Inferring a Multi-perspective Likelihood Graph from Black-Box Next Event Predictors

Yannik Gerlach, Alexander Seeliger(✉), Timo Nolle, and Max Mühlhäuser

Telecooperation Lab, Technical University of Darmstadt, Darmstadt, Germany
{gerlach,seeliger,nolle,max}@tk.tu-darmstadt.de

Abstract. Deep learning models for next event prediction in predictive process monitoring have shown significant performance improvements over conventional methods. However, they are often criticized for being black-box models. Without allowing analysts to understand what such models have learned, it is difficult to establish trust in their abilities.

In this work, we propose a technique to infer a likelihood graph from a next event predictor (NEP) to capture and visualize its behavior. Our approach first generates complete cases, including event attributes, using the NEP. From this set of cases, a multi-perspective likelihood graph is inferred. Including event attributes in the graph allows analysts to better understand the learned decision and branching points of the process.

The results of the evaluation show that inferred graphs generalize beyond the event log, achieve high F-scores, and small likelihood deviations. We conclude black-box NEP can be used to generate conforming cases even for noisy event logs. As a result, our visualization technique, which represents exactly this set of cases, shows what the NEP has learned, thus mitigating one of their biggest criticisms.

Keywords: Process mining · Next event predictors · Multi-perspective likelihood graph

1 Introduction

In process mining, more and more approaches make use of *machine learning* (ML) techniques, especially deep neural networks, to better capture the behavior of a process. ML algorithms are capable of incorporating more contextual factors of a case than conventional and hand-crafted models, making them a nice fit for the application in this field. For instance, predictive process monitoring has shown significant performance improvements with respect to predictive accuracy using recurrent neural networks [3,6,17,22].

The primary goal of process mining is to obtain valuable insights for analysts to improve process performance. On the one hand, many process mining

X. Franch et al. (Eds.): CAiSE 2022, LNCS 13295, pp. 19–35, 2022.
https://doi.org/10.1007/978-3-031-07472-1_2

tasks are human-centric, like process discovery or conformance checking, presenting results such that humans can actually see and understand them (e.g., process models). On the other hand, ML approaches to process mining are a popular choice as their predictive power can be much better [3,17,22], despite inhibiting direct inspection of trained models by analysts. Therefore, ML models are often rightfully criticized for being black-boxes [12,14,18,19,21,23]. Recent discussions in the community voiced the need for process mining methods that can be explained and interpreted [5] by analysts to build trust [11,12,15,23] and to reduce the vulnerability of prediction errors [19]. Explainability should encompass the visualization of processes as graphs or enriched process models with decision and constraint annotations [4]. While techniques for explaining predictions made by ML models exist, they often focus on relevant parts in the input, i.e., which part in the input case is important for the prediction, on a single prediction at a time, or are not directly tailored to process mining. Therefore, extracting knowledge about what the ML model has actually learned, to build trust in its capabilities and inspect what the model "thinks" the process is, remains challenging.

In this paper, we propose a technique for inferring a multi-perspective likelihood graph of events, transitions, and event attributes given an algorithm predicting the next event of a case. Such a *next event predictor* (NEP) can be a trained ML model but can also be any other algorithm performing this task (e.g., an ensemble of NEPs). The inferred graph can be visualized and explored by analysts to outline what the NEP has learned, similar to process models. The proposed approach is a post-hoc explanation technique because it is applied after training or creation of the NEP and aims at explaining the NEP's behavior. The visual explanation through a graph further contributes to the main goal of better understanding the behavior of the predictive model [1].

Our proposed approach consists of two main steps: First, using the NEP we generate all possible cases reflecting the process behavior. Second, we convert these cases to a graph by merging nodes which share the same behavior to remove redundancies and to compact the graph. The case generation is agnostic to the architecture of the NEP and, thus, can be adapted to work with any NEP.

In summary, our contributions are:

1. A novel method for revealing the process an NEP has learned from training on an event log.
2. A visual representation of the learned behavior in form of a likelihood graph that also includes event attributes.
3. A comprehensive evaluation showing our approach is capable of accurately representing the behavior of an NEP and its decision points in the process.

The rest of the paper is organized as follows. In Sect. 2 we provide an overview of the related work, followed in Sect. 3 by some preliminaries used throughout the paper. In Sect. 4 we describe the details of our approach. Then, in Sect. 5 we present the results of the evaluation of our approach. Finally, in Sect. 7 we conclude the paper with a discussion and future work.

2 Related Work

Next Event Predictors (NEP). ML-based NEPs have previously been used for generating cases from scratch. Tax et al. [17] iteratively predict the next activity and time, choosing the most likely continuation until a case is completed. Similarly, Camargo et al. [2] use *Long Short-Term Memory* (LSTM) networks to predict sequences of next events, their timestamps, and event attributes. Both of these approaches either focus on the most likely choice for the next activity or a random selection following the predicted probability distribution. As a result, depending on the complexity of the model, it is required to generate a large number of cases to capture the entire behavior of the NEP. In contrast, we aim to explore all non-noisy cases using a threshold heuristic.

Model Discovery. An approach more focused on explainability is introduced by Shunin et al. [13], who generate a finite state machine from a neural network NEP. Similar to our approach, the neural network is trained on an event log, and is then used to generate a set of cases from scratch. Then, a process model is generated from the finite state machine. Another process model-based approach is to learn a generic mapping between event log structures and process models [16]. The idea is that a graph neural network learns the generic dependencies between relationships of events to generate the structures of the process model. The learned mapping can then be applied to new, unseen event logs.

