

NeSy4PPM: A Python Library for Neuro-Symbolic Predictive Process Monitoring

Jamila Oukharjane^{1,*}, Ivan Donadello¹ and Fabrizio Maria Maggi¹

¹Faculty of Engineering, Free University of Bozen-Bolzano, NOI Techpark - via Bruno Buozzi, 1, Bolzano, 39100, Italy

Abstract

NeSy4PPM is the first Python-based library for Predictive Process Monitoring (PPM) that integrates neural models with symbolic background knowledge to improve suffix prediction under specific contextual circumstances. It supports suffix prediction while ensuring compliance with various types of background knowledge, including DECLARE, MP-DECLARE, PROBDECLARE, and procedural models such as Petri nets and BPMN. In this paper, we present the functionalities of NeSy4PPM and empirically evaluate its performance in terms of prediction efficiency, compliance with the input background knowledge, and overall effectiveness.

Keywords

Predictive Process Monitoring, Deep Learning, Symbolic Background Knowledge, Python API

Metadata description	Value
Tool name	NeSy4PPM
Current version	0.1
Legal code license	-
Languages, tools and services used	Python 3.10
Supported operating environment	Microsoft Windows, GNU/Linux
Download/Demo URL	https://pypi.org/project/nesy4ppm/
Documentation URL	https://nesy4ppm.readthedocs.io/en/latest/
Source code repository	https://github.com/JamilaOUKHARIJANE/NeSy4PPM
Screencast video	https://youtu.be/ig4QQGxu49M

1. Introduction

Predictive Process Monitoring (PPM) has received growing attention in recent years as organizations increasingly aim to anticipate the future evolution of ongoing process instances using historical execution data [1]. One of the main tasks in this domain is suffix prediction, which aims to forecast the remaining sequence of activities until a trace is completed.

Although various tools and frameworks have been developed for suffix prediction, they primarily focus on stationary processes (i.e., processes that do not evolve over time) and rely mainly on neural models, lacking support for incorporating background knowledge (\mathcal{BK}), such as constraints or logical rules in the prediction task. *NeSy4PPM* is the first general-purpose Python library for Neuro-Symbolic PPM that supports both stationary and evolving processes, i.e., those that evolve due to concept drift. It addresses suffix prediction for both activity sequences and activity-resource pairs by combining neural models (e.g., LSTM, Transformer) with diverse types of \mathcal{BK} , including probabilistic, declarative, and procedural knowledge.

NeSy4PPM provides an end-to-end pipeline, from data preprocessing to prediction evaluation, and enables users to integrate symbolic \mathcal{BK} to enhance neural predictions at testing time and better manage process changes over time. The library is modular, extensible, and designed to support both single-attribute (activity) and multi-attribute (activity and resource) suffix prediction tasks.

ICPM Doctoral Consortium and Demo Track 2025, October 20-24, 2025, Montevideo, Uruguay

*Corresponding author.

✉ jamila.oukharjane@unibz.it (J. Oukharjane); ivan.donadello@unibz.it (I. Donadello); maggi@inf.unibz.it (F. M. Maggi)

id 0000-0001-6155-0215 (J. Oukharjane); 0000-0002-0701-5729 (I. Donadello); 0000-0002-9089-6896 (F. M. Maggi)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. The NeSy4PPM Library

Figure 1 presents an overview of the core functionalities of the NeSy4PPM library, which implements the general NeSy framework introduced in [2]. The library is organized into four main packages based on their functionalities: *Data Preprocessing*, *Training*, *Prediction*, and *Evaluation*. Below, we detail the functionality of each core package.

