Hindawi
Complexity
Volume 2021, Article ID 7887159, 24 pages
https://doi.org/10.1155/2021/7887159

Research Article

Event-Tree Based Sequence Mining Using LSTM Deep-Learning Model

János Abonyi, Richárd Károly, and Gyula Dörgö

MTA-PE Lendület Complex Systems Monitoring Research Group, Department of Process Engineering, University of Pannonia, Egyetem u. 10, Veszprém H-8200, Hungary

Correspondence should be addressed to Gyula Dörgö; gydorgo@gmail.com

Received 10 June 2021; Accepted 31 July 2021; Published 16 August 2021

Academic Editor: Gonzalo Farias

During the operation of modern technical systems, the use of the LSTM model for the prediction of process variable values and system states is commonly widespread. The goal of this paper is to expand the application of the LSTM-based models upon obtaining information based on prediction. In this method, by predicting transition probabilities, the output layer is interpreted as a probability model by creating a prediction tree of sequences instead of just a single sequence. By further analyzing the prediction tree, we can take risk considerations into account, extract more complex prediction, and analyze what event trees are yielded from different input sequences, that is, with a given state or input sequence, the upcoming events and the probability of their occurrence are considered. In the case of online application, by utilizing a series of input events and the probability trees, it is possible to predetermine subsequent event sequences. The applicability and performance of the approach are demonstrated via a dataset in which the occurrence of events is predetermined, and further datasets are generated with a higher-order decision tree-based model. The case studies simply and effectively validate the performance of the created tool as the structure of the generated tree, and the determined probabilities reflect the original dataset.

1. Introduction

Nowadays, uncovering possible frequent event sequence scenarios has been a critical task across many disciplines. In the age of big data, when an immense amount of data is recorded into logs in the scope of the industry 4.0 trend, it is important for engineers to acquire as much knowledge about the industrial processes as possible [1, 2]. By using frequent pattern mining algorithms on event logs, we are able to identify sequences that can lead to given system states. This particular method has already proved its capability across numerous applications and industries. Taub et al. use sequence mining to distinguish efficient and nonefficient action patterns among their subjects in a game-based learning environment [3]. A similar frequent pattern identification method was used to give insight into successful learning patterns using Betty's brain computer-based learning environment [4]. A universal (language independent) algorithm was proposed for linguistical pattern discovery, where special attention was paid to a clear, easily understandable output [5]. Kant et al. proposed a new algorithm (*MC*PRISM) to mine min-closed sequences to identify comment section spam content on websites [6]. A new framework called malicious sequential pattern-based malware detection was developed by using a novel sequential pattern mining algorithm (MSPE) to recognize new, unseen malicious executables in computer systems [7]. Weiss uses a genetic algorithm for analyzing the temporal patterns in the alarm data of telecommunication systems to identify equipment failure [8]. Sequential pattern mining has been also used for event prediction in numerous applications [9, 10].

Although these examples are perfectly capable of fulfilling the sequential pattern mining task, traditional algorithms suffer greatly with runtime and accuracy when dealing with massive datasets [11]. Another drawback of the frequent pattern mining solutions is that their output data are proved to be challenging to interpret and

handle—especially when the number of the mined sequences is high—often introducing a new problem to solve [12]. To represent the yielded information, the frequent pattern tree proved to be a much more compact and workable data structure [13].

Machine learning techniques are excellent tools for processing massive datasets. Learning patterns from exemplary training sequences is a similar task as in the case of learning languages and the identification of frequent event sequences, where the use of long-short term memory (LSTM) yields better results compared to that of traditional recurrent neural networks (RNNs) [14]. The reason why LSTM is suitable for this application is the use of the forget gate in its cell, which is able to reset the internal state of the network [15]. The algorithm known as the seq2seq learning method was developed in 2014 by Sutskever et al. at Google for frequent sequence learning using LSTM to improve machine translation [16]. Ever since this method has been used in numerous applications. Karatzoglou et al. used it to improve location-based services by learning human semantic trajectories and better predicting their upcoming location [17]. The method's capabilities have also been demonstrated in finances by Rebane et al. who analyzed the performance for cryptocurrency price prediction [18]. A seq2seq model-based approach was used to improve query focused summarization performance [19]. Wu et al. described a novel method to create, store, and convert logs of Internet of Things big data systems to be later processed through their proposed seq2seq algorithm [20]. The method has also been applied in manufacturing systems. Hwang et al. used the algorithm to predict a furnace temperature based on other process variables with a very high accuracy [21]. The general application of this structure for event prediction has been described in detail by Dörgő et al. [22, 23].

Fundamentally, the output of a seq2seq approach is a single sequence, which consists of the items that have been found as the most probable at each prediction step. By using a heuristic search algorithm during inference, further information can be retained from each prediction step. This information can aid us to understand better the black-box model of the prediction [24]. This optimization is done using beam search, which retains several best items—the number usually referred to as beam width. Cohen and Beck studied the performance degradation in neural sequence models when an inappropriate beam width is chosen [25]. In recent years, the use of beam search instead of the traditional greedy search was favored because it usually provides much better results, although it is taxing on runtime [26].Li et al. used a seq2seq model with beam search decoder to realize a dependency parser with a direct head prediction with promising performance [27]. Williams et al. proposed the use of beam search to build an end-to-end speech recognition system, which is capable of adapting the inference process based on contextual signals at each prediction step [28]. Several different pruning strategies have been explored to be used with beam search to improve runtime [29]. A seq2seq model using the dynamic beam width was applied by Jahier Pagliari et al. to an embedded translation system in order to improve its efficiency [30]. A known drawback of the beam search algorithm is that it produces pretty similar output sequences in certain use cases. A solution for this phenomenon was proposed for image captioning [31].

This paper aims to create own implementation of the seq2seq learning method with a beam search decoder, which is referred to as the seq2probTree method later. This method will be realized in the Python environment, and it is able to create a probability tree that describes the alternative network of events based on a given input. The implemented tool is capable of displaying the output an easy to interpret, structured probability tree, thus giving a visualization of the prediction and aiding the debugging of seq2seq models, as the fault analysis of deep neural networks is a task with enormous importance, especially in the case of safety-critical application [32].

First, in Section 2, the methodology will be explained. Definitions will be given to the necessary expressions and the prediction task at hand. The LSTM deep-learning model will be described along with the tree creation process. The metrics used for the evaluation will also be defined in this section. In Section 3, the implementation process and the used toolboxes will be presented briefly. Then, the seq2probTree method will be put to the test by applying it on a first-order Markov chain model and later on a higher-order tree-based system, where the extent to which the method is able to reconstruct the tree is checked, and the necessary comparison score is defined. Finally, the real-life practical applicability is confirmed by using it on the alarm logs of a hydrofluoric acid alkylation production unit. Last, in Section 4, the findings and experiences of using the developed method will be summarized, and further steps in the subject will be proposed.

2. Methodology

In this section, the previously defined task will be explained in detail. The definition will be given to an event sequence and how its probability is calculated. The peculiarity of the seq2probTree method is explained, creating a whole sequence tree instead of only predicting the most likely scenario. Here, in addition to the theory of prediction, its extension to tree-based event-scenario generation is also provided. The metrics used for the evaluation of the predicted event scenarios are also explained in detail.

2.1. Sequences and the Prediction Task. Industrial processes frequently generate *event logs* those are logically consisting of *events* (denoted as e_i) related to production, safety, transportation, storage, sales, financial transactions, marketing, etc. An event log defined as D_T database is an ordered list of these events, where the events are arranged according to their start time in the ascending order. The D_T dataset can be segmented into *sequences* (denoted as Φ_n), which are the chronologically ordered lists of events $\Phi_k := e_1 \Rightarrow e_2 \Rightarrow \cdots \Rightarrow e_k$. According to different aspects, this segmentation can be carried out: causal connection of states, temporal segmentation, periodicity, etc. Therefore, a

sequence of k events is referred to as a k-length sequence and is denoted by Φ_k. These events represent the occurrence of n different states (type of events) of the set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$. The sequence Φ_k: $= e_1 \Rightarrow e_2 \Rightarrow \cdots \Rightarrow e_k$ can be divided chronologically at any part as $\Phi_k = (\Phi_{k'} \Rightarrow \Phi_{k''})$, where $\Phi_{k'}$ and $\Phi_{k''}$ are the antecedent and future sequence of states, respectively (naturally, $k = k' + k''$). Hereinafter, the "$'$" and "$''$" symbols denote the past and future sequences or states, respectively.

As single or multiple connected processes usually generate the data analyzed here, a causal flow connects the individual temporal instances of states (regardless of the type of the dataset, e.g., events, items, transactions, etc.), and the number of occurrences of different states is not independent of each other. Therefore, the probability of the occurrence of the Φ_k sequence $P(e_1 \Rightarrow e_2 \Rightarrow e_3 \Rightarrow \cdots \Rightarrow e_k)$ can be calculated by the chain rule and the conditional probabilities of transitions between the events according to the following equation:

$$P(\Phi_k) = P(e_1) \times P(e_2 \mid e_1) \times P(e_3 | e_1 \Rightarrow e_2) \times \cdots \cdots \times P(e_k \mid e_1 \Rightarrow e_2 \Rightarrow \cdots \Rightarrow e_{k-1}). \quad (1)$$

Therefore, according to the chain rule, the probability of a k-length sequence can be calculated as the product of the conditional probabilities of the step-by-step transition from the sequence of antecedent events to the present one. A conditional probability is the ratio of the number of occurrences of the more extended sequence and the shorter one, denoted by the supp value of the sequence, according to the following equation:

$$P(e_k \mid \Phi_{k-1}) = \frac{P(\Phi_k)}{P(\Phi_{k-1})} = \frac{\mathrm{sup}p(\Phi_k)}{\mathrm{sup}p(\Phi_{k-1})}. \quad (2)$$

This probability of transition reflects how confident is the next state knowing the previous sequence of states in Φ_{k-1}.

2.2. The Network of Alternative Events: Sequence Trees. The methodology where the prediction of the following state with the highest conditional probability is accepted was described by Dörgő and Abonyi [22]. However, the underlying processes and, hence, the resultant datasets can be highly complex. The ultimate goal of this method is being able to create an event sequence tree that describes the possible courses (all highly probable $\Phi_{k''}$) of events based on a given input sequence ($\Phi_{k'}$). Figure 1 shows the idea in detail. The horizontal axis indicates the time and illustrates how the possible future scenarios after $k\prime$ past events are ordered in a tree structure. The red branch of the tree indicates the scenario if the predictions of the highest probability are accepted in every prediction step, namely, by using the greedy search algorithm. The EOS tag indicates the end-of-sequence prediction.

So far, only the scenario with the highest probability has been predicted, ignoring the possibility of the occurrence of a less likely, however, highly informative and essential subsequence, which can indicate a different scenario of upcoming events. The added feature of this method is to uncover the information that these highly probable sequences may yield.

Therefore, accepting that the conditional probability-based prediction model often predicts several events with similar probability, here, the implemented beam search algorithm is described, thus not just the future sequence with the highest probability is accepted, but a scenario tree is formalized accepting all the predicted events above a certain probability threshold (P_{thr}). Therefore, after the occurrence of the first k' events, the prediction of the first future event e_1'' is accepted if its confidence of transition is above a specific P_{thr} limit as follows:

$$\left\{ e_1'' | P\left(e_1'' | \Phi_{k\prime}\right) > P_{\mathrm{thr}} \right\}. \quad (3)$$

Applying equation (3) in every prediction step, not a single future sequence but multiple sequences or possible future scenarios are predicted as depicted in Figure 1. Thus, as it is described by the prediction task, the $P(\Phi_{k''} | \Phi_{k\prime})$ conditional probability is to be determined among all possible future $\Phi_{k''}$ sequences.

In order to annotate the scenarios as well, a hierarchical annotation was introduced in the superscript of the predicted event: the numbers divided by commas after the "$\prime\prime$" mark indicate the likeliness of the predicted event in the prediction step as the number in order of the likeliness of the prediction, where 1 indicates the most likely future state. For instance, the tag $e^{\prime\prime,1,3,1}$ shows that this is the third predicted future event (three numbers are present after the "$\prime\prime$" mark), and this was the event with the highest probability for the first predicted state $e^{\prime\prime,1}$; then accepting this prediction, the second predicted event has the third highest probability $e^{\prime\prime,1,3}$ and accepting the first two predictions, the third predicted event had the highest probability in the given prediction step. Similarly, the $e^{\prime\prime,2,1}$ future state is the prediction with the second highest probability ($e^{\prime\prime,2}$) for the first future event and accepting this prediction, this is the prediction with the highest probability in the second prediction step. Therefore, continuously accepting the most likely predictions, the sequence $e^{\prime\prime,1} \Rightarrow e^{\prime\prime,1,1} \Rightarrow e^{\prime\prime,1,1,1} \Rightarrow \cdots$ is predicted, highlighted by the red arrow in Figure 1. However, in this sequence, the predictions with the highest probability are accepted in every step, the overall probability of the sequence is not maximal in every situation, since after the acceptance of a less likely prediction in a prediction step, the following predicted events could be of a high probability and then the overall probability of the occurrence of the sequence can be relatively high (the overall probability of the occurrence of a sequence is the product of the transition probabilities according to equation (1)).

