcal{M}$, the final marking M^{final} is reachable,

- and for any $t \in \mathcal{T}$, a firing sequence of events exists that enables t.

Furthermore, we define a function $\delta_p(g)$ for all $p \in \mathcal{P}$ measuring the average time between a token leaves a place p until a new token enters p based on an input trace g. Finally, τ_p describes the most recent time that a token entered a place p.

C. PROCESS MINING

Process mining defines the discovery, conformance, and enhancement of business processes [21], [38]. Process discovery is the algorithmic extraction of process models from event logs. One can carry out analysis on obtained models which are usually in the format of PNs, Business Process Modeling Notations (BPMN), Event Driven Process Chains (EPCs), or Casual Nets (CN). In this paper, we will focus on PNs only.

Conformance is defined as the evaluation of the quality of a discovered process model, i.e. if it is a good representation of the process recorded by an event log. It is commonly evaluated based on *fitness* and *precision* among other metrics [21]. Therefore, each trace of an event log is replayed by executing the events sequentially on top of the process model. *Fitness* metric functions evaluate the quality of a process model by quantifying deviations between an event log and the replay response of a process model to this event log. A process model should allow replaying the behavior seen in the event log [21]. *Precision* metric functions represent the alignment between simulated traces from the obtained process model and true traces from the event log. Ideally, each generated trace by the process model should be realistic, thus being present in the actual event log.

Enhancement considers discovered process models as well as event logs to improve or extend the models. Examples of process enhancement include structural corrections to allow the occurrence of specific behavior or extending a process model with performance data.

D. SPLIT MINER

Split miner [39] is a process discovery algorithm that creates sound labeled PNs with dedicated source and sink places from event logs and that is characterized by recent significant performance improvements in comparison to existing state-of-the-art methods [40]. It is currently the best algorithm to automatically obtain PN process models from event logs with high fitness and precision. This discovery method has been developed to engage with the tradeoff between fitness, precision, and the complexity of the obtained process model.

Split miner consists of the following five steps [39]. First, it discovers a directly-follows dependency graph and detects short loops. In the second step, the algorithm searches for concurrency and marks the respective elements as such. Afterward, *split miner* applies filtering such that each node is on a path from a single start node to an end node to guarantee *soundness*, the number of edges are minimal to

reduce complexity, and that every path from start to end has the highest possible sum of frequencies to maximize fitness. Fourth, the algorithm adds *split* gateways to capture choice and concurrency. As the final step, this discovery method detects *joins*.

Split miner encompasses two hyperparameters: a frequency threshold ε to control the filtering process and η which is a threshold to control parallelism detection. Both hyperparameters are percentiles, i.e. the numerical range is between 0 and 1. Moreover, this algorithm considers only the sequence of events without timestamp or other related information during process discovery. The discovery algorithm is publicly available as a Java application [41].

E. NEURAL NETWORK

A neural network is a computing methodology motivated by biological nervous systems. Such networks consist of a set of artificial neurons which receive one or multiple inputs and produce one output. This set is divided into a predefined number of disjoint subsets n where $n \geq 2$. Each subset represents a layer l_n in the form of a matrix containing outputs of the corresponding neurons. We refer to layer l_1 as the input and l_n as the output layer of the neural network. Multiple so-called hidden layers can exist in between. In a fully connected neural network, all neurons of a layer l_k are connected to all neurons of its adjacent layer l_{k+1} for $k \leq n-1$. A very basic neural network can be defined in the following way [42], [43].

A neuron j which belongs to layer l_k calculates its output based on the weighted outputs of each predecessor neuron of layer l_{k-1}. Each direct connection between two neurons i and j is associated with a weight $w_{i,j}$. Each neuron j comprises a differentiable activation function ρ_j which is used to calculate the output of a neuron. Thus, the output of a neuron j belonging to l_k based on its predecessor layer l_{k-1} can be calculated as

$$\theta_j(l_{k-1}) = \rho_j(\phi_j(l_{k-1})). \tag{4}$$

It follows that

$$\phi_j(l_{k-1}) = \sum_i \theta_i(l_{k-2}) * w_{i,j} + w_{0,j} \tag{5}$$

where $w_{0,j}$ is a bias term. Such a neural network is commonly modeled as an optimization problem where a cost function ξ is to be defined as a function of the difference between neural network outputs and true values and to be minimized by adapting the weights w of the neural network. This is called a supervised learning problem [43].

IV. APPROACH

The DREAM-NAP approach is a supervised learning algorithm to predict the next event given a partial trace of an event log. The method consists of three steps. First, we discover a PN model from an event log and associate each place of the PN with a decay function. Then, we replay the event log used for discovery and extract feature arrays incorporating decay function response values, token movement counts,

and utilized resources. Finally, we train a neural network to predict the next event based on these feature arrays. A flow diagram of the training and testing procedure of our approach is visualized in Figure 1. In this section, we introduce each component in detail.

The source code of the proposed approach is available in our GitHub repository.[1]

A. DECAY FUNCTION ENHANCEMENT

To discover a PN, the corresponding event log has to consist of at least one non-empty trace. We draw on an existing PN discovery algorithm called *split miner* which has been introduced in Section III-D.

Decay functions are used to model data values that decrease over time. Such functions are commonly applied to population trend modeling, financial domains, and physical systems. The basic form of a decay function is

$$f(\tau) = \beta - \alpha * \tau \tag{6}$$

where τ is time, α is the rate of decay, and β is a constant corresponding to the initial value. The decay function $f(\tau)$ can be easily modified to model more complex behavior such as exponential or squared declines. However, the linear decay function presented in Equation 6 is the simplest option.

We leverage the properties of these functions to expand discovered PN models by decaying the activation of places over time. A place activation is triggered through token arrivals during an event log replay, i.e. when the marking of a PN changes. In this way, we add a continuous time-based dimension to the discrete state representation of PNs. This approach overcomes limitations of the state-of-the-art next event prediction methods. In particular, such a decay mechanism leads to the incorporation of time as a continuous variable and can be used to detect and model event instance interarrival times through continuous state representations. At the same time, we preserve all advantages of an interpretable process model [21]–[24] in the form of PNs. Subsequently, we introduce the detailed process of decay function enhancement.

We associate each place of the PN generated by *split miner* on an event log $\mathcal{L}$ with a linear decay function $f_p(\tau)$. We denote the time difference between the current time, τ, and the most recent time a token has entered place p, τ_p, by $\Delta_p = \tau - \tau_p$.

$$f_p(\tau) = \begin{cases} \beta - \alpha * (\tau - \tau_p) & \text{if } \tau - \tau_p < \beta/\alpha, \\ 0 & \text{if otherwise.} \end{cases}$$

$$= \begin{cases} \beta - \alpha * \Delta_p & \text{if } \Delta_p < \beta/\alpha, \\ 0 & \text{if otherwise.} \end{cases} \tag{7}$$

We initially set $\Delta_p = \infty$ for all $p \in \mathcal{P}$ such that $f_p(\tau) = 0$. In this way, we reset all decay functions of the PN. A decay function $f_p(\tau)$ will *activate* as soon as a token enters a corresponding place p. The value of this function declines over

---

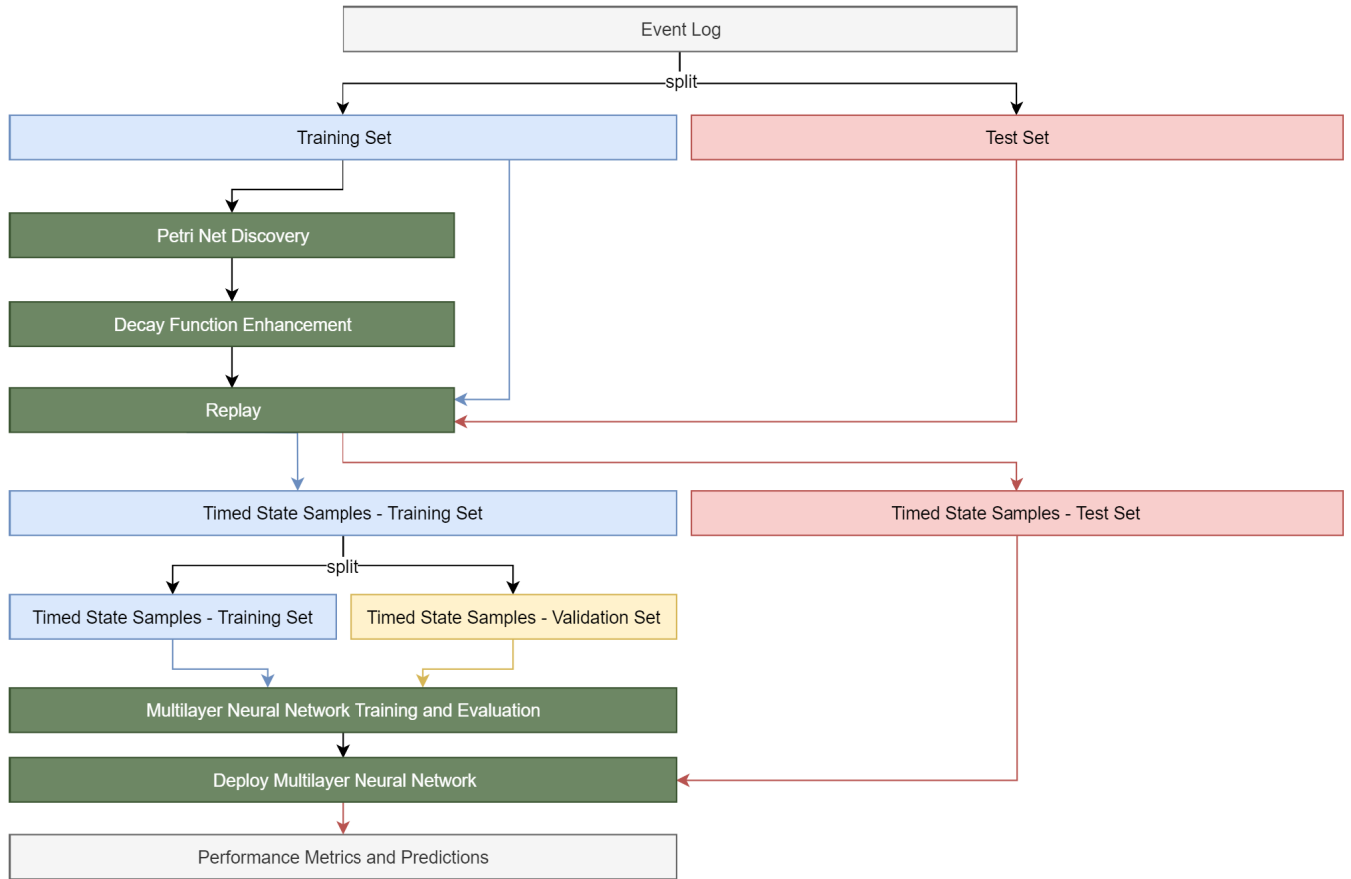[1] https://github.com/ProminentLab/DREAM-NAP

