Graphical Abstract

Neuro-Symbolic Predictive Process Monitoring

Axel Mezini, Elena Umili, Ivan Donadello, Fabrizio Maria Maggi, Matteo Mancanelli, Fabio Patrizi

Highlights

Neuro-Symbolic Predictive Process Monitoring

Axel Mezini, Elena Umili, Ivan Donadello, Fabrizio Maria Maggi, Matteo Mancanelli, Fabio Patrizi

- Integration of linear temporal logic in neural training: two automata-based logical losses.
- Better logical constraints satisfaction of the automata-based logical losses.
- Important step towards green AI: an automata-based logical loss drastically reduces the training epochs.

Neuro-Symbolic Predictive Process Monitoring

Axel Mezini[a,1], Elena Umili[b,1], Ivan Donadello[a], Fabrizio Maria Maggi[a], Matteo Mancanelli[b], Fabio Patrizi[b]

[a] *Faculty of Engineering, Free University of Bozen-Bolzano, NOI Techpark - via Bruno Buozzi, 1, Bolzano, 39100, Italy*
[b] *Department of Computer, Control and Management Engineering, Sapienza, Università di Roma, Via Ariosto, 25, Roma, 00185, Italy*

Abstract

This paper addresses the problem of suffix prediction in Business Process Management (BPM) by proposing a Neuro-Symbolic Predictive Process Monitoring (PPM) approach that integrates data-driven learning with temporal logic-based prior knowledge. While recent approaches leverage deep learning models for suffix prediction, they often fail to satisfy even basic logical constraints due to the lack of explicit integration of domain knowledge during training. We propose a novel method to incorporate Linear Temporal Logic over finite traces (LTL_f) into the training process of autoregressive sequence predictors. Our approach introduces a differentiable logical loss function, defined using a soft approximation of LTL_f semantics and the Gumbel-Softmax trick, which can be combined with standard predictive losses. This ensures that the model learns to generate suffixes that are both accurate and logically consistent. Experimental evaluation on three real-world datasets shows that our method improves suffix prediction accuracy and compliance with temporal constraints. We also introduce two variants of the logic loss (local and global) and demonstrate their effectiveness under noisy and realistic settings. While developed in the context of BPM, our framework is applicable to any symbolic sequence generation task and contributes to advancing Neuro-Symbolic AI.

Keywords: Suffix prediction, Neuro-Symbolic AI, Deep learning with logical knowledge, Linear Temporal Logic, Differentiable automata
2000 MSC: 68T05, 68T37

[1] Equal contribution

1. Introduction

Predictive Process Monitoring (PPM) [1] is a field within Business Process Management (BPM) that focuses on forecasting future events or outcomes of ongoing process instances based on historical event data. This paper addresses the PPM problem of suffix prediction by leveraging both data and prior logical temporal knowledge. Suffix prediction is particularly important in BPM for forecasting the continuation of a process trace, enabling better resource allocation, and anticipating future steps for effective decision-making.

Recently, there has been significant interest in employing deep learning techniques for suffix prediction in BPM [2], including the use of Recurrent Neural Networks (RNNs) [3], Transformers [4], and Deep Reinforcement Learning algorithms [5]. Despite these advances, several studies have shown that deep models can surprisingly fail to satisfy even the most basic logical constraints [6, 7], derived from commonsense reasoning or domain-specific knowledge. This occurs because such models are trained solely on data, and integrating logical knowledge into the training process remains an open challenge. As a result, in domains like BPM, where both data and formal knowledge about the process are often available, the latter is typically underutilized.

However, domain-specific knowledge can play a crucial role in characterizing the context in which a process is executed, providing valuable information not explicitly contained in the event data alone. For example, when the particular variant of a process being executed is known (such as a client-specific scenario or a seasonally influenced workflow), this knowledge can guide predictions in ways that purely data-driven models cannot. Similarly, in environments subject to concept drift, where process behavior evolves over time, logical constraints can help identify and adapt to changes more robustly than relying solely on historical data. Furthermore, logical knowledge is instrumental in filtering out noise in the data [8], enhancing model robustness and prediction accuracy, as demonstrated in our experimental evaluation.

This work explores a novel Neuro-Symbolic PPM approach aimed at bridging this gap by incorporating prior knowledge, expressed in Linear Temporal Logic over finite traces (LTL_f), into the training of a deep generative model for suffix prediction. Existing works on knowledge-constrained sequence generation using autoregressive models are typically designed for

test-time inference [9, 10, 11, 12, 13, 14, 15, 16, 17], rather than for integration during training. A notable exception is STLnet [7], which incorporates Signal Temporal Logic (STL) specifications into the training process through a student-teacher framework. However, STLnet is specifically designed for continuous-time sequences and STL constraints. Since STL generalizes LTL, we attempted to adapt STLnet to our symbolic domain by first translating our LTL_f formulas into STL and then applying STLnet. In practice, however, we found that the system was unable to construct the corresponding formula model without exceeding memory limits.[2] This highlights the limitations of STLnet in symbolic settings such as those encountered in PPM.

Our contribution is a Neuro-Symbolic method to integrate symbolic knowledge of certain process properties, expressed in Linear Temporal Logic over finite traces (LTL_f), into the training process of a neural sequence predictor. This enables us to leverage both sources of information (data and background LTL_f knowledge) during training. Specifically, we achieve this by defining a differentiable counterpart of the LTL_f knowledge and employing a differentiable sampling technique known as the Gumbel-Softmax trick [18]. By combining these two elements, we define a logical loss function that can be used alongside any other loss function employed by the predictor. This ensures that the network learns to generate traces that are both similar to those in the training dataset and compliant with the given temporal specifications *at the same time*.

We evaluate our method on several real-world BPM datasets and demonstrate that incorporating LTL_f knowledge at training time leads to predicted suffixes with both lower Damerau-Levenshtein (DL) distance [19] from the target suffixes and a significantly higher rate of satisfaction of the LTL_f constraints. This paper builds upon our previous work [20], extending the approach to make it more principled, robust, and practically applicable in real-world scenarios. The main differences between the present work and [20] are:

- We provide a more rigorous formalization of the problem of autore-

[2]This is likely due to the fact that, in STL, temporal operators are applied over dense-time intervals (e.g., $G_{[1,2]}$ means "globally over all times in the interval $[1, 2]$"), which are typically relatively short. In contrast, in LTL_f each operator ranges over the entire length of the trace. As a result, STL formulas obtained from LTL_f translations involve much larger temporal intervals, substantially complicating the application of the STLnet framework.

gressive sequence generation under logical temporal constraints in Section 3.1, which is entirely new. In the previous version, the problem setting and objectives were presented in a more concise and intuitive manner.

- Building on this formalization, we offer a more exhaustive analysis of the differences in *monitorability* [21] across various types of temporal constraints when related to autoregressive sequence generation. This analysis leads us to introduce the notions of *local* and *global* guidance, described in Sections 3.2 and 3.3, respectively.

- Based on these two concepts, we formulate *two* logical loss functions to guide the model toward satisfying the specification: one providing local, activity-level feedback, and another enforcing global, trace-level logical constraints during suffix prediction. The Local Logic Loss is entirely new, while the Global Logic Loss builds upon the loss proposed in [20] by introducing a Monte Carlo approximation phase. This addition both stabilizes learning and aligns the practical implementation of the loss with its theoretical formulation given in Section 3.1.

- We describe in more detail how we preprocess LTL_f knowledge to handle the sequence termination symbol (Section 3.5.1), and how we transition from the automaton representation to its neural counterpart, DeepDFA [22], which is used for loss computation (Section 3.5.2). Both of these components are new with respect to [20].

- Importantly, we substantially broaden the empirical evaluation of our framework, demonstrating both the effectiveness and generality of our approach. While [20] reported experiments only on synthetic datasets, here we present a comprehensive set of results on *real-world* BPM datasets under challenging and *noisy* experimental conditions. Consequently, Section 4 is also entirely new.

- Finally, we provide a more detailed discussion of the potential applications of our framework beyond PPM, as well as the limitations that may arise in such broader settings, together with possible mitigation strategies in Section 3.7, which is also new.

Overall, while our approach is instantiated in the context of PPM, its underlying principles are broadly applicable to any multi-step symbolic se-

quence generation task using autoregression. We believe the contributions of this work may be of general interest to the Machine Learning community, particularly in the area of Neuro-Symbolic AI.

The rest of the paper is structured as follows. Section 2 provides preliminary notions and notations necessary to understand the paper. Section 3 formulates the addressed problem and outlines the proposed solution, together with possible extensions and limitations. Section 4 presents an experimental evaluation of the solution. Section 5 discusses related work, while Section 6 concludes the paper and spells out directions for future research.

2. Background and Notation

This section introduces the fundamental notions and notations essential for understanding the concepts discussed in the paper. It lays the groundwork by defining key terms and formalizing the terminology used throughout the work.

2.1. Notation

In this work, we consider *sequential* data of various types, including both symbolic and subsymbolic representations. Symbolic sequences are also called *traces*. Each element in a trace is a symbol σ drawn from a finite alphabet Σ. We denote sequences using bold notation. For example, $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_T)$ represents a trace of length T.

Each symbolic variable in the sequence can be grounded either categorically or probabilistically. In the case of categorical grounding, each element of the trace is assigned a symbol from Σ, denoted simply as σ_i, where σ_i can be encoded as an index in $\{1, 2, \ldots, |\Sigma|\}$ or as a *one-hot vector* $\sigma_i \in \{0, 1\}^{|\Sigma|}$ such that $\sum_{j=1}^{|\Sigma|} \sigma_i[j] = 1$. In the case of probabilistic grounding, each symbolic variable is associated with a probability distribution over Σ, represented as a vector $\tilde{\sigma}_i \in \Delta(\Sigma)$, where $\Delta(\Sigma)$ denotes the probability simplex defined as:

$$\Delta(\Sigma) = \left\{ \tilde{\sigma} \in \mathbb{R}^{|\Sigma|} \;\middle|\; \tilde{\sigma}[j] \geq 0,\ \sum_{j=1}^{|\Sigma|} \tilde{\sigma}[j] = 1 \right\}.$$

Accordingly, we distinguish between categorically grounded sequences $\boldsymbol{\sigma}$ and probabilistically grounded sequences $\tilde{\boldsymbol{\sigma}}$, using the tilde notation.

Finally, we use subscripts to indicate time steps in the sequence, and $v[j]$ ($M[j]$) to denote the j-th component (tensor) of vector v (matrix M). For instance, $\tilde{\sigma}_i[j]$ denotes the j-th component of the probabilistic grounding of σ at time step i. We also use $+$ to denote trace concatenation; e.g., $\boldsymbol{a} + \boldsymbol{b}$ is the trace obtained by concatenating traces $\boldsymbol{a}$ and $\boldsymbol{b}$.

2.2. Linear Temporal Logic and Deterministic Finite Automata

Linear Temporal Logic (LTL) [23] is a formal language that extends traditional propositional logic with modal operators, allowing the specification of rules that must hold *through time*. In this work, we use LTL interpreted over finite traces (LTL_f) [24], which models finite, but length-unbounded, process executions, making it suitable for finite-horizon problems.

Given a finite set Σ of atomic propositions, the set of LTL_f formulas ϕ is inductively defined as follows:

$$\phi ::= \top \mid \bot \mid \sigma \mid \neg\phi \mid \phi \wedge \phi \mid X\phi \mid \phi U \phi, \tag{1}$$

where $\sigma \in \Sigma$. We use $\top$ and $\bot$ to denote $True$ and $False$, respectively. X (Strong Next) and U (Until) are temporal operators. Other temporal operators are N (Weak Next) and R (Release), defined as $N\phi \equiv \neg X \neg \phi$ and $\phi_1 R \phi_2 \equiv \neg(\neg\phi_1 U \neg\phi_2)$; G (Globally) $G\phi \equiv \bot R \phi$; and F (Eventually) $F\phi \equiv \top U \phi$.

A trace $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_T)$ is a sequence of propositional assignments to the propositions in Σ, where $\sigma_t \subseteq \Sigma$ is the set of all and only propositions that are true at instant t. Additionally, $|\boldsymbol{\sigma}| = T$ denotes the length of the trace. Since every trace is finite, $|\boldsymbol{\sigma}| < \infty$. If the propositional symbols in Σ are all *mutually exclusive*, i.e., the domain produces exactly one symbol true at each step, then we have $\sigma_t \in \Sigma$. As is customary in BPM, we make this assumption, known as the *mutual exclusivity assumption* [25]. By $\boldsymbol{\sigma} \vDash \phi$ we denote that the trace $\boldsymbol{\sigma}$ satisfies the LTL_f formula ϕ. We refer the reader to [24] for a formal description of the LTL_f semantics.

Any LTL_f formula ϕ can be translated into a Deterministic Finite Automaton (DFA) [24] $A_\phi = (\Sigma, Q, q_0, \delta, F)$, where Σ is the automaton alphabet,[3] Q is the finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$

[3]Here the alphabet of the equivalent DFA is Σ because of the mutual exclusivity assumption. In the general case, the automaton alphabet is 2^Σ.

is the transition function, and $F \subseteq Q$ is the set of final states. Additionally, we recursively define the extended transition function over traces $\delta^* : Q \times \Sigma^* \to Q$ as:

$$\begin{aligned} &\delta^*(q, \epsilon) = q \\ &\delta^*(q, \sigma + \boldsymbol{x}) = \delta^*(\delta(q, \sigma), \boldsymbol{x}), \end{aligned} \tag{2}$$

where $\sigma \in \Sigma$ is a symbol and $\boldsymbol{x} \in \Sigma^*$ is a trace. The automaton accepts the trace $\boldsymbol{\sigma}$ if $\delta^*(q_0, \boldsymbol{\sigma}) \in F$, and in that case we say that $\boldsymbol{\sigma}$ belongs to the language of the automaton, denoted as $L(A_\phi)$. We have that ϕ and A_ϕ are equivalent because, for any trace $\boldsymbol{\sigma} \in \Sigma^*$:

$$\boldsymbol{\sigma} \in L(A_\phi) \iff \boldsymbol{\sigma} \vDash \phi \tag{3}$$

Running Example. We consider a domain with three possible activities $\mathcal{A} = \Sigma = \{a, b, c\}$. For this domain, we are given both a set of traces describing example process executions and a constraint stating that activity a must always be eventually followed by activity b. This constraint corresponds to the formula $\phi = G(a \to F\, b)$, which is equivalent to the *Response* constraint in Declare. The corresponding DFA, defined over the alphabet $\Sigma = \{a, b\}$, is shown in Figure 2 (Original DFA).