By repeating the prediction task at each node, the sequence tree explained in Figure 1 may be created. After each prediction step, by meeting the confidence of all the possible events to the previously defined P_{thr} probability limit, we can make sure that we keep the complexity of the tree as low as necessary for the given task.

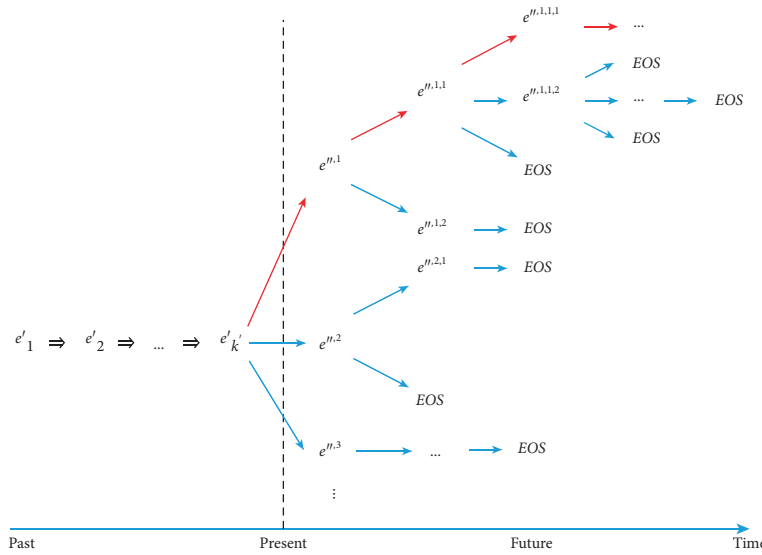

Figure 1: The predicted scenarios ordered in a tree structure (the EOS tag indicates the end-of-sequence prediction).

2.3. The LSTM Deep-learning Model. In the seq2seq machine learning method, the so-called long-short term memory is utilized as a recurrent neural network of choice. This network was specifically developed to deal with the problem of vanishing gradients with the least possible computational cost increase [33]. The LSTM network is well-known for its capability of classification, processing, and prediction making on time series data due to its relative insensitivity to gap length (lag) between discreet events, which property is welcome in the given use case. The LSTM structure is depicted in Figure 2.

The input of the model: Figure 2 highlights the structure of the input sequences. First, an end-of-sequence (EOS) tag is appended to the end of every sequence to indicate the end of the event series. The implemented EOS tag is added to the end of the sequences and handled similarly to all the other events in the subsequent steps. Moreover, the order of the events in the input sequence is reversed, since according to Sutskever et al. [16], the prediction accuracy significantly improves when the beginning of the input sequence is close to the beginning of the predicted sequence.

Embedding layer: The described sequence of input events needs to be transformed into a mathematically manageable vector of numerical values. Therefore, first, the symbols are encoded as one-hot encoded vectors, $\mathbf{oh}_t$ of binary values of length n_d, where n_d is the number of one-hot encoded symbols. In the one-hot encoded vectors, only one bit related to the encoded symbol is fired. A detailed explanation and visualization of one-hot encoding can be found in [34]. Then, the embedding layer transforms the one-hot coded vectors into a lower dimension (n_e) of continuous values using a $\mathbf{x}_t = \mathbf{W}_{\text{emb}}\mathbf{oh}_t$ linear transformation. Note that, in Figure 2, the embedded forms of the EOS symbol are denoted by the symbol EOS.

Encoder and decoder layers: The encoder LSTM layer processes the sequence of one-hot coded and then embedded symbols. Instead of calculating its output values, it maps the sequence into its internal states. These internal weights of the encoder layer represent the state of the process, which generated the events. These weights are used to condition the decoder layer, which means the transfer of information of that happened previously in the process and generally means copying the encoder layer's weights into the decoder layer, obtaining the same structure (of n_u LSTM units). These weights indicate the prediction required from the decoder layer. After the input of an (embedded) start-of-sequence symbol, the decoder layer predicts the next event of the predicted sequence iteratively, consistently applying the previously predicted event as the input for the prediction of the next event. This procedure is repeated until an end-of-sequence symbol is predicted or the maximum sequence length is reached.

Dense layer: After the decoder layer maps the input event $\mathbf{x}_{k''}$ into a vector of real values $\mathbf{h}''$ represented as $\mathbf{h}'' = [h''_1, \ldots, h''_{n_U}]$, these values are used to calculate the probabilities of occurrence of the events using the softmax activation function of the dense layer in Figure 2,

$$P\left(e_{(t+1)''} \mid \mathbf{x}_{t''}\right) = P\left(e_{(t+1)''} \mid \mathbf{h}_{t''}\right) = \frac{\exp\left(\left(\mathbf{h}_{t''}\right)^T \mathbf{w}_{s,j} + b_j\right)}{\sum_{j=1}^{n_d} \exp\left(\left(\mathbf{h}_{t''}\right)^T \mathbf{w}_{s,j} + b_j\right)}, \tag{4}$$

where $\mathbf{w}_{s,j}$ represents the j-th column vector of the weight matrix of the output dense layer of the network $\mathbf{W}_s$, and b_j represents the degree of bias. Once the probability of each state in our dictionary is determined, all the predictions above the defined threshold P_{thr} is accepted as the next event of the related future scenario,

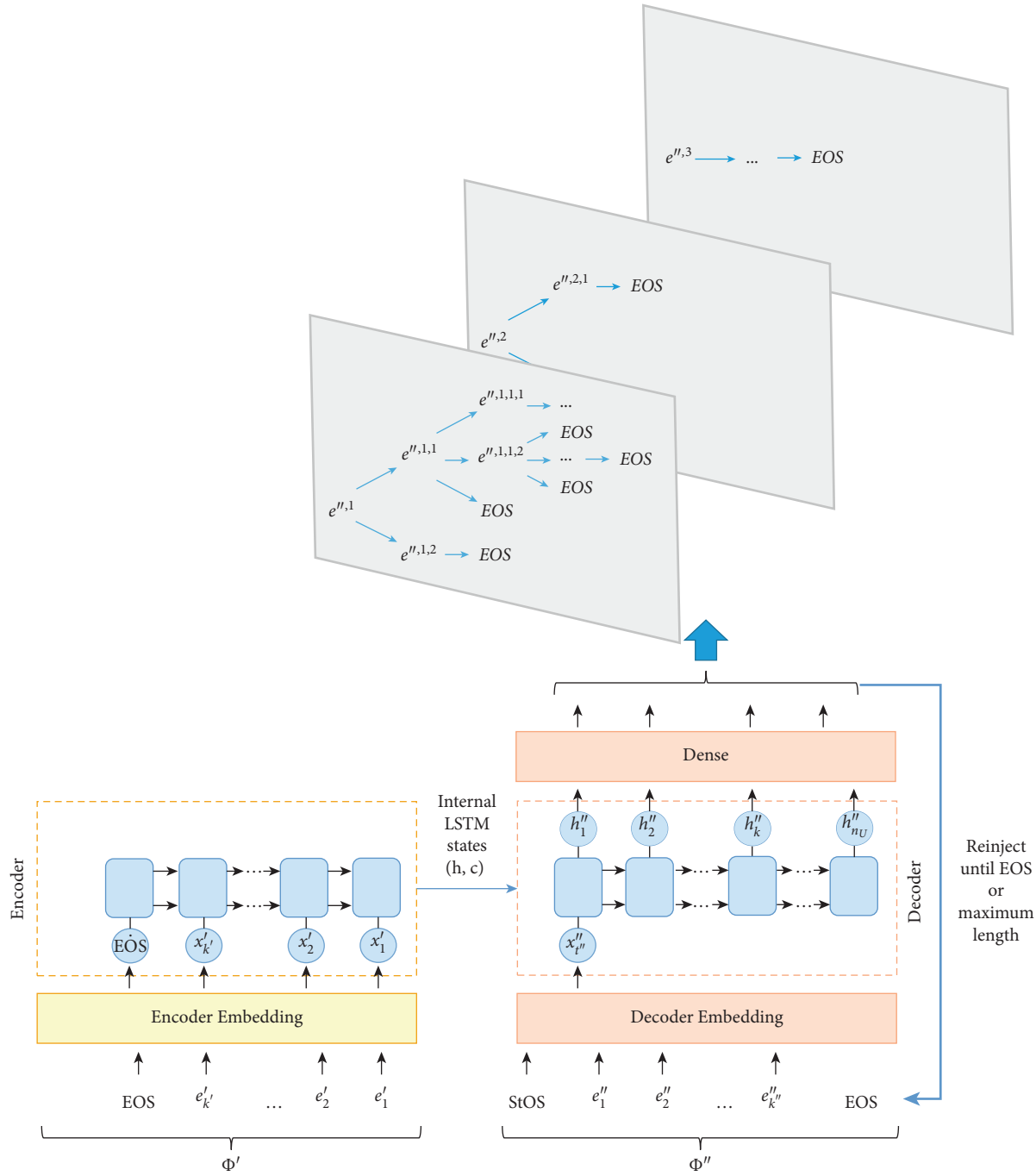

FIGURE 2: The illustration of the structure of the sequence-to-sequence event-scenario prediction. The encoder model maps the states of the input sequence into a fixed-length vector-based representation. Using these vector-based representations of input events as the initial state, the decoder model determines the next event. However, using the probabilities calculated by the dense layer, not just the event with the highest probability is recorded, but event scenarios are predicted using every prediction above a predefined threshold. The StOS and EOS tags mark the start-of-sequence and end-of-sequence tags, respectively.

$$\left\{e_{(t+1)''} \mid P\left(e_{(t+1)''} \mid \mathbf{h}_{t''}\right) > P_{\text{thr}}\right\}. \tag{5}$$

2.4. Creation and Traversal of the Probability Trees. Prior to prediction, the sequence of events that defines the state of the process is to be transformed to the internal state of the encoder layer. Then, these internal states of the encoder layer containing information on the history of the process are transferred to the decoder layer. The prediction starts with the input of a start-of-sequence symbol (marked as StOS in Figure 2). The decoder network generates the prediction of the next event, which is reintroduced into the input of the decoder network and applied as the input in the next time

step. By utilizing the original seq2seq learning method, the generated events are continuously appended to the predicted sequence of events. The feature added by the seq2probTree method is that after the first prediction step following the start-of-sequence symbol, we do not simply accept an event as the next with the highest probability. However, we take the entire output vector and apply equation (3), thus pruning the candidates for the next possible event. Then, we further explore the network of alternative events during which the probability of each upcoming event is determined (and stored if that probability is adequate), thereby realizing the beam search algorithm. The prediction process is continued until the layer generates the end-of-sequence symbol or reaches the previously set limit of the length of the predicted sequence in the case of every scenario.

The method results in a probability tree that is explored and recorded in a depth-first manner (Figure 3). The resource demand of this approach is significantly increased as it is necessary to store all the internal LSTM states and the previous prediction's output for each step—depending on the original number of the possible events—could be a memory hog. In addition, an increase in the inference runtime is expected as the time demand of the depth-first search algorithm is $O(|V| + |E|)$, where V and E stand for the number of vertices and edges in the tree, respectively. The pseudocode for the tree traversal and the recursive prediction step is given below.

2.5. Evaluation and Metrics. The evaluation of the model was carried out using metrics that measure the potential applicability of the method. Since the focus is on the development of a prediction system that draws attention to the most possible outcomes of the process, three performance metrics have been identified for characterizing the sequence containing the events that are found suitable in every step by using equation (3). Therefore, for easier notation, we introduce $\widehat{\Phi}$, a sequence containing the events with only adequate prediction probabilities in every step.

First, S_1 is the percentage of the $\widehat{\Phi}$ sequences that include at least one well-predicted event. For mathematical formulation, Φ is the sequence of events that we aim to predict, while Φ'' is our prediction. N is the number of sequences in the analyzed database, the cardinality of a set is marked with $|^*|$, while the common elements in two sequences are marked as their intersection. Mathematically, S_1 is expressed as follows:

$$S_1 = \frac{\sum_{n=1}^{N}\left(\left|\Phi_n \cap \widehat{\Phi}_n''\right| \geq 1\right)}{N}. \tag{6}$$

Second, $S_{\%}$, a set-based similarity measure that describes the well-predicted events as a percentage of the length of the target sequence has been defined. The events do not have to be in the order of occurrence, and $S_{\%}$ measures how accurately the type of events are predicted,

$$S_{\%} = \frac{\sum_{n=1}^{N}\left|\Phi_n \cap \widehat{\Phi}_n''\right|/\left|\Phi_n\right|}{N}. \tag{7}$$

Finally, S_{ED} was proposed, which is an edit distance-based similarity metric that provides the edit distance between the actual (target) and predicted sequence as a percentage of the length of the more extended sequence among them. The edit distance yields the minimum number of elements that must be inserted or skipped in the compared sequences in order to be identical. The edit distance of two sequences is marked with ED, and equation (8) mathematically describes the S_{ED} edit distance-based similarity metric,

$$S_{\mathrm{ED}} = \frac{\sum_{n=1}^{N}\mathrm{ED}\left(\Phi_n, \widehat{\Phi}_n''\right)}{N}. \tag{8}$$

These performance metrics are calculated for each sequence on the tree, whenever a leaf is found, that is, EOS is predicted, or the maximum sequence length is reached. However, in order to make the resulting sequences even more comparable, their confidence is also calculated. Confidence for each $\widehat{\Phi}$ is defined as a product of the supports of all the containing events in the sequence. The support of the event is the probability the LSTM calculated for that item, given the sequence of the previous events. For the events in the input sequence, the support is determined as a value of 1,

$$\text{confidence} = \prod_{i=1}^{k} P\left(e_i|\widehat{\Phi}_{i-1}\right). \tag{9}$$

3. Implementation and Results

In this section, a summary is provided on the implementation of the proposed method. Then, the used validation techniques are detailed, and the obtained results are evaluated. Since the implemented tool is used for diagnostic purposes, the results should be easily reproducible. Thus, the validation is performed by applying the proposed methods on examples with different complexities. First, the realized system is validated on a simple first-order Markov chain where the method's capability to reproduce the sequence tree is examined. Then, to demonstrate the proposed method's capability to understand higher-order relationships between events, a more complex benchmark dataset is generated using a tree-based system. Finally, the method is tested on a real-life production unit.