FIGURE 1. This figure illustrates the flow diagram of the proposed DREAM-NAP approach. It also visualizes the training and testing procedures. The elements of the approach are shown in green, train datasets in blue, test datasets in red, and evaluation datasets in yellow colors. The flows are color-coded correspondingly.

time τ and *reactivates* with a response value β immediately when a token enters this place.

During replay, each event instance of an event log corresponds to a transition which fires immediately when a respective event is observed and token requirements are met. Instead of focusing on the fired transitions itself, we can also unambiguously identify the sequence of fired transitions by observing the movement of tokens between places. By enhancing each place with a decay function described in Equation 7, we assign a level of importance to recent token movements compared to past ones. This mechanism scales event time information into a range from 0 to β without discretization and loss of generality.

We control the level of importance using the two decay function parameters β and α. Ideally, α should be set such that the slope of $f_p(\tau)$ covers the whole range from β to 0 based on the *reactivation* durations of a place p. In other words, the slope should not be too steep such that $f_p(\tau) = 0$ for a small Δ_p, nor too flat such that $f_p(\tau) \approx \beta$ for a large Δ_p. This cannot be achieved using a single α value for all decay functions of the PN when applying this mechanism to real-world processes with varying durations of reactivation. For this reason, we estimate an individual decay rate α_p for each

place $p \in \mathcal{P}$. We define the set of all decay rates as $\mathcal{R}$ where the cardinality of $\mathcal{R}$, $|\mathcal{R}|$, equals to $|\mathcal{P}|$.

We estimate α_p by utilizing the event log $\mathcal{L}$ and the respective PN discovered by *split miner* on $\mathcal{L}$. Each trace $g \in \mathcal{L}$ consists of a finite number of event instances. We refer to the jth event instance of the ith trace of an event log $\mathcal{L}$ by $\mathcal{L}_{i,j}$, as mentioned in Section III-A. The maximum trace duration observed in $\mathcal{L}$ is denoted by $\Delta_{max}(\mathcal{L})$ and is defined with the following equation.

$$\Delta_{max}(\mathcal{L}) = max\left(\forall_{1 \leq i \leq |\mathcal{L}|}\left(\upsilon_{d_{ts}}(\mathcal{L}_{i,\gamma(\mathcal{L}_i)}) - \upsilon_{d_{ts}}(\mathcal{L}_{i,1})\right)\right) \quad (8)$$

For the estimation of α_p, it is inevitable to know if a value for Δ_p exists, i.e. if a place p gets activated only once or if reactivations occur. Therefore, we define a function $\upsilon_p(g)$ which returns the number of tokens that enter a place p when replaying a trace g. We estimate α_p for two different cases based on the outcome of the following condition.

$$max\left(\forall_{g \in \mathcal{L}} \upsilon_p(g)\right) \leq 1 \quad (9)$$

If Condition 9 holds, α_p will be set to a value such that the response of $f_p(\tau)$ will never equal to 0 before the last event instance of a corresponding trace $g \in \mathcal{L}$ occurred. By doing

so, we guarantee to carry information on the occurrence of a specific event in the response of the decay function until the end of a trace. Equation 10 defines α_p mathematically for this case.

$$\alpha_p(\mathcal{L}) = \frac{\beta}{\Delta_{max}(\mathcal{L})} \qquad (10)$$

If Condition 9 does not hold, we consider the average reactivation duration of a place p based on all traces of the respective event log. With this information, we set the decay rate to a value such that $f_p(\tau)$ provides a level of recent token movement importance for the average duration between reactivations. Consequently, the slope will neither be too steep nor too flat. Mathematically, we can estimate α_p by

$$\alpha_p(\mathcal{L}) = \frac{\beta}{mean\big(\forall_{g \in \mathcal{L}} \delta_p(g)\big)} \qquad (11)$$

where $mean(\cdot)$ is the arithmetic mean function.

B. EVENT LOG REPLAY

After estimating all α_p of $\mathcal{R}$ for each place p in $\mathcal{P}$, we can use the corresponding decay functions, $f_p(\tau)$, to obtain a decay function response for all p at a specific time τ. We write $F(\tau)$ as the vector of decay function response values. Each element of this vector corresponds to the response value of one specific place in the PN, i.e. the ith element in $F(\tau)$ corresponds to the response of the decay function at time τ associated to the ith place in $\mathcal{P}$.

Since $F(\tau)$ constitutes only the most recent activation of $\mathcal{P}$, we introduce a counting vector $C(\tau)$ of size $|\mathcal{P}|$ elements where the ith element corresponds to the ith place in $\mathcal{P}$. We initialize the counting vector at time 0, $C(\tau = 0)$ by setting each element to 0. When a token enters a specific place p at time τ, the corresponding counter element will be incremented by 1 such that $C(\tau)$ reflects the number of tokens which have entered each place from time 0 to τ.

Similarly, we introduce a counting vector $R(\tau)$ which counts the occurrence of each unique non-mandatory event instance attribute value from time 0 to τ when replaying an event log. Continuous attribute values require discretization in advance.

We replay the event log $\mathcal{L}$ on the PN which has been enhanced using decay functions. $F(\tau)$, $C(\tau)$, $R(\tau)$, and the PN marking M at time τ, $M(\tau)$, will be reset before a trace $g \in \mathcal{L}$ will be replayed. We then obtain vectors and PN states at each time τ corresponding to the timestamp values of the replayed event instances in $\mathcal{L}$. We concatenate the vectors of decay function values, token and resource counts, and the PN marking at time τ to obtain a single vector that can be used to train a neural network. This concatenation of $F(\tau)$, $C(\tau)$, $R(\tau)$, and $M(\tau)$ is called a *timed state sample* $S(\tau)$,

$$S(\tau) = F(\tau) \oplus C(\tau) \oplus M(\tau) \oplus R(\tau) \qquad (12)$$

where $\oplus$ represents a vector concatenation. Therefore, a *timed state sample* $S(\tau)$ describes a PN process state in a

TABLE 1. Deep learning architecture of the *DREAM-NAP* model.

Parameter	Value
# layers	5
# neurons per layer	[input, input*1.2, input*0.6, input*0.3, output]
# dropout layer	4
dropout rate	0.2
# batch normalization layers	0
activation functions	[relu, relu, relu, relu, softmax]
loss	categorical crossentropy
optimizer	adam

TABLE 2. Deep learning architecture of the *DREAM-NAPr* model with an activation function hyperparameter.

Parameter	Value
# layers	6
# neurons per layer	[input, 300, 200, 100, 50, output]
# dropout layer	5
dropout rate	0.5
# batch normalization layers	5
activation functions	[relu, relu, relu, relu, relu, softmax] [sigm, sigm, sigm, sigm, sigm, softmax]
loss	categorical crossentropy
optimizer	adam

timed manner through decay function enhancement. It contains information about time-based token movements, i.e. when a token has entered a place the last time relative to the current time, token counts per place (loop information), and the current PN state using the marking. Optionally, if event instances of the event log contain non-mandatory attributes, the *timed state sample* also contains such information.

After replaying the event log $\mathcal{L}$, we obtain a set of *timed state samples*, $\mathcal{S}$, such that Condition 13 holds.

$$\forall_{1 \le i \le |\mathcal{L}|} \forall_{1 \le j \le \gamma(\mathcal{L}_i)} : S\big(\upsilon_{d_{ts}}(\mathcal{L}_{i,j})\big) \in \mathcal{S} \qquad (13)$$

C. DEEP LEARNING

We use the set of *timed state samples*, $\mathcal{S}$, to predict the next event. For each $S_i \in \mathcal{S}$ where $1 \le i \le |S|$, we predict the next event a of the upcoming instance E_{i+1} given that the *timed state sample* S_i does not contain the final marking M^{final}. This is a supervised classification problem as the event log $\mathcal{L}$ and the set of events $\mathcal{A}$ are known. An event log $\mathcal{L}$ usually consists of thousands of event instances across multiple traces. Hence, a deep neural network is a suitable method to conquer this problem due to a large amount of available data.