2.3. Deep Autoregressive Models and Suffix Prediction

Deep autoregressive models are a class of deep learning models that automatically predict the next component in a sequence by using the previous elements in the sequence as inputs. These models can be applied to both continuous and categorical (symbolic) data, finding applications in various generative AI tasks such as Natural Language Processing (NLP) and Large Language Models (LLM) [26, 27], image synthesis [28, 29], and time-series prediction [30]. They encompass deep architectures such as RNNs and Transformers and, in general, any neural model capable of estimating the probability of a token given the preceding elements:

$$P(x_t \mid x_1, x_2, \ldots, x_{t-1}) = P(x_t \mid \boldsymbol{x_{<t}}). \tag{4}$$

The probability of a sequence of data $\boldsymbol{x}$ can be calculated as:

$$P(\boldsymbol{x}) = \prod_{i=1}^{T} P(x_i \mid \boldsymbol{x_{<i}}). \tag{5}$$

In suffix prediction in BPM, given a subsequence (or prefix) of *past activities* $\boldsymbol{p_t} = (a_1, a_2, \ldots, a_t)$ that the process has produced up to the current time step t, with a_i in a finite set of activities $\mathcal{A}$, we aim to complete the trace by generating the sequence of future events, also called the *suffix*, $\boldsymbol{s_t} = (a_{t+1}, \ldots, a_{t+l}, \texttt{EOT})$. We use $\texttt{EOT} \notin \mathcal{A}$ to denote a special symbol marking the end of sequences, which is not included in $\mathcal{A}$.

Suffix prediction can be accomplished using autoregressive models, by choosing at each step the most probable next event according to the neural network, concatenating it with the prefix, and continuing to predict the next event in this manner until the EOT symbol is predicted or the trace has reached a maximum number of steps T. We define $\mathcal{A}^{\leq T}\texttt{EOT} \equiv \{\boldsymbol{a} + \texttt{EOT} \mid \boldsymbol{a} \in \mathcal{A}^{\leq T}\}$ as the set of possible complete traces $\boldsymbol{a} = \boldsymbol{p_t} + \boldsymbol{s_t}$ that can be generated in this way.

At each generation step, a symbol must be *sampled* from the next activity probability. A common way of selecting the next activity to feed into the autoregressor is to *greedily* choose the activity maximizing the next symbol probability at each step, as follows:

$$a_k = \underset{a \in \mathcal{A} \cup \{\texttt{EOT}\}}{\operatorname{argmax}} P(a_t = \sigma \mid a_1, \ldots, a_t, a_{t+1}, \ldots, a_{k-1}), \quad t < k \leq T. \tag{6}$$

This greedy search strategy may not produce the *most probable suffix*, i.e., the trace that maximizes the probability in Equation 5. Other non-optimal sampling strategies commonly used for this task include Beam Search, Random Sampling, and Temperature Sampling [5].

3. Method

This section defines the specific suffix prediction problem we aim to solve and outlines our Neuro-Symbolic PPM approach, which integrates prior knowledge expressed in Linear Temporal Logic over finite traces (LTL_f) into the training of a deep generative model. We also introduce two logic loss formulations: one providing *local*, activity-level feedback, and another enforcing *global*, trace-level constraints over the predicted suffix.

3.1. Problem Formulation

We assume an autoregressive neural model f_θ with trainable parameters θ that estimates an approximation P_θ of the probability of the next event a_t

given a trace of previous events $\boldsymbol{a}_{<t}$ (Equation 4):

$$\begin{aligned} &\tilde{y}_t = f_\theta(\boldsymbol{a}_{<t}) \\ &P(a_t = a_i \in \mathcal{A} \cup \{\texttt{EOT}\} \mid \boldsymbol{a}_{<t}) \approx \tilde{y}_t[i]. \end{aligned} \tag{7}$$

Note that we do not make any assumptions about the neural model, except that it can estimate the probability of the next activity given a sequence of previous ones. As a result, our approach is entirely *model-agnostic* and can be readily applied to any autoregressive model.

We denote by P_θ the probability of a trace according to the network approximation:

$$P_\theta(\boldsymbol{a}) = \prod_{t=1}^{|\boldsymbol{a}|} \tilde{y}_t[a_t]. \tag{8}$$

The model parameters are typically trained using a supervised loss $L_{\mathcal{D}}$, evaluated on a dataset $\mathcal{D}$ of ground-truth traces obtained by observing the process. The loss for a trace $\boldsymbol{a} \in \mathcal{D}$ of length T is defined as follows:

$$L_{\mathcal{D}}(\boldsymbol{a}) = \frac{1}{T} \sum_{t=1}^{T} \text{cross-entropy}(f_\theta(\boldsymbol{a}_{<t}), a_t). \tag{9}$$

This loss trains the network to predict the next symbol in a trace so as to closely mimic the data in the dataset.

In this work, we assume that certain properties of the process are also known and can be injected into the learning process during training. These properties, which form the *background* (or *prior*) knowledge about the process, are expressed as an LTL_f formula ϕ defined over an alphabet $\Sigma \subseteq \mathcal{A}$.

Our goal is for the language generated by the autoregressor f_θ to be strictly contained within the language of strings accepted by the formula, denoted as $L(A_\phi)$. However, the language produced by the network is *unbounded*, as it is only *softly assigned*: in other words, the network can generate any possible string, each with a different probability.

Our method therefore aims to maximize the probability $P_{\theta \vDash \phi}$ that traces $\boldsymbol{a} \sim P_\theta$, sampled from the autoregressor, satisfy the specification:

$$P_{\theta \vDash \phi} = \mathbb{E}_{\boldsymbol{a} \sim P_\theta}[\boldsymbol{a} \vDash \phi] = \sum_{\boldsymbol{a} \in \mathcal{A}^{\leq T}\texttt{EOT}} P_\theta(\boldsymbol{a})\, \mathbb{1}\{\boldsymbol{a} \vDash \phi\}. \tag{10}$$

Note that, in order to compute the exact probability of knowledge satisfaction, one would need to enumerate *all possible suffixes* of maximum length

T and sum their acceptance probabilities. However, this set has exponential size, making exact computation unfeasible. Furthermore, we aim to impose the maximization of this probability as a *training objective*. Therefore, our goal is to design a fast and differentiable procedure to approximate it at each optimization step of the autoregressor.

We propose two logic loss functions: a local, activity-level loss L_ϕ^{loc}, and a global, trace-level loss L_ϕ^{glob}. We show that both contribute positively to the learning process, demonstrating the effectiveness of integrating LTL_f knowledge into autoregressive training. The choice between the two logic losses depends on the type of LTL_f knowledge available. The global loss L_ϕ^{glob} can always be applied, regardless of the structure of the formula ϕ, but it is more computationally demanding, as it requires generating entire suffixes during training.

In general, the compliance of a trace with ϕ can only be assessed on complete traces. Indeed, whether the current partial prediction $\boldsymbol{a}^{\leq t}$ satisfies (or violates) ϕ does not guarantee that the final predicted trace $\boldsymbol{a}$ will also satisfy (or violate) it. This introduces a challenge in evaluating LTL_f constraints, as opposed to approaches that rely on local constraints [16], which can be verified step-by-step during generation.

However, some types of formulas, such as safety constraints, once translated into a DFA, contain *failing states*, which can be used to guide generation locally. A failing state is a state in the DFA from which no accepting state is reachable. When a partial trace enters such a state, it has irreversibly violated the formula, and no continuation can satisfy the knowledge. Our local loss L_ϕ^{loc} exploits the presence of failing states to guide the autoregressor at every generation step. As a result, it provides richer feedback at the activity level, but it can only be used with formulas that admit such a DFA structure. Nevertheless, this limitation is overcome during the DFA preprocessing phase, where a failing state is always added to handle the `EOT` symbol, as discussed in Section 3.5.1.

In the following sections, we describe how we compute the logic loss under the two scenarios. In either case, we combine it with the supervised loss $L_{\mathcal{D}}$ as follows:

$$L = \alpha L_{\mathcal{D}} + (1 - \alpha) L_\phi, \tag{11}$$

with α being a constant between 0 and 1 that balances the influence of each loss on the training process.

3.2. Local Guidance

In cases where the formula can be permanently violated, the corresponding DFA includes a set of failing states $Q^{\text{fail}} \subseteq Q$, from which the formula can no longer be satisfied. In this case, we aim to minimize the *probability that the next predicted activity will irreversibly violate prior knowledge*, denoted as $P(\boldsymbol{a}_t \not\models^{\times} \phi \mid \boldsymbol{a}_{<t})$. Here, the symbol $\not\models^{\times}$ denotes the permanent violation of the formula. Given a ground-truth trace $\boldsymbol{a} \in \mathcal{D}$, we define this probability as:

$$P(\boldsymbol{a}_t \not\models^{\times} \phi \mid \boldsymbol{a}_{<t}) = \sum_{a \in \mathcal{A} \cup \{\texttt{EOT}\}} f_\theta(\boldsymbol{a}_{<t})[a] \cdot \mathbb{1}\left\{\delta^*(q_0, \boldsymbol{a}_{<t} + a) \in Q^{\text{fail}}\right\}. \quad (12)$$

In other words, at each step t of the trace, we consider the probability distribution over the next symbol given by $f_\theta(\boldsymbol{a}_{<t})$. For each symbol a in the vocabulary (including EOT), we simulate the state reached by the extended trace $\boldsymbol{a}_{<t} + a$ in the DFA. If the resulting state belongs to Q^{fail}, we add the probability of a to the probability that the trace permanently violates the knowledge. We minimize this probability by minimizing the following loss, hereafter called the *Local Logic Loss* (LLL), on each ground-truth trace $\boldsymbol{a}$:

$$L_\phi^{\text{loc}} = \frac{1}{|\boldsymbol{a}|} \sum_{t=1}^{|\boldsymbol{a}|} -\log\left(1 - P(\boldsymbol{a}_t \not\models^{\times} \phi \mid \boldsymbol{a}_{<t})\right). \quad (13)$$

Note that traces can fall into one of the following categories: (i) they irreversibly violate the knowledge (by reaching a failing state either at termination or earlier); (ii) they do not satisfy the knowledge (by terminating in a non-accepting, non-failing state); (iii) they satisfy the knowledge (by terminating in an accepting state).

Both cases (i) and (ii) result in a violation of the knowledge. The loss L_ϕ^{loc} specifically targets only case (i), aiming to reduce occurrences of irreversible violations. By decreasing the probability in Equation 12, we increase the desired probability $P_{\theta \models \phi}$. However, knowledge satisfaction is not directly maximized, and it is still possible for traces with zero loss to violate the knowledge under case (ii). Despite this limitation, we observed in practice that L_ϕ^{loc} remains highly effective on many datasets, as we will show in Section 4.

3.3. Global Guidance

In the general case, an LTL_f formula ϕ cannot always be permanently violated. In such cases, it is impossible to supervise the suffix generation step-by-step, and the only guidance that can be provided to the autoregressor is *global*, i.e., applied to the entire generated trace. To this end, we define a second logic loss (hereafter called the *Global Logic Loss* (GLL)), L_ϕ^{glob}, which provides this global supervision to the autoregressor and can be applied to *any* LTL_f formula.

In particular, in the global case, we directly approximate the target probability $P_{\theta \vDash \phi}$ using a Monte Carlo estimation. We sample a set of complete traces $\{\boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}, \ldots, \boldsymbol{a}^{(N)}\} \sim P_\theta$ according to the distribution learned by the autoregressor, and compute an approximation of the target probability $\hat{P}_{\theta \vDash \phi}$ as the empirical average compliance with the formula over the sampled set:

$$\hat{P}_{\theta \vDash \phi} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{\boldsymbol{a}^{(i)} \vDash \phi\}. \tag{14}$$

However, the sampled traces $\{\boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}, \ldots, \boldsymbol{a}^{(N)}\}$ used for the Monte Carlo estimation are generated by a neural network that returns a probability distribution over activity names at each time step. Therefore, a crisp trace $\boldsymbol{a}^{(i)}$ is replaced by its probabilistic counterpart $\tilde{\boldsymbol{a}}^{(i)}$, where activity symbols are sampled from the probability distribution computed by the neural network. Moreover, the indicator function $\mathbb{1}$ in Equation 14 is replaced by a method that computes the (probabilistic) compliance of a probabilistic trace with the knowledge in a fast and differentiable way. To achieve this, our method relies on two key components:

1. Gumbel-Softmax sampling [18, 31] to generate differentiable, near one-hot suffixes during training. A probabilistic trace $\tilde{\boldsymbol{a}}^{(i)}$ is therefore obtained by concatenating an input prefix with the sampled suffix.

2. DeepDFA [22], a Neuro-Symbolic framework that encodes temporal logic properties as a recurrent layer, enabling efficient and differentiable evaluation of logical constraints. This allows us to compute the probabilistic compliance $P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(i)} \vDash \phi)$ of a sampled trace $\tilde{\boldsymbol{a}}^{(i)}$ with the knowledge ϕ. Therefore, Equation 14 becomes:

$$\hat{P}_{\theta \vDash \phi} = \frac{1}{N} \sum_{i=1}^{N} P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(i)} \vDash \phi). \tag{15}$$

By leveraging these two components, detailed in the following sections, we compute the global logic loss L_ϕ^{glob}, which enforces the satisfaction of prior knowledge over entire traces:

$$L_\phi^{\text{glob}} = -\log(\hat{P}_{\theta \vDash \phi}). \tag{16}$$

In the next sections, we first discuss further properties of the local and global guidance. Then, we describe how the LTL_f formula ϕ is preprocessed and encoded using DeepDFA. Finally, we illustrate how DeepDFA is integrated with suffix prediction during training through differentiable sampling based on Gumbel-Softmax.

3.4. Further Discussions about Local and Global Guidance

The local and global guidance can be further compared along two dimensions: the training procedure and the scope of the LTL_f properties that are exploited.