3.1. Realization of the seq2probTree Method. The described method was implemented in Python using the Spyder 4 Integrated Development Environment in the Anaconda open-source data science development platform. This platform was ideal for the task as most of the necessary libraries are included by default, thus minimizing the setup process for development. The LSTM RNN was implemented using Keras, a deep-learning application programming interface running on top of the TensorFlow end-to-end open-source machine learning platform. Keras API is well-known for its full-fledged documentation and high-quality example codes, which are usually very well commented for easy

Require: $\text{model}_{\text{LSTM}}$, $\text{eventseq}_{\text{input}}$
Create root node for $\text{Tree}_{\text{Event}}$
Append events in $\text{eventseq}_{\text{input}}$ to $\text{Tree}_{\text{Event}}$
$\text{input}_{\text{LSTM}} = \text{inputConversion}\ (\text{eventseq}_{\text{input}})$ \(⊳\) Conversion of input to match Encoder Embedding layer format
$\text{states}_{\text{LSTM}} = \text{encoder}_{\text{LSTM}}.\text{prediction}\,(\text{input}_{\text{LSTM}})$
$\text{RecursiveDecoding}(\text{input}_{\text{LSTM}}, \text{states}_{\text{LSTM}}, \text{Tree}_{\text{Event}})$

Algorithm 1: Preprocessing before prediction.

Require: $\text{input}_{\text{LSTM}}$, $\text{states}_{\text{LSTM}}$, $\text{Tree}_{\text{Event}}$
$\text{output}_{\text{LSTM}}, \text{states}_{\text{LSTM}} = \text{decoder}_{\text{LSTM}}.\text{prediction}\,(\text{input}_{\text{LSTM}}, \text{states}_{\text{LSTM}})$
$i = 0$
While There is $e'' | P(e'') > P_{\text{thr}}$ in $\text{output}_{\text{LSTM}}$ **AND** $i < N_{\text{Thr}}$ **do**
Add e'' to $\text{Tree}_{\text{event}}$
Delete e'' from $\text{output}_{\text{LSTM}}$
if not(e'' is EOS OR $\text{sequencelength}_{\text{MAX}}$ reached) **then**
$\text{RecursiveDecoding}(\text{input}_{\text{LSTM}} + e'', \text{states}_{\text{LSTM}}, \text{Tree}_{\text{Event}})$
$i = i + 1$

Algorithm 2: RecursiveDecoding function.

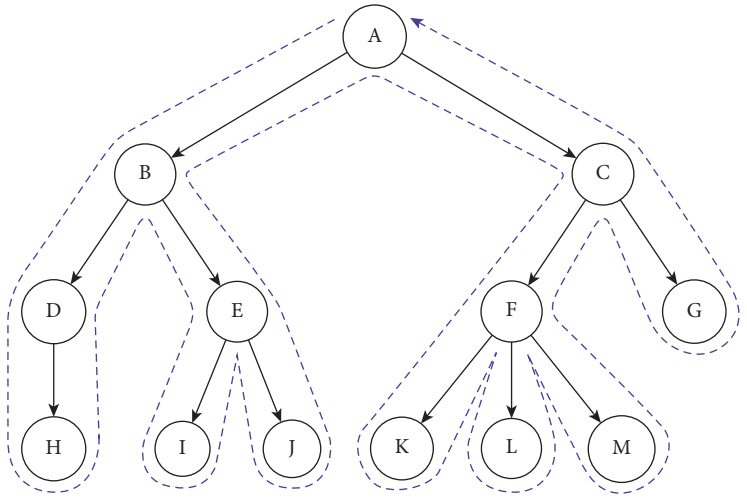

Figure 3: Depth-first traversal of a tree structure.

adaptation. In order to decrease the runtime of the training process of LSTM, the NVIDIA CUDA®Deep Neural Network (cuDNN) library was utilized. Since Keras is built on top of Tensorflow, which happened to be a cuDNN accelerated framework after the initial setup, the time required by the LSTM training was reduced tenfold. This speed increase was provided by an NVIDIA Geforce GTX 1080 Ti graphics processing unit.

The probability trees presented in this paper were generated using the ETE toolkit for Python, which provides a wide range of tree-handling options and node annotation features alongside a tree visualization system to output the resultant trees. The code of Markov chain models was created in MathWorks MATLAB environment for the ease of exporting the simulated data into *.xlsx* format and importing it into Python using the pandas library. However, due to the vast size of the training dataset for the third-order Markov model, MATLAB's *.m* format had to be utilized, which can be handled by SciPy (conveniently included in Anaconda).

The finished implementation consists of two routines. The first contains the selection of the desired dataset, the setup of the LSTM, the training procedure, and the creation of the training history plots. After the training process has been completed, the encoder and decoder models are saved, thus eliminating the necessity of running the model training with each subsequent session of the application of the tool. The second routine consists of loading the LSTM models, the recursive decoding, and all the functions necessary for the metric calculation and the tree generation and output.

3.2. Validation on First-Order Markov Model. In this section, a brief summary will be given on how the proposed method has been implemented. For the ease of validation, a simple Markov chain is used. The model consists of 12 states that follow each other in a row as a rule of thumb. The only 2 exceptions are state 4 and 7, which break this rule. While transitioning from state 4, there is a probability of 0.35 that the system will "reset," thus returning to state 1. If the system reaches state 7, there is a 30% chance that the system skips the following 2 states and goes right to 10. This behavior can be observed in Figure 4.

The dataset was established by creating 10000 sequences utilizing the described Markov chain. Each sequence starts from a randomly selected system state, and the length is also randomly determined between 9 and 12. After the generation of the dataset, the LSTM model was trained by using the following parameters:

(i) Embedding dimensions = 6
(ii) Latent dimensions = 15
(iii) Batch size = 256
(iv) Epochs = 70

The training's accuracy and loss can be observed in Figure 5. In order to validate the model's performance, a cross-check was made by feeding each state as an input to the encoder, thus initializing the internal LSTM states. It is important to note here that to initialize the encoder for the validation, not only the state from which the prediction starts needs to be used as the input but also the previous two states; as for the model training, each sequence in the database was separated after the third state as input and output. Then, one prediction step is completed, and the output of the LSTM is recorded. This is repeated for each state, creating the validation transition matrix, which is then compared to the transition matrix of the first-order Markov chain (part (a) in Figure 4). In Figure 6, each predicted value is illustrated in function of the original transition probability. The calculated coefficient of determination for this simple example is as high as 0.9994.

After the training was completed, the seq2probTree method was utilized with $P_{\text{thr}} = 0.2$ and by giving the input sequence of $[1, 2, 3]$ to the taught LSTM model. The maximum output sequence length was set to 12.

Figure 7 gives visual aid about the metrics placed at each node on the probability tree, while the acquired results can be observed in Figure 8. Each node on the tree has at least three properties: name, support, and confidence (top and bottom values, respectively). The EOS nodes also have the three performance metrics calculated for the given sequence: S_1, $S_{\%}$, and S_{ED}, values of which can be found in the right column in the specified order from top to bottom. For example, it can be observed from Figure 7 that the seq2probTree method predicted state 11 after the subsequence ending with state 10 with a probability of 0.49. In addition, the calculated probability of ending the sequence after state 11 is 0.5. We can also see that the confidence of Φ_k—thus, the whole sequence ending with EOS—is 0.04. The S_1 value also shows the highlighted Φ_k sequence that every entry (1.0) in the input database starting with the given $\Phi_{k'}$ subsequence—in this case [1 2 3]—has at least one state that has been predicted in $\Phi_{k''}$ by the method. $S_{\%}$ being 0.68 gives us the idea that the states predicted in $\Phi_{k''}$ occur in 68% of the database entries starting with [1 2 3]. The last metric of this EOS node on the probability tree—S_{ED}—shows that the average edit distance—thus the number of changes that need to be made to match the sequence—is 4.49, given the aforementioned $\Phi_{k'}$.

The properties of the first-order Markov chain are observable in the results. Both of the distinguished transitions are identifiable, and the predicted transition probabilities are within a margin of error of the Markov chains. The tree also reflects all the different length variants of each possible sequences.

3.3. Validation on Higher-Order Tree-Based System. As the LSTM-based deep-learning networks are explicitly developed to capture the long-term relationship in datasets, a higher-order system is used for further evaluation. The behavior of the system is based on a probability tree, which was pseudorandomly generated. Each node on the tree may have up to three children, with the system stating that it represents and the probability that state occurs also generated randomly. The sum of the probabilities of states originating from the same node is normalized to 1. The depth of the tree was determined randomly between 8 and 9—without considering the root (StOS) and leaf (EOS) nodes. The number of applied states is set to 4 to facilitate easier understanding and reconstruction of the results. However, at this complexity, it is already a difficult task. The states are represented by letters A, B, C, and D. The complexity of the system can be observed in Figure 9, while the inspected transition probabilities—thus the highlighted areas—are visible more transparently in Figures 10–12.

To utilize the seq2probTree method, a training dataset was created consisting of 10,000 simulations of the system starting from the root node and randomly determining the path—based on the transition probabilities—until a leaf node is reached. After the given amount of simulations were concluded, the resultant dataset was copied six times, as during the training, the sequences are split to input and target and this position, where the sequences are separated. as input and target is randomly selected. The reason for the six times multiplication is that the position of the cut is varied between the 1 st and the 6 th state in the sequence —separating the input and target after the selected state. The generated dataset was used for the training of the LSTM model by using the following parameters:

(i) Embedding dimensions = 2
(ii) Latent dimensions = 15
(iii) Batch size = 64
(iv) Epochs = 25

During the training on the dataset produced by the simulation of the proposed tree-based system, the accuracy and loss functions were also recorded. They can be

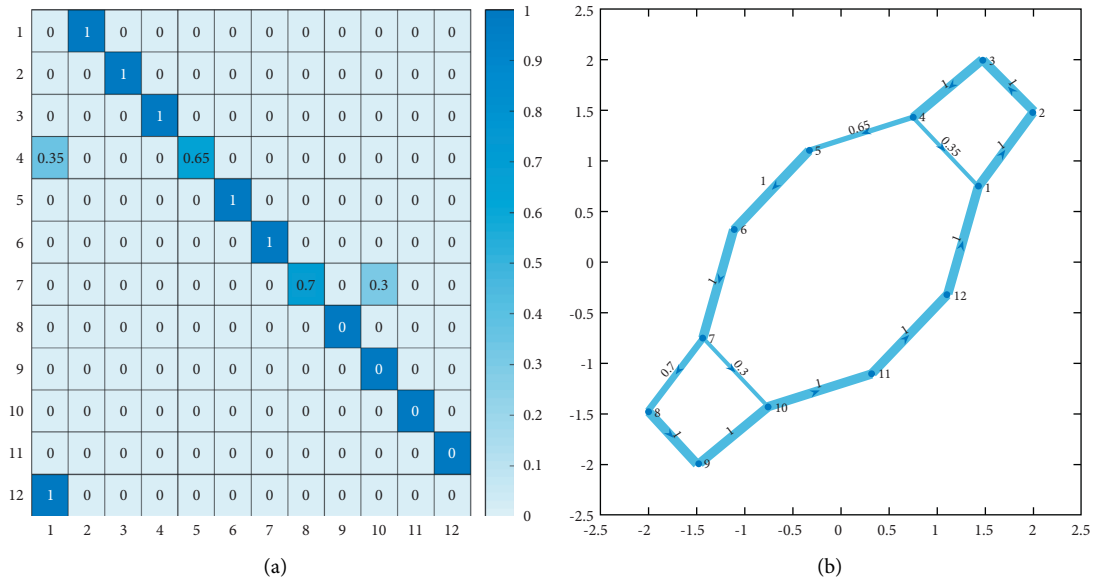

Figure 4: The transitional probabilities (a) and the directional graph (b) of the first-order Markov chain.

