SuTraN: an Encoder-Decoder Transformer for Full-Context-Aware Suffix Prediction of Business Processes

Brecht Wuyts[1], Seppe vanden Broucke[2], Jochen De Weerdt[1]

[1]*Research Center for Information Systems Engineering (LIRIS), KU Leuven, Leuven, Belgium*
[2]*Department of Business Informatics and Operations Management, Ghent University, Belgium*
brecht.wuyts@kuleuven.be, seppe.vandenbroucke@ugent.be, jochen.deweerdt@kuleuven.be

***Abstract*—Predictive Process Monitoring (PPM) in Process Mining (PM) uses predictive analytics to forecast business process progression. A key challenge is suffix prediction, forecasting future event sequences with activity labels, timestamps, and remaining runtime. Current techniques often focus on one-step-ahead predictions, rely on iterative feedback loops for suffix generation, and underutilize payload data. Additionally, many lag behind recent advancements in model architectures, sticking to LSTM-based models. Addressing these gaps, we propose *SuTraN*, a novel transformer architecture for PPM suffix prediction. *SuTraN* avoids iterative prediction loops and utilizes all available data, including event features, to forecast entire event suffixes in a single forward pass. Our approach integrates autoregressive suffix generation, data awareness, and seq2seq learning. Experimental results on real-life event logs demonstrate *SuTraN*'s superior performance in suffix prediction, highlighting its contributions often overlooked in current research.**

***Index Terms*—Process Mining, Predictive Process Monitoring, Deep Learning, Transformers, Suffix Prediction, Remaining Time Prediction**

I. Introduction

Predictive Process Monitoring (PPM) enhances Process Mining (PM) by using predictive analytics to forecast future business process trajectories. Recent advancements, including advanced machine learning models like neural networks, have significantly improved PPM's accuracy, especially in less structured processes. These advancements have led to the development of deep learning (DL)-based techniques in PPM, particularly focusing on complex tasks like suffix prediction. Suffix prediction involves forecasting future event sequences (suffix) based on the preceding events (prefix), encompassing activity labels, timestamps, and remaining runtime estimates. These predictions are crucial for resource planning, bottleneck anticipation, deadline management, and customer service optimization in various business applications.

Despite the widespread adoption of *"sequence-to-sequence"* (seq2seq) deep learning networks for generating event suffixes, few techniques are trained under this paradigm, creating a gap between training methodologies and practical deployment. This limitation, i.e. the focus on single-step prediction [1]–[5], not only leads to inaccuracies when extended to predict entire suffixes, but also prohibits the use of dynamic event features, limiting contextual understanding. Currently, only a few techniques have proposed models specifically trained to predict entire event suffixes [6]–[8]. Moreover, many approaches overlook additional payload data (e.g. [1]–[5], [7], [8]), which could enhance predictive capabilities further.

Most existing suffix prediction techniques, e.g., [1]–[6], [8], rely heavily on Long Short-Term Memory (LSTM) networks [9], a type of Recurrent Neural Network (RNN) [10]. In contrast, recent advancements in seq2seq modeling, particularly through Transformer models [11], remain largely underexplored in PPM [7], [12], [13].

This paper introduces the *Suffix Transformer Network* (*SuTraN*), a novel full-context-aware encoder-decoder transformer neural network tailored for *event suffix* and *remaining runtime* prediction in PPM. *SuTraN* eliminates iterative prediction loops by forecasting entire event suffixes in a single forward pass. By leveraging seq2seq learning, all available features of prefix events, and employing autoregressive (AR) inference, *SuTraN* enriches its contextual understanding and adaptability to evolving process sequences.

Extensive experiments evaluating *SuTraN* on four real-life event logs demonstrate its superiority over existing techniques on all prediction tasks. This evaluation not only emphasizes *SuTraN*'s high performance, but also underscores the importance of AR inference, data awareness, and seq2seq learning in achieving accurate predictions in PPM. By addressing these often overlooked aspects, our work provides valuable insights into the critical components required for effective suffix prediction.

The remainder of our paper is structured as follows: Section II reviews related work, Section III introduces *SuTraN*'s architecture and methodology, Section IV presents the experimental evaluation, Section V discusses the results, and Section VI concludes our findings.

II. Related Work

Recent advancements in PPM highlight the superiority of DL methods over classical machine learning (ML) and process model-dependent techniques across various prediction tasks, including suffix prediction [5], [6], [14], [15]. Our literature analysis reveals a proliferation of DL-based suffix prediction techniques, diverging from traditional process-model-based approaches. Notably, Table I provides a comparative analysis of existing techniques, distinguishing between *Single Event Prediction* (*SEP*) and *Complete Remaining Trace Prediction* (*CRTP*) methodologies. The table's last column categorizes prediction targets as activity suffix ('A'), timestamp suffix ('T'), remaining time ('RT'), and others ('O').

SEP techniques [1]–[5] are initially trained as sequence-to-vector models for next event prediction. Only upon inference, they adopt a seq2seq approach for suffix generation using an *iterative feedback loop* that updates event prefixes after each prediction, treating them as new instances in subsequent iterations. While this AR inference enhances contextual relevance by conditioning each suffix prediction on cumulative predictions made up to that point, it limits the use of informative event features unless they are also predicted, thereby restricting *comprehensive feature utilization*. Moreover, these models

TABLE I
COMPARISON OF MOST RELEVANT APPROACHES FOR ACTIVITY AND TIMESTAMP SUFFIX PREDICTION.

	Approaches	Sequence-sequence training	AR Inference	Comprehensive Feature Utilization	Transformer (-based)	Encoder-Decoder	Prediction Targets
SEP	Evermann et al. [3]	×	✓	×	×	×	A
	Tax et al. [5]	×	✓	×	×	×	A / T / RT
	Lin et al. [4]	×	✓	×	×	×	A / O
	Di Francescomarino et al. [2]	×	✓	×	×	×	A
	Camargo et al. [1]	×	✓	×	×	×	A / T / RT / O
CRTP	Taymouri et al. [8]	✓	✓	×	×	✓	A / T / RT
	Ketykó et al. [7]	✓	✓	×	✓	✓	A / T / RT
	Gunnarsson et al. [6]	✓	×	✓	×	×	A / RT
	SuTraN (this paper)	✓	✓	✓	✓	✓	A / T / RT

do not qualify as encoder-decoder architectures [16] because they rely on an external AR mechanism rather than generating suffixes in a single forward pass where only the decoder operates autoregressively.

All SEP techniques in Table I employ LSTM layers, each with unique characteristics. For instance, Evermann et al. [3] introduced a basic LSTM network for predicting next activities based solely on activity label prefixes. Lin et al. [4] enhanced this with a 'modulator' to weigh input sequence parts differently for each prediction. The authors in [2] improved predictions by incorporating additional knowledge of process executions during inference. Tax et al. [5] explored various shared and specialized LSTM layer combinations to predict activity labels and timestamps concurrently, with additional remaining runtime predictions derived from predicted timestamps. Camargo et al. [1] adopted similar architectures, additionally predicting roles, which were mined prior to training.