During the training procedure, both types of guidance learn a similar conditional distribution, namely the probability that a trace is compliant with the knowledge given the context (i.e., the prefix), but they differ in how such a context is conditioned during training. The local guidance adopts the *teacher forcing* training procedure, which trains the neural network model by feeding the true previous tokens $\boldsymbol{a}_{<t}$ as inputs at every step and minimizing the negative log-likelihood in Equation 13. In general, this procedure yields stable and low-variance gradients. On the other hand, the global guidance exploits *autoregressive training*, which feeds the neural network model's own predictions $\tilde{\boldsymbol{a}}_t^{(i)}$ back as inputs. In this way, the loss becomes an expectation over model-generated prefixes, making optimization noisier and more sensitive to early mistakes. In general, teacher forcing is preferred for efficiency and stability, while autoregressive training better matches inference but is harder to optimize. The Monte Carlo estimation through sampling in GLL is performed to obtain a more stable autoregressive training procedure.

Regarding the scope of LTL_f properties, Declare constraints are characterized by two well-known LTL_f properties: safety and liveness. Safety constraints express that a certain behavior must never occur, whereas liveness constraints require that a condition will eventually be satisfied. When a

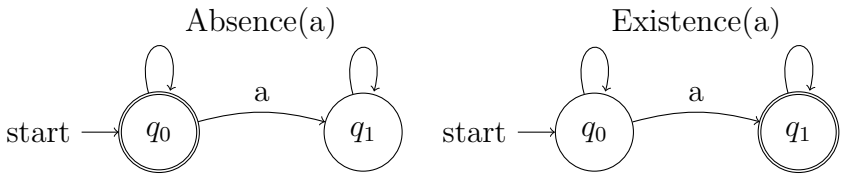

Figure 1: DFAs obtained after the conversion of the Absence (on the left) and Existence (on the right) constraints for activity a. The Absence DFA has a failing state, q_1, from which no accepting states can be reached (safety property). The Existence DFA has no failing states; from all states, an accepting state can be reached (liveness property).

safety constraint is translated into a deterministic finite automaton, a dedicated failing state is always generated to capture violations. In contrast, this is not the case for liveness constraints, since a trace can only be considered violated at its termination if the required condition has never been fulfilled. Figure 1 illustrates the DFAs for the Absence (i.e., a certain activity cannot occur) and Existence (i.e., a certain activity must occur) Declare constraints, which exhibit the safety and liveness properties, respectively.

The Absence constraint exhibits the safety property, whereas the Existence constraint exhibits the liveness property. The local guidance leverages the presence of failing states by penalizing the model whenever it assigns a high probability to transitions leading to such states. This mechanism enables the model to receive immediate feedback whenever a violation occurs and, therefore, is best suited for constraints that exhibit the safety property. Conversely, liveness constraints do not inherently produce explicit failing states. The only exploitable failing state is therefore the one introduced during the preprocessing phase (see Section 3.5.1), which can be reached exclusively through the end-of-trace signal. As a result, in this case, the violation signal can only be provided at the conclusion of the trace, similarly to the behavior of the global guidance.

3.5. Knowledge Preprocessing and Tensorization

In this section, we describe how the LTL_f formula ϕ is preprocessed and *tensorized* to enable the integration of prior knowledge into the learning process. Note that these steps are performed only once, before training begins, and are required to compute both the local and global losses.

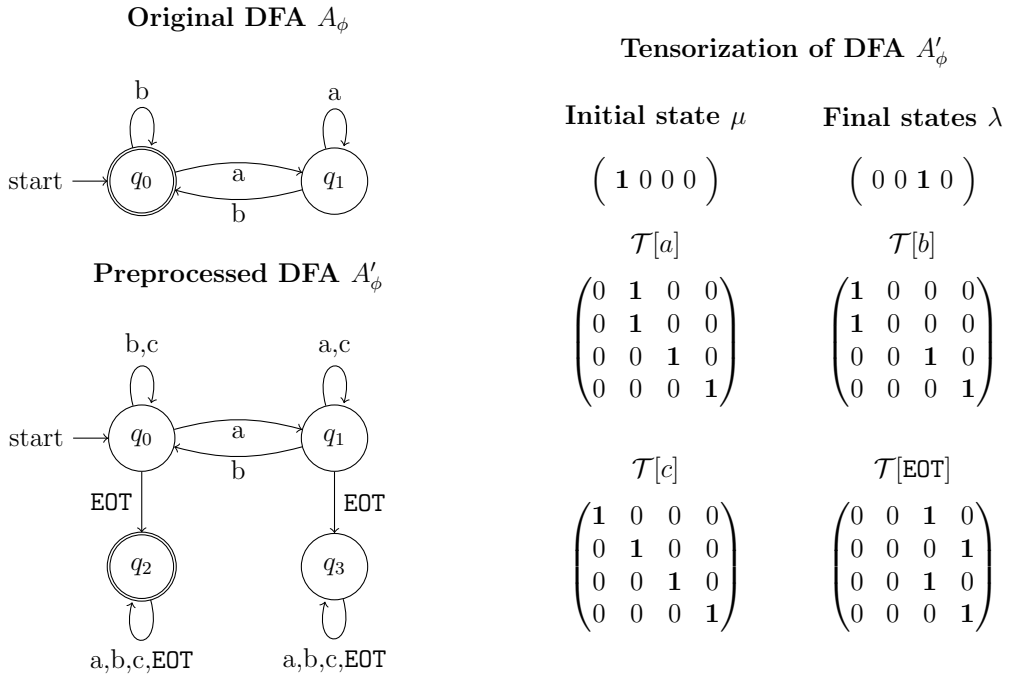

Figure 2: Preprocessing and tensorization for the running example. Left: original DFA A_ϕ (top) and preprocessed DFA A'_ϕ (bottom). Right: tensorization of A'_ϕ with initial state vector μ, final state vector λ, and transition matrices $\mathcal{T}[a], \mathcal{T}[b], \mathcal{T}[c], \mathcal{T}[\texttt{EOT}]$, where a 1 in position (i, j) of $\mathcal{T}[x]$ indicates a transition $q_i \xrightarrow{x} q_j$ in the automaton.

3.5.1. Knowledge Preprocessing

First, we translate the LTL_f formula ϕ into an equivalent deterministic finite automaton (DFA) $A_\phi = (\Sigma, Q, q_0, \delta, F)$ using the automatic translation tool `ltlf2DFA` [32]. This step is necessary because all existing approaches for integrating temporal specifications into learning pipelines rely on automata-based representations [22, 33, 34], while the direct integration of LTL_f formulas remains an open challenge [35, 36]. Although this translation has worst-case *double-exponential* complexity [24], it is often efficient in practice, and several scalable techniques exist to perform it for a given formula ϕ [37, 38, 39]. Moreover, in this work, we focus on *Declare* formulas [40], a standard for declaratively specifying business processes [41], which are known to yield DFAs of *polynomial size* with respect to the input formula [42].

Second, we adapt the DFA alphabet $\Sigma \subseteq \mathcal{A}$ so that it includes all activity

symbols present in the event log $\mathcal{A}$. Specifically, for each symbol $s \in \mathcal{A} \setminus \Sigma$ and each state $q \in Q$, we add a self-loop $\delta(q, s) = q$ to the transition function δ. This ensures that symbols not constrained by the formula can still be processed by the DFA without affecting its acceptance behavior. Note that while this technique is feasible in the BPM setting, it may become impractical in other application domains where the autoregressor's symbol space is excessively large, such as in LLM-based applications [10].

Additionally, we extend the DFA to handle the special `EOT` (End-of-Trace) symbol. In particular, we define as accepted all and only those traces of the form $t{+}\texttt{EOT}{+}z$ such that $t \in \mathcal{A}^*$, $t \vDash \phi$, and z is any (possibly empty) trace in $(\mathcal{A} \cup \{\texttt{EOT}\})^*$. This implies that:

(i) a non-terminating trace is considered *non-compliant* with the specification;

(ii) a trace is evaluated *against the specification only at the first occurrence of `EOT`* – whether the specification is violated or satisfied *before* this point is irrelevant, as are any symbols in z occurring after the first `EOT`.

To implement this behavior, we add `EOT` to the alphabet and introduce two terminal states: a success state q_t^s and a failure state q_t^f. For each state $q \in Q$, we add the transition $\delta(q, \texttt{EOT}) = q_t^s$ if $q \in F$, and $\delta(q, \texttt{EOT}) = q_t^f$ otherwise. States q_t^s and q_t^f are *terminal*, meaning they are absorbing: once entered, the automaton cannot exit. Therefore, for each symbol $s \in \mathcal{A} \cup \{\texttt{EOT}\}$, we add the transitions $\delta(q_t^s, s) = q_t^s$ and $\delta(q_t^f, s) = q_t^f$. We designate q_t^s as the *only accepting state* of the extended DFA.

Finally, we compute the set of failure states Q^{fail}, which is only necessary when employing the local logic loss L_ϕ^{loc}. The final DFA after preprocessing is therefore:

$$A'_\phi = (\mathcal{A} \cup \{\texttt{EOT}\}, Q \cup \{q_t^s, q_t^f\}, q_0, \delta', \{q_t^s\}),$$

with the extended transition function δ' defined as:

$$\delta'(q, s) = \begin{cases} \delta(q, s) & \text{if } q \in Q \wedge s \in \Sigma, \\ q & \text{if } s \in \mathcal{A} \setminus \Sigma \vee q \in \{q_t^s, q_t^f\}, \\ q_t^s & \text{if } s = \texttt{EOT} \wedge q \in F, \\ q_t^f & \text{if } s = \texttt{EOT} \wedge q \notin F. \end{cases} \tag{17}$$

Note that the input background knowledge could contain conflicting constraints, meaning that their conjunction cannot be satisfied. Therefore, the

corresponding automaton is empty, with no accepting states. In this case, our system is still able to learn, but the contribution of the conflicting input knowledge is ignored. Indeed, both logical losses have a constant value that cannot be reduced by updating the neural network weights. Consequently, only the task loss is able to drive the learning.

Running Example: DFA Preprocessing. The original DFA shown in Figure 2 is transformed into A'_ϕ, illustrated in the figure. Note that the new DFA is defined over all symbols in the domain plus the termination symbol, i.e., $\Sigma' = \mathcal{A} \cup \{\texttt{EOT}\}$, and contains the same states as A_ϕ together with two terminal states: one accepting terminal state ($q_t^s = q_2$) and one failing terminal state ($q_t^f = q_3$).

3.5.2. Knowledge Tensorization

Given the final DFA A'_ϕ, we transform it into a neural layer using DeepDFA [22]. DeepDFA is a neural, probabilistic relaxation of a standard deterministic finite-state machine, where the automaton is represented in matrix form and the input symbols, states, and outputs are *probabilistically grounded*.

We define the transition function of the DFA in matrix form as $\mathcal{T} \in \mathbb{R}^{|\Sigma| \times |Q| \times |Q|}$, and the initial and final states as the vectors $\mu \in \mathbb{R}^{|Q|}$ and $\lambda \in \mathbb{R}^{|Q|}$, respectively. While this matrix representation is traditionally used for Probabilistic Finite Automata (PFA), here it is applied to deterministic automata to leverage tensor operations for fast and differentiable computation of state transitions and outputs. The DeepDFA model is defined as follows:

$$\begin{aligned} \tilde{q}_0 &= \mu, \\ \tilde{q}_t &= \sum_{j=1}^{|\Sigma|} \tilde{\sigma}_t[j] \cdot (\tilde{q}_{t-1} \cdot \mathcal{T}[j]), \\ \tilde{o}_t &= \tilde{q}_t \cdot \lambda^\top. \end{aligned} \tag{18}$$

Here, $\tilde{\sigma}_t$, $\tilde{q}_t$, and $\tilde{o}_t$ represent the probabilistic representations of the input symbol, the automaton state, and the output (i.e., whether the trace is accepted or rejected) at time t. In the GLL, the probabilistically grounded sequence $\tilde{\boldsymbol{\sigma}}$ is the sampled trace $\tilde{\boldsymbol{a}}^{(i)}$; therefore, the output $\tilde{o}_t$ corresponds to the probability P_{DDFA}, see Equation 15. The neural network parameters μ,

$\mathcal{T}$, and λ are initialized from the DFA as:

$$\mu[j] = \begin{cases} 1 & \text{if } q_j = q_0, \\ 0 & \text{otherwise,} \end{cases}$$
$$\mathcal{T}[s, q_i, q_j] = \begin{cases} 1 & \text{if } \delta(q_i, s) = q_j, \\ 0 & \text{otherwise,} \end{cases} \tag{19}$$
$$\lambda[j] = \begin{cases} 1 & \text{if } q_j \in F, \\ 0 & \text{otherwise.} \end{cases}$$

Running Example: DFA Tensorization. The preprocessed DFA A'_ϕ shown in Figure 2 is converted into the three-dimensional tensor $\mathcal{T}$ encoding the transition function, and into the two vectors μ and λ, which represent the initial state and the accepting states, respectively, as illustrated in the same figure. Note that a 1 in position (i, j) of $\mathcal{T}[x]$ indicates a transition $q_i \xrightarrow{x} q_j$ in the preprocessed automaton.

3.6. Differentiable Sampling

For the computation of the global logic loss, our goal is to generate complete suffixes and evaluate their compliance with the knowledge constraint during training. To this end, note that the next-activity predictor is trained on *perfectly one-hot* (i.e., symbolic) input sequences $\boldsymbol{a}_{\leq t}$ and produces continuous probability vectors $\tilde{y}^{(t)}$ as output, which can differ significantly from one-hot vectors. As a result, we cannot directly feed these probability vectors back into the network as inputs in subsequent steps, as doing so may lead to unpredictable behavior. Instead, we need to *sample* from these distributions to recover one-hot-like inputs. At the same time, this sampling must remain differentiable to enable backpropagation.

The computation of the global logic loss proceeds as follows:

1. Given a prefix $\boldsymbol{p}_t$, generate N suffixes $\tilde{\boldsymbol{s}}_t^{(i)}$ that are simultaneously: (i) highly probable under the network's distribution; (ii) differentiable; and (iii) *nearly symbolic*, i.e., as close as possible to one-hot vectors;

2. Evaluate whether the generated complete traces $\boldsymbol{p}_t + \tilde{\boldsymbol{s}}_t^{(i)}$ satisfy the LTL_f formula ϕ using DeepDFA, which supports evaluation over both

categorically grounded and probabilistically grounded traces;[4]

3. Maximize the estimated satisfaction probability $\hat{P}_{\theta \models \phi}$, computed as the empirical mean over the sampled traces.