Different from our approach, both related works do not consider noisy event logs. Furthermore, only the control-flow of a process is discovered, making it difficult to analyze decision points.

Evaluation Frameworks and Metrics. As an alternative to explanatory approaches, related work also presents evaluation frameworks and metrics. Camargo et al. [2] compare the generated cases of an NEP to the event log to evaluate their quality. However, the metric does not consider noisy cases that are typically present in real-life event logs, leading to an increase of both precision and fitness. Despite this property being undesirable, the evaluation measures do not reflect it. A framework investigating the generalization ability of neural networks is introduced by Peeperkorn et al. [9]. Their metrics are based on removing variants from an event log, training the neural network and then checking how many of the removed variants the network can reconstruct. The authors conclude that overfitting avoidance methods are important to allow the neural network to generalize beyond the training data. The approach is restricted to control-flow only and does not consider noise. Different from our approach, the approach samples from the probability distribution instead of an exhaustive search. Additionally, both presented approaches do also not visualize the generated cases, preventing efficient exploration and inspection.

3 Background

Throughout the paper, we refer to the event log, event, case, and attribute definitions from [20]:

Definition 1 (Case, Event, and Event Log). *Let $\mathcal{A} = \{a_1, \dots, a_A\}$ be a set of ordered attributes and $\mathcal{V}$ be a set of attribute values, where $\mathcal{V}_{a_i}$ is the set of possible values for attribute $a_i \in \mathcal{A}$. Furthermore, let a_1 be the activity and a_j with $j = 2, \dots, A$ the event attributes.*

Let $\mathcal{C}$ be the set of all cases and $\mathcal{E}$ be the set of all events. A case $c \in \mathcal{C}$ is an event sequence $c \in \mathcal{E}^*$, where $\mathcal{E}^*$ is the set of all sequences over $\mathcal{E}$. An event log is a set of cases $\mathcal{L} \subseteq \mathcal{C}$. A prefix is an ongoing, incomplete case, where the prefix length is the number of completed events.

Furthermore, let $|c|$ be the number of events in case c, $C = |\mathcal{L}|$ be the number of cases in $\mathcal{L}$, $A = |\mathcal{A}|$ be the number of attributes and $E = \max_{c \in \mathcal{L}} |c|$ be the case with the most events in $\mathcal{L}$. We denote start and end symbol by $\blacktriangleright$ and $\blacksquare$, respectively.

In process mining, next event prediction is a set of A multi-class classification problems. For the classification problem of a_i with $i = 1, \dots, A$, the classes to be predicted are its values $\mathcal{V}_{a_i}$. The features, based on which the event at position i is determined, are the previous events, the prefix $1, \dots, i-1$ of the input case.

Definition 2 (Next Event Predictor). *A next event predictor is any algorithm $\mathcal{N}$ receiving a prefix as input and producing a set of next events, annotated with likelihoods for each attribute value.*

From the likelihoods for each attribute value, the likelihood of the event can be calculated and the most likely one returned, for example. The above definition of an NEP allows our approach to be applicable independently of the NEP's specific architecture.

4 Inferring Graphs from Next Event Predictors

Our approach is a post-hoc explanation technique that infers a directed and acyclic likelihood graph from an NEP to reveal the process it has learned from an event log. The resulting graph compactly represents the predictions made by the NEP and serves as a visual explanation of the NEP's behavior. The semantics of the likelihood graph is similar to that of a stochastic process model [10]. However, it may contain nodes with the same labels multiple times to model loop behavior more accurately. This allows us to model different probabilities of the same activity for different paths through the process. Additionally, the graph includes event attributes to better capture the behavior of the NEP at decision points. These can reveal dependencies between activities and event attributes that remain hidden when only constructing a process map.

The proposed approach follows two main steps to generate a likelihood graph of the determined process by the NEP:

1. **Exhaustive Case Generation.** Our approach explores the NEP to generate an exhaustive set of cases by repeatedly extending prefixes with predicted events. A heuristic strategy is applied to distinguish between normal and noisy behavior.

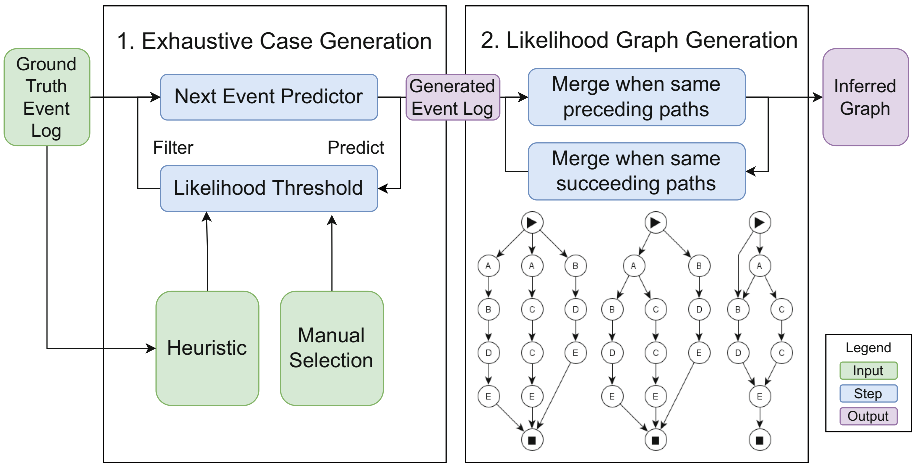

Fig. 1. Overview of the components of the proposed approach.