In contrast, few approaches are trained under a sequence-to-sequence paradigm explicitly tailored for suffix prediction. For example, [6] employs an LSTM-based model to predict entire activity and remaining runtime suffixes, diverging from the commonly used timestamp suffix approach. Similar to *SuTraN*, their method leverages all prefix features, including event features, but predicts entire suffixes directly rather than using an AR approach. Additionally, Taymouri et al. [8] introduced an encoder-decoder LSTM architecture where the decoder sequentially generates activity and timestamp suffixes in an AR manner, dynamically integrating context to enhance accuracy and relevance of the generated sequence. They complement supervised training with adversarial learning, showing performance gains with larger beam widths in beam search. In comparison, Ketykó et al. [7] compared a variety of sequential DL architectures for suffix prediction, including a Transformer encoder-decoder. Both [7] and [8] represent prefix and suffix events similarly, focusing solely on activity labels and timestamps without leveraging additional payload data. Our experimental evaluation highlights notable performance enhancements across all prediction tasks by fully utilizing payload data, indicating a promising shift towards comprehensive, context-aware suffix prediction in PPM.

Besides the transformer-based techniques evaluated in [7], only a few other approaches integrating transformer components have been proposed exclusively for next event prediction [12], [13], [17], all using an encoder-only architecture. However, these methods might be adapted for suffix prediction by employing the same iterative feedback loop utilized by SEP techniques.

III. METHODOLOGY

A. Preliminaries

1) Event Log Data: In PM, an *event log* $\boldsymbol{L} = \{\boldsymbol{\sigma}_i | 1 \leq i \leq |\boldsymbol{L}|\}$ records cases or instances of a business process (with $|\boldsymbol{L}|$ being the total number of cases). Each case $\boldsymbol{\sigma}_i$ can be represented as a sequence of chronologically ordered events $\langle \boldsymbol{e}_{i,1}, \dots, \boldsymbol{e}_{i,n} \rangle$, with n being the number of events executed for that particular case. For notational simplicity, the subscript i referring to the case is omitted. We define an event $\boldsymbol{e}_j$ $(\in \boldsymbol{\sigma})$ as follows:

Definition 1 (Event). *An event is a tuple* $\mathbf{e} = (a, c, t, cf_1, \dots, cf_{m_1}, ef_1, \dots, ef_{m_2})$ *with* a *the activity label,* c *the case ID,* t *the timestamp,* $cf_1, \dots, cf_{m_1}$ *(with* $m_1 \geq 0$*) the potential case features and* $ef_1, \dots, ef_{m_2}$ *(with* $m_2 \geq 0$*) the potential event features. All elements comprising the event tuple* $\mathbf{e}$ *can be accessed individually, and are denoted by means of the subscript of the event. E.g., the activity label of the j-th event* $\boldsymbol{e}_j$ $(j \in \{1, \dots, n\})$ *is denoted by* a_j*, while the timestamp of that same event is denoted by* t_j*.*

All events $\boldsymbol{e}_j \in \boldsymbol{\sigma}$ pertaining to the same case share the same case ID $(\forall j,k \in \{1,\dots,n\} : c_j = c_k)$ and the same value for all case features $(\forall j,k \in \{1 \dots n\}; \forall \alpha \in \{1,\dots,m_1\} : cf_{\alpha,j} = cf_{\alpha,k})$. In contrast, (dynamic) event features do vary between events pertaining to the same case.

2) Suffix and Remaining Time Prediction.: Furthermore, from each *(complete)* case $\boldsymbol{\sigma}$, a set of *prefix-suffix pairs* $\{\{\boldsymbol{\sigma}_k^p, \boldsymbol{\sigma}_k^s\} | 1 \leq k \leq n\}$ can be derived, with prefix $\boldsymbol{\sigma}_k^p = \langle \mathbf{e}_1, \dots, \mathbf{e}_k \rangle$ containing the first k events, and suffix $\boldsymbol{\sigma}_k^s = \langle \mathbf{e}_{k+1}, \dots, \boldsymbol{e}_n \rangle$ containing the $n-k$ last occurring events.

Definition 2 (Suffix Prediction). *The goal of suffix prediction, given a prefix* $\boldsymbol{\sigma}_k^p$*, is to predict the sequence of activities and timestamps pertaining to event suffix* $\boldsymbol{\sigma}_k^s$*; i.e.* $\langle (a_{k+1}, t_{k+1}), \dots, (a_n, t_n), (EOS) \rangle$*. The End Of Sequence (EOS) token is added during preprocessing to denote the end of a case. Additionally, remaining time prediction is concerned with predicting the total remaining time* r_k $(= t_n - t_k)$*, from the last observed prefix event* $\mathbf{e}_k$*, until case completion (*$\mathbf{e}_n$*).*

3) Preprocessing: To enable predictive algorithms to interpret timestamps as well as categorical variables, it is necessary to convert them into numerical proxies. Timestamp information is captured by deriving two numeric features: the *time elapsed since the previous event* $t_j^p = t_j - t_{j-1}$ $(\forall j = 2, \dots, n)$ and *time elapsed since case start* $t_j^s = t_j - t_1$. For the first event $\boldsymbol{e}_1$, both numerics are set to zero. Accordingly, also the timestamp suffix in Def. 2, is proxied by $\langle t_{k+1}^p, \dots, t_n^p, 0 \rangle$[1] Categorical features, including the activity label, can numerically be represented by one-hot encoded vectors. These sparse vectors can be fed to the model as-is [5], [8], or further processed by either pre-trained [1] or learnable [3], [6], [7], [17] linear projections

[1]The timestamp proxy accompanying the *EOS* token is set to 0.

(embeddings). We opted for the latter[2] (Sect. III-B). The final prefix representation presented to *SuTraN*, from hereon referred to as the *(sequence of) prefix event tokens*, can formally be defined as follows:

Definition 3 (Sequence of Prefix Event Tokens). *Suppose each event $\boldsymbol{e_j} \in \boldsymbol{\sigma}_k^p$ contains m_1^c categorical and m_1^n $(= m_1 - m_1^c)$ numerical case features, as well as m_2^c categorical and m_2^n $(= m_2 - m_2^c)$ numerical event features. The preprocessed sequence of prefix event tokens is defined as $\tilde{\boldsymbol{\sigma}}_k^p = \langle \boldsymbol{e}_1^p,\ldots,\boldsymbol{e}_k^p\rangle$, with every prefix event token $\boldsymbol{e}_j^p$ $(\in \tilde{\boldsymbol{\sigma}}_k^p)$ being represented as follows:*

$$\boldsymbol{e}_j^p = \big(\boldsymbol{a}_j, t_j^p, t_j^s, (cf_{1,j}^c,\ldots,cf_{m_1^c,j}^c), (ef_{1,j}^c,\ldots,ef_{m_2^c,j}^c), (cf_{1,j}^n,\ldots,cf_{m_1^n,j}^n), (ef_{1,j}^n,\ldots,ef_{m_2^n,j}^n)\big)$$

Continuous features, including targets, undergo standardization to a normal distribution $(\sim N(0,1))$, using mean and standard deviation from the training set, subsequently applied to the test and validation sets. For numerical features with missing values, we introduce additional binary indicator features (1 for missing, 0 otherwise), while imputing the original feature with 0. Categorical features adopt an additional `MISSING` category for absent values and an *'Out Of Vocabulary'* (`OOV`) category for all test set levels unseen in the training set.