Thanks to the differentiability of both the knowledge evaluator and the sampling process, the resulting loss can be back-propagated through the generated suffixes and used to update the parameters θ of the next-activity predictor, as illustrated in Figure 3. To sample the next activity $\tilde{a}_t$ from the probability distribution $\tilde{y}_t$ produced by the network (while ensuring differentiability), we employ the Gumbel-Softmax reparameterization trick [18, 31]:

$$\tilde{a}_t = \text{softmax}\left(\frac{\log(\tilde{y}_t) + G}{\tau}\right), \tag{20}$$

where G is a random vector sampled from the Gumbel distribution, and τ is a temperature parameter that controls the sharpness of the output distribution. As $\tau \to 0$, the output approaches a discrete one-hot vector, while for $\tau = 1$, it remains close to the original continuous probabilities in $\tilde{y}_t$. Since the next activity is only *probabilistically* grounded, we denote it as $\tilde{a}_t$.

Running Example: Local and Global Loss Computation. We now have the full theoretical framework for computing the two loss functions for our running example characterized by the automaton A'_ϕ of Figure 2.

The local loss is computed under the *teacher forcing* training procedure, meaning that at each time step, the model is conditioned on the ground-truth prefix of the sequence and penalized for assigning probability mass to labels that would cause a permanent violation. To compute this penalty, the prefix is simulated step by step through the DFA in order to identify the labels that would lead to a rejecting state and to sum the probability mass assigned to them. Consider the following sequence, which satisfies the automaton A'_ϕ:

$$\boldsymbol{a} = (c, a, c, b, \texttt{EOT}).$$

[4]Note that the prefix $\boldsymbol{p}_t$ is categorically grounded, whereas the suffix $\tilde{\boldsymbol{s}}_t$ is probabilistically grounded.

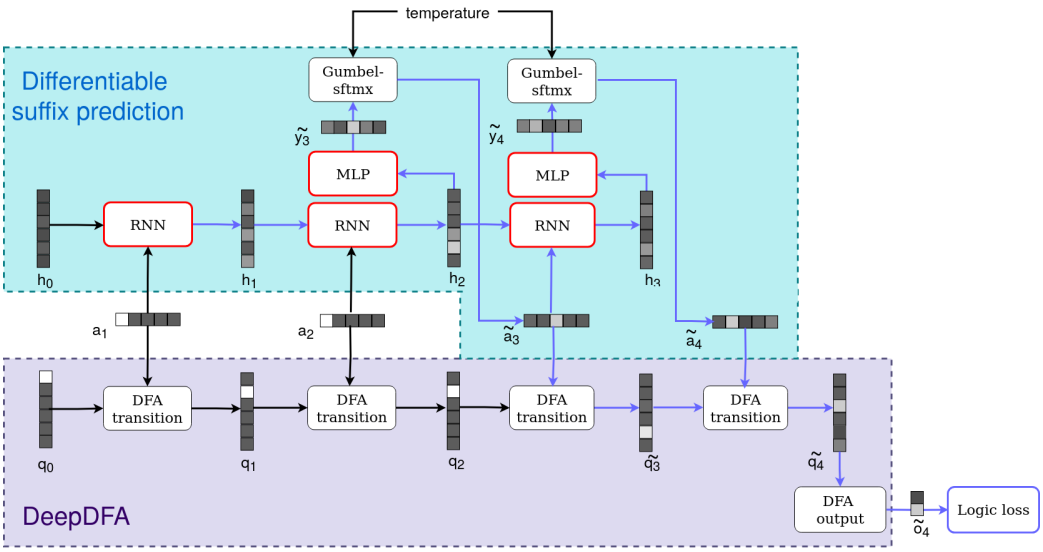

Figure 3: Global logic loss computation using a differentiable procedure for both suffix generation and formula evaluation. Violet arrows indicate the connections through which the loss is back-propagated, while components highlighted with a red border denote the modules whose parameters are updated by the logic loss. The figure is the same as in [20], with adapted notation.

At each time step, the current prefix is simulated through the automaton, and the model outputs a probability distribution over the alphabet:

$$
\begin{aligned}
\boldsymbol{a}_{<1} &= () && f_\theta(\boldsymbol{a}_{<1}) = [p(a), p(b), p(c), p(\texttt{EOT})] = [0.35,\ 0.15,\ 0.4,\ 0.1],\\
\boldsymbol{a}_{<2} &= (c) && f_\theta(\boldsymbol{a}_{<2}) = [0.55,\ 0.25,\ 0.15,\ 0.05],\\
\boldsymbol{a}_{<3} &= (c, a) && f_\theta(\boldsymbol{a}_{<3}) = [0.1,\ 0.65,\ 0.1,\ 0.15],\\
\boldsymbol{a}_{<4} &= (c, a, c) && f_\theta(\boldsymbol{a}_{<4}) = [0.1,\ 0.15,\ 0.55,\ 0.2],\\
\boldsymbol{a}_{<5} &= (c, a, c, b) && f_\theta(\boldsymbol{a}_{<5}) = [0.05,\ 0.1,\ 0.15,\ 0.7].
\end{aligned}
$$

The automaton A'_ϕ encodes the Response Declare constraint that has the liveness property. Recall that such a constraint can be permanently violated only if the sequence terminates before the constraint is satisfied. In this example, once the symbol a has been observed, the constraint requires that b must eventually follow before the sequence ends. Therefore, at prefixes where a has occurred but b has not yet been seen (steps 3 and 4), predicting `EOT` would cause a permanent violation. In these cases, the invalid probability mass corresponds solely to the probability assigned to `EOT`. The violating

mass at each time step is computed using Equation 12 and is therefore:

$$\begin{aligned}
&P(\boldsymbol{a}_1 \not\models^{\times} \phi \mid \boldsymbol{a}_{<1}) = 0,\\
&P(\boldsymbol{a}_2 \not\models^{\times} \phi \mid \boldsymbol{a}_{<2}) = 0,\\
&P(\boldsymbol{a}_3 \not\models^{\times} \phi \mid \boldsymbol{a}_{<3}) = p(\texttt{EOT}) = 0.15,\\
&P(\boldsymbol{a}_4 \not\models^{\times} \phi \mid \boldsymbol{a}_{<4}) = p(\texttt{EOT}) = 0.2,\\
&P(\boldsymbol{a}_5 \not\models^{\times} \phi \mid \boldsymbol{a}_{<5}) = 0.
\end{aligned}$$

After computing the invalid mass at each step, the local loss is obtained by applying Equation 13, which averages the negative logarithm of the valid mass across all time steps:

$$\begin{aligned}
L_{\phi}^{\text{loc}} &= \frac{1}{5}\Big[0 + 0 - \log(1 - 0.15) - \log(1 - 0.2) + 0\Big]\\
&= \frac{-\log(0.85) - \log(0.8)}{5} \approx 0.0771.
\end{aligned}$$

Regarding the global loss, we recall that this loss is computed as follows: (i) we generate N traces of length T, and (ii) we compute over them the probability that the autoregressive model satisfies the specification encoded by the automaton, denoted as $\hat{P}_{\theta \models \phi}$ and defined in Equation 15. Consider the case where we simulate three traces of length four ($N = 3, T = 4$):

$$\boldsymbol{a}^{(1)} = (a, b, b, b), \quad \boldsymbol{a}^{(2)} = (a, c, b, \texttt{EOT}), \quad \boldsymbol{a}^{(3)} = (a, a, \texttt{EOT}, b).$$

Among these, only $\boldsymbol{a}^{(2)}$ satisfies the automaton A'_{ϕ}. Indeed, it is the only trace that satisfies the specification *and then* properly terminates. Trace $\boldsymbol{a}^{(1)}$ satisfies the specification but does not terminate, while $\boldsymbol{a}^{(3)}$ terminates one step before T but does not satisfy the specification prior to termination. The satisfaction probability is therefore

$$P_{\phi} = \frac{1}{3}(0 + 1 + 0) = 0.33.$$

For simplicity, this example assumes perfectly categorical sampling of symbols. In practice, however, sampling is performed via the Gumbel–Softmax in order to preserve differentiability. The Gumbel–Softmax *approximates* categorical sampling; thus, the traces $\boldsymbol{a}^{(i)}$ may in practice be sampled

as follows (in bold the highest probability value):

$$\begin{aligned}
\tilde{\boldsymbol{a}}^{(1)} =&([\mathbf{0.99}, 0, 0.01, 0], \\
&[0.05, \mathbf{0.82}, 0.05, 0.08], \\
&[0.05, \mathbf{0.9}, 0.05, 0], \\
&[0, \mathbf{0.75}, 0.05, 0.2])
\end{aligned}
\qquad
\begin{aligned}
\tilde{\boldsymbol{a}}^{(2)} =&([\mathbf{0.99}, 0, 0.01, 0], \\
&[0, 0.02, \mathbf{0.98}, 0], \\
&[0, \mathbf{1}, 0, 0], \\
&[0, 0.2, 0, \mathbf{0.8}])
\end{aligned}
\qquad
\begin{aligned}
\tilde{\boldsymbol{a}}^{(3)} =&([\mathbf{0.99}, 0, 0.01, 0], \\
&[\mathbf{0.96}, 0.02, 0.02, 0], \\
&[0.1, 0.2, 0.04, \mathbf{0.66}], \\
&[0.05, \mathbf{0.7}, 0.2, 0.05]),
\end{aligned}$$

where each time step is represented by a probability vector over the alphabet, whose largest component corresponds to symbol a at the first step, c at the second, and so on. In this case, the probability that this probabilistic grounding of the trace satisfies the specification is no longer exactly one, but remains close to it. DeepDFA naturally handles such probabilistic groundings.

We now compute the satisfaction probability $P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(i)} \vDash \phi)$ by applying Equation 18 using the matrices shown in Figure 2. $P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(1)} \vDash \phi)$ is computed as:

$$\begin{aligned}
\tilde{q}_1 &= [0.01, 0.99, 0, 0], \\
\tilde{q}_2 &= [0.82, 0.1, 0, 0.08], \\
\tilde{q}_3 &= [0.87, 0.05, 0, 0.08], \\
\tilde{q}_4 &= [0.73, 0, 0.17, 0.09], \\
P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(1)} \vDash \phi) &= \tilde{o}_4 = 0.17.
\end{aligned}$$

$P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(2)} \vDash \phi)$ is computed as:

$$\begin{aligned}
\tilde{q}_1 &= [0.01, 0.99, 0, 0], \\
\tilde{q}_2 &= [0.03, 0.97, 0, 0], \\
\tilde{q}_3 &= [1, 0, 0, 0], \\
\tilde{q}_4 &= [0.2, 0, 0.8, 0], \\
P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(2)} \vDash \phi) &= \tilde{o}_4 = 0.8.
\end{aligned}$$

$P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(3)} \vDash \phi)$ is computed as:

$$\begin{aligned}
\tilde{q}_1 &= [0.01, 0.99, 0, 0], \\
\tilde{q}_2 &= [0.02, 0.98, 0, 0], \\
\tilde{q}_3 &= [0.2, 0.14, 0.01, 0.65], \\
\tilde{q}_4 &= [0.28, 0.05, 0.02, 0.66], \\
P_{\text{DDFA}}(\tilde{\boldsymbol{a}}^{(3)} \vDash \phi) &= \tilde{o}_4 = 0.02.
\end{aligned}$$

The final numeric value of the GLL is therefore

$$L_{\phi}^{\text{glob}} = -\log\left(\frac{0.17 + 0.8 + 0.02}{3}\right) = 1.11.$$

3.7. Applications beyond PPM and Limitations

In this paper, we focus on integrating logical and autoregressive losses in the context of PPM. However, our approach is general and can be applied to other domains, which we do not address here and leave for future work. Indeed, suffix prediction in PPM is simply a specific instance of discrete sequence generation. Many applications fall under this same paradigm, including language generation in NLP [43] and neural machine translation [44]. Given the remarkable success of Transformers across a wide range of tasks, a current trend is to employ discrete autoregression even in originally continuous domains, after discretization or "tokenization" [45]. Across all these applications, our framework provides a principled way to integrate prior knowledge – expressed as temporal formulas or automata – into sequential data modeling.

That said, PPM tasks typically involve medium-scale problem sizes (especially when compared to LLMs), in terms of both vocabulary size and trace length. The imposed specifications are also of reasonable complexity [5]. Applying our approach to substantially larger domains or more complex specifications may introduce the following challenges, which we summarize below together with possible mitigation strategies:

1. **Large vocabularies.** As discussed in Section 3.5.1, in the current version of the framework the automaton is preprocessed so as to explicitly include all symbols in the vocabulary $\mathcal{A}$. As a result, for very large vocabularies the size of the DeepDFA model can grow substantially. However, only the symbols appearing in the knowledge alphabet Σ are actually required to check specification satisfaction, and typically $|\Sigma| \ll |\mathcal{A}|$. This issue can therefore be addressed by mapping all symbols irrelevant to the specifications into a single special token, thereby minimally increasing the size of the automaton model.

[5] In our datasets, the number of activities and the trace length range from 10 to 50 and from 10 to 100, respectively. The number of states of the automata varies between 6 and 13.

2. **Very long sequences.** Recall that, to compute the global logic loss, suffix prediction must be carried out until the end of the sequence (or up to a maximum number of steps) during training. Consequently, applying the framework to very long sequences may significantly slow down the computation of the global loss. This limitation can be mitigated in different ways depending on the application domain. In natural language processing, for instance, one may segment sentences into sub-sentences, each expressing a distinct *concept* [46], and formulate the knowledge over a concept-level vocabulary rather than over individual tokens. Similar forms of *hierarchical* aggregation may also be meaningful in reinforcement learning settings [45, 47]. Ultimately, the most appropriate strategy depends on the specific domain.

3. **Large automata/complex specifications.** Depending on the complexity of the temporal specification we want to enforce on the dataset, we may encounter: (i) long conversion times to build the automaton and/or (ii) very large automata. Indeed, given an LTL_f formula ϕ, theoretical results show that the size of the equivalent automaton A_ϕ is double-exponential in ϕ in the worst case [24]. However, scalable techniques are available for computing A_ϕ from ϕ [37, 38, 39], and in practice the resulting automaton is often quite small and can be constructed in a reasonable amount of time. This is especially true for Declare formulas, which have been shown to generate DFAs that are polynomial in the size of the original formula [42]. Moreover, note that automaton construction in our framework is performed *offline* and only once, so its computation time does not significantly affect overall performance. For very complex knowledge bases, conversion times and automaton size can be further reduced by representing the specification as a conjunction of simpler formulas (e.g., $\phi = \phi_1 \wedge \phi_2 \wedge \phi_3$), converting each subformula into its own automaton ($A_{\phi_1}, A_{\phi_2}, A_{\phi_3}$), and monitoring the states of all automata in parallel. The knowledge is considered satisfied if all automata reach a final state at the end of the trace.