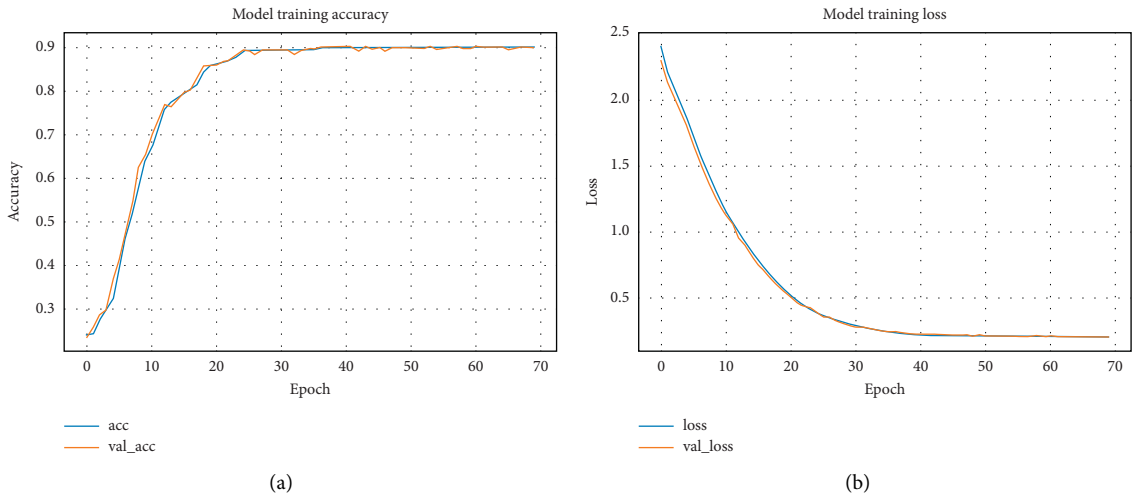

Figure 5: Training statistics on the first-order Markov chain.

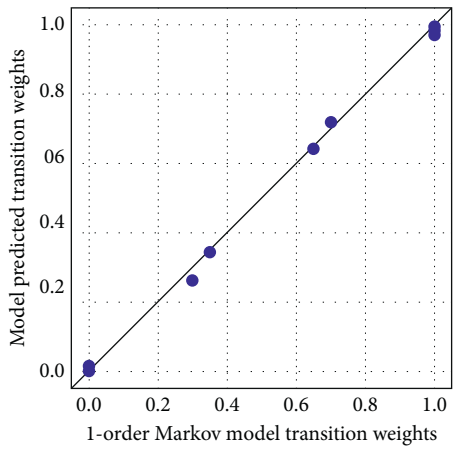

Figure 6: Crossvalidation of the transition probabilities of the first-order Markov chain.

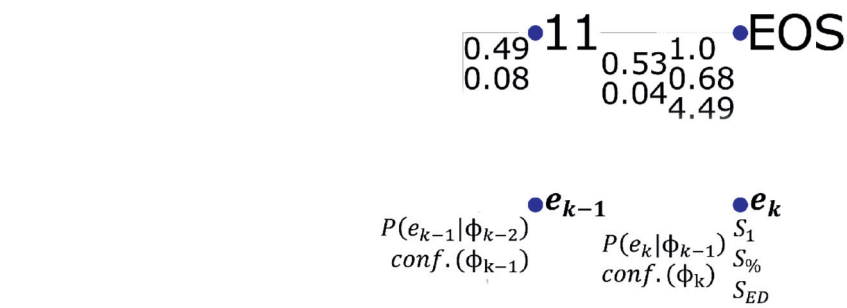

Figure 7: Explanation of the metrics located at the nodes of the probability tree.

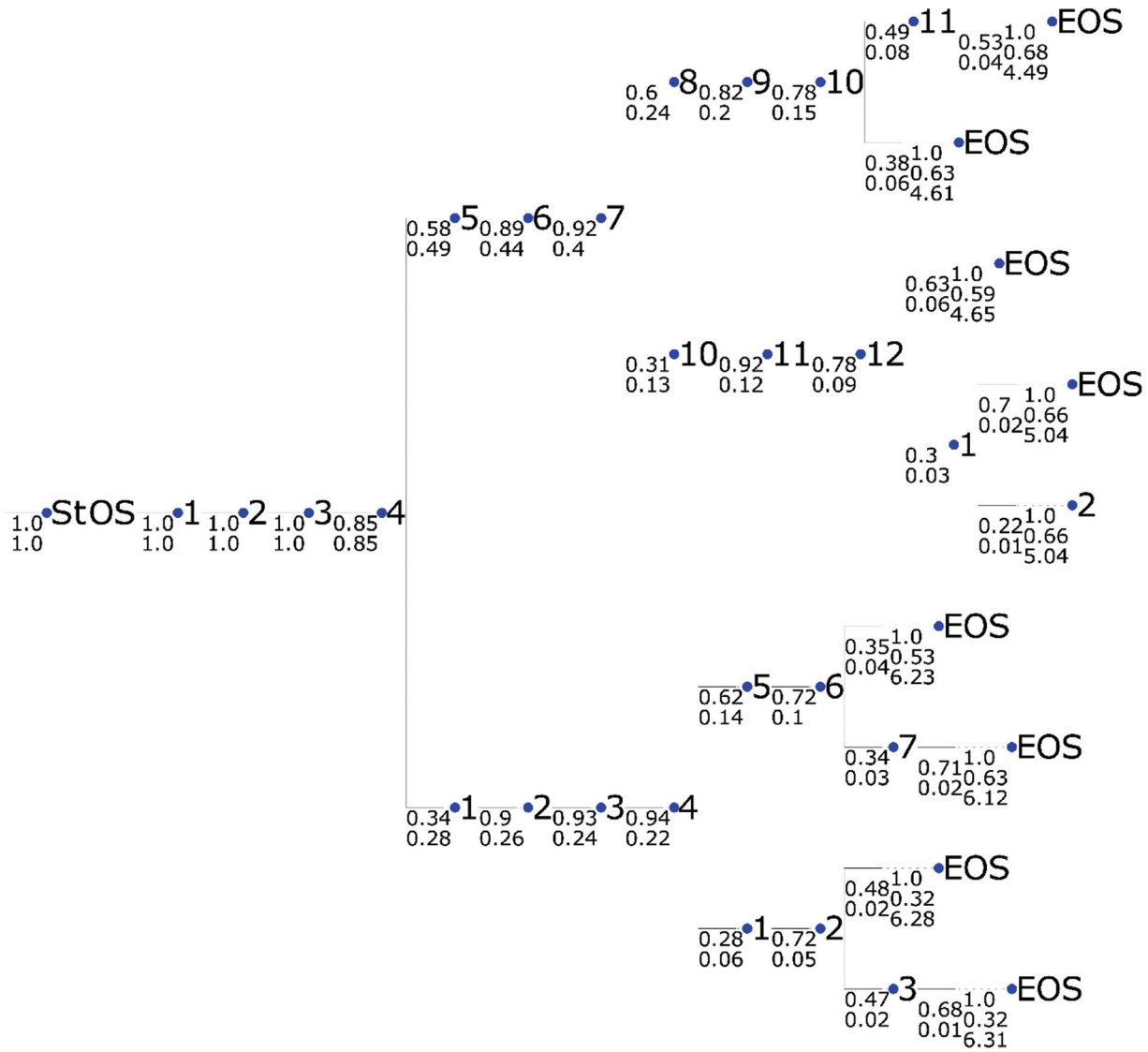

Figure 8: Output of the seq2probTree learning method for the first-order Markov model.

observed form Figure 13. It is important to note that during the training, 20% of the dataset was used as validation data, while dropout was not utilized in the LSTM layer.

After the training of the LSTM model on the aforementioned dataset, the seq2probTree method was applied with input sequences leading to the highlighted areas in Figures 14–16—[A D], [D D], and [B] respectively.

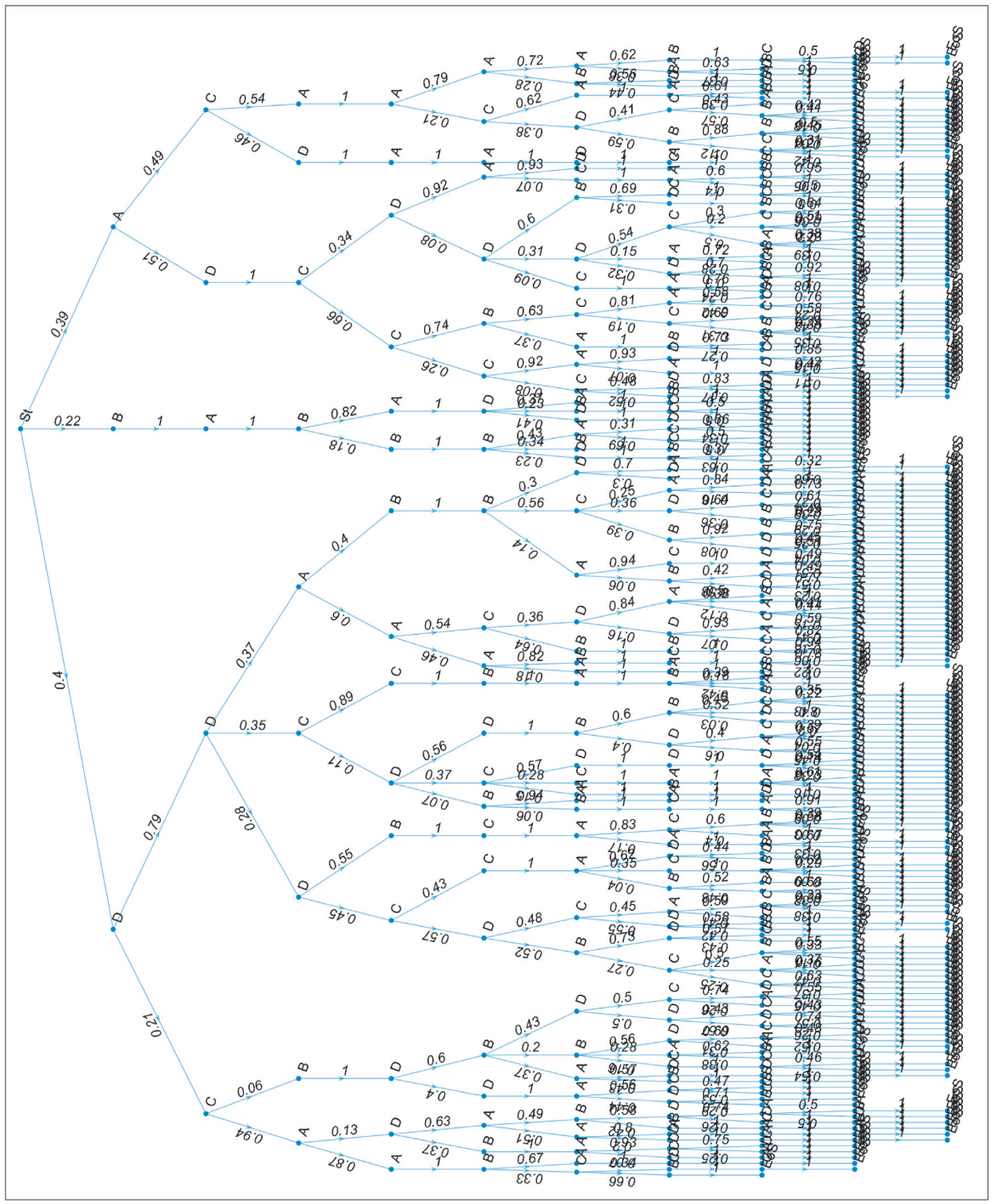

Figure 9: The state transition probabilities of the full tree-based system. The complexity of the system is easily observable.

In order to generate the smallest possible trees, consisting of only the states with the highest probability, the P_{thr} was set to a higher value of 0.25. In addition, TopN_{thr} parameter was introduced as the beam strength with a value of 2, which represents that only the two states with the highest probabilities are taken into consideration during the construction of the tree. With the maximal sequence length set to 9, these measures made sure that the size of the resultant tree is adequate and appropriate for evaluation.