B. SuTraN - Suffix Transformer Network

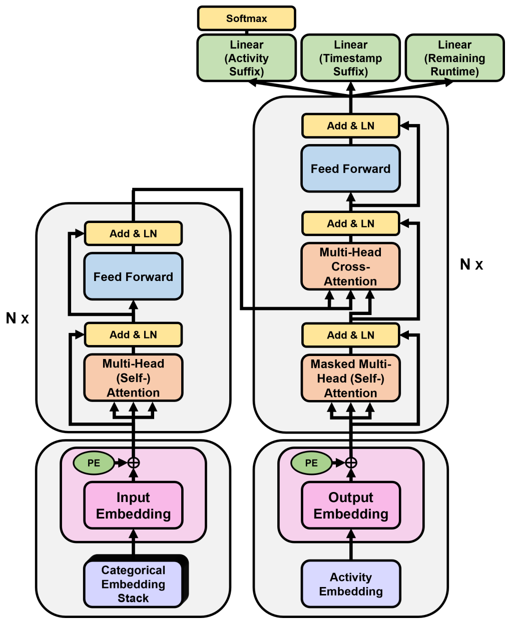
Softmax
Linear (Activity Suffix)
Linear (Timestamp Suffix)
Linear (Remaining Runtime)
Add & LN
Feed Forward
Add & LN
Multi-Head Cross-Attention
Add & LN
Masked Multi-Head (Self-) Attention
N x
Add & LN
Feed Forward
Add & LN
Multi-Head (Self-) Attention
N x
PE
Input Embedding
PE
Output Embedding
Categorical Embedding Stack
Activity Embedding

Fig. 1. *SuTraN* model architecture

The multimodal (set of) prediction(s) $\boldsymbol{\pi}(\boldsymbol{o}_k^p)$ produced by *SuTraN* (Fig. 1) when being presented with the sequence of prefix event tokens $\tilde{\boldsymbol{\sigma}}_k^p$ (Def. 3), is represented by Eq. 1.

$$\boldsymbol{\pi}(\boldsymbol{o}_k^p) = \begin{cases} \langle \hat{a}_{k+1},\ldots,\hat{a}_{k+D-1}, EOS\rangle & \textit{(1. activity suffix)} \\ \langle \hat{t}_{k+1}^p,\ldots,\hat{t}_{k+D-1}^p\rangle & \textit{(2. timestamp suffix)} \\ \hat{r}_k & \textit{(3. remaining time)} \end{cases} \quad (1)$$

SuTraN consists of two main components, the *encoder* and the *decoder*. The encoder processes and encodes each prefix event $\boldsymbol{e}_j$ $(\in \boldsymbol{\sigma}_k^p)$. It takes as input the sequence $\tilde{\boldsymbol{\sigma}}_k^p = \langle \boldsymbol{e}_1^p,\ldots,\boldsymbol{e}_k^p\rangle$ of prefix event tokens (Def. 3) and transforms it into an equal-length sequence of d_m-dimensional prefix event embeddings $\langle \boldsymbol{h}_1^{p,N},\ldots,\boldsymbol{h}_k^{p,N}\rangle$. These embeddings capture the essential features of each prefix event and serve as input for the decoder. The decoder utilizes these prefix event representations to generate predictions $\boldsymbol{\pi}(\boldsymbol{o}_k^p)$ (Eq. 1) in an AR manner. More specifically, at each consecutive decoding step $d = 1,\ldots,D$, it predicts the next suffix event $\boldsymbol{\pi}_d(\boldsymbol{o}_k^p)$ $(\in \boldsymbol{\pi}(\boldsymbol{o}_k^p))$ (Eq. 2) based on the sequence of encoder embeddings $\langle \boldsymbol{h}_1^{p,N},\ldots,\boldsymbol{h}_k^{p,N}\rangle$ and the current sequence of *suffix event tokens* $\langle \boldsymbol{e}_0^s,\ldots,\boldsymbol{e}_{d-1}^s\rangle$, which represent the suffix generated so far. At the end of each decoding step, a new *sufix event token* $\boldsymbol{e}_d^s = (\hat{a}_{k+d}, \hat{t}_{k+d}^p, \hat{t}_{k+d}^s)$ is derived from $\boldsymbol{\pi}_d(\boldsymbol{o}_k^p)$. The first suffix token, $\boldsymbol{e}_0^s$, is populated with the *activity a_k*, *time since previous t_k^p* and the *time since start t_k^s* features of the last observed prefix event token $\boldsymbol{e}_j^p$, and can be seen as the multimodal proxy for the *Start Of Sequence (SOS)* token in Natural Language Processing (NLP). This AR decoding continues until it has predicted the *EOS* token.

$$\boldsymbol{\pi}_d(\boldsymbol{o}_k^p) = \begin{cases} (\hat{a}_{k+d}, \hat{t}_{k+d}^p) & \textit{if } d > 1 \\ (\hat{a}_{k+d}, \hat{t}_{k+d}^p, \hat{r}_k) & \textit{if } d = 1 \end{cases} \quad (2)$$

Only at the first decoding step $d = 1$, an additional scalar remaining time prediction $\hat{r}_k$ is produced. Existing approaches [1], [5], [7], [8] derive $\hat{r}_k$ by simply computing the sum over the predicted timestamp suffix, i.e. $\hat{r}_k = \sum_{i=1}^{D-1} \hat{t}_{k+i}^p$. Initial experiments show that adding an additional remaining time prediction head produces significantly more accurate results compared to the sum over a suffix of timestamp predictions, of which each timestamp prediction itself is subject to errors.

This setup, in which a separate encoder first processes the entire prefix, followed by the AR decoder which at each decoding step considers both the k prefix embeddings and the suffix predicted so far, allows for the utilization of all additional features in the prefix events, including (dynamic) event features (cf. [6]), while still advantageously conditioning each consecutive suffix prediction directly on the cumulative predictions made up to that point (cf., e.g., [7], [8]). This structured and informed approach to sequential decision-making enables nuanced and context-aware predictions that are grounded in both historical *(fully data-aware prefix events)* and emergent data patterns *(suffixes generated so far)*, more closely resembling the real-life nature of how cases evolve. In what follows, both the encoder and decoder are discussed in more detail.