Before describing these packages in detail, we briefly recall some preliminary definitions. An event log is a set of traces, where each trace σ represents an execution of a business process. An event is a tuple (c, a, t, r) where c is the case id, $a \in \mathcal{A}$ is the activity name, t is the timestamp of the event, and $r \in \mathcal{R}$ is the allocated resource for activity a . We denote the components of an event $e = (c, a, t, r)$ using the notation $e.c$, $e.a$, $e.t$, and $e.r$. Given a trace $\sigma = \langle e_1, e_2, \dots, e_n \rangle$, the prefix $\sigma_{\leq k}$ of length $k \in \{1, \dots, n\}$ is the sub-trace including the first k events of σ , i.e., $\sigma_{\leq k} = \langle e_i \rangle_{i=1}^k$, with $|\sigma_{\leq k}| = k$. The corresponding suffix $\sigma_{>k}$ is the sub-trace obtained by removing the prefix, i.e., $\sigma_{>k} = \langle e_i \rangle_{i=k+1}^n$, with $|\sigma_{>k}| = n - k$.

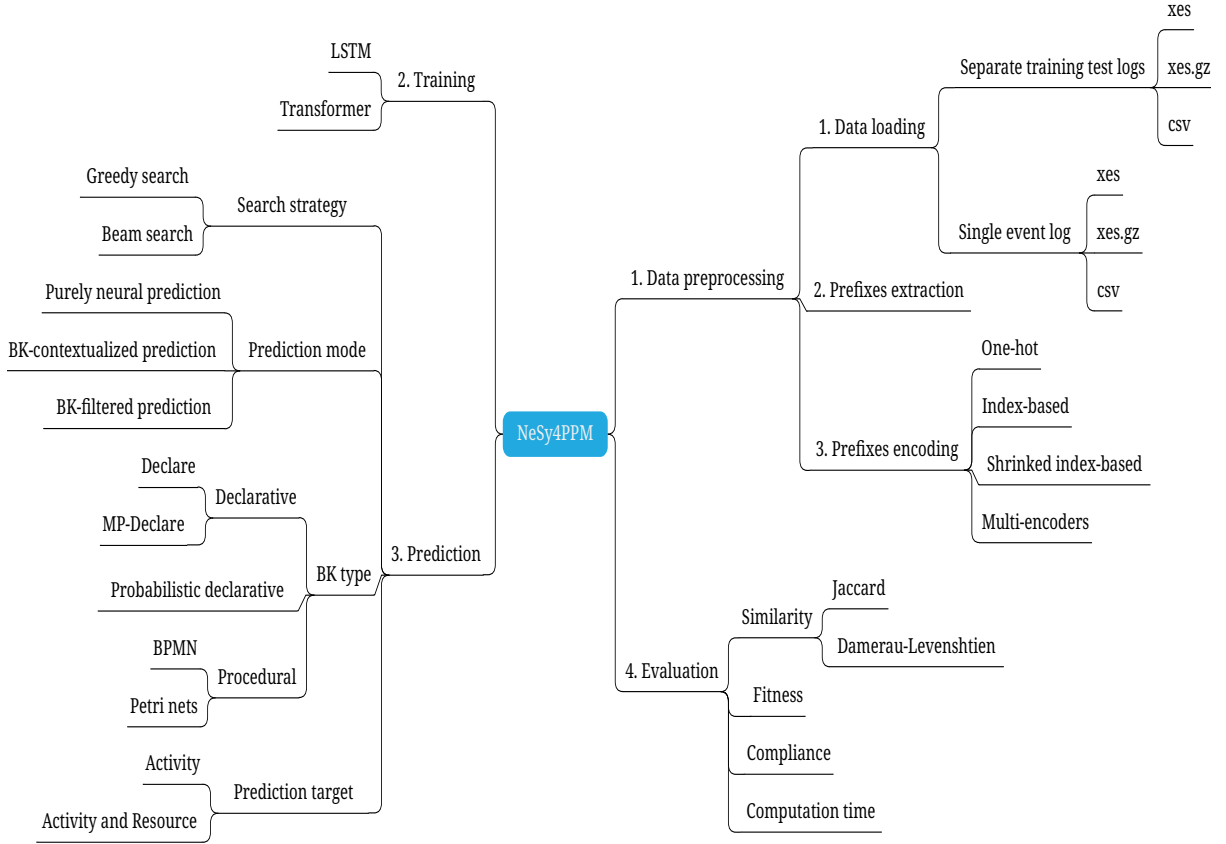


Figure 1: Overview of the NeSy4PPM library.