We propose two fully connected neural network architectures. One which ignores attribute value count vectors $R(\tau)$ in S_i, and another one which considers each S_i as is. With *DREAM-NAP*, we refer to the first neural network architecture, whereas *DREAM-NAPr* refers to the second one considering event attributes. The details of the architecture for *DREAM-NAP* are illustrated in Table 1 whereas the details of *DREAM-NAPr* are illustrated in Table 2. Both architectures have been developed in Python using Keras [44] with a Tensorflow backend [45].

The *DREAM-NAP* neural network consists of five layers. The first layer has the same size as the vector length of $3 * |\mathcal{P}|$ and correspondingly called *input*. The second layer has

1.2 times, the third 0.6 times, and the fourth 0.3 times the size of the *input* layer. Each of these layers use Rectified Linear Unit (ReLU) activation functions, which have proven major performance advantages over sigmoid and hyperbolic tangent ones in deep learning architectures [46]. Since the proposed architecture is shallow and traditional neural network activation functions can perform well on such architectures [47]–[50], we examined the impact of ReLU and sigmoid activation functions on the predictive accuracy. In all our experimental cases, ReLU-based architectures performed with a higher or equal accuracy score compared to sigmoidal ones. Hence, we propose a ReLU-based *DREAM-NAP* architecture. The final layer is the output layer with a size equal to $|\mathcal{A}|$.

The output layer utilizes a softmax activation function since we are interested in the probability of a specific $a \in \mathcal{A}$. We use dropout [51] for regularization applied between each hidden layer as well as between the fourth and the output layer. We decide on the Adam optimizer [52] to train the neural network. Batch normalization [53] layers are not used in this architecture since no further regularization is required. Moreover, batch normalization did not improve the results of the *DREAM-NAP* architecture, as we will demonstrate in Section V.

The *DREAM-NAPr* architecture is similar to the *DREAM-NAP* one. However, in this architecture, we use fixed layer sizes that are a result of a comprehensive grid search over the number of layers and number of neurons per layer. For the architecture search, we considered three, four, and five layers with each either 50, 100, 150, 200, 300, 400, or 500 neurons per layer. The results of this search have shown that the *DREAM-NAPr* model with the architecture described in Table 2 performs best across all benchmark training datasets. We specifically propose a fixed number of neurons per layer since a dynamic assignment based on the size of $R(\tau)$ could easily result in an unreasonably large number of neurons.

Since this architecture is most likely confronted with a higher probability of overfitting due to the number of event instance attribute values, we increase the dropout rate and consider batch normalization layers. Furthermore, the type of neural network activation function is a hyperparameter of this model. We examined the impact of ReLU over sigmoid activation functions on this architecture with inconclusive results. In half of our experimental cases, *DREAM-NAPr* architectures with sigmoid functions performed better whereas in the other half ReLU lead to higher accuracy scores. As a consequence, the choice of activation function type is application-specific and therefore not suggested to be fixed.

V. EVALUATION
In this section, we evaluate our proposed approach using the *DREAM-NAP* and *DREAM-NAPr* models introduced in Section IV-C on nine popular benchmark datasets and compare to the most recent peer-reviewed methods in the literature. We contrast our method specifically to the algorithms of

Tax *et al.* [15], Evermann *et al.* [16], Breuker *et al.* [12], and Lee et al [33]. The source codes of these methods are publicly available. Hence, we evaluate and perform all experiments on the same dataset splits. Unfortunately, a fair comparison to the method of Mehdiyev *et al.* [18] is not possible since the authors of that paper did not disclose the corresponding source code and deep learning parameter sets that were required to reproduce the results. We, therefore, exclude this method from our statistical comparison.

We first provide an overview of the datasets, followed by the introduction of metrics we will use. We then report and comment on the conformance of the discovered PN process models of all datasets. Afterward, we describe the preprocessing steps of the timed state samples before feeding them to our proposed deep learning architectures. Finally, we evaluate the prediction performance of the neural network models.

We perform the discovery of PNs using *split miner* and the transformation of event logs to timed state samples using our *DREAM* approach on a computer running Windows 10 with an Intel i7-6700 CPU and 16GB RAM. This task took between 30 minutes and 4 hours depending on the size of the dataset. The training of the *DREAM-NAP* and *DREAM-NAPr* neural networks were performed on Tesla K80 and NVidia GeForce RTX 2080 Ti GPUs and took between 15 minutes and 2 hours per dataset.

A. DATASETS
Our evaluation is based on three real-life benchmark datasets, specifically the Helpdesk [54], the Business Process Intelligence Challenge 2012 (BPIC12) [9], and the Business Process Intelligence Challenge 2013 (BPIC13) [55] dataset.

The Helpdesk dataset comprises events from a ticketing management process of an Italian software company. Each event instance contains the mandatory event type and associated timestamp. No further attributes are used.

The BPIC12 dataset originates from a Dutch financial institute and represents the process of a loan application. It can be split into three subprocesses related to the *work*, the *application* itself, and the *offer*. All event instances contain the required attributes as well as further non-mandatory resource information. Moreover, each event instance describes a *lifecycle* status which is either *complete*, *scheduled* or *start*. Finally, the event instances of this event log carry information about the requested loan amount. We split the dataset into multiple subprocesses to be able to compare our results to the results of existing methods. We consider the complete event log without any filtering, denoted by *BPIC12 - all*. *BPIC12 - all complete* considers only event instances of *lifecycle* value *complete*. Similarly, we filter the original event log by *work* related events only and consider all events, code-named as *BPIC12 - work all*, and events with *lifecycle* attribute value *complete* as *BPIC12 - work complete*. Additionally, we consider the subprocesses of *offer*s and *application*s separately as *BPIC12 - O* and *BPIC12 - A* to perform our evaluation on the same datasets as the state-of-the-art methods [12], [16], [18], [33].

Dataset	# event instances	# events	# traces	# resources
Helpdesk	13,710	9	3,804	0
BPIC12 - all	262,200	24	13,087	2
BPIC12 - all complete	164,506	23	130,897	2
BPIC12 - work complete	72,413	6	9,658	2
BPIC12 - work all	170,107	7	9,658	2
BPIC12 - O	31,244	7	5,015	2
BPIC12 - A	60,849	10	13,087	2
BPIC13 - Incidents	65,533	4	7,554	7
BPIC13 - Problems	8,599	4	1,758	8

These subprocess event logs consist of event instances with *complete* lifecycle values only.

The third log originates from Volvo IT and describe events from an incident and problem management system. Each event instance contains the required attributes and non-mandatory information about the lifecycle, group, responsible employee, resource country, organization country, involved organizations, impact, and the product. Events that are associated with a problem rather than an incident contain a further attribute which describes the role of the affected organization. We split this dataset into two separate event logs handling incidents and problems independently. We call these two event logs *BPIC13 - Incidents* and *BPIC13 - Problems*.

An overview of all datasets and their number of event instances, events, traces, and resource attributes is given in Table 3.

B. METRICS

We utilize 10-fold cross-validation to perform our evaluation. Therefore, we consider 90% of the actual traces for training and 10% for testing. The training set is used to discover multiple process models, as we describe in Section V-C. We measure the quality of the obtained PNs using a basic conformance checking function called *token-based replay fitness* (Equation 14) [21]. The function calculates the fitness by replaying each trace of an event log based on the number of missing, consumed, remaining, and produced tokens. The higher its score, the higher the alignment between the event log and the process model. Such fitness functions are common process mining metrics to evaluate the quality of process models [56]–[59], as described in Section III-C.

$$fitness = \frac{(1 - \frac{missing}{consumed})}{2} + \frac{(1 - \frac{remaining}{produced})}{2} \quad (14)$$

We select the best fitting process model after process discovery based on the fitness function of Equation 14. This model is then used for decay function enhancement and estimation of the corresponding decay function parameters, as described in Section IV-A.

Afterward, the enhanced PN model is used to replay the training as well as the test set to obtain timed state samples. Moreover, we split the training set after replaying into a 90% training and 10% holdout evaluation set. We finally obtain three disjoint datasets for training, validation, and

testing a deep learning model. We train deep learning models on the training set only and select the best one based on the validation set. The best predictive model is chosen at the lowest validation loss which is an effective and widely used approach to train neural networks called early stopping [60]–[62]. An overview of this procedure is visualized in Figure 1.

We evaluate the predictive performance of our approach based on averaged accuracy, precision, and recall, as well as F-score and the *area under the curve* (AUC) of the receiver operating characteristic. All of the metrics are used to compare against the earlier-introduced next event prediction techniques. The subsequent definitions of metrics are based on [18], [63], [64].

Accuracy is defined as

$$\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{A}|} n_i * \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i} \quad (15)$$

where $|\mathcal{S}|$ is the total number of timed state samples and n_i the number of timed state samples with a next event equal to the ith event in $\mathcal{A}$. Moreover, tp, tn, fp, and fn represent *true positive*, *true negative*, *false positive*, and *false negative* respectively.