2. **Likelihood Graph Generation.** We use the generated set of cases to build a likelihood graph that captures the predictions made by the NEP. A merging strategy is used to compact the graph such that similar behavior is grouped.

Next, we provide a detailed description of each step. An overview of the approach is depicted in Fig. 1.

4.1 Exhaustive Case Generation

In this step, our approach extracts the knowledge about the process the NEP has learned. We perform an exhaustive search to explore all possible cases that the NEP generates and to capture its behavior in form of an event log, representing its predictions given certain prefixes. In order to avoid pursuing all possible and very unlikely cases, which may lead to a huge search space, we use a threshold heuristic to only consider case continuations with at least a certain likelihood. As a result, activities and attribute values below this threshold are not considered in the case generation; they are considered to be noise.

Algorithm 1 shows our case generation algorithm, which is a worklist algorithm that starts with an empty prefix of a case, only containing the artificial start symbol (▶) of the NEP. For each remaining prefix (line 5), the algorithm obtains all possible combinations of next events with their likelihoods in the sequence from the NEP, denoted by $\langle \dots \rangle$ (line 8). The prefix is then extended by the continuation to generate a new prefix $\boldsymbol{c}_{new}$ (line 11). If the activity of the next event is the end symbol (■, line 12), the entire case is added to the list of completed cases ($c_{complete}$). Otherwise, the new prefix is added to the worklist.

In most scenarios, our algorithm terminates when all possible cases are generated. However, in certain cases it may be that the NEP predicts looping behavior (e.g., activities a and b are predicted alternatingly). To allow the algorithm to

Algorithm 1. Extract all cases with normal behavior from an NEP.

```
Require: NEP $\mathcal{N}$, likelihood thresholds $t_1, ..., t_A$
 1: function GenerateCases
 2:     $c_{complete} = \varnothing$
 3:     $\boldsymbol{c}_{initial} = \langle \blacktriangleright^A \rangle \in \mathbb{R}^{1 \times A}$
 4:     $q = [\boldsymbol{c}_{initial}]$
 5:     while $|q| > 0$ do
 6:         $\boldsymbol{c} = q[0]$
 7:         remove $\boldsymbol{c}$ from $q$
 8:         $e_{next} = \mathcal{N}(\boldsymbol{c})$
 9:         for each $(\boldsymbol{e} \in \mathbb{R}^A, \boldsymbol{l} \in \mathbb{R}^A)$ in $e_{next}$ do
10:             if $\forall i \in \{1, \ldots, A\} : l_i \geq t_i$ then
11:                 $\boldsymbol{c}_{new} = \langle \boldsymbol{c}, \boldsymbol{e} \rangle \in \mathbb{R}^{(|c|+1) \times A}$
12:                 if $\boldsymbol{e}_0 = \blacksquare$ then
13:                     $c_{complete} = c_{complete} \cup c_{new}$
14:                 else
15:                     $q = [\boldsymbol{c}_{new}, q]$
16:     return $c_{complete}$
```

terminate in all scenarios, cases are discarded if an activity occurs more than ϵ-times. If a too high percentage of cases is discarded, the algorithm is terminated, indicating either a too low likelihood threshold or an inaccurate NEP. For filtering out noisy behavior, we choose to set *likelihood thresholds* $t_1, \ldots, t_A$ for each attribute (line 10). Therefore, for a prediction for attribute a_i to be further considered in the case generation, its probability needs to be at least t_i. We use a separate threshold for each attribute to allow a more precise filtering of noise. Related work [7] has shown that the likelihoods vary significantly between different attributes, thus, specifying individual thresholds allows fine grained control (see Sect. 4.3).

4.2 Likelihood Graph Generation

In this step, our approach constructs the likelihood graph from the set of cases, generated by the NEP. We opted for a graph because analysts are familiar with this structure (e.g., process models in process discovery), which are also used for visual exploration. As a result, we obtain a compact representation of the generated cases that serves as a visual explanation of the NEP. For obtaining such a likelihood graph, we use a merge strategy to eliminate redundant sequences of events if they behave similarly. Different from process discovery techniques, we explicitly include event attributes in the graph to better reflect branching decisions depending on the attribute probabilities.

Basic Graph Structure. We start by converting the set of cases to nodes and edges in a graph. The initial graph consists of an independent path for each case, only afterwards we merge. For each event, we convert it into a list of nodes, one for each attribute value in the event. Each node is assigned a label (activity or event attribute value name) and an unique identifier. The nodes representing attribute values are ordered and connected following the order of the attributes. Events are then linked in the order they appear in the case. To be more precise, when linking two events, the last attribute value node of the previous event is connected to the activity value node of the current event. All cases share the same start and end symbol. The conversion approach is illustrated in Fig. 1.