These considerations provide useful guidelines for extending the framework to substantially larger domains. They are not required in our current experiments, as the proposed approach is well suited to PPM-sized problems, and we leave their systematic exploration to future work.

4. Experiments

This section describes the experimental setup, including the datasets, evaluation metrics, and comparative approaches, followed by a detailed analysis of the results obtained. The experiments are reproducible using our implementations of the GLL and LLL at https://github.com/axelmezini/nesy-suffix-prediction-dfa.

4.1. Experimental Setup

Our methods have been evaluated using three real-world datasets: the Sepsis event log [6], the BPI Challenge 2013 (closed problems) [7], and the BPI Challenge 2020 (travel permit) event log [8]. Given an event log, the traces are ordered by the timestamp of the first event and split into training and test sets using an 80–20 ratio. Knowledge is extracted from the test set in the form of Declare constraints using the Declare Miner [48], each with a minimum support value of 85%, that is, they are satisfied in at least 85% of the traces. This value allows us to obtain a reasonable number of constraints and test traces that satisfy them. These constraints are then translated into their corresponding LTL_f formulas and combined into an LTL_f model using conjunctions. To enable controlled experiments, traces that violate the extracted LTL_f model (i.e., that do not satisfy all the discovered constraints) are removed from both the training and test sets.

Table 1 summarizes the key statistics of the datasets used in our experiments. The datasets vary in size and complexity, with the number of training traces ranging from 547 to 1665 and the number of distinct activity names ranging from 7 to 51. The semantic complexity of the background knowledge can be inspected from the table by analyzing the complexity of the generated automaton. The number of DFA states and transitions gives an idea of the complexity of the conjunction of the Declare constraints. While all DFAs have a reasonable number of states, the DFA of BPIC 2020 has a high number of transitions, suggesting a high semantic complexity of the underlying Declare model. The Declare model of BPIC 2013, on the other

[6] `http://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`, checked on June 2026

[7] `http://doi.org/10.4121/uuid:c2c3b154-ab26-4b31-a0e8-8f2350ddac11`, checked on June 2026

[8] `http://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51`, checked on June 2026

	Sepsis	BPIC 2013	BPIC 2020
# Training traces	547	590	1665
# Training variants	438	61	50
# Testing traces	143	228	730
# Testing variants	121	24	24
# Activities	16	7	51
# Constraints	39	7	56
# DFA states	10	6	13
# Failure states	2	2	2
# DFA transitions	170	48	676

Table 1: Key statistics of the datasets and background knowledge used in our experiments.

hand, presents the lowest semantic complexity. The mined Declare models are available in the shared GitHub repository. Correspondingly, the number of discovered Declare constraints and the size of the resulting deterministic finite automata (DFA) differ across datasets, reflecting their behavioral complexity. Despite these differences, all datasets include a small number of failure states in their DFAs, which indicate states violating the discovered constraints.

In addition to the training set containing only positive traces, four levels of noise are introduced at the event level: 10%, 20%, 30%, and 40%. Noise is applied by randomly replacing the activity label of selected events with other labels from the activity vocabulary. Introducing noise into the training set after filtering makes it possible to train on non-compliant traces in a more controlled manner, providing a more realistic scenario, as one may encounter in production. Starting from a set of fully compliant traces and progressively injecting different levels of noise allows the assessment of how sensitive the extracted knowledge of a specific dataset is to random activity label perturbations.

Table 2 reports the resulting proportions of traces in the training sets that remain compliant with respect to the extracted constraints after noise injection. The BPIC 2020 and Sepsis datasets are particularly sensitive, retaining only slightly more than 50% compliant traces at a noise level of 10% and less than 5% at a noise level of 40%. In contrast, BPIC 2013 appears more robust, which is expected given its lower number of constraints. As a result, it maintains at least 35% compliant traces even at the highest noise

Noise	Sepsis	BPIC 2013	BPIC 2020
10%	0.534	0.800	0.513
20%	0.296	0.597	0.255
30%	0.135	0.451	0.094
40%	0.044	0.349	0.048

Table 2: Compliance ratio of the training sets after noise injection.

level. During GLL training and GLL and LLL testing, different prefix lengths are used: $\lfloor M/2 \rfloor$, $\lfloor M/2 \rfloor + 1$, and $\lfloor M/2 \rfloor + 2$, where M is the median trace length of the training log, as done in [9].

Compared Methods. We base our evaluation on two classes of next-activity predictors: recurrent and attention-based architectures. For the recurrent setting, we consider: (i) a baseline LSTM trained only with supervised loss, (ii) an LSTM trained with supervised and local logic loss, and (iii) an LSTM trained with supervised and global logic loss. All recurrent models are two-layer LSTMs with 100 hidden units per layer, trained using a batch size of 64 and the Adam optimizer. To assess architectural generality, we additionally evaluate Transformer-based models. The Transformer consists of an input projection layer followed by positional embeddings, a stack of Transformer encoder layers with multi-head self-attention (using GELU activation and pre-layer normalization), and a linear output projection to the activity vocabulary. A causal mask is applied to enforce autoregressive generation for GLL. We evaluate baseline, baseline+LLL, and baseline+GLL variants under the same training protocol as the LSTMs. We highlight the fact that the base models (simple LSTM and Transformer) can be considered an ablation study in which the logical loss in Equation 11 has been removed.

For each setting, two different sampling strategies are tested: temperature-based sampling (at testing time for both GLL and LLL and at training time for LLL) with temperature parameter $\tau = 0.7$ (see Equation 20), and greedy decoding at testing time for both GLL and LLL (i.e., the activity that maximizes the next-symbol probability is chosen). For each distinct configuration, 15 runs are executed. We compare the proposed approaches using two evaluation metrics: the satisfaction rate with respect to the LTL_f model and the normalized Damerau–Levenshtein similarity with respect to the ground-truth traces. Both metrics range from 0 to 1, following the "higher is better"

rule.

4.2. Results

Figures 4 and 5 show the performance of LSTM models (with $\alpha = 0.75$ for GLL and $\alpha = 0.25$ for LLL) across all datasets and noise levels for the satisfiability and similarity metrics, respectively. The empirical results demonstrate that integrating background knowledge during training consistently increases the satisfaction rate of the predicted traces, regardless of the sampling strategy. These improvements are especially pronounced under higher noise levels, confirming the utility of incorporating logical background knowledge when predictive uncertainty increases. Notably, the satisfaction rate remains close to 100% even at the highest noise level of 40%.

Importantly, this increase in satisfaction rate does not negatively affect the model's ability to learn from the training data. The similarity to the ground-truth data remains consistent across configurations. These results suggest that the integrated knowledge helps the model distinguish compliant traces from noisy patterns in the training sets. Overall, the impact of the sampling strategy is limited, although greedy decoding often shows slightly better performance than temperature-based sampling for both metrics.

To further validate these findings, we replicate the same empirical evaluation using Transformer architectures (with $\alpha = 0.75$ for GLL and $\alpha = 0.25$ for LLL). Figures 6 and 7 report the corresponding results and reveal behavior consistent with that observed for LSTMs. In particular, when either GLL or LLL is employed, the satisfiability remains close to 100% across all noise levels, whereas in the baseline configuration it deteriorates substantially as noise increases. As with LSTMs, the improvement in satisfiability does not come at the expense of predictive quality: the suffix similarity remains comparable to the baseline across configurations. These results confirm that the benefits of incorporating the logic-based loss generalize across architectures and are not specific to recurrent models.

Differences observed across datasets likely reflect their inherent characteristics, such as vocabulary size, number and frequency of variants, and training data volume. Overall, our empirical evaluation confirms that integrating background knowledge at training time consistently improves performance and that the methodology is applicable across various real-world datasets and scenarios. Across all datasets and noise levels, the most important result is the clear improvement obtained by introducing the logic-based loss

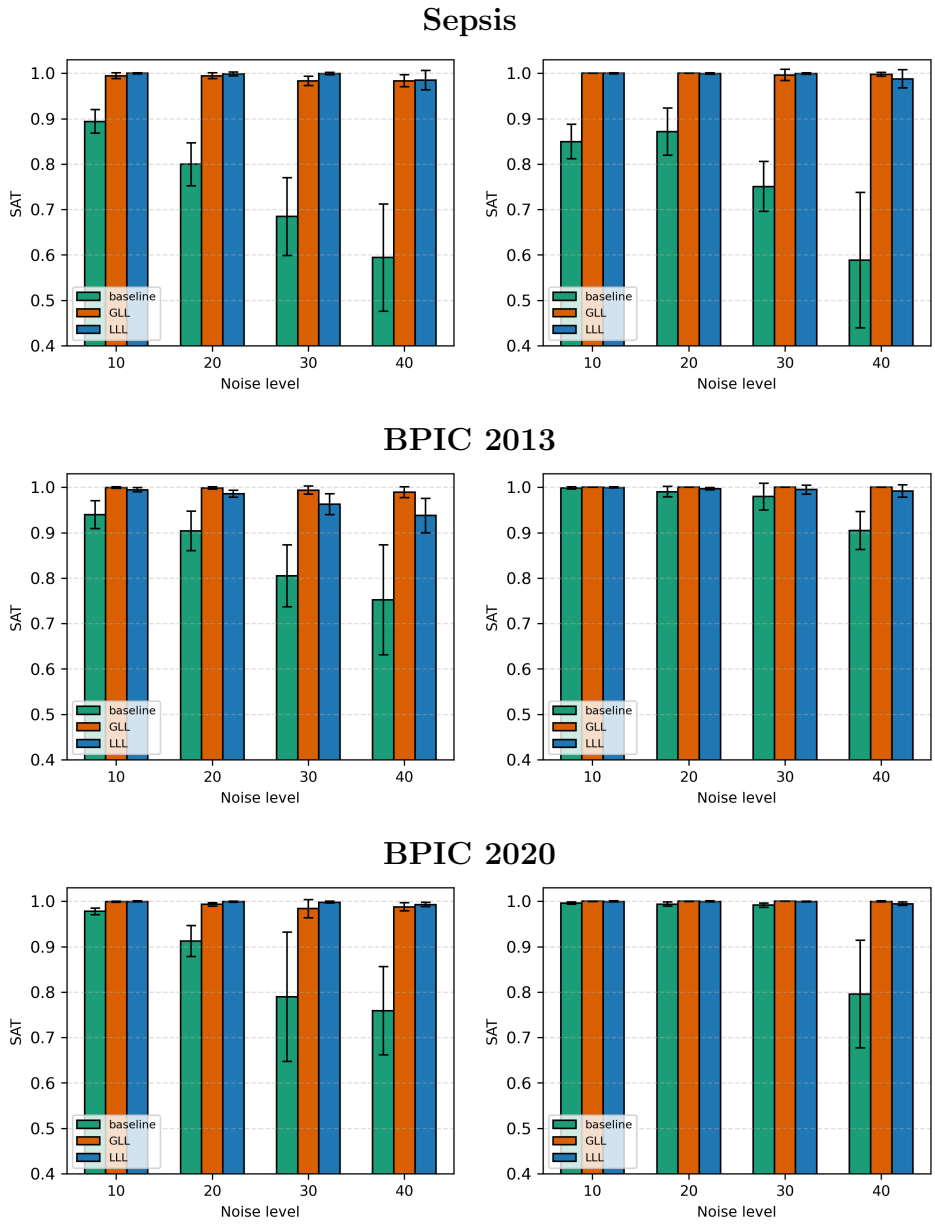

Figure 4: Satisfiability (SAT) of predicted traces under increasing noise levels for LSTM models. Columns correspond to the sampling strategies (temperature on the left, greedy on the right). Results are averaged over prefix lengths; error bars indicate the standard deviation over prefix lengths.

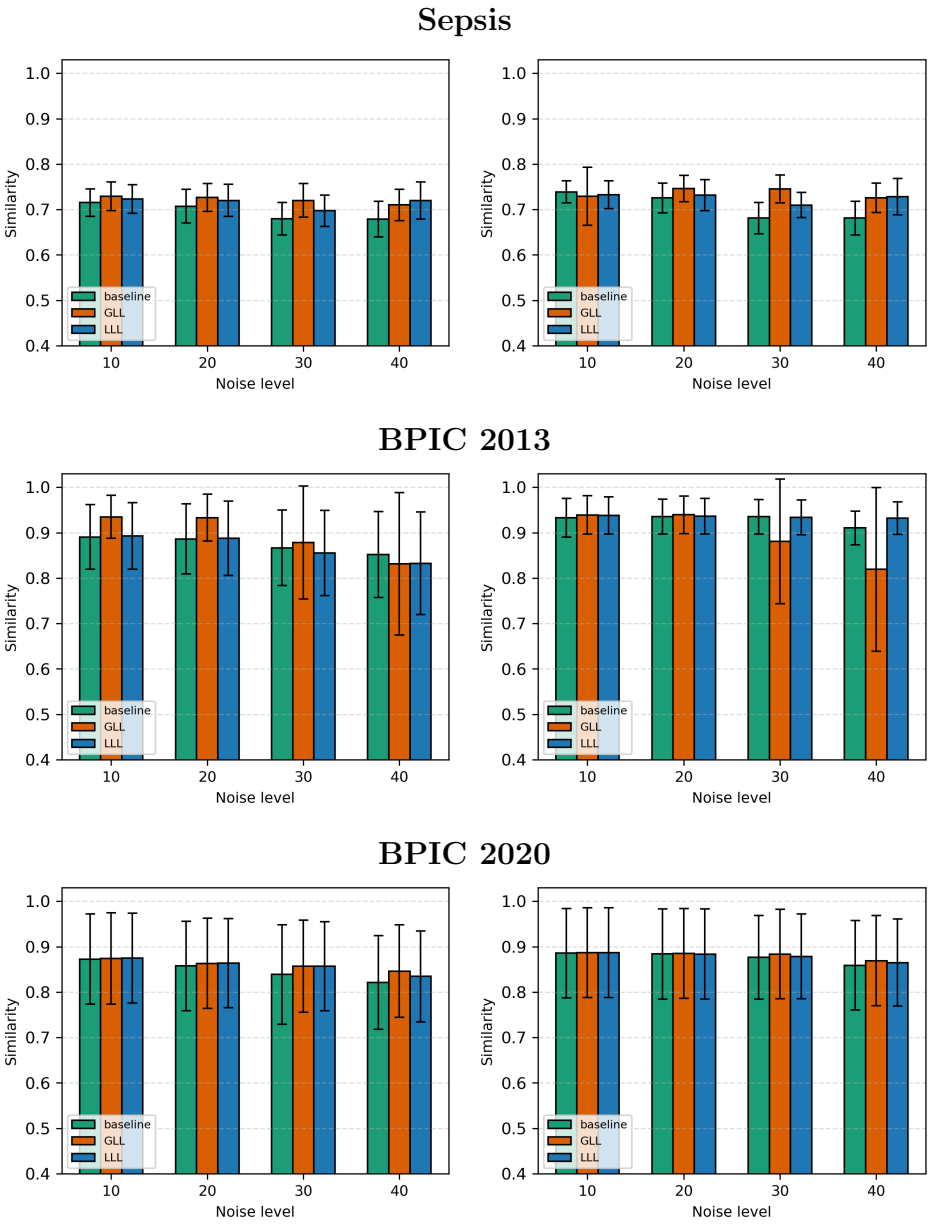

Figure 5: Normalized Damerau–Levenshtein similarity under increasing noise levels for LSTM models. Columns correspond to the sampling strategies (temperature on the left, greedy on the right). Results are averaged over prefix lengths; error bars indicate the standard deviation over prefix lengths.