Precision is defined as

$$\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{A}|} n_i * \frac{tp_i}{tp_i + fp_i}. \quad (16)$$

Recall is defined as

$$\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{A}|} n_i * \frac{tp_i}{tp_i + fn_i}. \quad (17)$$

In addition, we report the *F-score* for each dataset. This measure is the harmonic mean of precision and recall and provides information on how precise and robust an algorithm is. *F-Score* is defined as

$$\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{A}|} n_i * \frac{precision * recall}{precision + recall}. \quad (18)$$

Finally, we report the AUC of the receiver operating characteristic. It is a common classification analysis to determine which model predicts classes best. The closer an AUC value is to 1, the better the model is. *Multiclass AUC* is defined as

$$\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{A}|} n_i * \int_0^1 tpr_i d(fpr_i) \quad (19)$$

where tpr_i and fpr_i is the true positive and false positive rate for the ith event.

In a final step, we compare the overall performance of our approach with the ones of the state-of-the-art algorithms using a rank test. Also, we perform a sign test to determine statistical significant improvements. This test method is a variation of a binomial test and considers the number of times an algorithm performed best [65].
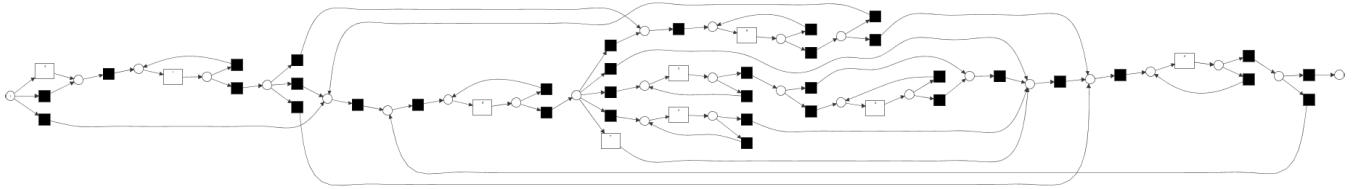
FIGURE 2. This figure shows the interpretable PN obtained from the first training set of the Helpdesk dataset.

C. PETRI NET DISCOVERY

We initially utilize *split miner* to discover multiple PN process models for all benchmark event logs. We perform hyperparameter optimization to obtain the best combination of ε and η for each of the 10-fold cross-validation training sets of each dataset. ε and η are initially set to 0.0 and are increment in 0.1 steps. A PN is discovered for each of the 100 hyperparameter combinations. Based on Equation 14, for each fold, we select the process model with the highest fitness score for decay function enhancement. The closer the value is to 1, the better the PN represents the logical behavior of an underlying process. This logic does not have to be learned by a neural network.

Table 4 illustrates the averaged fitness scores over the best process models for all 10 folds per each benchmark training dataset. It can be seen that *split miner* can detect PNs with fitness values above 85%. The models obtained on *BPIC12 - work all*, *BPIC12 - A*, and *BPIC13 - Incidents* even reach fitness scores above 95%. This shows that process discovery techniques can unveil and model basic logical behavior from event logs which we leverage in our enhancement approach. However, none of the process model evaluations result in a perfect fitness score of 1. This is because *split miner* filters infrequent behavior, i.e. discards information, which does not seem to correspond to the main process behavior.

We visualize the best obtained PN from the first fold of the Helpdesk training set in Figure 2. The white rectangles represent the 9 events recorded in the event log, whereas black rectangles correspond to hidden transitions, i.e. transitions which are mapped to the non-observable event $\perp$. The basic behavior of the underlying process can be observed, interpreted, and analyzed in different contexts. This underscores one of the advantages of process models.

D. DEEP LEARNING PREPROCESSING

After selecting the best process model for each fold of all benchmark datasets, we enhance these PNs using decay functions and replay the training and testing traces to create timed state samples, as described in Section IV and as visualized in Figure 1. However, several preprocessing steps are necessary before feeding the timed state samples to the proposed *DREAM-NAP* and *DREAM-NAPr* deep learning architectures.

All datasets originating from *BPIC12* contain one continuous and one categorical resource attribute. We discretize the

TABLE 4. This table shows the averaged cross validated fitness scores of the PN models obtained from each dataset. It can be seen that *split miner* is able to capture basic logical process behavior for all benchmark event logs.

Dataset	Averaged Fitness
Helpdesk	0.928
BPIC12 - all	0.919
BPIC12 - all complete	0.88
BPIC12 - work complete	0.892
BPIC12 - work all	0.961
BPIC12 - O	0.856
BPIC12 - A	0.951
BPIC13 - Incidents	0.955
BPIC13 - Problems	0.925

continuous attribute by quantizing its values using disjoint intervals of size 20.

For the event logs originating from the *BPIC13* dataset, we consider only the categorical attributes of *resource country*, *organization country*, *involved organization*, *impact*, and, if applicable, *role of the affected organization*. The number of unique values for the excluded resources are too large, hence these resources do not contribute beneficial and generalizable information.

We normalize each component of the timed state samples of the training and validation set separately, i.e. $F(\tau)$, $C(\tau)$, $M(\tau)$ and $R(\tau)$, to zero mean and unit variance. The mean and standard deviation of each vector before normalization is used to normalize the test set.

E. RESULTS

We train the *DREAM-NAP* and *DREAM-NAPr* models on the preprocessed training timed state samples of each benchmark dataset using early stopping, but continue training for in total 100 epochs for visualization purposes. The training batch size of *DREAM-NAP* is set to 64 whereas the *DREAM-NAPr* batch size is set to 100 to accelerate the training process. We train the methods of Evermann *et al.* [16], Tax *et al.* [15], Breuker *et al.* [12], and Lee *et al.* [33] using the reported sets of hyperparameters. However, the methods of Evermann et al., Tax et al., and Lee et al. are not designed to predict the first event of a sequence. According to the corresponding authors, the inputs of the methods must be padded with null/zero events such that the first predictable event equals the actual first event of the trace. In this way, all

TABLE 5. This table illustrates the results obtained by the proposed approach and contrasts them to existing state-of-the-art methods. Bold values designate that the proposed model outperforms state-of-the-art results. * denotes datasets that do not contain resources, therefore *DREAM-NAPr* is not applicable. ** denotes that the source code of Breuker *et al.* [12] was not able to produce results on this dataset.

Dataset	Approach	Model	Accuracy	Precision	Recall	F-Score	AUC
Helpdesk *	DREAM	DREAM-NAP	**0.829**	0.768	**0.829**	**0.795**	**0.876**
	Tax et al. [15]		0.814	**0.799**	0.814	0.792	0.872
	Evermann et al. [16]		0.599	0.625	0.599	0.550	0.716
	Breuker et al. [12]		0.801	0.759	0.801	0.777	0.861
	Lee et al. [33]		0.801	0.772	0.801	0.782	0.861
BPIC12 - all	DREAM	DREAM-NAP	0.847	0.838	0.847	0.820	0.910
	DREAM	DREAM-NAPr	**0.896**	**0.895**	**0.896**	**0.888**	**0.942**
	Tax et al. [15]		0.850	0.848	0.850	0.834	0.917
	Evermann et al. [16]		0.789	0.801	0.789	0.774	0.885
	Breuker et al. [12]		0.770	0.698	0.770	0.722	0.871
	Lee et al. [33]		0.695	0.730	0.695	0.694	0.835
BPIC12 - all complete	DREAM	DREAM-NAP	0.789	0.778	0.789	0.746	0.884
	DREAM	DREAM-NAPr	**0.863**	**0.871**	**0.863**	**0.856**	**0.926**
	Tax et al. [15]		0.802	0.794	0.802	0.767	0.892
	Evermann et al. [16]		0.684	0.693	0.684	0.645	0.829
	Breuker et al. [12]		0.721	0.639	0.721	0.654	0.847
	Lee et al. [33]		0.681	0.722	0.681	0.678	0.830
BPIC12 - work complete **	DREAM	DREAM-NAP	**0.747**	**0.748**	**0.747**	**0.720**	**0.827**
	DREAM	DREAM-NAPr	**0.823**	**0.823**	**0.823**	**0.818**	**0.872**
	Tax et al. [15]		0.698	0.732	0.698	0.688	0.804
	Evermann et al. [16]		0.709	0.723	0.709	0.693	0.805
	Lee et al. [33]		0.547	0.526	0.547	0.524	0.689
BPIC12 - work all	DREAM	DREAM-NAP	**0.878**	**0.882**	**0.878**	**0.877**	**0.919**
	DREAM	DREAM-NAPr	**0.902**	**0.903**	**0.902**	**0.900**	**0.934**
	Tax et al. [15]		0.855	0.875	0.855	0.861	0.909
	Evermann et al. [16]		0.855	0.876	0.855	0.861	0.908
	Breuker et al. [12]		0.850	0.856	0.850	0.848	0.901
	Lee et al. [33]		0.784	0.784	0.784	0.778	0.862
BPIC12 - O	DREAM	DREAM-NAP	**0.852**	0.888	**0.852**	**0.834**	**0.918**
	DREAM	DREAM-NAPr	**0.935**	**0.932**	**0.935**	**0.927**	**0.963**
	Tax et al. [15]		0.837	0.876	0.837	0.824	0.910
	Evermann et al. [16]		0.807	0.876	0.807	0.797	0.893
	Breuker et al. [12]		0.826	0.894	0.826	0.820	0.905
	Lee et al. [33]		0.822	0.826	0.822	0.812	0.900
BPIC12 - A	DREAM	DREAM-NAP	**0.805**	**0.748**	**0.805**	**0.761**	**0.893**
	DREAM	DREAM-NAPr	**0.951**	**0.958**	**0.951**	**0.950**	**0.974**
	Tax et al. [15]		0.791	0.739	0.791	0.746	0.885
	Evermann et al. [16]		0.601	0.640	0.601	0.530	0.765
	Breuker et al. [12]		0.800	0.745	0.800	0.755	0.890
	Lee et al. [33]		0.805	0.748	0.805	0.761	0.893
BPIC13 - Incidents	DREAM	DREAM-NAP	**0.703**	0.689	**0.703**	0.657	0.685
	DREAM	DREAM-NAPr	**0.882**	**0.879**	**0.882**	**0.879**	**0.880**
	Tax et al. [15]		0.697	0.708	0.697	0.667	0.699
	Evermann et al. [16]		0.637	0.642	0.637	0.619	0.667
	Breuker et al. [12]		0.697	0.709	0.697	0.661	0.686
	Lee et al. [33]		0.637	0.655	0.637	0.618	0.714
BPIC13 - Problems **	DREAM	DREAM-NAP	**0.675**	0.604	**0.675**	**0.593**	0.558
	DREAM	DREAM-NAPr	**0.791**	**0.777**	**0.791**	**0.779**	**0.779**
	Tax et al. [15]		0.630	0.608	0.630	0.584	0.591
	Evermann et al. [16]		0.624	0.654	0.624	0.565	0.568
	Lee et al. [33]		0.597	0.593	0.597	0.570	0.627