Merging Nodes Strategy. We merge nodes based on their context to remove redundancies and improve comprehensiveness of the resulting likelihood graph. Given any two nodes n_1 and n_2 in the graph with the same label (e.g., same activity) but different identifiers. If the two nodes share the same preceding sequences of labels (even though the nodes in the sequences might have different identifiers), one of the nodes is redundant, i.e., their behavior cannot be distinguished, up until that point in the sequence. Therefore, we merge these nodes into a single node n with possibly two outgoing edges. As a result, nodes with the same label are not merged if their behavior is different. With the same motivation, we also apply the merge strategy if two nodes share the same succeeding sequences of labels. The presented merging strategy does not change the set of cases the graph represents but only the structure of the nodes. As a result, no information (e.g., decision points or attribute values) gets lost due to merging.

To remove both types of redundancies, our approach merges once based on preceding and succeeding sequences, respectively. For generating the preceding paths of a node n, all paths between the start node and n are determined by following the ingoing edges of n backwards. For the succeeding paths of a node n, all paths between n and the end node are determined by following the outgoing edges of n forwards. Afterwards, only behavior impacting the control-flow remains. The order of merging does not affect the final result because the set of preceding or succeeding label sequences is not changed but only the structure of the graph. Note that the event attribute nodes are not entirely separated from their activity node. While activity nodes are always checked to see if they can be merged, the user may specify which event attribute nodes to attempt to merge. The proposed merging procedure contrasts process discovery algorithms, which typically merge activities with the same name, regardless of their context. Merging event attributes can further reduce the size of the graph at the expense of run time.

Example. Let us consider the following three cases: $\langle \blacktriangleright, a, b, d, e, \blacksquare \rangle$, $\langle \blacktriangleright, a, c, c, e, \blacksquare \rangle$ and $\langle \blacktriangleright, b, d, e, \blacksquare \rangle$. In the first step, the initial graph is created where every case is a single, isolated and independent path (Fig. 1, left). In the second step, nodes are merged based on their predecessor paths. Both nodes with activity a have the same preceding paths and are, thus, merged together, resulting in the middle graph in Fig. 1. After merging nodes based on predecessor paths, nodes are merged based on successor paths. Note that only nodes with the same activities are compared. The first and third case are merged because the third one is a suffix of the first one. Furthermore, activity e is always followed by the end symbol, resulting in the right graph in Fig. 1. No other nodes can be merged without changing the behavior of the graph.

4.3 Likelihood Thresholds

A main component of our approach is the threshold heuristic strategy for identifying noisy behavior predicted by the NEP. Here, the set of likelihood thresholds $t_1, \ldots, t_A$ control the predictions being considered in the extension of a prefix. While low thresholds might lead to noise being included in cases, high thresholds lead to a likelihood graph only focused on common variants. As such, the thresholds control the granularity of the inferred likelihood graph and the extent to which filter out noisy behavior.

As a strategy to determine a likelihood threshold for each attribute, we use the *(mean-centered) lowest plateau heuristic*, earlier introduced for anomaly detection in [8]. The heuristic provides an out-of-the-box threshold, which serves as a starting point for further adjustments, e.g., decreasing in steps if only few cases are generated, based solely on the prediction probabilities.

An alternative approach is to find candidate thresholds by optimizing the F-score of the likelihood graph, given the event log that was used to train the NEP. Particularly, we systematically explore different thresholds to maximize the F-score for the resulting likelihood graph. If the training event log contains much of the ground truth process behavior, this behavior will then also be reflected in the resulting likelihood graph. However, the noise that is present in the training event log, can, if incorporated in the inferred likelihood graph, have a negative impact because including the noise improves the graph's F-score, therefore resulting in an undesired optimization.

5 Evaluation

In this section, we evaluate our approach using synthetic and real-life event logs. The primary objective of our experiments is to find out how well our approach is able to capture the behavior of the ground truth event log. To ensure transparency and reproducibility of the results, we publish our source code[1] and rely on publicly available datasets. The repository also contains additional example graphs for real-life event logs.

5.1 Datasets

We use process models introduced by Nolle et al. [7] to generate likelihood graphs of activities and event attributes that serve as the ground truth in our evaluation. These graphs allow us to compare the inferred graphs to the desired outcome and, thus, are referred to as the *ground truth likelihood graphs*. These graphs vary in complexity w.r.t. the number of activities, depth, and width. We playout these graphs to obtain a *ground truth event log* that represents the entire process designed in the ground truth likelihood graph. To generate a *sampled event log*, random walks through the ground truth likelihood graph are performed in compliance with the transition probabilities. In addition, we introduce noise to

[1] https://github.com/yannikgerlach/likelihood-graphs-from-neps.

Table 1. Statistics of the synthetic event logs used in the evaluation

Name	# Cases	# Events	# Variants	# Activities	Avg. case length
Small	2 148	43 498	670	39	8.7
Wide	6 414	31 810	563	56	6.4
Medium	16 672	31 928	684	63	6.4
p2p	21 608	43 265	610	25	8.7
Paper	52 020	50 413	777	27	10.1
huge	102 475	43 249	890	107	8.7

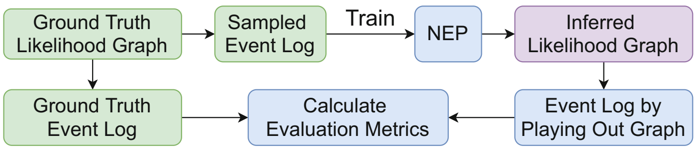
Ground Truth Likelihood Graph
Sampled Event Log
Train
NEP
Inferred Likelihood Graph
Ground Truth Event Log
Calculate Evaluation Metrics
Event Log by Playing Out Graph

Fig. 2. Relation between ground truth graph, event log, NEP and inferred graph.

the sampled event log by randomly applying noise (skip, insert, rework, early, and late activities as well as attribute anomalies) [7] to 30% of the cases.