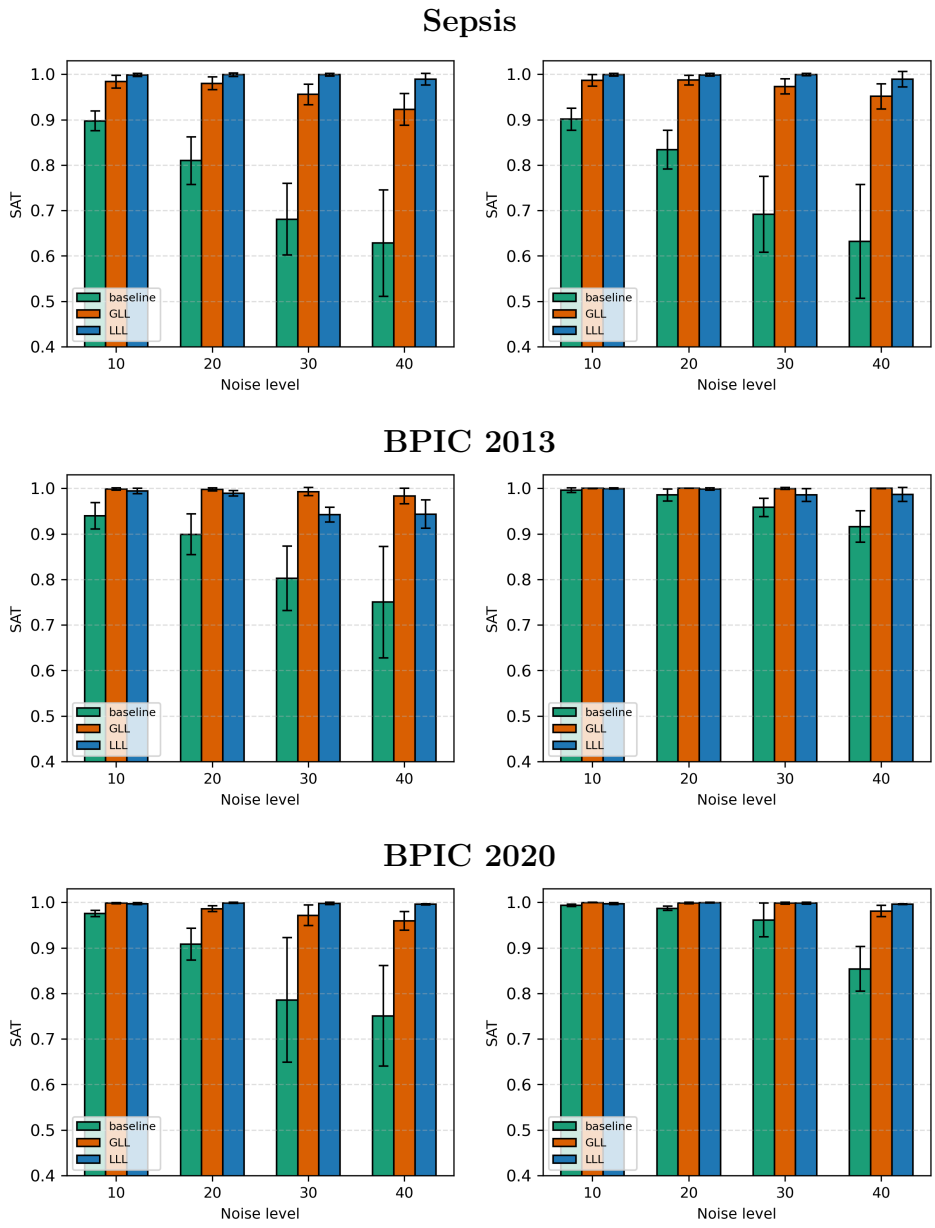

Figure 6: Satisfiability (SAT) of predicted traces under increasing noise levels for Transformer models. Columns correspond to the sampling strategies (temperature on the left, greedy on the right). Results are averaged over prefix lengths; error bars indicate the standard deviation over prefix lengths.

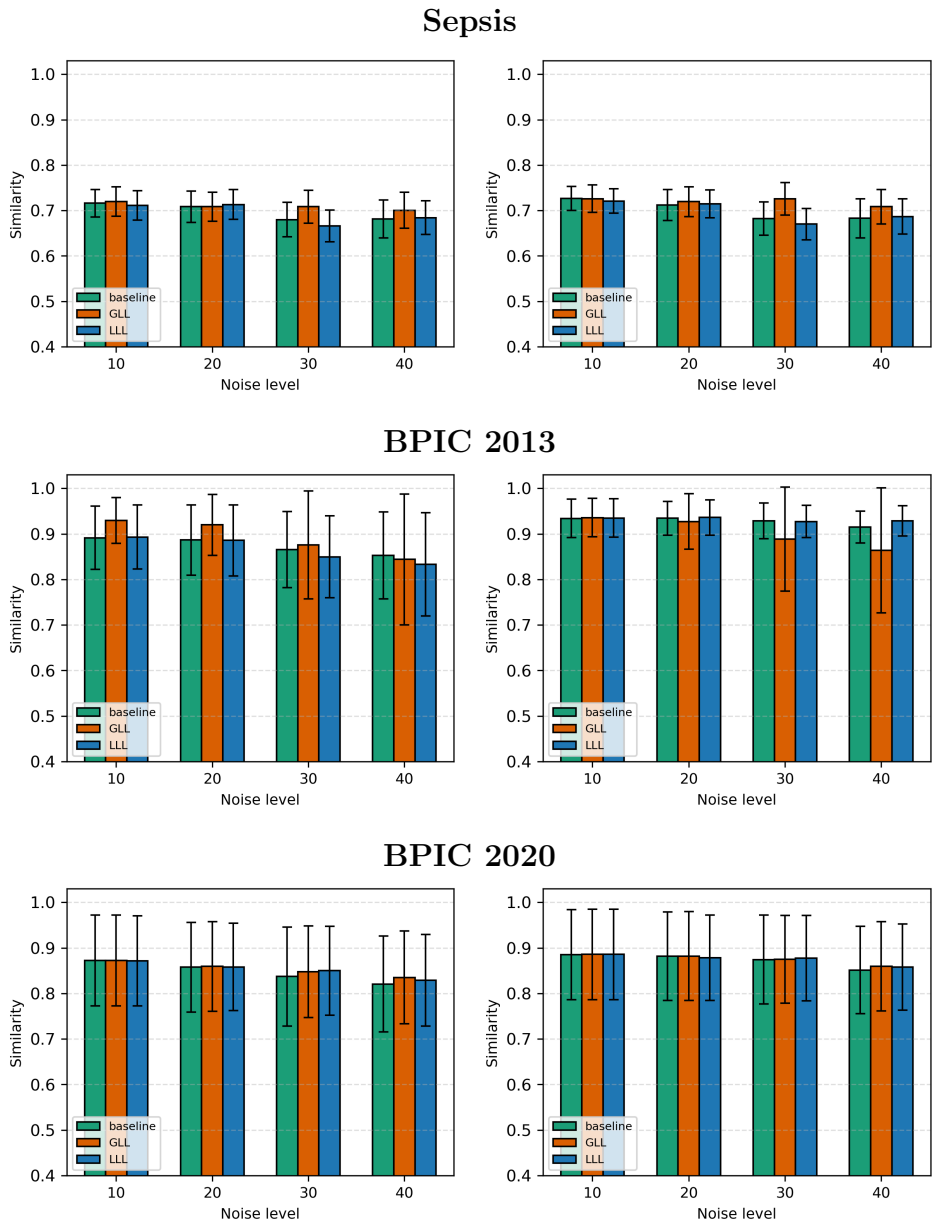

Figure 7: Normalized Damerau–Levenshtein similarity under increasing noise levels for Transformer models. Columns correspond to the sampling strategies (temperature on the left, greedy on the right). Results are averaged over prefix lengths; error bars indicate the standard deviation over prefix lengths.

compared to the baseline. This result depends on a fixed value of the α hyperparameter in the combined loss function in Equation 11.

Dataset	Noise	Global Guidance										Baseline
		0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95	1
Sepsis	10%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.7%	**100%**	99.8%	98.5%	85.0%
	20%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.8%	**100%**	99.8%	98.2%	87.1%
	30%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.9%	99.6%	99.6%	96.8%	75.1%
	40%	**100%**	**100%**	**100%**	99.8%	99.9%	99.5%	99.8%	99.8%	99.3%	92.4%	58.9%
BPIC13	10%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.8%
	20%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.5%	99.0%
	30%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.6%	98.0%
	40%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.6%	90.5%
BPIC20	10%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.9%	99.6%
	20%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.9%	99.4%
	30%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.9%	99.5%	99.1%
	40%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.9%	88.6%	79.6%

		Local Guidance										Baseline
		0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95	1
Sepsis	10%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	85.0%
	20%	**100%**	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	99.9%	87.1%
	30%	99.9%	**100%**	99.9%	99.9%	**100%**	**100%**	**100%**	99.9%	**100%**	99.9%	75.1%
	40%	98.6%	98.2%	**98.8%**	98.7%	97.8%	97.9%	**98.8%**	**98.8%**	98.4%	**98.8%**	58.9%
BPIC13	10%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.8%
	20%	**99.7%**	**99.7%**	**99.7%**	**99.7%**	**99.7%**	**99.7%**	**99.7%**	**99.7%**	**99.7%**	**99.7%**	99.0%
	30%	99.3%	**99.5%**	**99.5%**	99.4%	99.4%	99.4%	99.4%	99.4%	**99.5%**	99.4%	98.0%
	40%	99.3%	99.3%	99.1%	99.2%	99.2%	98.9%	**99.4%**	99.2%	99.2%	99.2%	90.5%
BPIC20	10%	**99.9%**	99.8%	99.8%	99.8%	**99.9%**	99.8%	**99.9%**	**99.9%**	99.8%	**99.9%**	99.6%
	20%	**100%**	**100%**	**100%**	**100%**	99.9%	99.9%	99.9%	**100%**	99.9%	99.9%	99.4%
	30%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	99.9%	99.9%	99.1%
	40%	**99.6%**	**99.6%**	99.5%	99.5%	99.5%	99.5%	99.5%	99.5%	99.5%	99.5%	79.6%

Table 3: Satisfiability scores obtained with the LSTM model for different values of the parameter α using greedy sampling. The rightmost column reports the baseline setting ($\alpha = 1$), in which no logic-based loss is used. The best results are in bold.

We now explore the impact of this hyperparameter by showing how performance differs across different values of α. As shown in Table 3, satisfiability in the baseline setting decreases noticeably as noise increases, while the inclusion of the logic loss keeps satisfiability close to perfect over a broad range of α values. This demonstrates that the logic component is crucial for ensuring that generated outputs satisfy the required constraints, especially under noisy conditions. For similarity, the effect of the logic loss depends on the guidance strategy, as shown in Table 4. With Global Guidance, giving more weight to the logic loss can sometimes reduce similarity, indicating a trade-off between strictly enforcing constraints and staying close to the orig-

Dataset	Noise	Global Guidance										Baseline
		0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95	1
Sepsis	10%	52.2%	52.9%	57.2%	61.9%	68.6%	69.9%	70.8%	72.9%	72.9%	73.1%	**73.9%**
	20%	49.5%	50.8%	63.3%	65.3%	72.7%	74.1%	73.1%	**74.6%**	71.9%	73.4%	72.5%
	30%	47.6%	49.2%	54.9%	65.9%	68.7%	71.6%	72.3%	**74.5%**	72.2%	68.2%	68.1%
	40%	55.5%	55.6%	55.4%	66.0%	69.0%	70.3%	72.2%	72.6%	**73.8%**	69.9%	68.1%
BPIC13	10%	81.3%	81.4%	81.5%	81.6%	81.5%	82.7%	86.8%	**93.9%**	**93.9%**	**93.9%**	93.3%
	20%	81.2%	81.5%	81.5%	81.4%	81.4%	81.4%	81.3%	**93.9%**	**93.9%**	93.7%	93.5%
	30%	81.2%	81.3%	81.2%	81.2%	81.2%	81.2%	81.2%	88.1%	**93.9%**	93.7%	93.5%
	40%	81.2%	81.3%	81.3%	81.4%	81.2%	81.3%	81.2%	81.9%	**93.2%**	93.1%	91.1%
BPIC20	10%	88.4%	88.5%	**88.6%**	**88.6%**	**88.6%**	**88.6%**	**88.6%**	**88.6%**	**88.6%**	88.5%	**88.6%**
	20%	88.3%	88.4%	**88.5%**	**88.5%**	**88.5%**	**88.5%**	**88.5%**	**88.5%**	**88.5%**	88.4%	88.4%
	30%	88.4%	**88.5%**	**88.5%**	**88.5%**	**88.5%**	**88.5%**	**88.5%**	88.4%	88.3%	88.2%	87.7%
	40%	87.4%	88.3%	**88.4%**	88.0%	87.7%	87.2%	87.6%	86.9%	86.8%	86.3%	85.9%