methods can be applied and evaluated on the same datasets. The detailed results are listed in Table 5.

DREAM-NAP outperforms seven out of nine benchmark datasets in terms of accuracy and recall, six out of nine in terms of F-score, and five out of nine in terms of AUC. Especially on the *BPIC12 - work complete* dataset, we demonstrate that the decay mechanism in combination with token movement counts is extremely beneficial to predict the next

event. We outperform the current state-of-the-art by 3.8% in accuracy and recall as well as with a 2.5% higher precision. This leads to an F-Score value of 72%. The state-of-the-art method with the closest F-Score value is Evermann *et al.* [16] with 69.3%. Lee *et al.* [33] perform the worst on this dataset with an F-Score of 52.4%. This underscores that our approach performs significantly better on this dataset than existing state-of-the-art methods. *DREAM-NAP* also

surpasses the existing methods on *Helpdesk*, *BPIC12 - work all*, *BPIC12 - O*, *BPIC13 - Incidents*, and *BPIC12 - Problems* in terms of accuracy and recall. However, the improvement is less significant. On the *Helpdesk* dataset, the F-Score of *DREAM-NAP* is only 0.3% greater than the state-of-the-art obtained by Tax *et al.* [15]. Furthermore, the method of Evermann *et al.* [16] results in the lowest observed F-Score value with 55% on this dataset, though performing very well on *BPIC12 - work complete*. This indicates that not all state-of-the-art methods perform consistently well across all datasets. For the *BPIC12 - A* dataset, we obtain slightly better scores compared to the state-of-the-art across all observed metrics. Our proposed *DREAM-NAP* model does not outperform the existing methods when considering all process events of the BPIC12 dataset, i.e. on *BPIC12 - all complete* and *BPIC12 - all*, though performing with scores close to the state-of-the-art. We show that our approach achieves a comparatively stable performance across the evaluated benchmark datasets, whereas other state-of-the-art methods perform more unstable, such as the earlier mentioned method of Evermann *et al.* [16]. In terms of precision, one observes that *DREAM-NAP* only outperforms the state-of-the-art in one-third of the datasets. It often falls to the LSTM-based approaches of Tax *et al.* [15] and Evermann *et al.* [16]. Future research is suggested to increase the precision of the *DREAM-NAP* model to reduce the false positive rate. In five out of nine cases, our approach outperforms the state-of-the-art in terms of F-score and AUC. For all *BPIC12* datasets, the obtained F-scores values are in the range of 72% to 87% describing satisfactory performance, but leaving room for improvement. Especially when analyzing the *BPIC13* datasets, one observes F-score values of 65.7% and 59.3% which are comparatively good, but which also leave room for improvements. Accordingly, the AUC values for these datasets leave opportunities for further enhancements, too.

Ultimately, *DREAM-NAP* scores consistently average to high ranks without considering resource information. This underscores that PNs extended with decay functions and token movement counters carry important information to predict the next event in running process cases. However, we also see that further research should be conducted to improve the quality of our predictions, especially in terms of precision.

The *DREAM-NAPr* architecture outperforms the state-of-the-art in terms of accuracy, precision, recall, F-score, and AUC on all eight out of eight datasets containing event resource information. Similar to the *DREAM-NAP* model, the slightest improvements are observed on the *BPIC12* datasets that consider all types of events. In these two cases, we outperform the state-of-the-art by 4.6% and 6.3% in accuracy and recall. At the same time, we improve the precision on these datasets resulting in higher and more desirable F-score values. The AUC scores of 94.2% and 92.6% indicate strong and worthwhile classification results. The results of the *BPIC12* subprocesses show accuracy, precision, and recall uptakes between 3.8% and 21%. Especially the results on *BPIC12 - work complete* and *BPIC12 - A* show that the

incorporation of event resource information can dramatically increase the predictive performance. As a result, the F-score and AUC values for these datasets are much higher than the ones of the state-of-the-art indicating consistent results with desired well-balanced false positive and false negative rates. In the same way, the results on the *BPIC13* datasets show significant improvements between 12.3% and 18.5% in terms of accuracy, precision, and recall. A large amount of available resource information contains critical information to predict the next event. Although we are improving the overall performance, predicting the next event for *BPIC13 - Problems* remains difficult. None of our reported metric scores are greater than 80% leaving space for further enhancement.

Overall, it can be seen that the predictive performance of the proposed approaches is significantly larger compared to the existing methods. Moreover, our models perform with well balanced scores across all benchmark datasets resulting in comparatively better F-score and AUC values. Solely *DREAM-NAP* on *Helpdesk*, *BPIC12 - A*, and *BPIC13 - Problems* has a 6.1%, 5.7%, and 7.1% lower precision compared to its accuracy and recall.

Figure 3 shows the training, evaluation, and validation accuracy and loss over 100 epochs of the *DREAM-NAP* architecture for each dataset. It can be seen that none of the models tend to overfit. This confirms that batch normalization layers are not required for this neural network architecture. All models demonstrate a smooth learning curve and converge after a few training epochs.

Figure 4 visualizes the same metrics scores over training epochs for the *DREAM-NAPr* models. In comparison to the previous figure, all datasets tend to overfit early. Especially on *BPIC12 - work complete*, our architecture overfits and demonstrates the importance of early stopping. It can be noted that the models which overfit the earliest and strongest, are the models that do not improve much compared to the state-of-the-art. Specifically, *DREAM-NAPr* on *BPIC12 - work complete* shows strong overfitting. At the same time, this model is more than 17% below a perfect accuracy. Similarly, overfitting can be observed on *BPIC13 - Incidents* and *BPIC13 - Problems*; two further models that result in low outperforming accuracies in comparison to all benchmarks.

The diagram shown in Figure 5 indicates the superiority of our proposed architectures in terms of accuracy. *DREAM-NAP* scores an average arithmetic rank of 2.1 whereas *DREAM-NAPr* scores on average first. In comparison, the method proposed by Tax *et al.* [15], which performs with competitive scores across all metrics and datasets, and which beats the *DREAM-NAP* model in accuracy in two of the datasets, scores an average rank of 3.