1) SuTraN *Components:* The *encoder* and *decoder* are highly similar. The *encoder* consists of an *Initial Prefix Embedding Block*, followed by a series of N identical *Encoder Blocks*. Similarly, an *Initial Suffix Embedding Block* and N identical *Decoder Blocks* constitute the *decoder*. Both the *Encoder* and *Decoder Blocks*, are identical to the ones introduced in [11]. The main ingredients of both the *Encoder* and *Decoder Blocks*, are the Multi-Head Attention (MHA) layers. An MHA layer updates each vector $(\in \mathbb{R}^{d_m})$ in sequence $\boldsymbol{H}^q$ $(\in \mathbb{R}^{l_q \times d_m}) = \langle \boldsymbol{h}_1^q,\ldots,\boldsymbol{h}_{l_q}^q\rangle$ by considering information from each vector $(\in \mathbb{R}^{d_m})$ in sequence $\boldsymbol{H}^k$ $(\in \mathbb{R}^{l_k \times d_m}) = \langle \boldsymbol{h}_1^k,\ldots,\boldsymbol{h}_{l_k}^k\rangle$. In case of self-attention, sequences $\boldsymbol{H}^q$ and $\boldsymbol{H}^k$ are the same.

Every subsequent *Encoder Block* l $(\in \{1,\ldots,N\})$ further updates the k prefix event embeddings received from the previous block $l-1$ into the updated sequence $\langle \boldsymbol{h}_1^{p,l},\ldots,\boldsymbol{h}_k^{p,l}\rangle$. Each one itself consists of two subcomponents: the *Multi-Head Self-Attention (MHSA)* layer, followed by a (position-wise) *Feed Forward (FF)* layer. The *MHSA* sublayer updates each prefix event embedding by allowing it to smartly focus on and learn from other relevant vectors within the same

[2]Embedding layers, converting integer-encoded categoricals into embedded vectors, are mathematically equivalent with linearly projecting their one-hot-encoded representations without a bias term.

sequence, enriching its understanding of the sequence's context and relationships. The subsequent FF layer further refines each vector individually, applying a set of transformations that enhance its representation for capturing complex patterns and relationships within the data. A residual connection is placed around each sublayer, followed by Layer Normalization (LN). Each sublayer both receives, further processes, and outputs a sequence of k d_m dimensional prefix event vectors.

Each subsequent *Decoder Block* l $(\in \{1,...,N\})$ further processes the sequence of suffix event embeddings $\langle \boldsymbol{h}_0^{s,l-1},...,\boldsymbol{h}_{d-1}^{s,l-1} \rangle$ received from the previous block, into the updated suffix event embeddings $\langle \boldsymbol{h}_0^{s,l},...,\boldsymbol{h}_{d-1}^{s,l} \rangle$. In comparison to the *Encoder Blocks*, an additional sublayer, the *Multi-Head Cross-Attention layer (MHCA)*, is included. This layer performs cross-attention, further updating each suffix event embedding with relevant information gathered from all prefix event embeddings $\boldsymbol{H}^k = \langle \boldsymbol{h}_1^{p,N},...,\boldsymbol{h}_k^{p,N} \rangle$ produced by the encoder, thereby incorporating global context from the event prefix into the generation process. Moreover, the decoder's self-attention layer (*Masked Multi-Head Self-Attention Layer*) differs from the encoder's by preventing each suffix event (embedding) from accessing information from future events in the sequence, thereby maintaining the AR property in suffix generation.

In order for the *Encoder* and *Decoder Blocks* to be able to further process the prefix and suffix event embeddings respectively, both the prefix event tokens $\langle \boldsymbol{e}_1^p, \dots, \boldsymbol{e}_k^p \rangle$ and the suffix event tokens $\langle \boldsymbol{e}_0^s,...,\boldsymbol{e}_{d-1}^s \rangle$ need to be projected into the d_m-dimensional latent space in which the *Encoder* and *Decoder Blocks* operate. This transformation is carried out by the *Initial Prefix Embedding Block* and the *Initial Suffix Embedding Block* respectively.

The *Initial Prefix Embedding Block* comprises a *Categorical Embedding Stack*, which contains, for each of the $m_1^c + m_2^c$ categorical features and the activity label, a separate learnable embedding layer with its embedding dimension d_x determined by the following commonly applied rule of thumb: $d_x = min\big(600, round(1.6c_x^{0.56})\big)$, with c_x the cardinality of that specific categorical. Following concatenation with original numeric features (including t_j^p and t_j^s), the *Input Embedding* sublayer further refines these vectors into d_m-dimensional embeddings $\boldsymbol{h}_j^{p,0}$ (Eq. 3) via a linear projection W^{input}, enriched with Positional Encoding (PE) vectors $\boldsymbol{pe}_j$ and layer normalization (LN).

$$\boldsymbol{h}_j^{p,0} = LN_0^p(W^{input}\tilde{\boldsymbol{e}}_j^p + \boldsymbol{pe}_j) \quad (\forall j = 1,...,k) \tag{3}$$

These PE vectors are the same as the ones used by [11], and serve to inject sequence order information into the otherwise order-agnostic architecture. Similarly, the *Initial Suffix Embedding Block* performs a function akin to the *Initial Prefix Embedding Block*. Given that suffix event tokens only consist of the suffix event's activity label and two time features, it first embeds the activity label (using the same weights as the activity embedding in the prefix), after which the concatenation of the embedding and the two time features is projected into a d_m-dimensional vector by the *Output Embedding* layer, resulting in the d initial suffix event embeddings $\boldsymbol{h}_j^{s,0} \in \mathbb{R}^{d_m}$ $(j \in \{0,...,d-1\}$ after adding PE vectors and applying LN.

Finally, at each decoding step $d = 1, \dots, D$, the final suffix embeddings $\boldsymbol{h}_j^{s,N}$ $(j \in 0,...,d-1)$ generated by the last *Decoder Block*, culminates in the embedding of the last generated suffix prediction $\boldsymbol{h}_{d-1}^{s,N} \in \mathbb{R}^{d_m}$ being forwarded to each of the different prediction heads for suffix forecasting. The three prediction heads for activity, timestamp and remaining time, all consisting of a single linear layer, can be represented by the learned weight matrices $W^{act} \in \mathbb{R}^{d_m \times n^a}$, with n^a the number of possible next activities, $W^t \in \mathbb{R}^{d_m \times 1}$ and $W^{rt} \in \mathbb{R}^{d_m \times 1}$ respectively. The softmax activation converts the activity output $\boldsymbol{h}_{d-1}^{s,N} W^{act}$ $(\in \mathbb{R}^{n^a})$ into a probability distribution over all n^a activities. Only the scalar runtime prediction produced at the first decoding step $(\boldsymbol{h}_0^{s,N} W^{rt})$ is retained. Notably, $\boldsymbol{h}_{d-1}^{s,N}$, uniquely integrates both the complete suffix and historical context from the encoder, ensuring *full-context* awareness for each consecutive prediction.