By comparing the acquired probability trees to the tree that is defining the system's behaviour, it is observable that the seq2probTree method based on the LSTM model can capture the long-term relationship of the states of a system. Given the training accuracy as 0.86, the acquired results represent the original probability tree on which the system is based quite accurately. A few prediction errors are observable in the results. These discrepancies could be explained by pointing out that the input sequence is

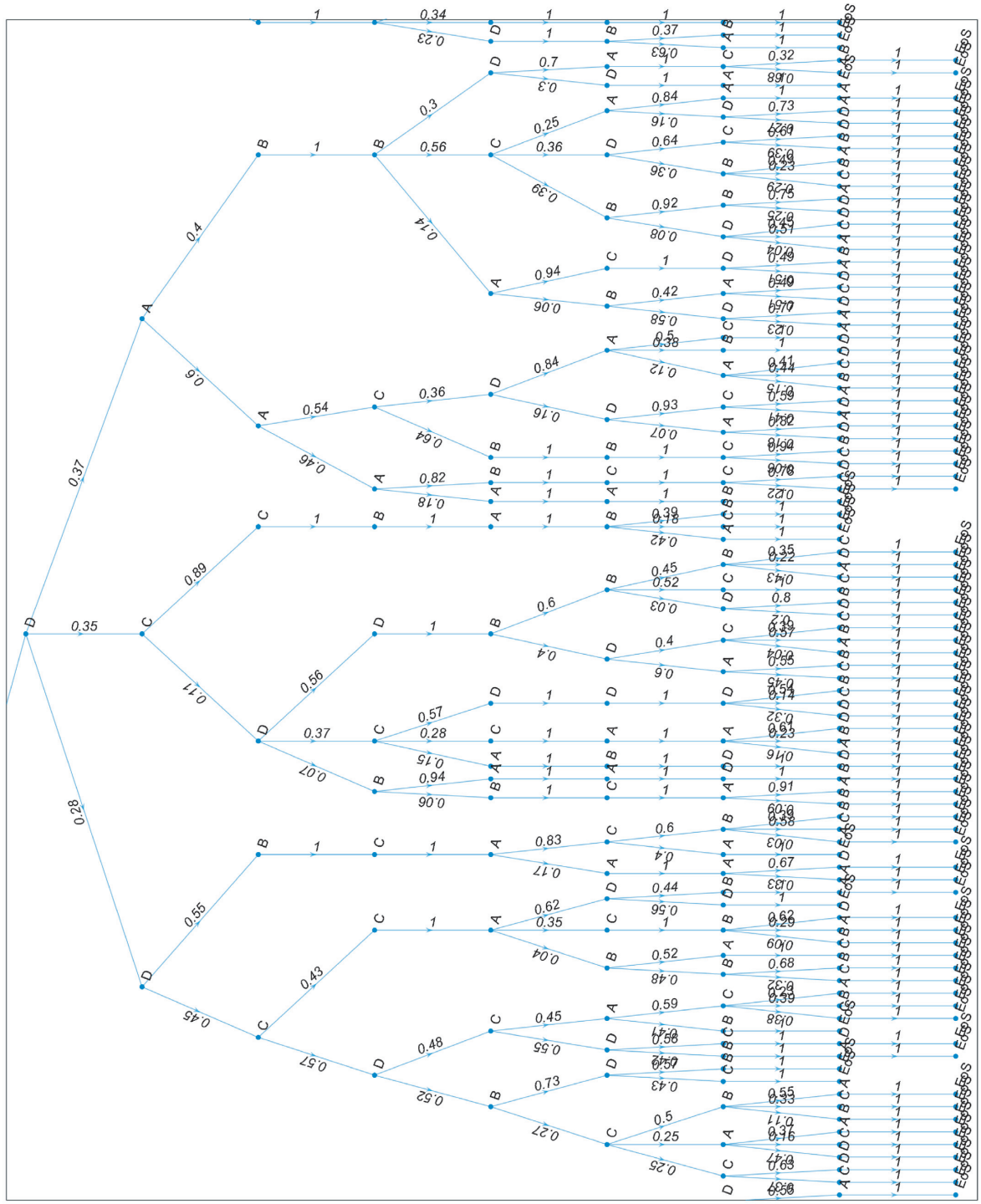

Figure 10: The state transition probabilities of the full tree-based system, assuming [D D] event history.

relatively scarce; thus, shorter patterns with high confidences can "mislead" the model.

As the seq2probTree method is proposed as a tool capable of online dynamic process supervision, the visual information it provides is crucial. While understanding the prediction tree with sparse input is an overwhelming task, as the input sequence expands with more system states, the less complex the probability tree structure becomes. Figures 16–21 represent the method's visual output, while step-by-step appending the input sequence starting from [B] to [B A B A D D] following the most probable path shown in Figure 16 (also the path of the sequence with the lowest S_{ED} metric). The results clearly show how the complexity of the acquired probability trees decreases by expanding the input sequences. The inferred state sequences are diversified by providing scarce input for the LSTM model, and a few erroneous conclusions are drawn. An excellent example for

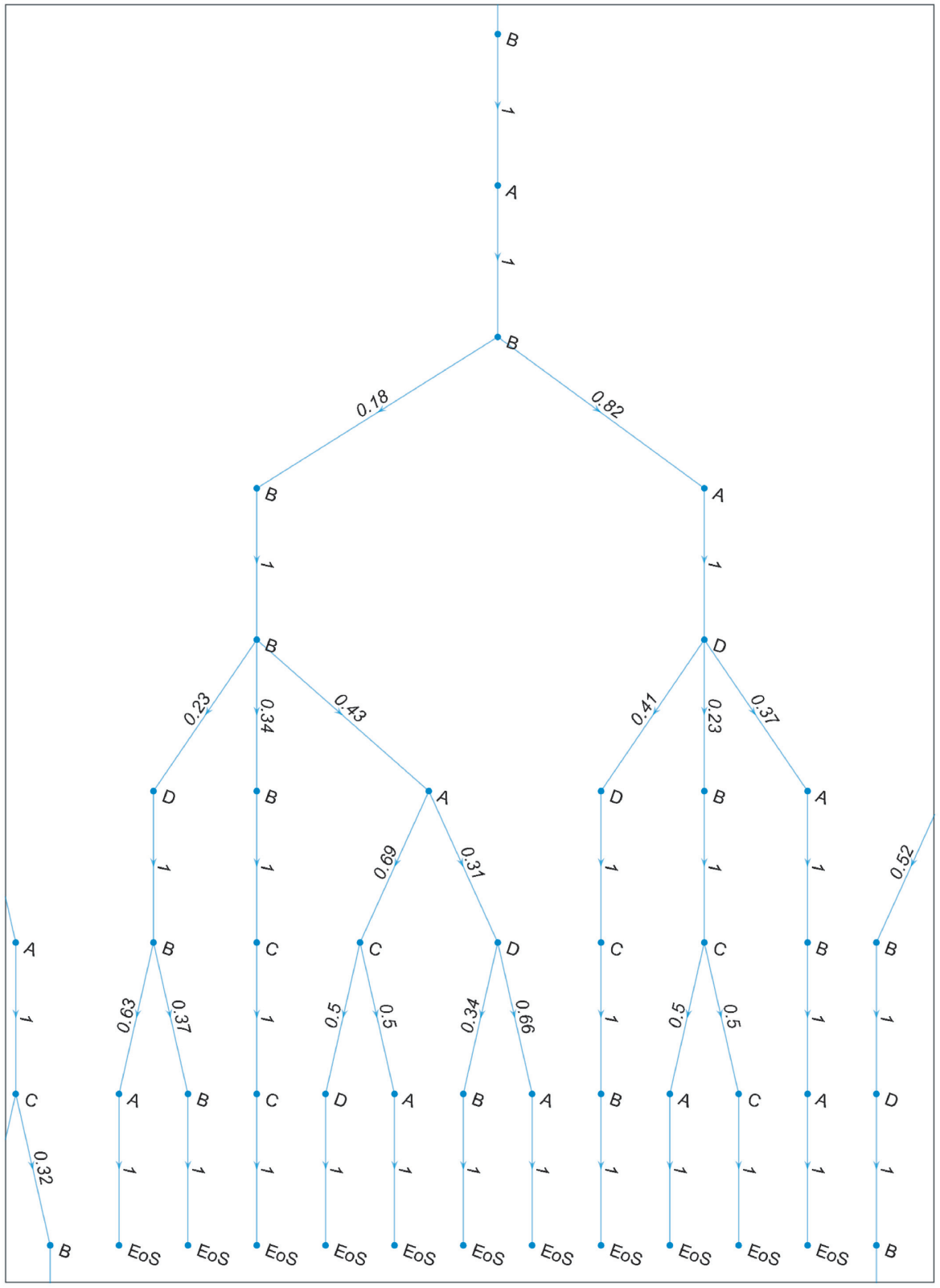

Figure 11: The state transition probabilities of the full tree-based system, assuming [B] event history.

this behavior is found in Figure 17, where after the [B A] input sequence, state A was predicted with a probability that fit P_{thr} along with state B, which should have been a sure transition.

To quantify the accuracy of the model for each aforementioned input sequence, an average error has been calculated and may be observable on Table 1. The error—just

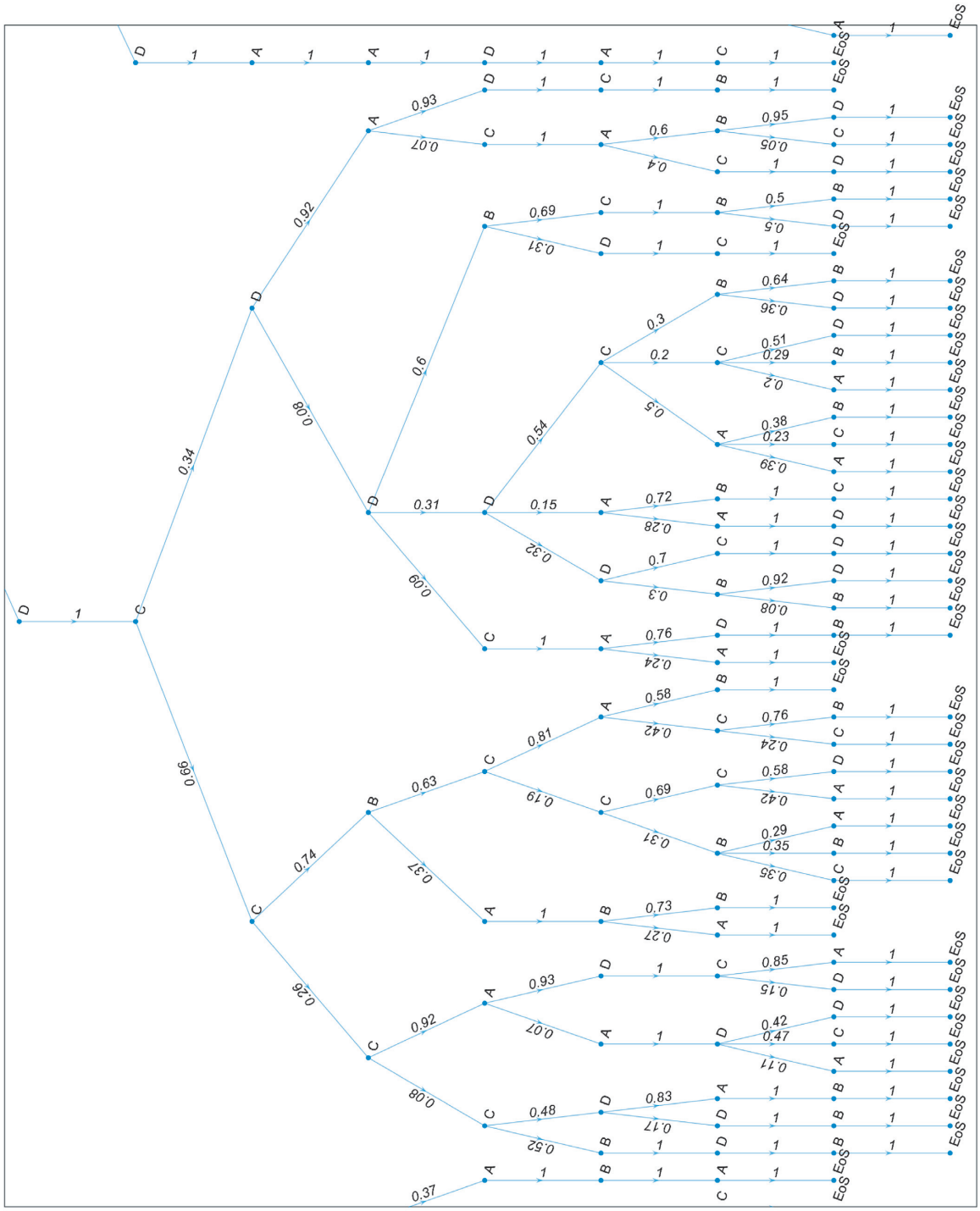

Figure 12: The state transition probabilities of the full tree-based system, assuming [A D] event history.

like the S metrics—is calculated for each predicted event sequence on the probability tree by simply counting how many elements from the end of the sequence are not found in the probability tree defining the system. The determined errors are then averaged out.