We can further statistically test whether the improvements in accuracy of our proposed approach are significant by comparing our architectures against the best state-of-the-art algorithm on each dataset. We are using a sign test due to the small number of available samples, i.e. 9 for *DREAM-NAP*
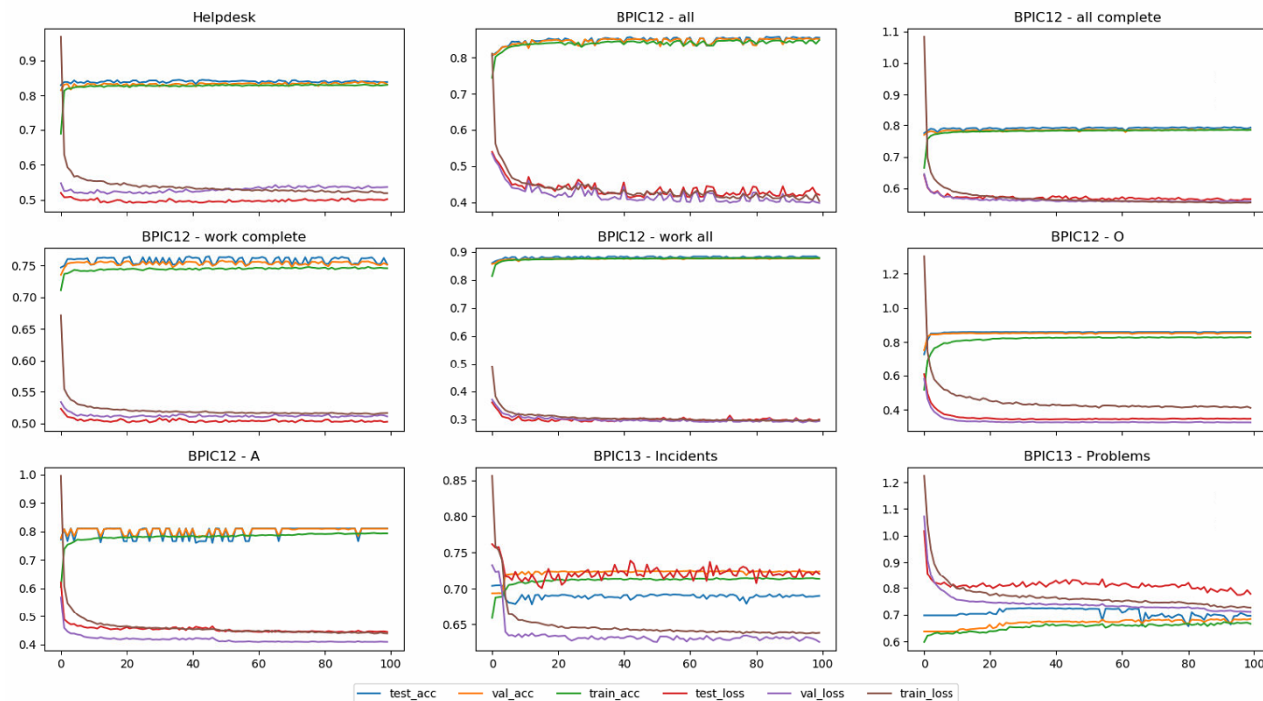
FIGURE 3. This figure shows the training, test, and validation accuracy and loss (y-axis) over 100 training epochs (x-axis) for each dataset without considering non-mandatory event attributes. Each plot shows the first cross-validation run representative for all ten runs.
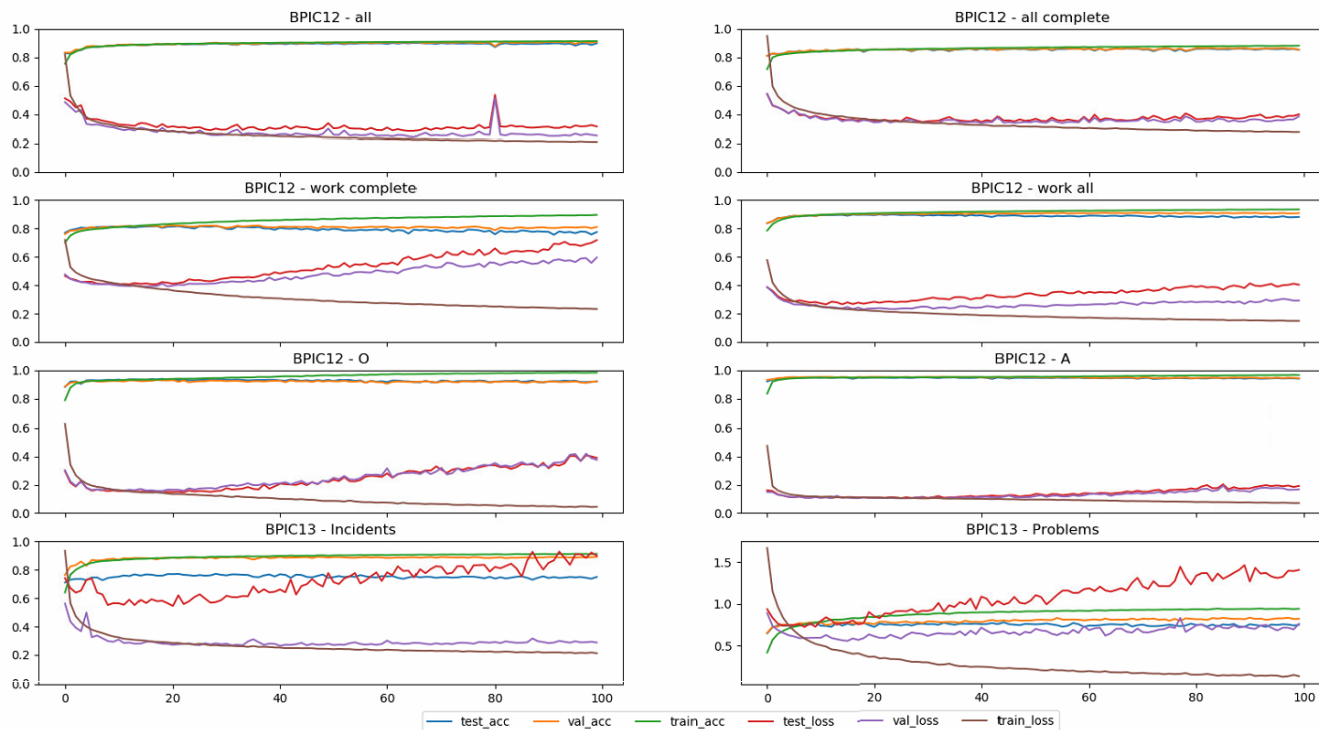

FIGURE 4. This figure shows the training, test, and validation accuracy and loss (y-axis) over 100 training epochs (x-axis) for each dataset with utilized non-mandatory event attributes. *BPIC12 - work complete, BPIC12 - O, BPIC13 - Incidents* and *BPIC13 - Problems* start overfitting comparatively early. Each plot shows the first cross-validation run representative for all ten runs.

and 8 for *DREAM-NAPr*. We set the level of significance to $\alpha = 0.05$ and adjust it using the Dunn-Sidak correction [66] to control the type I error. Therefore, the level of significance

for *DREAM-NAP* is

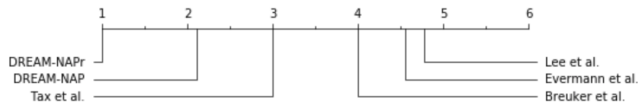$$\alpha_{nap} = 1 - (1 - 0.05)^{1/9} = 0.0057 \qquad (20)$$

FIGURE 5. Arithmetic means of ranks of the state-of-the-art and proposed approaches.

and for *DREAM-NAPr* is

$$\alpha_{napr} = 1 - (1 - 0.05)^{1/8} = 0.0064. \qquad (21)$$

The sign test for *DREAM-NAP* results in a *p*-value of 0.0898 whereas it results in a value smaller than 0.00001 for *DREAM-NAPr*. We can see that $0.00001 \leq 0.0064$, thus *DREAM-NAPr* shows significant improvements in accuracy over the state-of-the-art. This further underscores the superiority of our proposed method.

The results show that the *DREAM-NAP* model performs with consistently high metric scores across a diverse set of event logs without considering event resource information. The performance is further improved when incorporating available non-mandatory attributes using *DREAM-NAPr*. Hence, we can deduce that the proposed *DREAM* approach adds significant value to the deep learning predictor. Overall, we demonstrate statistical superiority over the state-of-the-art methods, therefore presenting major improvements.

VI. CONCLUSION

In this paper, we introduced a novel approach to predict next events in running process cases called *DREAM-NAP*. Specifically, we extended the places of PN process models with decay functions to obtain timed state samples when replaying an event log. These timed samples are used to train a deep neural network which accurately predicts the next event in a running process case. Our results surpass many state-of-the-art techniques. We obtain cross-validated accuracies above 90% and show robust, precise performances across a diverse set of real-world event logs. This underscores the feasibility and usefulness of our proposed approach.

We have shown that decay functions are a suitable tool to express a traditionally discrete PN state as a continuous representation during process runtime. In this way, we can incorporate timing information of processes directly into the process model. This is important for predictive tasks such as predicting the next event since the duration between two event instances might be correlated with a subsequent occurring event.

While most recent techniques model processes implicitly, our approach is based on explicit process models. Therefore, our method is easier to interpret than algorithms which are based exclusively on deep learning. While decision making of neural networks is naturally hard to understand and explain [67]–[69], we are retaining an interpretable process model in combination with a simple deep learning architecture. Therefore, organizations will still be able to debug their processes using graphical representations of PNs while taking advantage of the predictive capabilities. A sensitivity analysis can be performed to interpret the decision making of the neural network which performs on top of the decay function extended PN process model.

This paper introduced a promising novel approach with many potential real-world applications. High quality next event predictions are beneficial for the efficient control of real-time processes. Predicting future events helps organizations to improve scheduling, model flexible demand, and reduce system waste which are high impact problems to tackle issues like climate change [70]. Other applications can be found in atypical process mining disciplines such as in black box controller logic estimation [71] that has the potential to release huge amounts of engineers from heavy-duty of repeated controller design work. The proposed approach can further be applied in the novel process mining discipline of Human-Computer Interaction [72]. Predicting upcoming user interactions with a given computer system might unveil error-prone or inefficient interfaces and can be used to create accessible and interactive devices to overcome e.g. user impairments [73]. Finally, DREAM-NAP might have potential applications in Healthcare to understand patient's medical records with the ultimate goal to optimize treatments and diagnoses.