IV. Experimental Setup

The experiments were implemented with Python 3.10, Pytorch 2.0 and CUDA 11.0, using an NVIDIA Quadro RTX 5000 GPU with 16GB RAM. *SuTraN* was implemented using $N = 4$ encoder and decoder blocks, and embedding dimensionality $d_m = 32$, which initial experimentation indicated to be a good trade-off between model performance and computational complexity. All multi-head attention layers contained $M = 8$ parallel attention heads of dimensionality 4 $(\frac{d_m}{M} = \frac{32}{8})$. The *FF* layers all consisted of a hidden sublayer of dimensionality $d_{ff} = 128$ and ReLu activation. *SuTraN* was trained for a maximum of 200 epochs using the AdamW optimizer with a decay rate of 0.0001 and an initial learning rate of 0.0002, progressively adjusted by an exponential learning rate scheduler with a decay factor of 0.96. Early stopping was triggered if there was no improvement in the validation scores for any of the three prediction targets over 24 consecutive epochs. The final weights were chosen based on the epoch that provided the best overall trade-off in validation performance across the three prediction tasks. Teacher forcing was used for both the activity suffix and timestamp suffix predictions to streamline the learning process and highlight the model's core capabilities in its initial exploration. Furthermore, a multi-task learning approach with joint loss optimization was used.

A separate loss function was computed for each of the three targets:

1) **Activity Suffix Prediction - Categorical Cross Entropy**: Let $\hat{a}_{i,t}$ be the predicted probability for the t-th activity label in the suffix of the i-th instance, and $a_{i,t}$ be the true activity label. The cross-entropy loss for the activity suffix is given by:
$$\mathscr{L}_{\text{activity}} = -\frac{1}{\sum_{i=1}^{B} N_i} \sum_{i=1}^{B} \sum_{t=1}^{N_i} \sum_{c=1}^{C} a_{i,t,c} \log(\hat{a}_{i,t,c}) \tag{4}$$
where B $(=128)$ is the batch size, N_i is the number of events in the (ground-truth) suffix of the i-th instance, and C is the number of possible activity labels.
2) **Timestamp Suffix Prediction - Mean Absolute Error (MAE)**: Let $\hat{t}_{i,t}$ be the predicted timestamp for the t-th event in the suffix of the i-th instance, and $t_{i,t}$ be the true timestamp. The MAE loss for the timestamp suffix is given by:
$$\mathscr{L}_{\text{timestamp}} = \frac{1}{\sum_{i=1}^{B} N_i} \sum_{i=1}^{B} \sum_{t=1}^{N_i} |\hat{t}_{i,t} - t_{i,t}| \tag{5}$$
3) **Remaining Runtime Prediction - Mean Absolute Error (MAE)**: Let $\hat{r}_i$ be the predicted remaining runtime for the i-th instance, and r_i be the true remaining runtime. The MAE loss for the remaining runtime is given by:
$$\mathscr{L}_{\text{runtime}} = \frac{1}{B} \sum_{i=1}^{B} |\hat{r}_i - r_i| \tag{6}$$

Ultimately, the loss function used for multi-task training is defined by taking the unweighted sum $\mathscr{L}_{\text{batch}} = \mathscr{L}_{\text{activity}} + \mathscr{L}_{\text{timestamp}} + \mathscr{L}_{\text{runtime}}$. Standardizing the remaining time and timestamp suffix labels better aligned the MAE loss scale with the categorical cross-entropy loss. MAE, rather than Mean Squared Error, was chosen to enhance model robustness against significant timestamp and remaining time outliers in event logs.

A. Data

SuTraN is evaluated against several benchmark techniques using four real-life event logs, three of which are publicly available (BPIC17[3], BPIC17-DR, BPIC19[4]). Table II summarizes their main properties following preprocessing discussed in this subsection. Notably, the BPIC17 event log exhibits frequent immediate repetitions of the same activity. To mitigate this, we derived a BPIC17-DR *('Duplicates Removed')* version, maintaining the original process and cases but eliminating subsequent repetitions of the same activity. This preprocessing step reduced the number of events by over 400,000 and decreased the number of unique activity sequences (control-flow variants) from 14,745 to 3,592. Simplifying BPIC17-DR enhances the clarity and reliability of predictive model evaluations, suggesting that the complexity of BPIC17 may have been artificially inflated by arbitrary repetitions, potentially introducing noise into the data. The BAC event log, sourced from a large European airport's luggage handling system, cannot be publicly shared.

Each case $\boldsymbol{\sigma} = \langle e_1, \dots, e_n \rangle$ is parsed into n prefix event token sequences $\tilde{\boldsymbol{\sigma}}_k^p$ $(k \in \{1,\dots,n\})$ (see Def. 3). Each prefix is associated with an activity suffix, timestamp suffix (represented by the *time elapsed since previous event*), and scalar remaining time labels. To facilitate teacher forcing for *SuTraN* (and the *ED-LSTM* benchmark discussed in Sect. IV-B), corresponding suffix event token sequences (from Sect. III-B) were also derived for each prefix. We adopted a chronological 75-25 % out-of-time train-test split method, following the guidelines proposed by [18][5], to prevent data leakage between training and test sets. The resulting training set was further divided into final training and validation sets by assigning the last 20 % of cases to the validation set. For the BPIC19 dataset, a modified split procedure was necessary due to an observed anomaly [19] where the average throughput time declined sharply from September 2018, likely due to extraction method issues. This anomaly is believed to be attributable to the event log's extraction method. This adjustment aimed to mitigate bias in the test set caused by this anomaly[6].

Prior to prefix generation, outliers comprising the top 1.5% of cases with the highest event counts were excluded to address their disproportionate influence. Including these outliers would have required excessively large sequence lengths for training, leading to high computational costs and unnecessary complexity in model training. Numeric features and targets were standardized using means and standard deviations computed from training set prefixes rather than cases, ensuring consistent scaling across multiple distinct prefixes within each case.

B. Benchmark techniques

As highlighted in [7], [18], disparities in preprocessing and evaluation setups often complicate or prevent direct comparisons of results across studies. To ensure a fair and controlled assessment of *SuTraN*'s performance against existing techniques, and specifically to compare different modeling setups (one-step-ahead (SEP) vs. whole suffix prediction (CRTP)), network components (LSTM vs. transformer components) and context-awareness (Data Aware (DA) vs. Non-Data-Aware (NDA)), we re-implemented existing techniques, standardizing data preprocessing and scaling across all benchmarks[7].