2.1. Data Preprocessing

The *Data Preprocessing* package transforms event logs into neural-compatible inputs. It handles event log loading, prefix extraction, and encoding into formats suitable for Neural Network (NN) models:

1. **Data loading.** This step supports two configurations: (i) a *single event log*, automatically split into training and test sets based on the chronological order of trace start timestamps and a user-defined train/test ratio; or (ii) a pair of *separate training and test logs*. Logs can be provided in `.xes`, `.xes.gz`, or `.csv` format.
2. **Prefix extraction.** All possible prefixes are extracted from the training log traces. Each prefix is turned into an input-label pair (x, y) for NN training. Given a trace $\sigma = \langle e_1, \dots, e_n \rangle$, this step generates training pairs by selecting prefixes $\langle e_i \rangle_{i=1}^j$ as input x , and the subsequent event e_{j+1} as

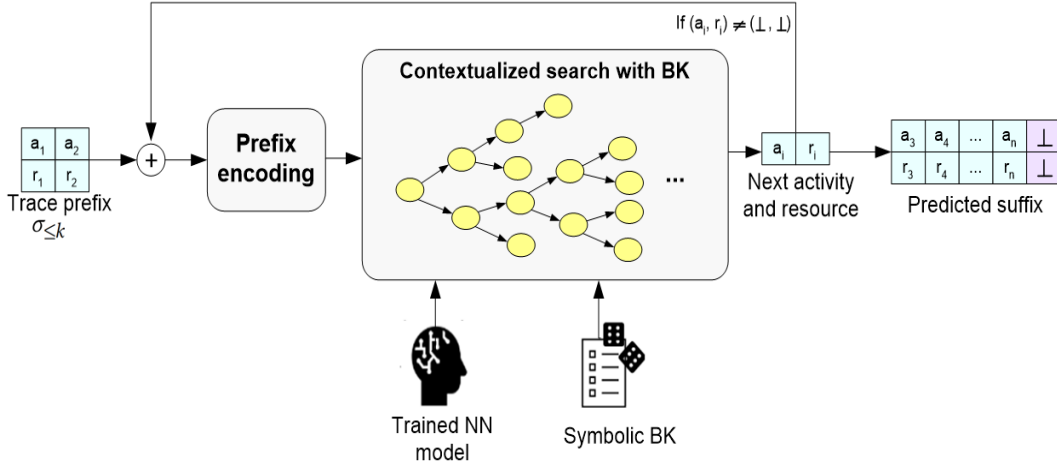


Figure 2: Overview of the Symbolic[Neuro] prediction component.

the target label y , for each $j = 1, \dots, n - 1$. Specifically, $e_{j+1}.a$ serves as the next activity label y_a , and $e_{j+1}.r$ as the next resource label y_r .

3. **Prefixes encoding.** Since NN inputs must be numerical, prefixes are transformed into feature vectors \mathbf{v} using one of the following methods: *one-hot*, *index-based*, *shrunk index-based*, or *multi-encoders*, which we describe in more detail below. Since vectors \mathbf{v} have variable lengths, zero-padding is applied to ensure fixed-size inputs for NN training.

In *one-hot* encoding, each event $e_j = \langle a_j, r_j \rangle$ is represented by a binary vector $\mathbf{v}_j = [\mathbf{v}_{a_j}, \mathbf{v}_{r_j}]$, where \mathbf{v}_{a_j} and \mathbf{v}_{r_j} are one-hot vectors for activity $a_j \in \mathcal{A}$ and resource $r_j \in \mathcal{R}$. The feature vector \mathbf{v} for prefix σ is a matrix of these row vectors.