Further research can be conducted in the following three directions. First, the predictive quality might be able to be improved by incorporating quality performance measures of process discovery algorithms apart from fitness scores to investigate the impact of the process model quality on the proposed approach. Additionally, repair methods might be beneficial to increase the process quality and may have a positive impact on the predictive performance [74]–[76]. The outcome of such studies might further increase the metric scores on the next event prediction that we have reported. Second, we have applied the simplest kind of decay function. A comprehensive study of different decay function types might improve the predictive performance of our approach. Moreover, we proposed two deep learning architectures that have shown satisfying results on the evaluated benchmark datasets. Further optimized architectures might exist that increase the quality of predicting next events and that might overcome the low precision scores reported in Section V-E. Finally, the presented approach has been applied to next event prediction only, but might apply to further predictive process management tasks such as remaining case time prediction, next event timestamp prediction, or anomalous process state predictions.

APPENDIX: NOTATIONS

$\varnothing$	empty set
$\perp$	*non-observable event*
a	*event*
$\mathcal{A}$	finite set of all *events*
α	decay rate
α_p	decay rate for a specific place p
β	constant parameter of a decay function

$C(\tau)$	token counting vector from time 0 to τ, each element represents the number of tokens which entered a specific place
d	attribute
d_{ts}	timestamp attribute
$\mathcal{D}$	finite set of all possible attributes
$\delta_p(g)$	function of average time between a token is consumed in place p until a new token is produced in p based on an input trace g
Δ_p	time difference between current time and most recent time a token has entered place p
$\Delta_{max}(\mathcal{L})$	Maximum observed trace duration in an event log $\mathcal{L}$
E	event instance vector
ε	*split miner* filtering threshhold hyperparameter
η	*split miner* parallelism threshhold hyperparameter
$\mathcal{F}$	set of all *arcs* of a PN
$f_p(\tau)$	decay function of place p
$F(\tau)$	decay function response vector
fn	false negative
fp	false positive
fpr	false positive rate
g	*case* or *trace*
$\mathcal{G}$	finite set of all possible *traces*
$\gamma(g)$	function returning the number of event instances of a trace g
l	denotes a neural network layer in form of a matrix
$\mathcal{L}$	event log which is a set of traces
$|\mathcal{L}|$	number of traces of an event log $\mathcal{L}$
$|\mathcal{L}_i|$	number of event instances of the ith trace of an event log $\mathcal{L}$
$\mathcal{L}_{i,j}$	jth event instance in the ith trace of an event log $\mathcal{L}$
M	vector representing the *marking* of a PN
M^{final}	final marking
M^{init}	initial marking
M_i	ith element of marking M
$M(\tau)$	vector representing the *marking* of a PN at time τ
$\mathcal{M}$	set of all *markings*
$mean(\cdot)$	arithmetic mean function
$\mathcal{N}$	set of all possible *event instances*
$\nu_p(g)$	number of tokens a place p produces when replaying a trace g
p	place
$\mathcal{P}$	set of all *places*
$|\mathcal{P}|$	cardinality of set of places
PN	mathematical definition of a labeled PN
$\phi_j(l_k)$	weighted input of neuron j from previous layer l_k
π	function which maps a transition to either a single observable event or to the non-observable event
$R(\tau)$	attribute value counting vector from time 0 to τ

$\mathcal{R}$	set of all α_p of a PN
$|\mathcal{R}|$	cardinality of the set $\mathcal{R}$
$\rho_j(l_k)$	activation function based on input layer l_k
$S(\tau)$	timed state sample at time τ
$\mathcal{S}$	set of timed state samples
$\sigma(p)$	function returning number of tokens of a place p
t	transition
tn	true negative
tp	true positive
tpr	true positive rate
$\mathcal{T}$	set of all *transitions*
τ	time
τ_p	most recent time that a token entered place p
$\theta_j(l_k)$	output function of a neuron based on input layer l_k
$\upsilon_d(E)$	function returning value of *attribute d* of event instance E
$w_{i,j}$	weight of direct connection between two neurons i and j
$\bullet x$	set of input nodes of a node x
$x\bullet$	set of output nodes of a node x
ξ	cost function
$\mathcal{Z}$	set of all non-negative integers
i, j, k, n, x, y	indices, integers, and variables used in different contexts

REFERENCES

[1] C. Janiesch, A. Koschmider, M. Mecella, B. Weber, A. Burattin, C. Di Ciccio, A. Gal, U. Kannengiesser, F. Mannhardt, J. Mendling, A. Oberweis, M. Reichert, S. Rinderle-Ma, W. Song, J. Su, V. Torres, M. Weidlich, M. Weske, and L. Zhang, "The Internet-of-Things meets business process management: Mutual benefits and challenges," 2017, *arXiv:1709.03628*. [Online]. Available: https://arxiv.org/abs/1709.03628

[2] R. Seiger, U. Assmann, and S. Huber, "A case study for workflow-based automation in the Internet of Things," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Apr./May 2018, pp. 11–18.

[3] I. Hwang and Y. J. Jang, "Process mining to discover shoppers' pathways at a fashion retail store using a WiFi-base indoor positioning system," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 4, pp. 1786–1792, Oct. 2017.

[4] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. van der Aalst, "Process mining can be applied to software too!" in *Proc. 8th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Sep. 2014, p. 57.

[5] H. Darabi, W. L. Galanter, J. Y.-Y. Lin, U. Buy, and R. Sampath, "Modeling and integration of hospital information systems with Petri nets," in *Proc. IEEE/INFORMS Int. Conf. Service Oper., Logistics Informat.*, Jul. 2009, pp. 190–195.

[6] S. Goedertier, J. De Weerdt, D. Martens, J. Vanthienen, and B. Baesens, "Process discovery in event logs: An application in the telecom industry," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 1697–1710, 2011.

[7] M. Haji and H. Darabi, "Petri net based supervisory control reconfiguration of project management systems," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Sep. 2007, pp. 460–465.

[8] N. Wightkin, U. Buy, and H. Darabi, "Formal modeling of sequential function charts with time Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 19, no. 2, pp. 455–464, Mar. 2011.

[9] B. B. Van Dongen. (2012). *BPI Challenge 2012*. [Online]. Available: https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

[10] W. M. P. van der Aalst, "Process-aware information systems: Lessons to be learned from process mining," in *Transactions on Petri Nets and Other Models of Concurrency II*, K. Jensen and W. M. P. Aalst, Eds. Berlin, Germany: Springer-Verlag, 2009, pp. 1–26.

[11] I. Verenich, "A general framework for predictive business process monitoring," in *Proc. CAiSE Doctoral Consortium (CAiSE)*, Ljubljana, Slovenia, Jun. 2016. [Online]. Available: https://eprints.qut.edu.au/97539/

[12] D. Breuker, M. Matzner, P. Delfmann, and J. Becker, "Comprehensible predictive models for business processes," *MIS Quart.*, vol. 40, no. 4, pp. 1009–1034, Dec. 2016.

[13] B. Kang, D. Kim, and S.-H. Kang, "Real-time business process monitoring method for prediction of abnormal termination using KNNI-based LOF prediction," *Expert Syst. Appl.*, vol. 39, no. 5, pp. 6061–6068, Apr. 2012.

[14] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime prediction of service level agreement violations for composite services," in *Proc. ICSOC/ServiceWave Workshops Service-Oriented Comput.* Berlin, Germany: Springer, 2009, pp. 176–186. doi: 10.1007/978-3-642-16132-2_17.

[15] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Cham, Switzerland: Springer, 2017, pp. 477–492.

[16] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decis. Support Syst.*, vol. 100, pp. 129–140, Aug. 2017.

[17] A. Khan, H. Le, K. Do, T. Tran, A. Ghose, H. Dam, and R. Sindhgatta, "Memory-augmented neural networks for predictive process analytics," 2018, *arXiv:1802.00938*. [Online]. Available: https://arxiv.org/abs/1802.00938

[18] N. Mehdiyev, J. Evermann, and P. Fettke, "A novel business process prediction model using a deep learning method," in *Business & Information Systems Engineering*. Wiesbaden, Germany: Springer, 2018, pp. 1–15. doi: 10.1007/s12599-018-0551-3.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–11.

[20] W. Reisig and G. Rozenberg, Eds., *Lectures on Petri Nets I: Basic Models Advances in Petri Nets*. London, U.K.: Springer-Verlag, 1998.

[21] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed. Berlin, Germany: Springer, 2011. doi: 10.1007/978-3-642-19345-3.

[22] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan, "Business process intelligence," *Comput. Ind.*, vol. 53, no. 3, pp. 321–343, Apr. 2004.

[23] W. M. P. Van Der Aalst, "Business process management: A comprehensive survey," *ISRN Softw. Eng.*, vol. 2013, Sep. 2013, Art. no. 507984.

[24] C. J. Turner, A. Tiwari, R. Olaiya, and Y. Xu, "Process mining: From theory to practice," *Bus. Process Manage. J.*, vol. 18, no. 3, pp. 493–512, 2012.

[25] M. Polato, A. Sperduti, A. Burattin, and M. de Leoni, "Time and activity sequence prediction of business process instances," *Computing*, vol. 100, no. 9, pp. 1005–1031, 2018.

[26] N. Navarin, B. Vincenzi, M. Polato, and A. Sperduti, "LSTM networks for data-aware remaining time prediction of business process instances," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov./Dec. 2017, pp. 1–7.

[27] C. Di Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, and A. Yeshchenko, "An eye into the future: Leveraging *a-priori* knowledge in predictive business process monitoring," in *Business Process Management*, J. Carmona, G. Engels, and A. Kumar, Eds. Cham, Switzerland: Springer, 2017, pp. 252–268.