If the sampled event log does not contain all cases of the ground truth event log, we can be sure the NEP has not seen all ground truth cases during training. We can therefore effectively measure its generalization ability. For the used event logs, for which an overview is given in Table 1, sampled event logs with 5 000 cases each allows us to do this.

5.2 Experimental Setup

We compare the cases obtained by playing out the inferred likelihood graph to the ground truth event log, which does not contain noisy behavior. This improves upon related work, which evaluates the generated cases on the sampled event log, as described in Sect. 2. It allows to better estimate how well an NEP extracts relevant behavior from a sampled event log and metrics such as precision and fitness are more meaningful. Figure 2 shows the evaluation setup for our approach.

Training of the NEPs. For our experiments, we use the NEP by Nolle et al. [7] which is a multi-perspective NEP, predicting the next activity and event attributes. When predicting the event attributes, the predicted activity is considered, enabling the NEP to better learn dependencies.

Algorithm 2. Calculate fitness and precision between two event logs.

```
1: function WeightedLevenshteinDistance($\mathcal{L}_1, \mathcal{L}_2$)
2:     $\hat{\mathcal{L}_1} = \mathcal{L}_1 \setminus \mathcal{L}_2$, $\hat{\mathcal{L}_2} = \mathcal{L}_2 \setminus \mathcal{L}_1$
3:     $d = \sum_{c_2 \in \hat{\mathcal{L}_2}} \min_{c_1 \in \mathcal{L}_1} LV(c_2, c_1)$
4:     $d_{weighted} = d / \sum_{c \in \mathcal{L}_2} |c|$
5:     return $1 - d_{weighted}$
```

We train an ensemble of 10 NEPs (with different seeds) on and for each sampled event log. To compare the prediction quality to a single NEP, we also use each NEP individually and average the resulting evaluation measures. For training, we use 90% of the data; the remaining 10% are used for validation.

Optimizing Likelihood Threshold. In Sect. 4.3 we propose optimizing the threshold for F-score on the sampled event log as a heuristic. To evaluate this approach, we also determine the threshold achieving the best F-score on the ground truth event log, i.e., an optimal threshold. However, this optimal threshold cannot be determined without a ground truth likelihood graph. The optimization on the sampled event log can always be performed. For optimizing, we use Bayesian optimization with 10 initial steps and 10 optimization steps. We investigate thresholds in the range of 0.02 to 0.4.

5.3 Evaluation Measures

We evaluate our approach with respect to the following aspects:

(1) We measure fitness, precision, and generalization ability to evaluate the quality of the resulting likelihood graph. For a better comparison, we also show the F_1*-measure*. To measure generalization ability, we take the set of generated cases, remove the cases present in the event log, and only take from the remaining cases those that are also present in the ground truth event log. We then divide the size of this set by the number of generated cases.

(2) We measure both the difference in likelihood prediction in single events and in the likelihood of a case to compare how well the inferred event and case likelihoods approximate the likelihoods defined in the ground truth graph.

We calculate fitness and precision between two event logs, the set of *generated cases* (GCs) and the set of *ground truth cases* (GTCs) in ground truth event log. Similar to Camargo et al. [2], who use a related measure in their evaluation, we use the normalized Levenshtein (LV) edit distance (see Algorithm 2) to compute the distance between two cases, excluding the start and end symbol. Note that for cases that are contained in both event logs, the distance is 0. Let $\mathcal{L}_1$ be the GTCs and $\mathcal{L}_2$ be the GCs. Then Algorithm 2 calculates the precision of the inferred likelihood graph. When switching the sets of cases, i.e. $\mathcal{L}_1$ are the GCs and $\mathcal{L}_2$ are the GTCs, Algorithm 2 calculates the fitness.

We define the *(average) absolute likelihood difference per event and attribute* (ALDE), as well as the *mean squared error per event and attribute* (MSEE). In comparison to ALDE, MSEE penalizes greater differences in the likelihoods more. Regarding case likelihoods, their magnitudes can be vastly different depending on the length of a case and the values are usually quite small. These two observations make the evaluation of absolute case likelihoods difficult and unintuitive. We mitigate this problem by dividing the difference in case likelihoods by the ground truth case likelihood. As a result, we obtain the relative likelihood difference, in the following called *normalized (average) likelihood difference per case* (NLDC).

As a baseline, we compare the sampled event log with the ground truth event log. Depending on the predictive performance of the NEP, it should be able to outperform the baseline. First, the NEP should filter out noise and, thus, have higher precision than the sampled event log. Second, the NEP should generalize the seen behavior and, thus, have higher fitness. Nevertheless, the reconstructed likelihood graph does not contain all attribute values and transitions because unlikely ones are filtered out using the likelihood thresholds. In contrast, the event log contains all attribute values and transitions, even those below the likelihood thresholds.