In *index-based* encoding, each event e_j is represented as $\mathbf{v}_j = [idx_{a_j}, idx_{r_j}]$, where idx_{a_j} and idx_{r_j} are indices in \mathcal{A} and \mathcal{R} . The feature vector \mathbf{v} is the concatenation of the \mathbf{v}_j vectors. Both encodings can be adapted for activity-only sequences by omitting the resource vector.

In *shrunk index-based* encoding, a unique integer index is assigned to each activity-resource pair (a_j, r_j) using a joint index function idx_{AR} , resulting in $\mathbf{v}_j = idx_{AR}(a_j, r_j)$. The feature vector \mathbf{v} is then obtained by concatenating the \mathbf{v}_j values.

In *multi-encoders*, separate embeddings \mathbf{E}_a and \mathbf{E}_r are created for activities and resources. Combined embeddings are computed as $\mathbf{v} = \tilde{\mathbf{E}}_a \cdot b_a + \tilde{\mathbf{E}}_r \cdot b_r$, where $\tilde{\mathbf{E}}_a = \mathbf{E}_a \oplus (\mathbf{E}_a \otimes \mathbf{E}_r)$ and $\tilde{\mathbf{E}}_r = \mathbf{E}_r \oplus (\mathbf{E}_r \otimes \mathbf{E}_a)$, with b_a and b_r as alignment weights computed via a shared modulator [3].

2.2. Training

For training, NeSy4PPM implements state-of-the-art NN architectures, supporting both LSTM (Long Short-Term Memory) [4] and Transformer [3] models. This component is responsible for learning predictive models from the encoded input data x and corresponding target labels y . In its basic configuration, the selected NN model learns to predict the next activity \hat{y}_a . If the target labels y also include resource information y_r , the NN is trained in a multi-output setting to jointly predict the next activity \hat{y}_a and its associated resource \hat{y}_r .

2.3. Prediction

Prediction in NeSy4PPM is performed using an autoregressive Symbolic[Neuro] component according to Kautz's taxonomy [5, Section 2], where an NN model is invoked as a subroutine within a symbolic reasoner that checks the compliance of the NN predictions with \mathcal{BK} (see Figure 2). This component takes as input a prefix $\sigma_{\leq k}$, a trained NN model, and \mathcal{BK} .

The prediction process begins by encoding the input prefix $\sigma_{\leq k}$ using the same encoding method applied during training. The encoded prefix is then passed to the *search predictor*, which performs suffix prediction via beam search in an autoregressive manner. At each prediction step h , the predictor evaluates possible continuations based on: (i) the probability distribution over next activities and resources predicted by the *trained NN*, and (ii) a \mathcal{BK} compliance score. These are combined using a weight $w \in [0, 1]$ to balance prediction likelihood and compliance:

$$\text{score}_{\sigma_{\leq k+h}} = \mathcal{G}(\text{NN}(\sigma_{\leq k+h-1}, a), \text{NN}(\sigma_{\leq k+h-1}, r))^{1-w} \cdot \mathcal{C}(\langle \sigma_{\leq k+h-1}, a, r \rangle, \mathcal{BK})^w \quad (1)$$

Here, $\mathcal{G}(\text{NN}(\sigma_{\leq k+h-1}, a), \text{NN}(\sigma_{\leq k+h-1}, r))$ computes an aggregation (we implemented the average, but other strategies are possible) of the predicted probabilities for the next activity and resource, and the compliance function $\mathcal{C}(\langle \sigma_{\leq k+h-1}, a, r \rangle, \mathcal{BK})$ returns a score in $[0, 1]$, indicating how well the predicted continuation satisfies \mathcal{BK} . Our implementation supports several conformance checkers for \mathcal{C} , including: the MP-DECLARE conformance checker [6] for declarative \mathcal{BK} , the PROBDECLARE checker for probabilistic declarative \mathcal{BK} [7], and the procedural conformance checker [8] for procedural \mathcal{BK} such as Petri nets and BPMN. At each step, the selected activity and resource are appended to the prefix, and the search continues until a termination symbol (\perp) is predicted.