One question that arises during the utilization of the seq2probTree method is regarding the necessary length of the input sequence, after which the output is considered reasonably accurate. Table 1 gives us the idea that if the input sequence is at least four-element long, then the probability trees generated by the seq2probTree method will show no discrepancies when compared to the tree on which the behavior of the system is based. Thus, the probability trees for every possible 4-length input sequence were generated, and the aAverage errors were calculated. Since the same accuracy cannot be expected for all the input sequences—for

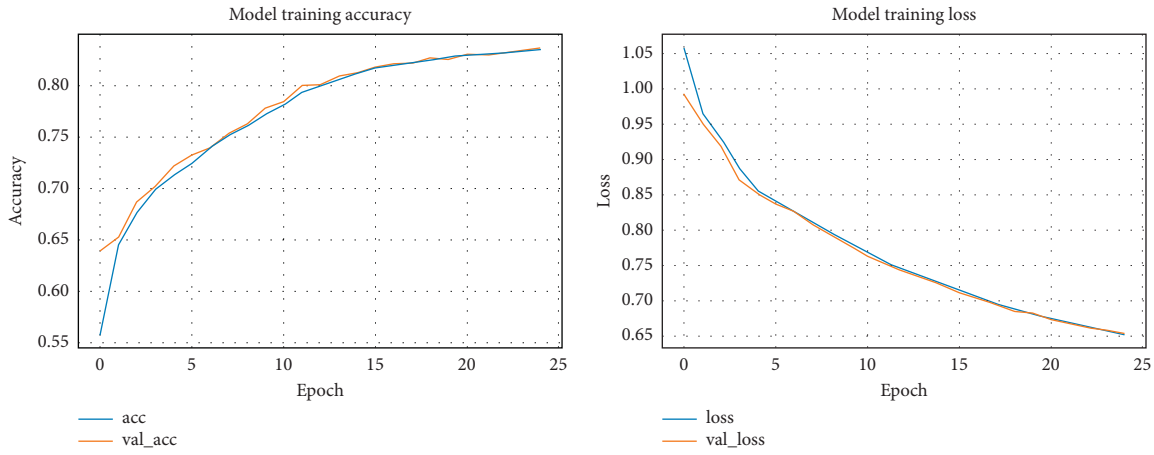

Figure 13: Training statistics on the tree-based system.

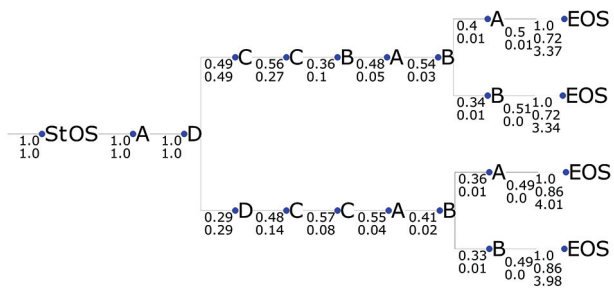

Figure 14: Output of the seq2probTree method for the tree-based system with input sequence [A D].

the sake of comparison—an additional weighting was applied to the calculated average error. The weight is calculated by determining the confidence for each input sequence and normalizing them based on the highest value. The resultant weighted average error values can be observed from Table 2. Based on the obtained results, it can be stated that after the 4 th input element, the prediction is quite precise for this example system. More significant discrepancies were observed in low confidence sequences, where even after the input sequence (several) diversions are possible.

By utilizing the seq2probTree method on this tree-based system, the capability of the algorithm for predicting higher-order event relationships has been verified with success. The average error value has been introduced to help the evaluation of the results when a direct comparison is possible with an original probability tree.

3.4. Case Study: Alarm Scenarios of a Hydrofluoric Acid Alkylation Production Unit. The proposed method has been applied to an alarm log of a hydrofluoric acid alkylation production unit to check the real-life performance. The process flow diagram of the technology can be observed in Figure 22.

The log used for this experiment was created by the operation of the production unit over a four-month-long (121 days) period, where all the incoming alarm and other events have been recorded. The unprocessed log contains precisely 200, 802 entries of which 30, 168 messages are unsuppressed alarm events. 8, 721 of these are alarms that were considered significant, thus were not shelved by the operators. The event sequences for the input of the tool were created by grouping them based on a time window while preserving their sequential temporal property. Thus, whenever a 600 sec gap is found after the last event, the two events are not considered related, and a new sequence is started. By using this strategy, the significant alarms were separated into 3, 330 sequences. Then, by considering only the event sequences with a minimum length of two, the number of valuable sequences got further reduced to 762. It is also important to note that this event database has a very high unique state count compared to the previous examples—the sequences are composed of 354 individual states. Due to confidentiality reasons, the name (the meaning) of the alarm tags has been removed.

Then, this sequence database was analyzed for frequent events that start sequences. To carry out the analysis for this case study, the four most frequent events were selected to be utilized using the seq2probTree method. The name of the selected events and their number of occurrence as the first in a sequence are highlighted in Table 3.

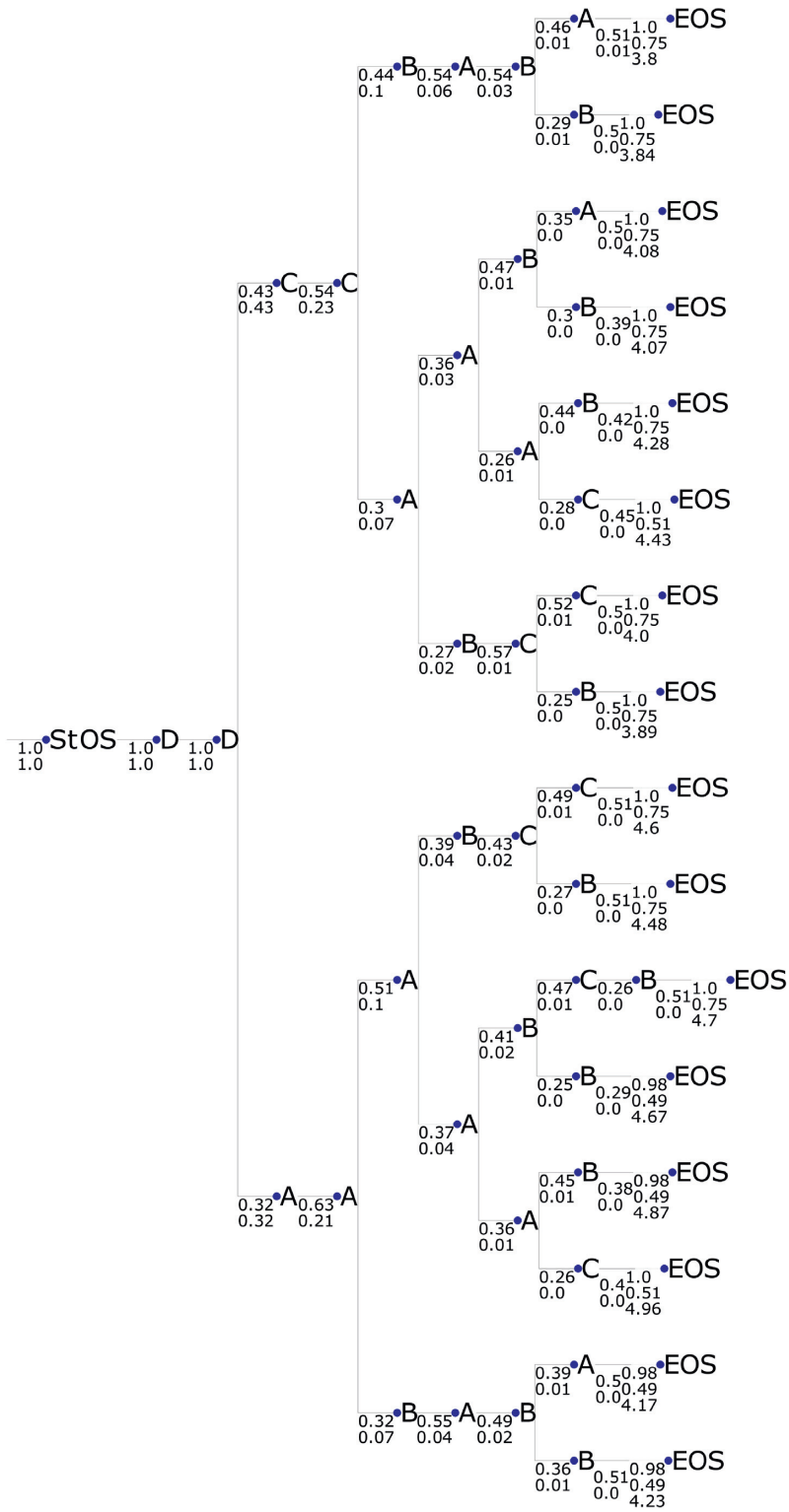

Figure 15: Output of the seq2probTree method for the tree-based system with input sequence [D D].

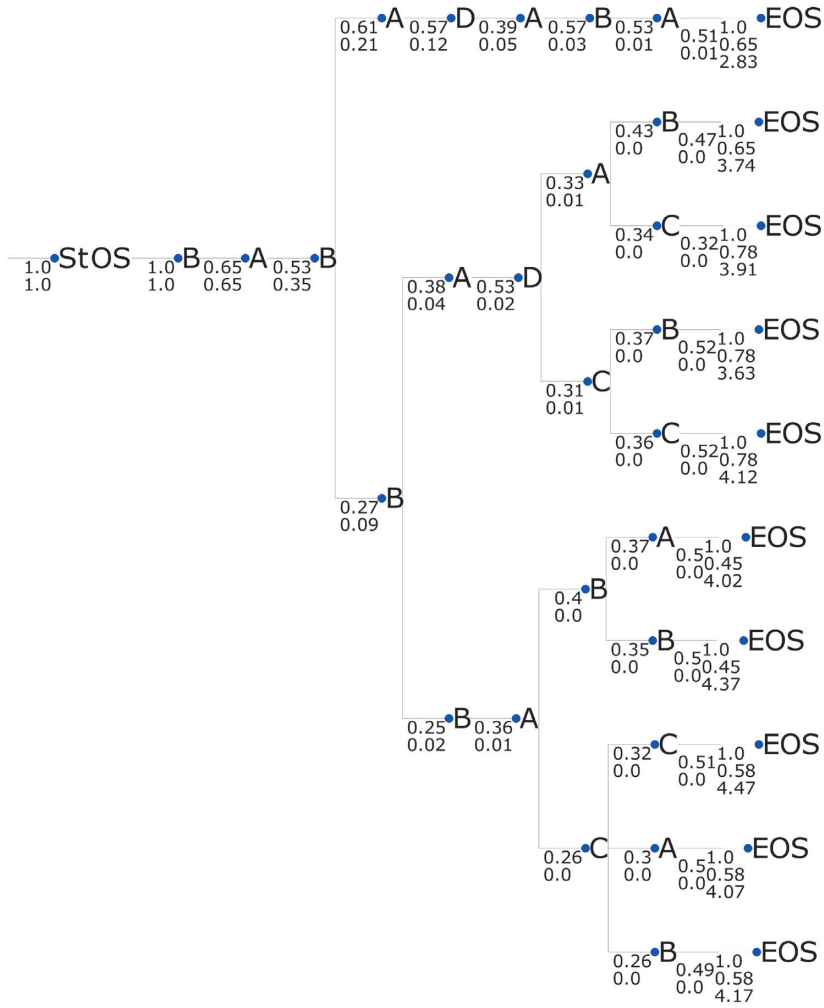

Figure 16: Output of the seq2probTree method for the tree-based system with input sequence [B].

After processing the event log, the seq2probTree method has been applied to the database. The training results from Figure 23 have been acquired by using the following LSTM and training parameters:

(i) Embedding dimensions = 5

(ii) Latent dimensions = 25

(iii) Batch size = 32

(iv) Epochs = 500

By using the aforementioned events as inputs for the seq2probTree method, the probability trees in Figures 24–27 have been created. The parameters of the beam search algorithm—P_{thr} and TopN_{thr}—were set as 0.065 and 3, respectively. Analyzing the trees, it is clear that the seq2probTree method is capable of learning and identifying the possible event scenarios. However, since the dataset is vastly diverse—especially since the seq2probTree method is also sequential position-sensitive—the probabilities of the individual transitions are pretty low; thus, the shallow P_{thr} value is justified. Moving lower with the probability threshold would have resulted in immense trees; thus, only the most frequent transitions are displayed in the figures. In Figure 27, one drawback of the method is also observed: in the longer sequences, which contain or start with [136711], often a recurring [361835] is present. This transition is so prominent that the LSTM model keeps on predicting it with a high probability. In these cases, only the defined maximal output sequence length parameter kept the seq2probTree method from creating an ever-growing branch on the tree.

Figure 24 illustrates well the different alarm sequences related to the depropanizer. The tree is initialized with the alarm message of the depropanizer pressure [136769], which can be followed by either the level alarm of one of the vessels of the depropanizer [137161] or an alarm of a pump [136711]. After the alarm on the depropanizer vessel, the alarm of the depropanizer pressure [136769] or the depropanizer feed can come in [353848].