5.4 Results: Synthetic Event Logs

Table 2 lists the results for each synthetic event log. The baseline is outperformed by our approach for all event logs regarding the control-flow considerably, except for the P2P event log. Due to the structure of the P2P event log it appears to be difficult to find automatic thresholds. In general, however, optimization based on the ground truth event log yields good F-scores, with the optimization on the sampled event log being a reasonable choice as well. Interestingly, across all settings, the achieved precision is often close to 1.0, implying that mostly correct cases are generated. However, usually not all of the correct cases are generated, as indicated by the fitness values. Disregarding P2P, fitness appears to decrease with increasing size of the ground truth event log as the sampled event log size stays constant. As a result, more behavior is missing in the sampled event log. We also observe good generalization ability in the tested NEP, implying the NEP generates a good amount of correct cases that it has not seen in training.

When presented with a real-life event log, the choice is to use the lowest-plateau heuristic or optimization on the sampled event log. In the results we observe no clear tendency for one or the other, although the optimization on the event log appears to be slightly better, but also more time-intensive. Regarding the found thresholds, the threshold for the event attribute appears to be especially important for achieving good F-scores. As an activity often has multiple valid event attribute values, getting them right appears to have more impact on the F-score than getting the activity right.

When comparing the performance of single NEPs to ensembles, we observe improved prediction performance (especially when optimizing for an optimal threshold), indicating its effectiveness.

Table 2. The found thresholds (t_1 for the activity and t_2 for the event attribute), F_1-score, precision, fitness, generalization ability and quality of likelihoods for the synthetic event logs and NEP. Optimized on Event Log (EL), mean lowest-plateau threshold Heuristic (HR) or Optimal (OPT) threshold. Using a Single Model (SM) or Ensemble (EM). Note that averaging the thresholds for single models is not meaningful.

Log	Model	t_1	t_2	F_1	Prec.	Fit.	Gen.	NLDC	ALDE	MSEE
SMALL	Baseline	–	–	0.905	0.840	0.981	–	–	–	–
	EL SM	–	–	0.988	1.000	0.977	0.183	0.486	0.005	0.001
	EL EM	0.213	0.020	**0.994**	1.000	0.989	0.247	0.579	**0.004**	0.001
	HR EM	0.364	0.090	0.983	1.000	0.967	0.150	**0.367**	0.005	0.001
	OPT SM	–	–	0.991	0.999	0.982	0.205	0.533	0.005	0.001
	OPT EM	0.368	0.021	**0.994**	1.000	0.989	0.247	0.579	**0.004**	0.001
WIDE	Baseline	–	–	0.847	0.828	0.866	–	–	–	–
	EL SM	–	–	0.895	0.975	0.830	0.583	1.353	0.071	0.028
	EL EM	0.143	0.051	0.892	0.946	0.845	0.591	1.014	0.066	0.026
	HR EM	0.210	0.090	0.844	1.000	0.730	0.485	2.009	0.081	0.033
	OPT SM	–	–	0.914	0.946	0.886	0.594	0.760	0.053	0.018
	OPT EM	0.219	0.020	**0.953**	0.989	0.918	0.755	**0.416**	**0.045**	0.017
MEDIUM	Baseline	–	–	0.878	0.873	0.883	–	–	–	–
	EL SM	–	–	0.941	0.994	0.894	0.700	0.991	0.026	0.005
	EL EM	0.136	0.083	0.926	0.990	0.863	0.680	0.998	0.025	0.005
	HR EM	0.420	0.070	0.932	1.000	0.872	0.695	0.931	**0.020**	0.003
	OPT SM	–	–	0.961	0.972	0.950	0.632	0.671	0.023	0.004
	OPT EM	0.214	0.020	**0.974**	0.989	0.918	0.727	**0.519**	**0.020**	0.003
P2P	Baseline	–	–	**0.854**	0.858	0.849	–	–	–	–
	EL SM	–	–	0.782	0.982	0.652	0.469	2.270	0.030	0.005
	EL EM	0.073	0.095	0.794	0.993	0.662	0.633	1.476	0.023	0.003
	HR EM	0.140	0.080	0.802	1.000	0.670	0.713	**0.999**	**0.021**	0.003
	OPT SM	–	–	0.786	0.965	0.668	0.494	2.401	0.028	0.004
	OPT EM	0.073	0.095	0.794	0.993	0.662	0.633	1.476	0.023	0.003
PAPER	Baseline	–	–	0.883	0.899	0.868	–	–	–	–
	EL SM	–	–	0.911	0.999	0.836	0.683	5.084	0.042	0.013
	EL EM	0.102	0.112	0.908	1.000	0.831	0.660	4.626	0.039	0.013
	HR EM	0.351	0.030	**0.943**	0.990	0.900	0.837	**2.863**	**0.034**	0.011
	OPT SM	–	–	0.928	0.988	0.875	0.746	3.667	0.040	0.013
	OPT EM	0.295	0.070	0.927	0.999	0.865	0.813	3.195	0.035	0.012
HUGE	Baseline	–	–	0.843	0.890	0.801	–	–	–	–
	EL SM	–	–	0.881	0.985	0.799	0.522	3.509	**0.023**	0.006
	EL EM	0.400	0.057	**0.912**	0.987	0.848	0.620	1.781	**0.023**	0.004
	HR EM	0.130	0.050	0.842	0.827	0.857	0.274	1.452	0.028	0.007
	OPT SM	–	–	0.892	0.933	0.856	0.366	1.993	0.030	0.007
	OPT EM	0.329	0.021	**0.912**	0.917	0.907	0.124	**1.034**	0.030	0.007