The search predictor is modular and configurable, supporting various settings across four dimensions. *Prediction modes* include purely neural prediction (with $w = 0$), BK-contextualized prediction ($w > 0$), and BK-filtered prediction, in which the full predicted trace is retained only if compliant with \mathcal{BK} upon termination. *Search strategies* include beam search for exploring multiple candidate continuations and greedy search for selecting only the most likely continuation. The predictor supports different \mathcal{BK} types as described above. Finally, this NeSy component supports various *prediction targets*, allowing for prediction of both activity and resource, or activity only. This flexible configuration enables adaptation to a wide range of predictive monitoring scenarios and knowledge models.

2.4. Evaluation

To evaluate the performance of neuro-symbolic predictions, the *Evaluation* package supports the following metrics: (i) the *Damerau-Levenshtein Similarity (DLS)* between the predicted suffix $\hat{\sigma}_{>k}$ and the ground truth suffix $\sigma_{>k}$; (ii) the *Jaccard similarity* between the predicted suffix $\hat{\sigma}_{>k}$ and the ground truth suffix $\sigma_{>k}$; (iii) the *Compliance*, defined as the proportion of predicted traces (i.e., the prefix concatenated with the predicted suffix) that satisfy \mathcal{BK} (i.e., DECLARE or MP-DECLARE constraints); (iv) the *Fitness*, representing the degree to which the predicted traces align with a procedural model; and (v) *Computation time*, referring to the average and standard deviation of prediction times.

3. Maturity

The maturity of NeSy4PPM is demonstrated by its support for a broad range of features, including: (i) multiple *encoding methods*; (ii) *neural architectures*, including LSTM and Transformer models; (iii) symbolic \mathcal{BK} representations, including declarative models (DECLARE, MP-DECLARE, PROBDECLARE) and procedural models (Petri nets and BPMN); (iv) *prediction search strategies*, such as greedy and beam search; (v) *prediction modes*, including purely neural, \mathcal{BK} -contextualized, and \mathcal{BK} -filtered predictions; and (vi) *prediction targets*, supporting both single-attribute (activity) and multi-attribute (activity and resource) prediction tasks. These features enable flexible and comprehensive evaluation across diverse experimental configurations.

In addition, we evaluated the computational performance and prediction accuracy of NeSy4PPM using two real-world event logs: Helpdesk¹ and Request For Payment², under concept drift assumptions. For each log, traces were chronologically sorted: the first 80% were used for training, and the remaining 20% for testing. To simulate concept drift, \mathcal{BK} was mined from under-represented behaviors in the

¹https://data.4tu.nl/articles/_/12675977/1

²https://data.4tu.nl/articles/_/12706886/1

Table 1

Performance results of the different configurations of NeSy4PPM. Best results in bold.