[28] M. Le, B. Gabrys, and D. Nauck, "A hybrid model for business process event prediction," in *Research and Development in Intelligent Systems XXIX*, M. Bramer and M. Petridis, Eds. London, U.K.: Springer, 2012, pp. 179–192.

[29] J. Becker, D. Breuker, P. Delfmann, and M. Matzner, "Designing and implementing a framework for event-based predictive modelling of business processes," in *Enterprise modelling and Information Systems Architectures—EMISA*, F. Feltz, B. Mutschler, and B. Otjacques, Eds. Bonn, Germany: Gesellschaft für Informatik e.V., 2014, pp. 71–84.

[30] M. Ceci, P. F. Lanotte, F. Fumarola, D. P. Cavallo, and D. Malerba, "Completion time and next activity prediction of processes using sequential pattern mining," in *Discovery Science*, S. Džeroski, P. Panov, D. Kocev, and L. Todorovski, Eds. Cham, Switzerland: Springer, 2014, pp. 49–61.

[31] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, "A Markov prediction model for data-driven semi-structured business processes," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 97–126, Jan. 2015.

[32] M. Unuvar, G. T. Lakshmanan, and Y. N. Doganata, "Leveraging path information to generate predictions for parallel business processes," *Knowl. Inf. Syst.*, vol. 47, no. 2, pp. 433–461, May 2016.

[33] W. L. J. Lee, D. Parra, J. Munoz-Gama, and M. Sepúlveda, "Predicting process behavior meets factorization machines," *Expert Syst. Appl.*, vol. 112, pp. 87–98, Dec. 2018.

[34] Q. Guo, L. Wen, J. Wang, Z. Yan, and P. S. Yu, "Mining invisible tasks in non-free-choice constructs," in *Business Process Management*, H. R. Motahari-Nezhad, J. Recker, and M. Weidlich, Eds. Cham, Switzerland: Springer, 2015, pp. 109–125.

[35] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Proc. IEEE 15th Int. Enterprise Distrib. Object Comput. Conf.*, Sep. 2011, pp. 55–64.

[36] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, "Soundness of workflow nets: Classification, decidability, and analysis," *Formal Aspects Comput.*, vol. 23, no. 3, pp. 333–363, May 2011.

[37] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits, Syst., Comput.*, vol. 8, no. 1, pp. 21–66, 1998.

[38] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. van der Aalst, "Process mining can be applied to software too!" in *Proc. 8th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*. New York, NY, USA: ACM, 2014, pp. 57:1–57:8.

[39] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy, "Split miner: Automated discovery of accurate and simple business process models from event logs," *Knowl. Inf. Syst.*, vol. 59, no. 2, pp. 251–284, May 2018.

[40] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 686–705, Apr. 2018.

[41] R. Conforti. (2016). *Research Code*. [Online]. Available: https://github.com/raffaeleconforti/ResearchCode

[42] A. Zell, "Komponenten neuronaler modelle," in *Simulation Neuronaler Netze*. Reading, MA, USA: Addison-Wesley, 1994, ch. 6, pp. 87–95.

[43] I. Goodfellow, Y. Bengio, and A. Courville, "Deep feedforward networks," in *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. 6, pp. 164–223.

[44] F. Chollet *et al.* (2015). *Keras*. [Online]. Available: https://keras.io

[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and S. Ghemawat, (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: http://tensorflow.org/

[46] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn. (ICML)*. Madison, WI, USA: Omnipress, 2010, pp. 807–814.

[47] B. Karlik and A. Vehbi, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2015.

[48] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 8, pp. 1553–1565, Aug. 2014.

[49] X. Zhai, A. A. S. Ali, A. Amira, and F. Bensaali, "MLP neural network based gas classification system on Zynq SoC," *IEEE Access*, vol. 4, pp. 8138–8146, 2016.

[50] W. Sun, S. Shaoa, R. Zhaob, R. Yana, X. Zhangc, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, Jul. 2016. doi: 10.1016/j.measurement.2016.04.007.

[51] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Int. Conf. Mach.*, 2015, pp. 448–456.

[54] M. M. Polato. (2017). *Dataset Belonging to the Help Desk Log of an Italian Company*. [Online]. Available: https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb

[55] W. Steeman. (2013). *BPI Challenge 2013*. [Online]. Available: https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07

[56] A. Rozinat and W. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Inf. Syst.*, vol. 33, no. 1, pp. 64–95, 2008.

[57] A. Rozinat and W. M. P. van der Aalst, "Conformance testing: Measuring the fit and appropriateness of event logs and process models," in *Business Process Management Workshops*, C. J. Bussler and A. Haller, Eds. Berlin, Germany: Springer, 2006, pp. 163–176.

[58] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from incomplete event logs," in *Application and Theory of Petri Nets and Concurrency*, G. Ciardo and E. Kindler, Eds. Cham, Switzerland: Springer, 2014, pp. 91–110.

[59] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs," *Inf. Syst.*, vol. 37, no. 7, pp. 654–676, 2012.

[60] C. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). Berlin, Germany: Springer-Verlag, 2006.

[61] I. Goodfellow, Y. Bengio, and A. Courville, "Regularization for deep learning," in *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. 7, p. 247.

[62] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," 2016, *arXiv:1611.03530*. [Online]. Available: https://arxiv.org/abs/1611.03530

[63] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.

[64] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, 1997.

[65] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.

[66] Z. Šidák, "Rectangular confidence regions for the means of multivariate normal distributions," *J. Amer. Stat. Assoc.*, vol. 62, no. 318, pp. 626–633, 1967.

[67] G. Marcus, "Deep learning: A critical appraisal," 2018, *arXiv:1801.00631*. [Online]. Available: https://arxiv.org/abs/1801.00631

[68] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *Proc. 41st Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2018, pp. 0210–0215.

[69] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," 2017, *arXiv:1708.08296*. [Online]. Available: https://arxiv.org/abs/1708.08296

[70] D. Rolnick, P. L. Donti, L. H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A. S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown, and A. Luccioni, "Tackling climate change with machine learning," 2019, *arXiv:1906.05433*. [Online]. Available: https://arxiv.org/abs/1906.05433

[71] J. Theis, I. Mokhtarian, and H. Darabi, "Process mining of programmable logic controllers: Input/output event logs," 2019, *arXiv:1903.09513*. [Online]. Available: https://arxiv.org/abs/1903.09513

[72] J. Theis and H. Darabi, "Behavioral Petri net mining and automated analysis for human-computer interaction recommendations in multi-application environments," in *Proc. ACM Hum.-Comput. Interact.*, vol. 3, pp. 13:1–13:16, Jun. 2019.

[73] J. O. Wobbrock, "Situationally aware mobile devices for overcoming situational impairments," in *Proc. ACM SIGCHI Symp. Eng. Interact. Comput. Syst. (EICS)*. New York, NY, USA: ACM, 2019, pp. 1:1–1:18.

[74] Z. He, Y. Du, L. Qi, and H. Du, "A model repair approach based on Petri nets by constructing free-loop structures," *IEEE Access*, vol. 7, pp. 24214–24230, 2019.

[75] Y. Xu, Y. Du, L. Qi, W. Luan, and L. Wang, "A logic Petri net-based model repair approach by constructing choice bridges," *IEEE Access*, vol. 7, pp. 18531–18545, 2019.

[76] W. T. Zheng, Y. Y. Du, L. Qi, and L. Wang, "A method for repairing process models containing a choice with concurrency structure by using logic Petri nets," *IEEE Access*, vol. 7, pp. 13106–13120, 2019.

JULIAN THEIS received the B.Eng. degree in media technology and the M.Eng. degree (Hons.) in media and communications technology from the RheinMain University of Applied Sciences, Wiesbaden, Germany, in 2014 and 2016, respectively. He is currently pursuing the Ph.D. degree with the Mechanical and Industrial Engineering Department, University of Illinois at Chicago.

Before joining Prominent Laboratory, the university's foremost research facility in process mining, he was working as a Software Engineer for a leading company in media and workflow automation, broadcast management, and OSS software in Germany. His research interests include process mining, its applications to industrial, IT, business processes, and deep learning.

HOUSHANG DARABI (S'98–A'00–M'10–SM'14) received the Ph.D. degree in industrial and systems engineering from Rutgers University, New Brunswick, NJ, USA, in 2000.

He is currently an Associate Professor with the Department of Mechanical and Industrial Engineering, University of Illinois at Chicago (UIC), where he is also an Associate Professor with the Department of Computer Science. He has been a contributing author of two books in the areas of scalable enterprise systems and reconfigurable discrete event systems. His research has been supported by several federal and private agencies, such as the National Science Foundation, the National Institute of Standard and Technology, the Department of Energy, and Motorola. He has extensively published on various automation and project management subjects, including wireless sensory networks for location sensing, planning and management of projects with tasks requiring multi-mode resources, and workflow modeling and management. He has published in different prestigious journals and conference proceedings, such as the IEEE Transactions on Robotics and Automation, IEEE Transactions on Automation Science and Engineering, IEEE Transactions on Systems, Man, and Cybernetics, and *Information Sciences*. His current research interests include the application of data mining, process mining, and optimization in design and analysis of manufacturing, business, project management, and workflow management systems.

● ● ●