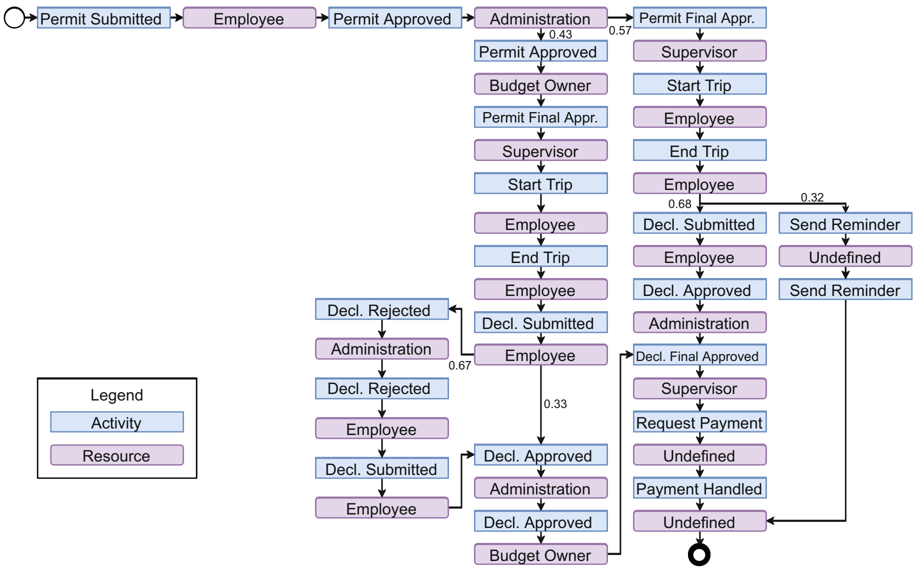

Fig. 3. Inferred graph on the travel permits log (BPIC 2020) with a single event attribute. *Undefined* user implies activity performed by the system.

The inferred transition likelihoods are typically quite close to the true likelihood (ALDE ≤ 0.04). The likelihoods of cases is also often not too far away from the true case likelihood, although it remains somewhat difficult to interpret. The best likelihood approximations correlate with the best F-scores, possibly due to the averaging of the outgoing edges.

5.5 Results: Real-Life Event Log

We apply our approach to publicly available real-life event logs from the BPI Challenge (2012, 2013, 2015, 2017, and 2020). Due to the space limitation of the paper, the details and the resulting likelihood graphs can be accessed from our repository. Here, we discuss the results of the Travel Permits event log from 2020[2], which consists of 7,065 cases and 86,581 events. The resulting likelihood graph is depicted in Fig. 3. Likelihoods of 1.0 are omitted for better readability. The NEP is trained with the activity and the resource event attribute; likelihood thresholds are set according to our threshold strategy. The likelihood graph shows three branching paths (permit approved vs. permit final approved, declaration approved vs. declaration rejected, and declaration submitted vs. send reminder) with different probabilities depending on the activity performed. Also, the additional activities executed for the permit approved-path are shown.

[2] van Dongen, Boudewijn (2020): BPI Challenge 2020. 4TU.ResearchData. Collection. https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.

6 Discussion

The results of our evaluation show that the inferred likelihood graph reflects the behavior that an ML-based NEP has learned. Similar to a stochastic process model in process discovery, it visualizes the process of the NEP for further exploration, inspection, and debugging of the NEP. However, it should be noted that our approach is not designed as a discovery algorithm, although it similarly infers a graph with directly-follows relations. Contrary to many process discovery algorithm, which usually solely focus on the activity, our approach is able to handle an arbitrary number of event attributes (within machine memory limits) in addition to the activity if the NEP predicts those. Our approach allows analysts to better understand the decision points due to the explicit inclusion of event attributes. Therefore, we support explaining the black-box NEP and the process, besides offering a graphical representation of what the NEP has learned [4]. Dependencies between activities and event attributes are represented as separate paths through the graph, each with their respective probabilities.

6.1 Limitations

We consider that our approach exhibits mainly two limitations. One is the potential exponential explosion in the number of generated cases due to the exhaustive search that explores all combinations of activities and attributes. Thus, generating the cases can be time-intensive for processes with very long cases and a large set of attribute value combinations. The second limitation is that our approach only supports the interleaving of concurrent activities, which is susceptible to different orderings of activities. This could be addressed using the same heuristics as process discovery algorithms. Also, our approach does not create loops but unrolls them to better model probabilities for different case prefixes and attribute values. However, this can lead to large graphs containing repeating sequences of nodes. We address this issue by limiting the number of activity occurrences within a case. If this limit is exceeded, the case is discarded and, thus, excluded from the case generation. As a result, it is not included in the inferred graph and also not considered in the evaluation.

6.2 Threats to Validity

While our evaluation on a deep learning model provided relevant observations, we did not explicitly include the evaluation of the NEP itself which directly influences the quality and the interpretability of the likelihood graph. Take, for instance, a NEP randomly predicting activities which is entirely disconnected from the original input data it was "trained" on. Since the likelihood graph is inferred from the generated cases, the likelihood graph will not contain any behavior of the original input but the behavior that the NEP predicts with the noise cut off. This is the expected behavior because we want to know what the NEP has learned, and in this example we can see that it did not learn the original process. Consequently, incorporating an extensive evaluation of the NEP itself may be needed to fully understand the likelihood graph.