Method	Encoder	Helpdesk				Request For Payment				
		Time Avg \pm Std (s)	DLS(Acts)	DLS(Res)	Compliance	Time Avg \pm Std (s)	DLS(Acts)	DLS(Res)	Compliance	
Greedy search	GS	One-hot	0.202\pm0.077	0.812	0.157	0.0	0.23\pm0.108	0.882	0.883	0.498
		Index-based [3]	0.214 \pm 0.092	0.809	0.165	0.0	0.234 \pm 0.115	0.883	0.883	0.5
		Shrunked index-based	0.215 \pm 0.095	0.81	0.161	0.0	0.234 \pm 0.108	0.882	0.883	0.498
		Multi-encoders [3]	0.272 \pm 0.217	0.814	0.175	0.0	0.342 \pm 0.251	0.882	0.883	0.498
	GS + BK _{filter}	One-hot	0.206 \pm 0.08	0.812	0.157	0.0	0.228 \pm 0.074	0.882	0.883	0.498
		Index-based	0.222 \pm 0.111	0.809	0.165	0.0	0.223 \pm 0.093	0.883	0.883	0.5
		Shrunked index-based	0.23 \pm 0.13	0.81	0.161	0.0	0.235 \pm 0.089	0.882	0.883	0.498
		Multi-encoders	0.266 \pm 0.203	0.814	0.175	0.0	0.352 \pm 0.249	0.882	0.883	0.498
	GS + BK _{contextualized} (MP-DECLARE)	One-hot	2.593 \pm 0.789	0.817	0.647	0.998	2.267 \pm 0.649	0.993	0.994	0.998
		Index-based	2.862 \pm 1.728	0.798	0.637	0.972	2.264 \pm 0.548	0.995	0.996	1.0
		Shrunked index-based	2.679 \pm 0.951	0.809	0.644	0.994	2.298 \pm 0.55	0.995	0.995	1.0
		Multi-encoders	2.661 \pm 0.723	0.816	0.663	1.0	2.304 \pm 0.647	0.995	0.996	1.0
	BS	one-hot	0.496 \pm 0.158	0.816	0.159	0.0	0.482\pm0.164	0.882	0.883	0.498
		Index-based	0.538 \pm 0.237	0.81	0.164	0.0	0.495 \pm 0.151	0.883	0.883	0.5
		Shrunked index-based	0.552 \pm 0.279	0.815	0.161	0.0	0.509 \pm 0.174	0.882	0.883	0.498
		Multi-encoders	0.602 \pm 0.314	0.816	0.174	0.0	0.822 \pm 0.395	0.883	0.883	0.5
Beam search	BS + BK _{filter}	One-hot [9]	0.62 \pm 0.26	0.816	0.159	0.001	0.515 \pm 0.176	0.882	0.883	0.498
		Index-based	0.678 \pm 0.314	0.81	0.202	0.04	0.674 \pm 0.308	0.883	0.883	0.5
		Shrunked index-based	0.783 \pm 0.316	0.815	0.166	0.006	0.572 \pm 0.223	0.882	0.883	0.498
		Multi-encoders	0.791 \pm 0.443	0.815	0.179	0.009	0.735 \pm 0.396	0.883	0.883	0.5
	BS + BK _{contextualized} (MP-DECLARE)	One-hot	11.669 \pm 3.261	0.831	0.658	1.0	4.782 \pm 1.618	0.995	0.996	1.0
		Index-based	12.417 \pm 4.35	0.825	0.662	0.999	5.861 \pm 1.678	0.995	0.996	1.0
		Shrunked index-based	12.565 \pm 4.225	0.827	0.66	1.0	4.921 \pm 1.751	0.995	0.996	1.0
		Multi-encoders	12.59 \pm 3.994	0.829	0.672	1.0	5.811 \pm 1.758	0.995	0.996	1.0

training set using the *DeclareMiner*³ tool, selecting constraints with less than 20% support. We extracted two MP-DECLARE constraints for the Helpdesk log and eight for the Request For Payment log. We then retained only those traces in the test sets that were compliant with the respective constraints. This resulted in MP-DECLARE-compliant test sets containing 820 traces for Helpdesk and 445 traces for Request For Payment.

We performed experiments using different prediction configurations, including Greedy Search (GS) and Beam Search (BS) with a beam size of $b_{size} = 3$. For each strategy, we evaluated: (i) purely neural predictions, (ii) filtered predictions (BK_{filter}), and (iii) contextualized predictions ($BK_{contextualized}$). All experiments used a Transformer architecture as the NN model.

Table 1 reports the average and standard deviation of prediction times (in seconds), as well as prediction accuracy using the DLS and compliance metrics for both activity and resource prediction, across all combinations of prediction methods – GS [3], GS + BK_{filter} , GS + $BK_{contextualized}$, BS, BS + BK_{filter} [9], and BS + $BK_{contextualized}$ – and all considered encoders.