		Local Guidance										Baseline
		0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95	1
Sepsis	10%	61.1%	**73.9%**	73.2%	73.2%	73.1%	72.7%	72.7%	72.9%	72.6%	73.3%	**73.9%**
	20%	68.8%	**73.2%**	**73.2%**	73.0%	72.9%	72.7%	72.7%	72.6%	72.6%	72.5%	72.5%
	30%	**73.3%**	70.6%	71.0%	69.2%	70.3%	70.6%	70.6%	70.8%	70.4%	70.9%	68.1%
	40%	71.1%	71.8%	**72.8%**	72.5%	**72.8%**	72.7%	72.7%	72.7%	72.5%	72.7%	68.1%
BPIC13	10%	93.7%	93.7%	**93.8%**	**93.8%**	**93.8%**	**93.8%**	93.6%	93.7%	93.6%	93.4%	93.3%
	20%	**93.6%**	**93.6%**	**93.6%**	**93.6%**	**93.6%**	**93.6%**	**93.6%**	**93.6%**	**93.6%**	**93.6%**	93.5%
	30%	93.3%	93.4%	93.4%	93.3%	**93.5%**	93.4%	93.2%	93.2%	**93.5%**	**93.5%**	**93.5%**
	40%	**93.3%**	**93.3%**	93.2%	93.2%	**93.3%**	93.1%	93.2%	93.2%	93.2%	93.2%	91.1%
BPIC20	10%	88.5%	**88.7%**	88.6%	88.6%	88.6%	88.6%	88.6%	88.6%	88.6%	88.6%	88.6%
	20%	88.3%	88.3%	**88.4%**	88.3%	**88.4%**	88.3%	**88.4%**	**88.4%**	**88.4%**	**88.4%**	**88.4%**
	30%	**88.3%**	87.8%	87.8%	87.7%	87.7%	87.6%	87.6%	87.6%	87.5%	87.5%	87.7%
	40%	**87.8%**	86.9%	86.5%	86.6%	87.1%	86.9%	86.8%	86.9%	87.0%	87.2%	85.9%

Table 4: Similarity scores obtained with the LSTM model for different values of the parameter α using greedy sampling. The rightmost column reports the baseline setting ($\alpha = 1$), in which no logic-based loss is used. The best results are in bold.

inal sequences. However, Global Guidance still achieves the best similarity scores in many settings when α is properly chosen, showing that strong logical guidance can also align well with task performance. Local Guidance, instead, provides more stable similarity across different α values and noise levels, while still benefiting from the large improvements in satisfiability. Overall, the logic-based loss is essential for improving constraint satisfaction over the baseline; Global Guidance often reaches the highest peak performance, and Local Guidance offers a more consistent balance between logical correctness and similarity.

Experiments with Transformers corroborate the previously observed trends, as shown in Table 5, which reports both satisfiability and similarity for $\alpha \in \{0.25, 0.75\}$, together with the baseline. As expected, increasing the

Dataset	Noise	Satisfiability					Similarity				
		GLL		LLL		Baseline	GLL		LLL		Baseline
		0.25	0.75	0.25	0.75	1	0.25	0.75	0.25	0.75	1
Sepsis	10%	**99.9%**	98.6%	**99.9%**	**99.9%**	90.1%	**73.1%**	72.6%	72.1%	72.0%	72.6%
	20%	**99.9%**	98.7%	**99.9%**	**99.9%**	83.4%	71.1%	71.9%	71.5%	**72.0%**	71.2%
	30%	99.7%	97.3%	99.9%	**100%**	69.2%	70.1%	**72.6%**	67.0%	69.5%	68.2%
	40%	**99.1%**	95.1%	98.9%	98.6%	63.2%	69.5%	**70.8%**	68.7%	69.5%	68.3%
BPIC 2013	10%	**100%**	**100%**	**100%**	99.9%	99.6%	81.7%	**93.6%**	93.5%	93.5%	93.4%
	20%	**100%**	**100%**	99.8%	99.7%	98.5%	81.5%	92.7%	**93.6%**	93.5%	93.4%
	30%	**100%**	**100%**	98.5%	98.9%	95.8%	81.2%	88.9%	92.7%	**93.0%**	92.9%
	40%	**100%**	**100%**	98.6%	98.6%	91.6%	81.2%	86.4%	**92.9%**	**92.9%**	91.5%
BPIC 2020	10%	**100%**	**100%**	99.7%	99.7%	99.4%	88.3%	**88.6%**	**88.6%**	**88.6%**	88.5%
	20%	**100%**	99.9%	99.9%	99.9%	98.7%	**88.3%**	88.2%	87.8%	88.2%	88.2%
	30%	**100%**	99.8%	99.8%	99.9%	96.1%	**87.9%**	87.5%	87.7%	**87.9%**	87.4%
	40%	**99.9%**	98.1%	99.6%	99.6%	85.4%	**86.5%**	85.9%	85.8%	86.1%	85.1%

Table 5: Satisfiability and similarity scores obtained with the Transformer model for different values of the parameter α using greedy sampling. The rightmost column corresponds to the baseline setting ($\alpha = 1$), in which no logic-based loss is used. The best results are in bold.

noise level in the training data consistently degrades baseline performance. In terms of satisfiability, both Global and Local Guidance keep constraint satisfaction near 100% across all noise levels, with Global Guidance providing the strongest results. Similarity again exhibits more varied behavior: Local Guidance yields higher similarity in some configurations, whereas Global Guidance is generally more robust under high noise. Notably, the baseline is uniformly worse across settings, underscoring the relevance of the logical component and the overall robustness of the proposed approach.

We also report the average number of training epochs in Table 6. We observe that incorporating the logic-based loss consistently reduces the number of epochs required for convergence across all datasets, noise levels, and neural architectures, with the effect being particularly pronounced for the LSTM models. This suggests that logic-guided objectives improve optimization efficiency in terms of training iterations. For LSTM models, GLL generally requires fewer epochs than LLL, in particular for the datasets (Sepsis and BPIC 2020) that have low compliance with the background knowledge (see Table 2). Indeed, the maximum difference in training epochs is around 310 for both datasets, whereas for BPIC 2013 the maximum difference is around 25 training epochs. This suggests that, for LSTM models, GLL can be preferred for datasets with high inconsistency with the background knowledge.

Transformers generally have the effect of lowering the number of training

Dataset	Noise	LSTM			Transformer		
		Baseline	GLL	LLL	Baseline	GLL	LLL
Sepsis	10%	1327.13	**568.37**	880.07	667.33	**559.87**	571.73
	20%	1460.93	**578.07**	785.13	615.20	571.00	**562.87**
	30%	1465.93	**578.17**	801.23	646.33	**563.20**	575.07
	40%	1606.60	**589.80**	657.40	641.13	570.20	**563.07**
BPIC 2013	10%	1477.73	**562.93**	587.60	571.07	563.47	**561.20**
	20%	1565.93	**566.80**	587.80	568.53	566.80	**558.13**
	30%	1895.73	**572.00**	583.77	571.47	572.77	**564.70**
	40%	1998.80	**575.90**	601.20	571.33	578.63	**564.20**
BPIC 2020	10%	1962.20	**556.13**	598.17	620.73	**553.07**	574.63
	20%	1843.73	**570.10**	618.03	641.33	**564.07**	573.83
	30%	1943.20	**568.27**	681.23	664.40	**564.07**	568.83
	40%	1999.00	**578.40**	884.57	770.40	**566.87**	576.77

Table 6: Average number of training epochs (the lower the better) required by each method across datasets, noise levels and neural architectures. Best results are in bold.

epochs. This is not surprising because the self-attention mechanism provides direct connections between all events in a prefix, enabling faster and more effective credit assignment than the recurrent propagation through time performed by LSTM models. Here, credit assignment refers to the identification of which past inputs are responsible for the current prediction error, and therefore which parameters should be updated to correct it. In addition, the absence of recurrence, aided by residual connections and layer normalization, avoids vanishing or noisy gradients. Moreover, Transformers process the entire prefix in parallel, rather than sequentially as in LSTM models, thus allowing the network to quickly identify the most relevant past activities for predicting the future suffix. As a consequence, meaningful dependencies are learned earlier in training, leading to a lower number of training epochs. For this architecture, the difference in terms of training epochs between GLL and LLL is much more contained.

5. Related Work

Recently, there has been significant interest in employing deep neural networks (NNs) in PPM for tasks such as next-activity prediction, suffix prediction, and attribute prediction [2]. Despite significant advances in the

field, nearly all works rely on training these models solely on data, without utilizing any formal prior knowledge about the process. They mainly focus on two aspects: (i) enhancing the neural model, ranging from RNNs [3, 49, 9, 50] and convolutional neural networks (CNNs) [51] to generative adversarial networks (GANs) [52, 50], autoencoders [50], and Transformers [50]; and (ii) wisely choosing the sampling technique used to query the network at test time to generate the suffix, mostly using greedy search [3], random search [49], or beam search [9], and more recently, policies trained with reinforcement learning (RL) [53, 5]. Among all these works, only one exploits prior process knowledge [9], expressed as a set of LTL_f formulas, but it uses this knowledge only at test time, modifying the beam search sampling algorithm to select traces that are potentially compliant with the background knowledge.

In this work, we take a radically different approach by introducing a principled way to integrate background knowledge expressed in LTL_f into a deep NN model for suffix prediction at *training time*. This is based on defining a logical loss that can be combined with the loss of any autoregressive neural model and used with any sampling technique at test time, drawing inspiration from the literature in Neuro-Symbolic AI [54]. In this field, many prior works focus on exploiting temporal logical knowledge in deep learning tasks, but none have addressed multi-step symbolic sequence generation.

T-leaf [55] creates a semantic embedding space to represent both formulas and traces and uses it in tasks such as sequential action recognition and imitation learning, which do not involve multi-step prediction. In [33], an extension of Logic Tensor Networks (LTNs) [56, 8] to represent fuzzy automata is proposed and employed to integrate LTL_f background knowledge into image sequence classification tasks. STLnet [7] adopts a student–teacher training scheme in which the student network proposes a suffix based on the data, which is then corrected by the teacher network to satisfy the formula. This work uses Signal Temporal Logic (STL) formulas and is applied to continuous trajectories rather than discrete traces. Our attempts to apply it to discrete data and LTL_f formulas translated into STL yielded poor results, as the resulting STL formulas were extremely challenging for the framework to handle.

A recent line of research focuses on constraining Large Language Models (LLMs) with structured temporal knowledge, either by employing constrained beam search [10, 11, 12, 57, 17], training auxiliary models [13, 14], or exploiting conditioned sampling techniques [15, 16]. However, all these approaches are exclusively designed for test-time inference and have no in-

fluence on the training of the LLM. While this may be reasonable in the context of LLMs, where prior knowledge is often available only for specific subtasks, in PPM structured global knowledge about the process may be available *before* data collection. In such cases, incorporating this knowledge during training, rather than only at inference time, can significantly benefit the learning process.

Our work is the first to integrate temporal knowledge into the generation of multi-step symbolic sequences at training time. It is based on encoding LTL_f formulas using a matrix representation that we have previously used for very different tasks, such as learning RL policies for non-Markovian tasks [58] and inducing automata from a set of labeled traces [22], which we adapt here for use in the generative task of suffix prediction.

6. Conclusions and Future Work

This paper introduces a novel Neuro-Symbolic approach that seamlessly integrates temporal logic knowledge, expressed in LTL_f, into the training of neural suffix predictors for PPM. By combining data-driven learning with formal background knowledge, our approach achieves improved prediction accuracy and higher compliance with logical constraints, even under noisy conditions. The proposed logical loss formulations, offering both local and global perspectives, demonstrate the effectiveness and generality of the method across different real-world datasets.

Future work will focus on extending the logical loss framework to support additional types of constraints that capture diverse process dimensions, such as resources, numeric attributes, and event timestamps. We also aim to assess the method in the presence of concept drift, where process behavior evolves over time. Finally, further investigation into the synergy between local and global constraints, as well as their integration with Large Language Models, could prove promising for advancing the state-of-the-art in multi-step symbolic sequence generation.

Acknowledgments

This work was carried out while Axel Mezini and Matteo Mancanelli were enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome in collaboration with the Free University of

Bozen-Bolzano. The work of Fabio Patrizi, Elena Umili, and Matteo Mancanelli was supported by the PNRR MUR project PE0000013-FAIR. This study was funded by the European Union – NextGenerationEU, within the framework of the iNEST – Interconnected Nord-Est Innovation Ecosystem (iNEST ECS00000043 – CUP I43C22000250006). We also acknowledge financial support under the National Recovery and Resilience Plan (NRRP), M4C2I1.1, funded by the European Union – NextGenerationEU – Project Title: *aRtificial intElligence for Process Analytics (REPA)* – Grant Assignment Decree No. 2022CJWPNA by the Italian Ministry of University and Research (MUR). 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.

References

[1] C. Di Francescomarino, I. Donadello, F. M. Maggi, Predictive Process Monitoring, Springer Nature Switzerland, 2026. doi:10.1007/978-3-032-17278-5.

[2] E. Rama-Maneiro, J. C. Vidal, M. Lama, Deep learning for predictive business process monitoring: Review and benchmark, IEEE Trans. Serv. Comput. 16 (1) (2023) 739–756. doi:10.1109/TSC.2021.3139807.

[3] N. Tax, I. Verenich, M. L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: E. Dubois, K. Pohl (Eds.), Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings, Vol. 10253 of Lecture Notes in Computer Science, Springer, 2017, pp. 477–492. doi:10.1007/978-3-319-59536-8_30.

[4] G. R. Lazo, R. Ñanculef, Multi-attribute transformers for sequence prediction in business process management, in: P. Poncelet, D. Ienco (Eds.), Discovery Science - 25th International Conference, DS 2022, Montpellier, France, October 10-12, 2022, Proceedings, Vol. 13601 of Lecture Notes in Computer Science, Springer, 2022, pp. 184–194. doi:10.1007/978-3-031-18840-4_14.