The alarm sequences in Figure 25 are related to another scenario of the depropanizer. As can be seen, the alarm

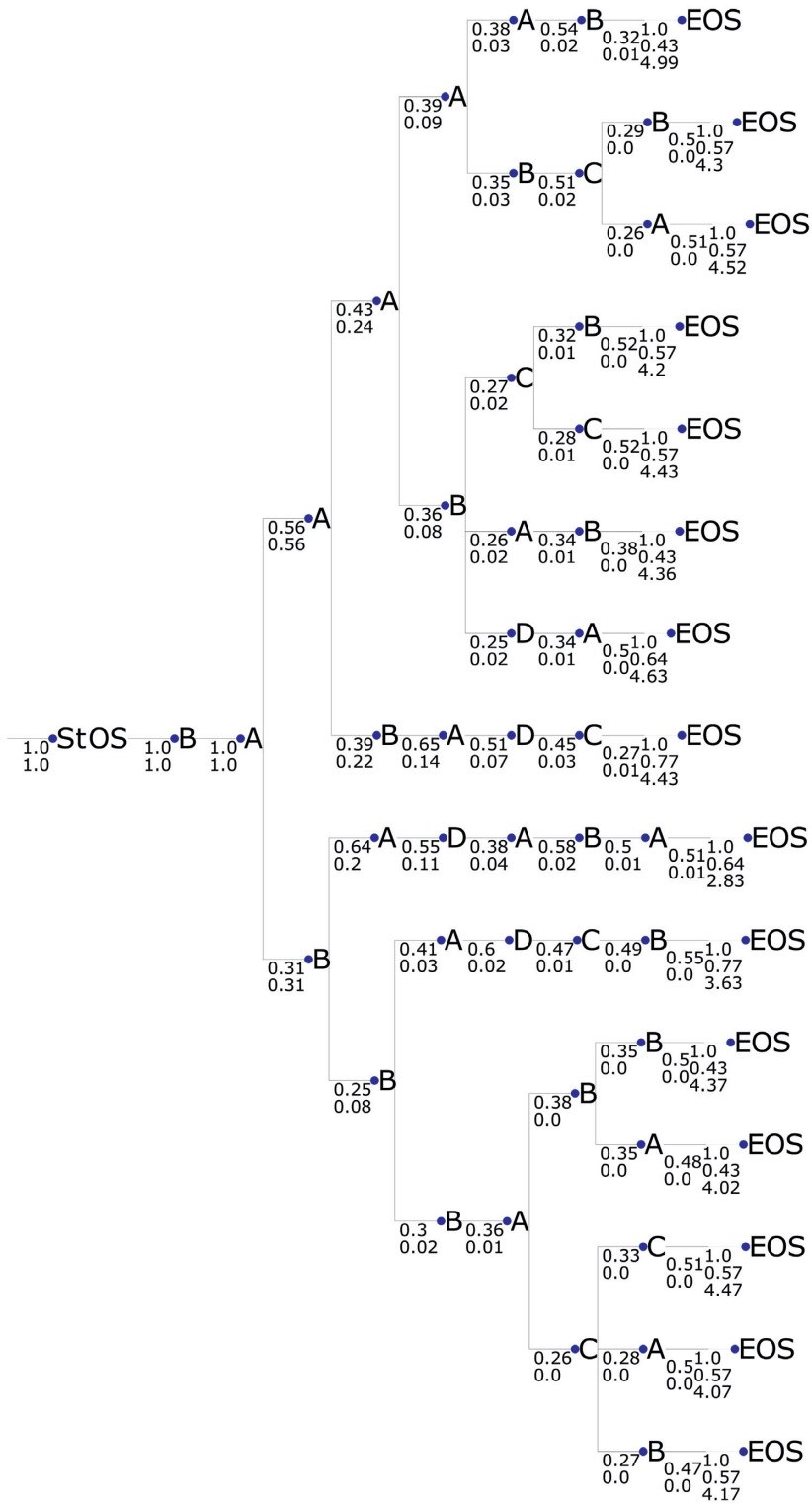

Figure 17: Output of the seq2probTree method for the tree-based system with input sequence [B A].

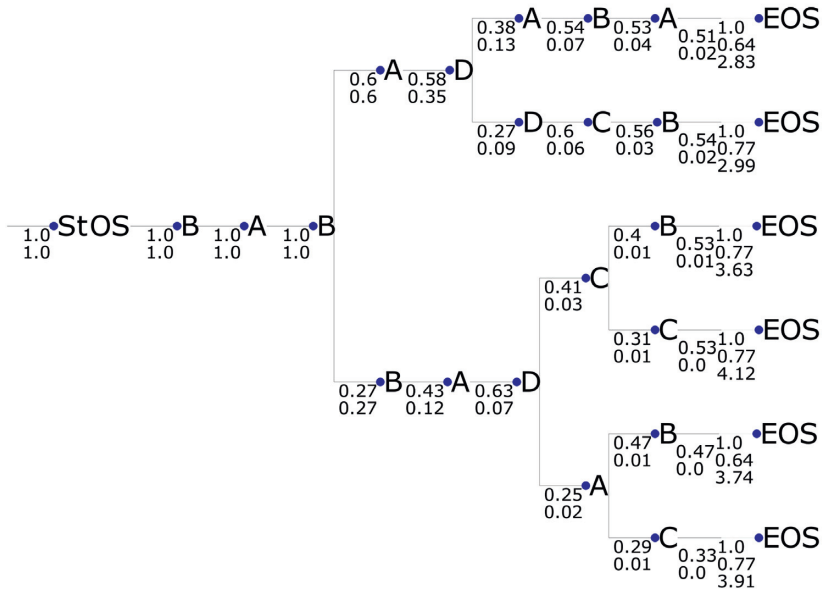

Figure 18: Output of the seq2probTree method for the tree-based system with input sequence [B A B].

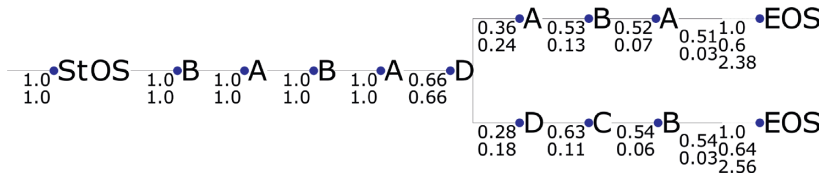

Figure 19: Output of the seq2probTree method for the tree-based system with input sequence [B A B A].

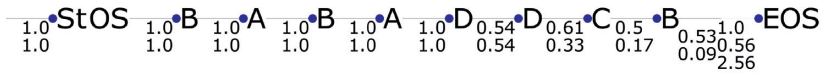

Figure 20: Output of the seq2probTree method for the tree-based system with input sequence [B A B A D].

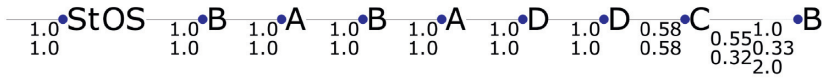

Figure 21: Output of the seq2probTree method for the tree-based system with input sequence [B A B A D D].

Table 1: The calculated average error values for different input sequences.

Input sequence	Average error
A D	3.5
D	2.5625
B	2.1
B A	3.4667
B A B	2.6667
B A B A	0
B A B A D	0
B A B A D D	0

Table 2: Prediction error statistics with every possible Φ_4' input.

SEQ	Conf.	Weight	Average error	Weighted A. E.
B A B A	0.1804	1	0	0
D C A A	0.0687	0.3808	2.8889	1.1001
D C A D	0.0013	0.0074	2.25	0.0166
D C B D	0.005	0.0279	1.8	0.0503
D D D C	0.0398	0.2207	4	0.8828
D D D B	0.0487	0.2698	0	0
D D C D	0.0122	0.0674	3.3333	0.2248
D D C C	0.0984	0.5456	2.5714	1.4031
D D A A	0.0702	0.3889	2.2222	0.8642
D D A B	0.0468	0.2592	4	1.0370
B A B B	0.0396	0.2195	1.8571	0.4077
A D C C	0.1313	0.7277	0	0
A D C D	0.0676	0.3749	0	0
A C D A	0.0879	0.4873	2.25	1.0964
A C A A	0.1032	0.572	2.4444	1.3983

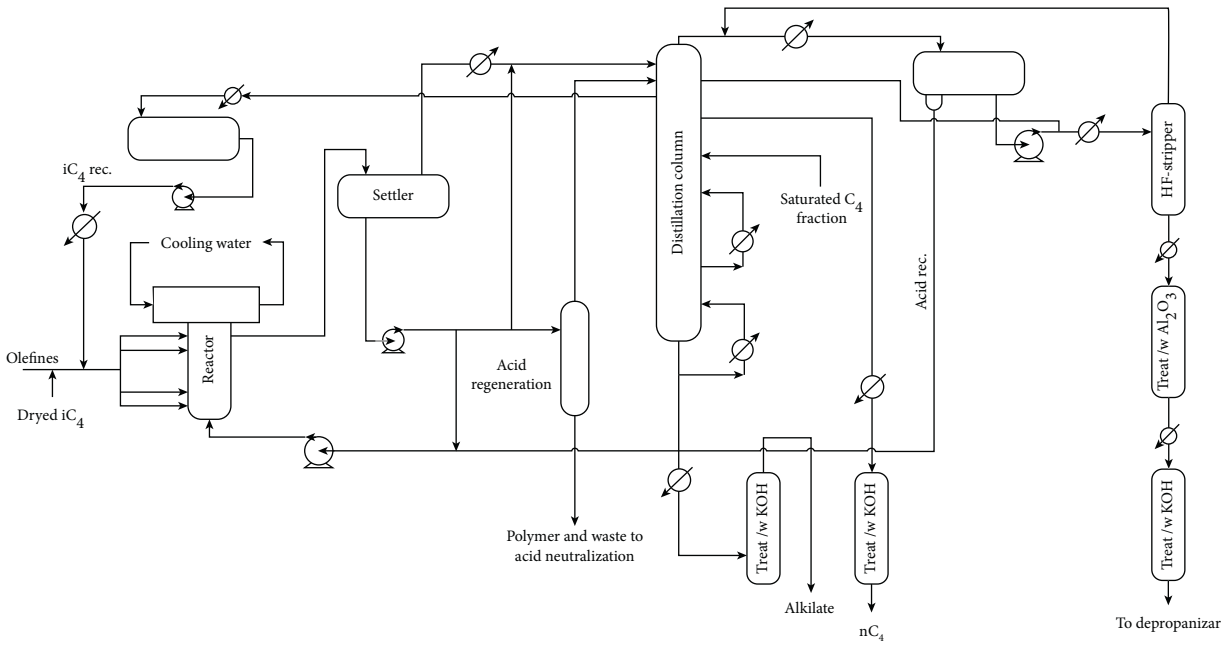

Figure 22: Process flow diagram of the hydrofluoric acid (HF) alkylation production unit.

Table 3: Occurrence statistics of the frequent sequence starting events in D_T.

Event ID	No. of occurrences
136711	127
136769	32
137438	31
137272	31

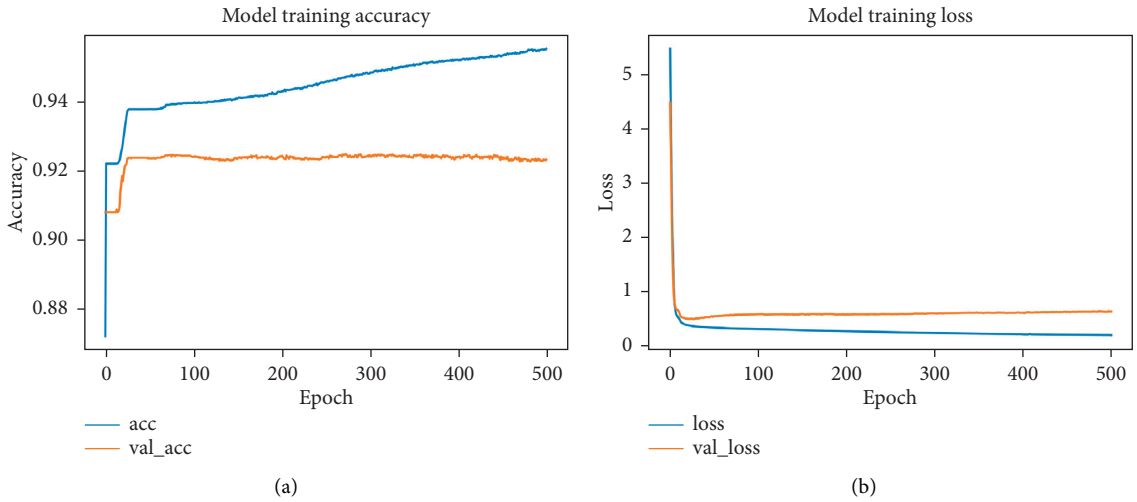

Figure 23: Training statistics on the log of the HF acid production unit.

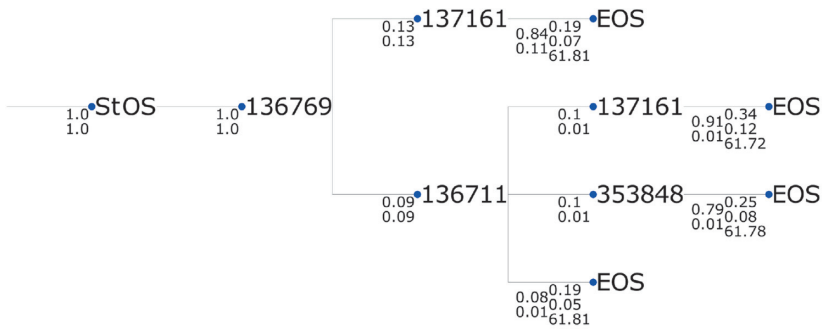

Figure 24: Output of the seq2probTree method for the HF production unit with input [136769].