[3]https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

[4]https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1

[5]The 'end of dataset debiasing step' recommended by the authors was omitted since the event logs used did not contain incomplete cases.

[6]See https://github.com/BrechtWts/SuffixTransformerNetwork

[7]I.e. we applied the preprocessing discussed in Sect. III-A to all benchmarks.

All DA techniques use the same prefix event representation defined in Def. 3. For NDA techniques, prefix event tokens are limited to the activity label and two time features, i.e., $(\boldsymbol{a}_j, t_j^p, t_j^s)$. Identical to *SuTraN*, categorical variables are processed using learned embeddings, which are concatenated with the numerical features. Targets specific to certain approaches, such as role prediction in [1], were excluded to isolate the performance impact attributable solely to architectural and modeling paradigm differences. This controlled experimental design minimizes external performance interference, enabling precise assessment of the factors under investigation.

Several previously proposed architectures were adapted and re-implemented based on task similarity, code availability, ease of implementation, and competitive performance. Table III provides an overview of all implementations along with their respective characteristics.

For SEP-LSTM techniques, we adopted the architecture introduced by [5], which closely resembles the multi-task model in [1], omitting the additional role prediction LSTM layer used in the latter. We trained using the best parameter settings reported by [5]. Regarding *'CRTP'* techniques specifically trained for suffix prediction, two benchmarks are implemented. The first, *'CRTP-LSTM'*, is the re-implementation of the DA technique proposed by [6], utilizing their best parameter settings for training. This technique directly generates suffixes without using an AR approach. Previous studies such as [7] and [8], employed an AR (encoder-decoder) LSTM architecture with varying numbers of LSTM layers for both encoder and decoder, and different model dimensions. Both implementations, focusing solely on timestamp and activity label information, are classified as NDA. To maintain comparability in model size across benchmarks, we implemented an *('ED-LSTM')* architecture with a four-layer LSTM encoder and decoder, and $d_m = 64$. As shown in Table III, this model still contains more than double the amount of parameters[8] compared to the NDA version of *SuTraN*. *ED-LSTM* was trained leveraging the same procedure as *SuTraN*. Finally, to assess the impact of DA accurately, both *SuTraN (NDA)* and *CRTP-LSTM (NDA)* were implemented.

All models generate predictions for activity suffix, timestamp suffix, and remaining runtime given a prefix. However, SuTraN uniquely provides explicit predictions for all three, while *SEP*- and *ED*-LSTM models explicitly generate activity and timestamp suffixes, implicitly deriving remaining runtime by summing predicted timestamps up to the index before the end-of-sequence (EOS) token. Consequently, both implementations utilize a loss function defined by the unweighted sum of the categorical cross-entropy and MAE, as specified in Equations 4 and 5, respectively. Note that for the *SEP-LSTM*, which is only trained to predict one step ahead, $N_i = 1$.

In contrast, the *CRTP-LSTM* explicitly predicts the activity and remaining runtime suffixes, using an additive loss function similar to the *ED-LSTM*, where MAE is calculated over predicted remaining runtime suffixes rather than timestamp suffixes. During inference, timestamp suffix predictions are derived from consecutive remaining runtime predictions: $\hat{t}_{k+i}^p = \hat{r}_{k+i-1} - \hat{r}_{k+i}$.

C. Evaluation

All AR models (*SuTraN*, *ED-LSTM*, and *SEP-LSTM*) iteratively generated activity and timestamp suffixes. At each decoding step, the predicted activity label and timestamp were used to update the

[8]The number of learnable parameters depends on the event log used. The ones reported in Table III are computed for the BPIC17-DR.

TABLE II
PROPERTIES OF THE EVENT LOGS INCORPORATED IN THE EXPERIMENTAL COMPARISON.

Event Log Name	Cases	Events	Variants	Activities	Mean - Standard Deviation Case Length	Mean - Standard Deviation Case Duration	Event Features	Case Features
BPIC17	30,078	1,109,665	14,745	25	36.90 − 14.55	20.52 − 10.81 (days)	8	3
BPIC17-DR	30,078	704,202	3,592	25	23.42 − 6.85	20.52 − 10.81 (days)	7	3
BPIC19	181,395	986,077	5,767	39	5.44 − 1.78	71.76 − 36.78 (days)	2	11
BAC	362,563	1,767,186	13,496	57	4.87 − 2.49	732 − 912.24 (seconds)	1	7

TABLE III
OVERVIEW OF MODELS INCLUDED IN THE EXPERIMENTAL COMPARISON.

Benchmark	Parameters	AR Inference	seq2eq	Encoder - Decoder	DA	Explicit RT prediction	Explicit timestamp prediction
SEP-LSTM	213726	✓	×	×	×	×	✓
CRTP-LSTM *(NDA)*	98459	×	✓	×	×	✓	×
CRTP-LSTM	117375	×	✓	×	✓	✓	×
ED-LSTM	241771	✓	✓	✓	×	×	✓
SuTraN (NDA)	121004	✓	✓	✓	×	✓	✓
SuTraN	125723	✓	✓	✓	✓	✓	✓

suffix event tokens, or in the case of SEP-LSTM, to create a new prefix event token. This iterative process continued until an *EOS* token was predicted for the activity suffix or until reaching a predefined maximum length, typically the longest case length in the dataset post-preprocessing. If this length was exceeded, an *EOS* token was automatically appended to the predicted activity suffix. Timestamp predictions of decoding steps past the *EOS* were padded with 0*s* (or the standardized 0-equivalent). Moreover, at each decoding step, the next activity label was chosen greedily, i.e. the one with the highest predicted probability was chosen, a method also applied in *CRTP-LSTM*.

Evaluation metrics include the (normalized) Damerau-Levenshtein Similarity (DLS) for activity suffix prediction and mean absolute error (MAE) for timestamp suffix and remaining time prediction. DLS measures sequence similarity normalized between 0 to 1, with 1 indicating identical sequences, and 0 completely dissimilar sequences. It is calculated as: $DLS(X,Y) = 1 - \frac{DL(X,Y)}{\max(|X|,|Y|)}$, with $DL(X,Y)$ the Damerau-Levenshtein distance between the sequences X and Y, and $\max(|X|,|Y|)$ the maximum length of both sequences. MAE for timestamp and runtime predictions, computed similarly to Eq. 5 and 6, respectively, scales by the number of test instances (B). Prior to MAE computation, predictions and labels for timestamp suffix and remaining runtime were de-standardized using the training set mean and standard deviation. The MAE is expressed in minutes unless otherwise specified.

V. RESULTS

Table IV shows the performance of *SuTraN* across the four event logs. The best results are in bold and underlined, while the second-best are only in bold. *SuTraN* consistently outperforms benchmarks in remaining time prediction and excels in three out of the four event logs for activity suffix and timestamp suffix prediction, except for *BPIC17*, where *CRTP-LSTM* achieves a higher DLS score, while *ED-LSTM* marginally surpasses *SuTraN* and its NDA version for timestamp suffix prediction.