[5] E. Rama-Maneiro, F. Patrizi, J. C. Vidal, M. Lama, Towards learning the optimal sampling strategy for suffix prediction in predictive monitoring, in: G. Guizzardi, F. M. Santoro, H. Mouratidis, P. Soffer (Eds.),

Advanced Information Systems Engineering - 36th International Conference, CAiSE 2024, Limassol, Cyprus, June 3-7, 2024, Proceedings, Vol. 14663 of Lecture Notes in Computer Science, Springer, 2024, pp. 215–230. doi:10.1007/978-3-031-61057-8_13.

[6] E. Giunchiglia, M. C. Stoian, S. Khan, F. Cuzzolin, T. Lukasiewicz, ROAD-R: the autonomous driving dataset with logical requirements, Mach. Learn. 112 (9) (2023) 3261–3291. doi:10.1007/S10994-023-06322-Z.

[7] M. Ma, J. Gao, L. Feng, J. A. Stankovic, STLnet: Signal temporal logic enforced multivariate recurrent neural networks, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.
URL https://proceedings.neurips.cc/paper/2020/hash/a7da6ba0505a41b98bd85907244c4c30-Abstract.html

[8] L. Serafini, A. S. d'Avila Garcez, S. Badreddine, I. Donadello, M. Spranger, F. Bianchi, Logic Tensor Networks: Theory and Applications, in: P. Hitzler, M. K. Sarker (Eds.), Neuro-Symbolic Artificial Intelligence: The State of the Art, Vol. 342 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2021, pp. 370–394. doi:10.3233/FAIA210498.

[9] C. D. 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: J. Carmona, G. Engels, A. Kumar (Eds.), Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings, Vol. 10445 of Lecture Notes in Computer Science, Springer, 2017, pp. 252–268. doi:10.1007/978-3-319-65000-5_15.

[10] V. Collura, K. Tit, L. Bussi, E. Giunchiglia, M. Cordy, TRIDENT: temporally restricted inference via dfa-enhanced neural traversal, CoRR abs/2506.09701 (2025). doi:10.48550/ARXIV.2506.09701.

[11] X. Lu, P. West, R. Zellers, R. Le Bras, C. Bhagavatula, Y. Choi, NeuroLogic decoding: (un)supervised neural text generation with predicate

logic constraints, in: K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, Y. Zhou (Eds.), Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 4288–4299. doi:10.18653/v1/2021.naacl-main.339.

[12] X. Lu, S. Welleck, P. West, L. Jiang, J. Kasai, D. Khashabi, R. Le Bras, L. Qin, Y. Yu, R. Zellers, N. A. Smith, Y. Choi, NeuroLogic A*esque decoding: Constrained text generation with lookahead heuristics, in: M. Carpuat, M.-C. de Marneffe, I. V. Meza Ruiz (Eds.), Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Seattle, United States, 2022, pp. 780–799. doi:10.18653/v1/2022.naacl-main.57.

[13] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. Joty, R. Socher, N. F. Rajani, GeDi: Generative discriminator guided sequence generation, in: M.-F. Moens, X. Huang, L. Specia, S. W.-t. Yih (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2021, Association for Computational Linguistics, Punta Cana, Dominican Republic, 2021, pp. 4929–4952. doi:10.18653/v1/2021.findings-emnlp.424.

[14] H. Zhang, P. Kung, M. Yoshida, G. V. den Broeck, N. Peng, Adaptable logical control for large language models, in: A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, C. Zhang (Eds.), Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024.
URL http://papers.nips.cc/paper_files/paper/2024/hash/d15c16cf5619a2b1606da5fc88e3f1a9-Abstract-Conference.html

[15] N. Miao, H. Zhou, L. Mou, R. Yan, L. Li, CGMH: constrained sentence generation by metropolis-hastings sampling, in: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelli-

gence, AAAI'19/IAAI'19/EAAI'19, AAAI Press, 2019. doi:10.1609/aaai.v33i01.33016834.

[16] J. Loula, B. LeBrun, L. Du, B. Lipkin, C. Pasti, G. Grand, T. Liu, Y. Emara, M. Freedman, J. Eisner, R. Cotterell, V. Mansinghka, A. K. Lew, T. Vieira, T. J. O'Donnell, Syntactic and semantic control of large language models via sequential monte carlo, in: The Thirteenth International Conference on Learning Representations, 2025.
URL https://openreview.net/forum?id=xoXn62FzD0

[17] I. Donadello, J. Ko, F. M. Maggi, J. Mendling, F. Riva, M. Weidlich, Knowledge-driven modulation of neural networks with attention mechanism for next activity prediction, CoRR abs/2312.08847 (2023). doi:10.48550/ARXIV.2312.08847.

[18] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.
URL https://openreview.net/forum?id=rkE3y85ee

[19] F. Damerau, A technique for computer detection and correction of spelling errors, Commun. ACM 7 (3) (1964) 171–176. doi:10.1145/363958.363994.

[20] E. Umili, G. P. Licks, F. Patrizi, Enhancing deep sequence generation with logical temporal knowledge, in: Proceedings of the 3rd International Workshop on Process Management in the AI Era (PMAI 2024) colocated with 27th European Conference on Artificial Intelligence (ECAI 2024), Santiago de Compostela, Spain, October 19, 2024, 2024, pp. 23–34.
URL https://ceur-ws.org/Vol-3779/paper4.pdf

[21] A. Bauer, M. Leucker, C. Schallhart, Runtime verification for LTL and TLTL, ACM Trans. Softw. Eng. Methodol. 20 (4) (2011) 14:1–14:64. doi:10.1145/2000799.2000800.

[22] E. Umili, R. Capobianco, Deepdfa: Automata learning through neural probabilistic relaxations, in: ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Com-

postela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024), 2024, pp. 1051–1058. doi:10.3233/FAIA240596.

[23] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, IEEE Computer Society, 1977, pp. 46–57. doi:10.1109/SFCS.1977.32.

[24] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI, 2013, pp. 854–860.
URL http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997

[25] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on LTL on finite traces: Insensitivity to infiniteness, in: C. E. Brodley, P. Stone (Eds.), Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada, AAAI Press, 2014, pp. 1027–1033. doi:10.1609/AAAI.V28I1.8872.

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and efficient foundation language models, CoRR abs/2302.13971 (2023). doi:10.48550/ARXIV.2302.13971.

[27] OpenAI, GPT-4 technical report, CoRR abs/2303.08774 (2023). doi:10.48550/ARXIV.2303.08774.

[28] A. van den Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, in: M. Balcan, K. Q. Weinberger (Eds.), Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, Vol. 48 of JMLR Workshop and Conference Proceedings, JMLR.org, 2016, pp. 1747–1756.
URL http://proceedings.mlr.press/v48/oord16.html

[29] T. Salimans, A. Karpathy, X. Chen, D. P. Kingma, PixelCNN++: Improving the pixelcnn with discretized logistic mixture likelihood and

other modifications, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.
URL https://openreview.net/forum?id=BJrFC6ceg

[30] A. Casolaro, V. Capone, G. Iannuzzo, F. Camastra, Deep learning for time series forecasting: Advances and open problems, Information 14 (11) (2023). doi:10.3390/info14110598.

[31] C. J. Maddison, A. Mnih, Y. W. Teh, The concrete distribution: A continuous relaxation of discrete random variables, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.
URL https://openreview.net/forum?id=S1jE5L5gl

[32] F. Fuggitti, LTLf2DFA (Mar. 2019). doi:10.5281/zenodo.3888410.

[33] E. Umili, R. Capobianco, G. De Giacomo, Grounding ltlf specifications in image sequences, in: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023, 2023, pp. 668–678. doi:10.24963/KR.2023/65.

[34] N. Manginas, G. Paliouras, L. D. Raedt, Nesya: Neurosymbolic automata, in: Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025, ijcai.org, 2025, pp. 5950–5958. doi:10.24963/IJCAI.2025/662.

[35] I. Donadello, P. Felli, C. Innes, F. M. Maggi, M. Montali, Conformance checking of fuzzy logs against declarative temporal specifications, in: Business Process Management - 22nd International Conference, BPM 2024, Krakow, Poland, September 1-6, 2024, Proceedings, 2024, pp. 39–56. doi:10.1007/978-3-031-70396-6_3.

[36] I. Donadello, P. Felli, C. Innes, F. M. Maggi, M. Montali, LTL-based conformance checking of fuzzy event logs, Process Science 2 (1) (2025). doi:10.1007/S44311-025-00020-W.

[37] S. Zhu, L. M. Tabajara, J. Li, G. Pu, M. Y. Vardi, Symbolic LTLf synthesis, in: C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 1362–1369. doi:10.24963/IJCAI.2017/189.

[38] S. Bansal, Y. Li, L. M. Tabajara, M. Y. Vardi, Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 9766–9774. doi:10.1609/AAAI.V34I06.6528.

[39] G. De Giacomo, M. Favorito, Compositional approach to translate LTLf/LDLf into deterministic finite automata, in: S. Biundo, M. Do, R. Goldman, M. Katz, Q. Yang, H. H. Zhuo (Eds.), Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021, AAAI Press, 2021, pp. 122–130.
URL https://ojs.aaai.org/index.php/ICAPS/article/view/15954

[40] M. Pesic, W. M. P. van der Aalst, A declarative approach for flexible business processes management, in: J. Eder, S. Dustdar (Eds.), Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings, Vol. 4103 of Lecture Notes in Computer Science, Springer, 2006, pp. 169–180. doi:10.1007/11837862_18.

[41] M. Pesic, H. Schonenberg, W. M. P. van der Aalst, DECLARE: full support for loosely-structured processes, in: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA, IEEE Computer Society, 2007, pp. 287–300. doi:10.1109/EDOC.2007.14.

[42] M. Westergaard, Better algorithms for analyzing and enacting declarative workflow languages using LTL, in: S. Rinderle-Ma, F. Toumani,

K. Wolf (Eds.), Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings, Vol. 6896 of Lecture Notes in Computer Science, Springer, 2011, pp. 83–98. doi:10.1007/978-3-642-23059-2_10.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 5998–6008.
URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[44] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.
URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[45] M. Janner, Q. Li, S. Levine, Offline reinforcement learning as one big sequence modeling problem, in: M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, J. W. Vaughan (Eds.), Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, 2021, pp. 1273–1286.
URL https://proceedings.neurips.cc/paper/2021/hash/099fe6b0b444c23836c4a5d07346082b-Abstract.html

[46] L. team, L. Barrault, P. Duquenne, M. Elbayad, A. Kozhevnikov, B. Alastruey, P. Andrews, M. Coria, G. Couairon, M. R. Costa-jussà, D. Dale, H. Elsahar, K. Heffernan, J. M. Janeiro, T. Tran, C. Ropers,

E. Sánchez, R. S. Roman, A. Mourachko, S. Saleem, H. Schwenk, Large concept models: Language modeling in a sentence representation space, CoRR abs/2412.08821 (2024). doi:10.48550/ARXIV.2412.08821.

[47] R. S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, Artif. Intell. 112 (1-2) (1999) 181–211. doi:10.1016/S0004-3702(99)00052-1.

[48] A. Alman, C. D. Ciccio, D. Haas, F. M. Maggi, A. Nolte, Rule mining with RuM, in: B. F. van Dongen, M. Montali, M. T. Wynn (Eds.), 2nd International Conference on Process Mining, ICPM 2020, Padua, Italy, October 4-9, 2020, IEEE, 2020, pp. 121–128. doi:10.1109/ICPM49681.2020.00027.

[49] J. Evermann, J. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decis. Support Syst. 100 (2017) 129–140. doi:10.1016/J.DSS.2017.04.003.

[50] I. Ketykó, F. Mannhardt, M. Hassani, B. F. van Dongen, What averages do not tell: predicting real life processes with sequential deep learning, in: J. Hong, M. Bures, J. W. Park, T. Cerný (Eds.), SAC '22: The 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, April 25 - 29, 2022, ACM, 2022, pp. 1128–1131. doi:10.1145/3477314.3507179.

[51] N. D. Mauro, A. Appice, T. M. A. Basile, Activity prediction of business process instances with Inception CNN models, in: AI*IA 2019 - Advances in Artificial Intelligence - XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19-22, 2019, Proceedings, 2019, pp. 348–361. doi:10.1007/978-3-030-35166-3_25.

[52] F. Taymouri, M. L. Rosa, S. M. Erfani, A deep adversarial model for suffix and remaining time prediction of event sequences, in: C. Demeniconi, I. Davidson (Eds.), Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021, SIAM, 2021, pp. 522–530. doi:10.1137/1.9781611976700.59.

[53] A. Chiorrini, C. Diamantini, A. Mircoli, D. Potena, A preliminary study on the application of reinforcement learning for predictive process

monitoring, in: S. J. J. Leemans, H. Leopold (Eds.), Process Mining Workshops - ICPM 2020 International Workshops, Padua, Italy, October 5-8, 2020, Revised Selected Papers, Vol. 406 of Lecture Notes in Business Information Processing, Springer, 2020, pp. 124–135. doi:10.1007/978-3-030-72693-5_10.

[54] T. R. Besold, A. S. d'Avila Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, G. Zaverucha, Neural-Symbolic Learning and reasoning: A Survey and Interpretation, in: P. Hitzler, M. K. Sarker (Eds.), Neuro-Symbolic Artificial Intelligence: The State of the Art, Vol. 342 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2021, pp. 1–51. doi:10.3233/FAIA210348.

[55] Y. Xie, F. Zhou, H. Soh, Embedding symbolic temporal knowledge into deep sequential models, in: IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021, IEEE, 2021, pp. 4267–4273. doi:10.1109/ICRA48506.2021.9561952.

[56] S. Badreddine, A. S. d'Avila Garcez, L. Serafini, M. Spranger, Logic tensor networks, Artif. Intell. 303 (2022) 103649. doi:10.1016/J.ARTINT.2021.103649.

[57] A. Alman, M. Comuzzi, C. Di Francescomarino, I. Donadello, F. M. Maggi, j. Oukharijane, Definitely maybe: Neuro-symbolic predictive process monitoring with probabilistic declarative knowledge, ACM Trans. Intell. Syst. Technol.Just Accepted (Apr. 2026). doi:10.1145/3810944.

[58] E. Umili, F. Argenziano, R. Capobianco, Neural reward machines, in: ECAI 2024 - 27th European Conference on Artificial Intelligence, 19-24 October 2024, Santiago de Compostela, Spain - Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024), 2024, pp. 3055–3062. doi:10.3233/FAIA240847.