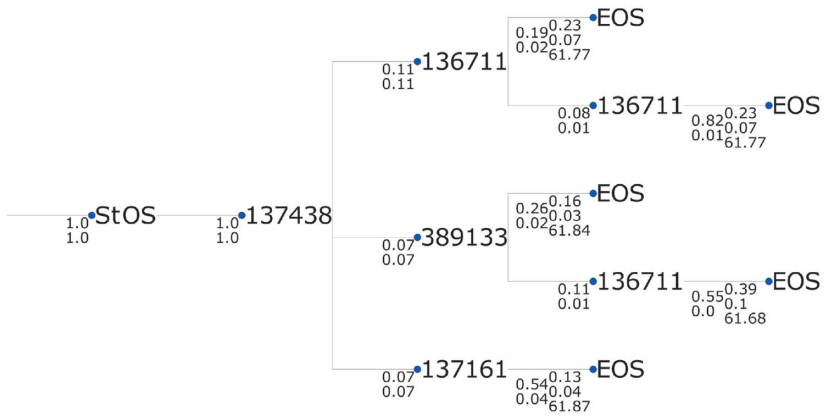

Figure 25: Output of the seq2probTree method for the HF production unit with input [137438].

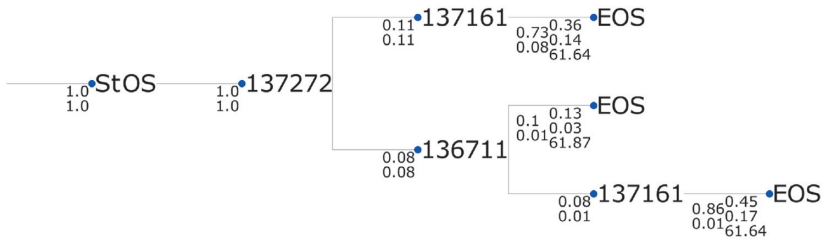

Figure 26: Output of the seq2probTree method for the HF production unit with input [137272].

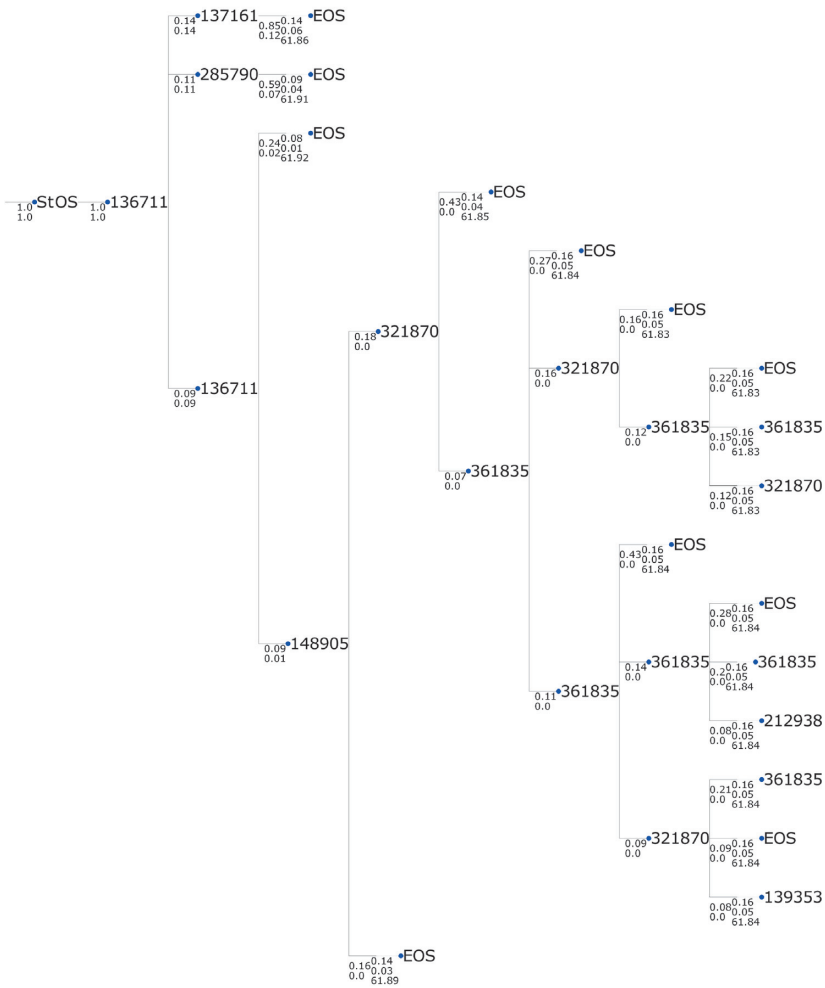

Figure 27: Output of the seq2probTree method for the HF production unit with input [136711].

message of the temperature alarm of the vessel [137438] can be followed by the pump alarm [136711] again, or the level alarm of the same vessel [137161], or another, rare alarm on a circulation pipeline.

Figure 26 is an excellent example that a problem in the bottom part of the stripper [137272] can generate a cascade of alarms on the connected units (pump and level of the vessel).

Similarly, very long alarm cascades of varying probabilities are generated in Figure 27. As we saw, the alarm of the pump [136711] reoccurs in many sequences, and, not surprisingly, it can induce the presence of several other alarms with different scenarios for the order of their occurrences.

4. Conclusion

By proposing the seq2probTree method, the application of the seq2seq learning algorithm is expanded by not only

considering the most probable item but also further exploring the alternative courses of an event sequence using the beam search algorithm during inference. This approach has been realized in Python environment by using state-of-the-art development tools.

The capability of the method has been demonstrated to reproduce the characteristics of a given system by applying it to a first-order Markov chain model. The provided transition probabilities were reasonably identified, but the approach was also capable of revealing the given unique attributes and quirks of the examined systems. The assumption that the seq2probTree method is capable of exploring higher-order relationships between events has been demonstrated and validated using a tree-based system as an example. In addition, the average error metric has been proposed to aid the user in determining the length of the input necessary for reliable prediction. Finally, the applicability of the proposed method was examined on a real-life practical example, where it produced valuable results even in the case of a highly diversified system. The proposed approach was able to map the typical alarm event scenarios and represent those in a visually interpretable manner in a hydrofluoric acid alkylation process.

Based on this evidence, it can be stated that the sequence trees created by the seq2probTree method properly represent the network of the possible alternate sequence of events. With this approach, the necessary visual output can be obtained for understanding and diagnostics of higher-order, complex systems.

Data Availability

The benchmark datasets and the code of the developed algorithms will be available on the GitHub profile and the website of the authors (https://www.abonyilab.com/) after the publication of the results.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the TKP2020-IKA-07 project financed under the 2020-4.1.1-TKP2020 Thematic Excellence Programme by the National Research, Development and Innovation Fund of Hungary. Gyula Dörgő was supported by the doctoral student scholarship program of the Co-operative Doctoral Program of the Ministry of Innovation and Technology financed from the National Research, Development, and Innovation Fund. The authors gratefully acknowledge the professional support of Ferenc Tandari who provided invaluable comments on the case study.

References

[1] D. Deitz, W. Irwin, G. Wilson et al., "Automatic linkage of process event data to a data historian," US Patent 7,275,062, 2007.

[2] I. W. Wilson and E. R. Heinzelmann, "Sequence of events recorder facility for an industrial process control environment," US Patent 7,840,285, 2010.

[3] M. Taub, R. Azevedo, A. E. Bradbury et al., "Using sequence mining to reveal the efficiency in scientific reasoning during STEM learning with a game-based learning environment," *Learning and Instruction*, vol. 54, pp. 93–103, 2018.

[4] J. S. Kinnebrew and G. Biswas, "Identifying learning behaviors by contextualizing differential sequence mining with action features and performance evolution," in *Proceedings of the International Conference on Educational Data Mining (EDM)*, Chania, Greece, June 2012.

[5] N. Béchet, P. Cellier, T. Charnois et al., "Discovering linguistic patterns using sequence mining," in *Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 154–165, Springer, New Delhi, India, March 2012.

[6] R. Kant, S. H. Sengamedu, and K. S. Kumar, "Comment spam detection by sequence mining," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, pp. 183–192, Seattle, WA, USA, February 2012.

[7] Y. Fan, Y. Ye, and L. Chen, "Malicious sequential pattern mining for automatic malware detection," *Expert Systems with Applications*, vol. 52, pp. 16–25, 2016.

[8] G. Weiss, "Predicting telecommunication equipment failures from sequences of network alarms," 2001.

[9] S. Laxman, V. Tankasali, and R. W. White, "Stream prediction using a generative model based on frequent episodes in event sequences," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 453–461, Las Vegas, NV, USA, August 2008.

[10] R. Karoly and J. Abonyi, "Multi-temporal sequential pattern mining based improvement of alarm management systems," in *Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 003870–003875, IEEE, Budapest, Hungary, October 2016.

[11] A. Belhadi, Y. Djenouri, J. C. W. Lin et al., "A general-purpose distributed pattern mining system," *Applied Intelligence*, pp. 1–16, 2020.

[12] M. d'Aquin and N. Jay, "Interpreting data mining results with linked data for learning analytics: motivation, case study and directions," in *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, pp. 155–164, New York, NY, USA, April 2013.

[13] M. El-Hajj and O. R. Zaïane, "Non-recursive generation of frequent k-itemsets from frequent pattern tree representations," in *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery*, pp. 371–380, Springer, Prague, Czech Republic, September 2003.

[14] F. A. Gers and E. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1333–1340, 2001.

[15] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with LSTM," in *Proceedings of the 1999 Ninth International Conference on Artificial Neural Networks ICANN 99*, vol. 2, Edinburgh, UK, September 1999.

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 3104–3112, MIT Press, Montreal Canada, December 2014.

[17] A. Karatzoglou, A. Jablonski, and M. Beigl, "A Seq2Seq learning approach for modeling semantic trajectories and predicting the next location," in *Proceedings of the 26th ACM*

SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 528–531, Seattle, WA, USA, November 2018.

[18] J. Rebane, I. Karlsson, P. Papapetrou et al., "Seq2Seq RNNs and ARIMA models for cryptocurrency prediction: a comparative study," in *Proceedings of the SIGKDD Fintech'18*, London, UK, August 2018.

[19] T. Baumel, M. Eyal, and M. Elhadad, "Query focused abstractive summarization: incorporating query relevance, multi-document coverage, and summary length constraints into Seq2seq models," 2018, https://arxiv.org/abs/1801.07704.

[20] P. Wu, Z. Lu, Q. Zhou et al., "Bigdata logs analysis based on seq2seq networks for cognitive internet of things," *Future Generation Computer Systems*, vol. 90, pp. 477–488, 2019.

[21] S. Hwang, G. Jeon, J. Jeong et al., "A novel time series based Seq2Seq model for temperature prediction in firing furnace process," *Procedia Computer Science*, vol. 155, pp. 19–26, 2019.

[22] G. Dörgő and J. Abonyi, "Learning and predicting operation strategies by sequence mining and deep learning," *Computers & Chemical Engineering*, vol. 128, pp. 174–187, 2019.

[23] G. Dörgő, P. Pigler, M. Haragovics, and J. Abonyi, "Learning operation strategies from alarm management systems by temporal pattern mining and deep learning," in *Proceedings of the 28th European Symposium on Computer Aided Process Engineering*, A. Friedl, J. J. Klemeš, S. Radl et al., Eds., vol. 43, pp. 1003–1008, Elsevier, Amsterdam, Netherlands, 2018.

[24] B. Carter, J. Mueller, S. Jain et al., "What made you do this? understanding black-box decisions with sufficient input subsets," in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics PMLR*, pp. 567–576, Naha, Okinawa, Japan, April 2019.

[25] E. Cohen and C. Beck, "Empirical analysis of beam search performance degradation in neural sequence models," in *Proceedings of the International Conference on Machine Learning PMLR*, vol. 97, pp. 1290–1299, Long Beach, CA, USA, June 2019.

[26] H. Scheidl, S. Fiel, and R. Sablatnig, "Word beam search: a connectionist temporal classification decoding algorithm," in *Proceedings of the 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pp. 253–258, IEEE, Niagara Falls, NY, USA, August 2018.

[27] Z. Li, J. Cai, S. He, and H. Zhao, "Seq2seq dependency parsing," in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 3203–3214, Santa Fem New Mexico, USA, August 2018.

[28] I. Williams, A. Kannan, P. S. Aleksic, D. Rybach, and T. N. Sainath, "Contextual speech recognition in end-to-end neural network systems using beam search," in *Proceedings of the Interspeech 2018*, pp. 2227–2231, Hyderabad, India, September 2018.

[29] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," 2017, https://arxiv.org/abs/1702.01806.

[30] D. Jahier Pagliari, F. Daghero, and M. Poncino, "Sequence-to-sequence neural networks inference on embedded processors using dynamic beam search," *Electronics*, vol. 9, no. 2, p. 337, 2020.

[31] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju et al., "Diverse beam search: decoding diverse solutions from neural sequence models," 2016, https://arxiv.org/abs/1610.02424.

[32] N. Humbatova, G. Jahangirova, G. Bavota et al., "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 1110–1121, Seoul, South Korea, June 2020.

[33] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proceedings of the Thirteenth Annual Conference of the International Speech Communication Association*, Portland, OR, USA, September 2012.

[34] G. Dorgo, P. Pigler, and J. Abonyi, "Understanding the importance of process alarms based on the analysis of deep recurrent neural networks trained for fault isolation," *Journal of Chemometrics*, vol. 32, no. 4, Article ID e3006, 2018.