Data awareness (DA) significantly improves model performance, with DA models (i.e., *SuTraN* and *CRTP-LSTM*) consistently outperforming their NDA counterparts in both activity suffix and remaining time prediction. This advantage is also observed for timestamp suffix prediction, although with two minor exceptions.

SuTraN's AR approach gives it an advantage over *CRTP-LSTM* in activity suffix prediction for three out of four event logs, excluding BPIC17. A similar, though less pronounced, pattern is observed for their NDA versions in two out of four event logs, with a tie on the BPIC19 event log.

Furthermore, *SuTraN* uniquely incorporates explicit training for remaining runtime, activity suffix, and timestamp suffix prediction (Table III). This comprehensive multi-task training approach proves advantageous, as evidenced by its superior performance in remaining runtime prediction across all logs and in timestamp suffix prediction for all but one log, where it remains highly competitive.

The superiority of CRTP techniques (*SuTraN, ED-LSTM, CRTP-LSTM*) over SEP techniques (*SEP-LSTM*) is clearly reflected in the results. *SEP-LSTM* generally performs worst, ranking lowest five times, closely followed by *CRTP-LSTM (NDA)*, ranking lowest four times. This underscores the critical importance of both DA suffix prediction and AR suffix generation. Specifically, making the CRTP-LSTM data-aware rather than non-data-aware significantly boosts its performance, moving it from generally the second worst performing model to the second best. The AR suffix generation models, including *SuTraN* (and its NDA counterpart) and *ED-LSTM*, perform better than non-AR models, as *SuTraN* implementations avoid the lowest ranking distinctions entirely, and *ED-LSTM* receiving it only twice, likely due to its reliance on timestamp predictions to derive remaining runtime, which inherently introduces more error.

Comparing *SuTraN (NDA)* with *ED-LSTM* underscores the transformer's potential for suffix generation in PPM. Despite having less than half the number of parameters, *SuTraN (NDA)* consistently competes well in activity suffix prediction and timestamp suffix prediction, while surpassing *ED-LSTM* in remaining runtime prediction (Table III).

As highlighted in [7], right-skewed trace lengths in event logs suggest that average metrics alone may not suffice for comparing DL architectures in PPM. Consequently, Fig. 2 illustrates activity suffix (DLS) and remaining time (MAE) prediction metrics as functions of prefix and suffix lengths. Each point on the plot represents the average DLS or MAE for test instances with specific prefix or suffix lengths, offering insights into how the models' performances vary with the number of observed events and the number of suffix events to be predicted, respectively. The right vertical axis displays the number of instances pertaining to each length.

Intuitively, one might expect performance to improve with longer prefixes and decrease with shorter suffixes. However, as observed in [7], this trend is not consistent especially with longer prefixes. These extended cases often feature repetitive activities lacking clear structure, questioning their representativeness in modeling and whether models should adapt to or remain resilient against such anomalies.

Moreover, these outliers are significantly underrepresented in both training and test sets, skewing performance evaluations, particularly for the longest prefixes and suffixes. Instances derived from these outlier cases additionally exert a disproportionate influence on average metrics across all lengths, contrasting with more typical case lengths that only affect measures up to their respective lengths. Concerns about unusually short cases also arise, albeit less prominently due to

TABLE IV
PERFORMANCE COMPARISON ACROSS DIFFERENT TECHNIQUES AND DATASETS.

	Damerau-Levenshtein Similarity (DLS)				MAE Timestamp Suffix Prediction				MAE Remaining Runtime Prediction			
	BPIC17-DR	BPIC17	BPIC19	BAC	BPIC17-DR	BPIC17	BPIC19	BAC (sec.)	BPIC17-DR	BPIC17	BPIC19	BAC (sec.)
SEP-LSTM	0.6733	0.2160	0.8425	0.7206	**1178**	762	16604	113	10139	11823	30572	420
CRTP-LSTM (NDA)	0.6525	0.3357	0.8435	0.7320	1391	1009	17182	113	8931	8906	29323	318
CRTP-LSTM	**0.6741**	**0.4095**	**0.8522**	**0.8374**	1556	996	**15708**	112	**8000**	**8685**	**21345**	**301**
ED-LSTM	0.6737	0.3239	0.8477	0.7424	1200	**739**	16485	**108**	9705	12160	31000	338
SuTraN (NDA)	0.6723	0.2669	0.8435	0.7355	1201	**745**	16621	109	8896	8860	29209	308
SuTraN	**0.7274**	**0.3843**	**0.8699**	**0.8461**	**1157**	749	**14542**	**106**	**7727**	**7913**	**20182**	**290**