From the table, we observe that while the use of $BK_{contextualized}$ in combination with greedy or beam search introduces additional computational overhead due to \mathcal{BK} conformance checking, it leads to a clear improvement in prediction accuracy. These results highlight the effectiveness of integrating symbolic \mathcal{BK} into the prediction algorithm. Notably, BS + $BK_{contextualized}$ achieves the best DLS scores for both activity and resource prediction, as well as the highest compliance on both logs. We acknowledge that this comes at the cost of increased computational time.

In contrast, the purely neural predictions methods (GS and BS) struggle with resource prediction, confirming that NN models alone fail to capture drifts in resource allocation. Although GS + BK_{filter} and BS + BK_{filter} methods apply symbolic \mathcal{BK} as a post-prediction filter, their overall performance remains similar to the purely neural predictions methods in terms of both prediction accuracy and compliance. This limited improvement can be attributed to the restricted beam size: when activity-resource pairs representing drift are rare or unseen during training, the NN model assigns them low probabilities. As a result, these pairs are often excluded from the beam due to the prioritization of higher-probability candidates.

As future work, we plan to implement multi-threading to parallelize the beam search to reduce the computational time. Furthermore, we aim to integrate fuzzy LTL_f compliance checking to support reasoning under uncertainty [10], incorporate \mathcal{BK} into the training phase via the loss function [11], and include explanation techniques to enhance interpretability.

³<https://rulemining.org>

Acknowledgments

This study was partially funded by the European Union - NextGenerationEU, in the framework of the iNEST - Interconnected Nord-Est Innovation Ecosystem (iNEST ECS00000043 – CUP I43C22000250006). The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union, nor can the European Union be held responsible for them. The study was also supported by Fondazione Cariverona within the ReSS-Pro project.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] P. Ceravolo, M. Comuzzi, J. De Weerd, C. Di Francescomarino, F. M. Maggi, Predictive process monitoring: concepts, challenges, and future research directions, *Process Science* 1 (2024) 2.
- [2] J. Oukharjane, I. Donadello, F. M. Maggi, A general framework for neuro-symbolic predictive process monitoring, in: *Business Process Management Workshops, Lecture Notes in Business Information Processing*, 2025.
- [3] G. R. Lazo, R. Nanculef, Multi-attribute transformers for sequence prediction in business process management, in: *DS*, volume 13601, 2022, pp. 184–194.
- [4] N. Tax, I. Verenich, M. L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: *CAiSE*, volume 10253 of *Lecture Notes in Computer Science*, Springer, 2017.
- [5] M. K. Sarker, L. Zhou, A. Eberhart, P. Hitzler, Neuro-symbolic artificial intelligence: Current trends, *AI Communications* 34 (2022) 197–209.
- [6] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, *Expert Syst. Appl.* 65 (2016) 194–211.
- [7] A. Alman, F. M. Maggi, M. Montali, R. Peñaloza, Probabilistic declarative process mining, *Inf. Syst.* 109 (2022) 102033.
- [8] A. Berti, W. M. P. van der Aalst, Reviving token-based replay: Increasing speed while improving diagnostics, in: *ATAED@Petri Nets/ACSD*, 2019, pp. 87–103.
- [9] C. Di Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, A. Yeshchenko, An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring, in: *BPM*, Springer, 2017, pp. 252–268.
- [10] I. Donadello, P. Felli, C. Innes, F. M. Maggi, M. Montali, Conformance checking of fuzzy logs against declarative temporal specifications, in: *BPM*, volume 14940, 2024, pp. 39–56.
- [11] E. Umili, G. P. Licks, F. Patrizi, Enhancing deep sequence generation with logical temporal knowledge, in: *PMAl@ECAI*, volume 3779 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024.