7 Conclusion

In this paper, we introduced a novel approach for inferring a multi-perspective likelihood graph from any NEP trained on an event log. Our proposed approach first explores the predictions of the NEP and then creates a compact visual representation of its revealed internals, combining the sequence of activities as well as the event attributes into a single likelihood graph. Therefore, the graph uncovers what black-box NEPs (e.g., ML models) have learned, thus visually explaining them, allowing analysts to better understand branching decisions and mitigating one of their biggest criticisms. Particularly, the visual explanation allows analysts to estimate if the NEP is sufficiently trained and good to deploy. By exploring the graph, the analyst can check assumptions about the process and if the learned process appears reasonable (e.g. the graph might be bigger in regions of uncertainty). As the inferred likelihood graph includes the event attributes, the analyst can identify decision points that lead to different paths and process outcomes (i.e. dependencies between activities and event attributes). For example might a process take a different path depending on the users executing a certain activity. Such decision points can provide valuable insight into how real processes are executed. In this regard, our likelihood graph also provides the probability distribution over the directly-followed activity and event attribute values, additionally allowing us to calculate likelihoods of cases and outcomes. The results of our experiments show that the inferred likelihood graphs accurately describe the original likelihood graphs from which the event logs were sampled and the NEPs were trained on.

As for future work, it is needed to systematically prune case continuations that lead to no additional knowledge about the NEP, i.e., paths or loops through the graph that are already covered by other paths. Also, our approach does not consider the concurrent execution of activities, which could reduce the complexity and increase expressiveness of the likelihood graph. Regarding the evaluation, a study with real users might give valuable insights into how business analysts can use the presented approach to generate value.

References

1. Barredo Arrieta, A., et al.: Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. Inf. Fusion **58**, 82–115 (2020)
2. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate LSTM models of business processes. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 286–302. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_19
3. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
4. Gal, A., Senderovich, A.: Process minding: closing the big data gap. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 3–16. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_1

5. Galanti, R., Coma-Puig, B., de Leoni, M., Carmona, J., Navarin, N.: Explainable predictive process monitoring. In: International Conference on Process Mining (2020)
6. Mehdiyev, N., Evermann, J., Fettke, P.: A novel business process prediction model using a deep learning method. Bus. Inf. Syst. Eng. **62**(2), 143–157 (2018). https://doi.org/10.1007/s12599-018-0551-3
7. Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: BINet: multi-perspective business process anomaly classification. In: Information Systems (2019)
8. Nolle, T., Seeliger, A., Mühlhäuser, M.: BINet: multivariate business process anomaly detection using deep learning. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 271–287. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_16
9. Peeperkorn, J., Vanden Broucke, S.K.L.M., De Weerdt, J.: Can deep neural networks learn process model structure ? An assessment framework and analysis. In: International Workshop on Leveraging Machine Learning in Process Mining (2021)
10. Polyvyanyy, A., Moffat, A., García-Bañuelos, L.: An entropic relevance measure for stochastic conformance checking in process mining. In: International Conference on Process Mining, pp. 97–104 (2020)
11. Rehse, J.-R., Mehdiyev, N., Fettke, P.: Towards explainable process predictions for industry 4.0 in the DFKI-Smart-Lego-Factory. KI - Künstliche Intelligenz **33**(2), 181–187 (2019). https://doi.org/10.1007/s13218-019-00586-1
12. Rizzi, W., Di Francescomarino, C., Maggi, F.M.: Explainability in predictive process monitoring: when understanding helps improving. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 141–158. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_9
13. Shunin, T., Zubkova, N., Shershakov, S.: Neural approach to the discovery problem in process mining. In: Analysis of Images, Social Networks and Texts, pp. 261–273 (2018)
14. Sindhgatta, R., Moreira, C., Ouyang, C., Barros, A.: Exploring interpretable predictive models for business processes. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 257–272. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_15
15. Sindhgatta, R., Ouyang, C., Moreira, C., Liao, Y.: Interpreting predictive process monitoring benchmarks. arXiv (2019)
16. Sommers, D., Menkovski, V., Fahland, D.: Process discovery using graph neural networks. In: International Conference on Process Mining, pp. 40–47 (2021)
17. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
18. Tax, N., van Zelst, S.J., Teinemaa, I.: An experimental evaluation of the generalizing capabilities of process discovery techniques and black-box sequence models. In: Gulden, J., Reinhartz-Berger, I., Schmidt, R., Guerreiro, S., Guédria, W., Bera, P. (eds.) BPMDS/EMMSAD -2018. LNBIP, vol. 318, pp. 165–180. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91704-7_11
19. Theis, J., Darabi, H.: Decay replay mining to predict next process events. IEEE Access **7**, 119787–119803 (2019)
20. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
21. Verenich, I., Dumas, M., La Rosa, M., Nguyen, H.: Predicting process performance: a white-box approach based on process models. J. Softw. Evol. Process (2019)

22. Weinzierl, S., et al.: An empirical comparison of deep-neural-network architectures for next activity prediction using context-enriched process event logs. arXiv (2020)
23. Weinzierl, S., Zilker, S., Brunk, J., Revoredo, K., Matzner, M., Becker, J.: XNAP: making LSTM-based next activity predictions explainable by using LRP. In: Del Río Ortega, A., Leopold, H., Santoro, F.M. (eds.) BPM 2020. LNBIP, vol. 397, pp. 129–141. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66498-5_10