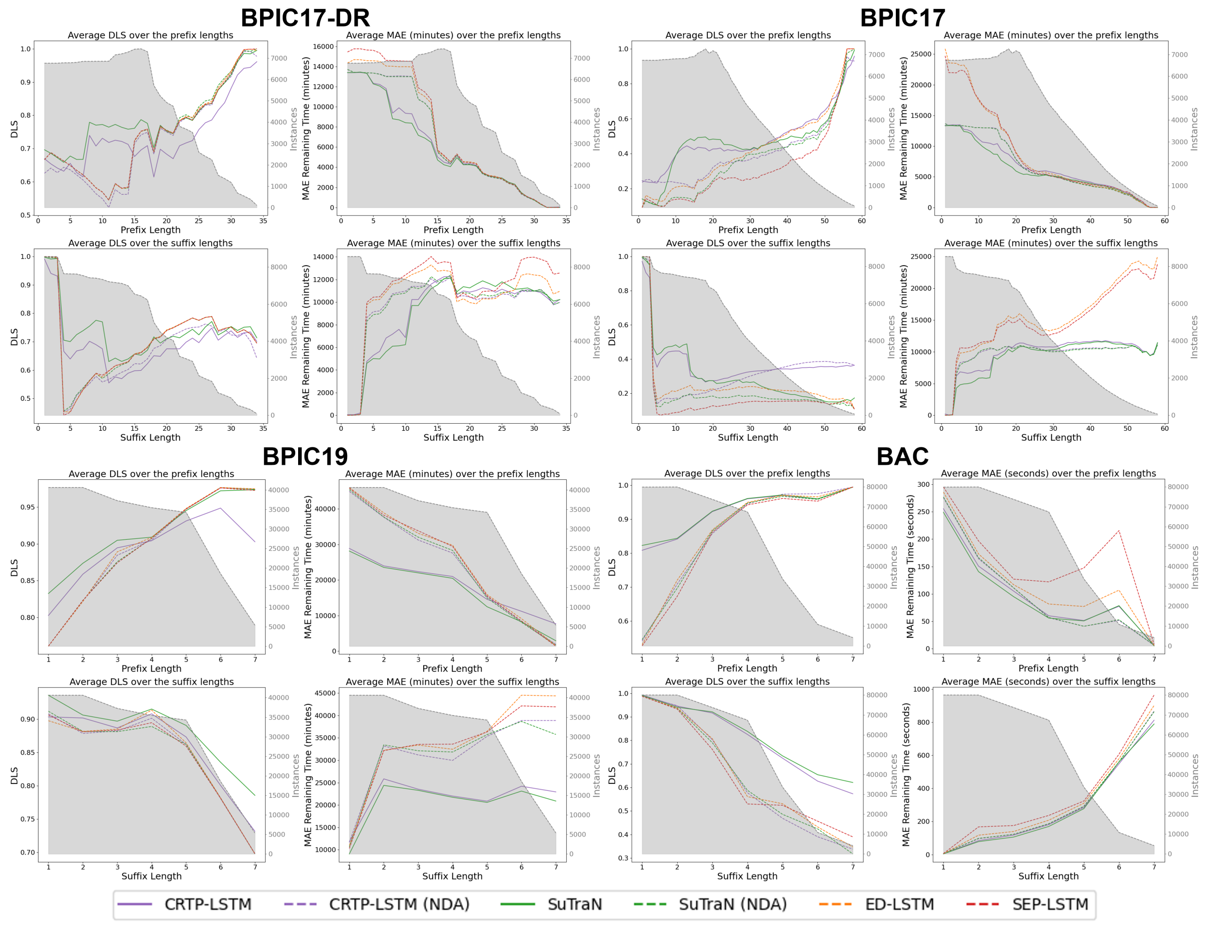

Fig. 2. Performance metrics in function of prefix and suffix lengths.

their lower representation in the right-skewed case length distribution. To mitigate these issues and ensure a representative assessment of model performance across varying prefix and suffix lengths, we focus on test instances derived from cases within the 5th and 90th percentiles of case lengths.

Analysis of Fig. 2 reveals several interesting patterns. DA models consistently excel in predicting activity suffixes from early stages (shorter prefix lengths), contrasting with NDA models that typically catch up only as more events unfold (higher prefix lengths). This trend is consistent across all event logs. Conversely, while NDA models perform competitively with shorter suffixes, their performance deteriorates more rapidly as suffix lengths increase compared to DA techniques. This indicates that DA models produce higher quality activity suffixes across longer sequences.

A notable pattern emerges in the DLS-suffix plot of the BPIC17-DR log, where DLS scores initially increase with suffix length before decreasing. In contrast, in the BPIC17 log, primarily the CRTP-LSTM's DLS scores increase with suffix length, likely explaining its high DLS score reported in Table IV. The longer cases in BPIC17 often feature numerous consecutive repetitions of the same activity. Given CRTP-LSTM's non-AR activity suffix generation, it likely overfits on these repetitions, getting stuck in them during generation, which the DLS metric unfortunately rewards for noisy suffixes.

Similar patterns appear in the MAE evolution for remaining runtime predictions. DA models consistently obtain lower remaining time errors for shorter prefixes, leveraging comprehensive data inputs. Conversely, NDA models exhibit faster degradation in prediction accuracy as suffix length increase. This effect is least pronounced in the BAC event log, which exhibits significant variability in throughput times relative to the mean (Table II). Factors such as flight delays, strikes, and holidays, not captured in the data, likely contribute to this variability. Finally, explicit remaining runtime prediction is particularly advantageous, as shown by the MAE versus suffix length plots for the BPIC17 and BPIC19 event logs. *SuTraN* and *CRTP-LSTM*, as well as their NDA counterparts, report lower MAE values for longer suffix lengths compared to *SEP-* and *ED-LSTM*. The latter two techniques' remaining time predictions rely not only on the sum of timestamp suffix predictions—each subject to prediction errors—but also on the correctness of the *EOS* in their predicted activity suffix, which could be predicted too early or too late. As the quality of predicted activity suffixes degrades with longer suffix lengths, this further deteriorates the remaining time predictions for these techniques. This impact is most notable in the BPIC17 event log, characterized by repetitive and noisy activity patterns in longer cases. In contrast, *SuTraN*, *CRTP-LSTM*, and their NDA counterparts maintain relatively stable MAE values across varying suffix lengths.

VI. Discussion & Conclusion

In this study, we introduced *SuTraN*, a full-context-aware encoder-decoder transformer network designed for multi-task suffix prediction in PPM. *SuTraN* addresses limitations of existing techniques by forecasting entire event suffixes in a single forward pass, uniquely integrating seq2seq learning, AR suffix generation, explicit remaining runtime prediction, and comprehensive data awareness.

To ensure a rigorous assessment of *SuTraN*'s performance, we re-implemented various existing techniques from scratch, maintaining uniformity in data preprocessing and scaling across all models. This standardized approach enabled a thorough comparison against the existing literature, highlighting the performance gains facilitated by *SuTraN*'s integrated features. Our results demonstrate *SuTraN*'s superiority in jointly predicting remaining time, and activity and timestamp suffixes. Our findings highlight the importance of AR decoding, data awareness, seq2seq learning, and explicit remaining time predictions—features often overlooked in existing techniques but uniquely integrated within *SuTraN*—as key factors for superior performance in PPM tasks. *SuTraN* excels across these dimensions, outperforming benchmarks on all tasks, showcasing its effectiveness and robustness.

Moreover, our analysis of performance across varying prefix and suffix lengths revealed that DA models exhibit significantly higher accuracy early in processes (short prefix lengths) and for instances requiring the generation of longer suffixes. Explicit remaining runtime prediction, in particular, proves advantageous for scenarios involving longer suffix lengths and event logs with complex control-flows, where implicit prediction methods fail to maintain accuracy due to their reliance on less accurate timestamp suffix predictions.

While our results suggest the potential and suitability of transformers for PPM, decisive conclusions regarding the superiority of transformer-based versus LSTM-based models require further investigation. Future research could explore this further by designing controlled experiments tailored to compare these architectures comprehensively across different logs and their specific characteristics.

For transparency and collaboration within the research community, we have made the code for this study, including all re-implementations, publicly available at https://github.com/BrechtWts/SuffixTransformerNetwork. The repository features meticulous documentation, comprehensive explanations of preprocessing steps, detailed descriptions of all re-implementations, and supplementary materials aimed at facilitating reproducibility and advancing the field of PPM.

References

[1] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate lstm models of business processes," in *Business Process Management*, T. Hildebrandt, B. F. van Dongen, M. Röglinger, and J. Mendling, Eds. Cham: Springer International Publishing, 2019, pp. 286–302.

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

[3] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decision Support Systems*, vol. 100, pp. 129–140, 2017, smart Business Process Management. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923617300635

[4] L. Lin, L. Wen, and J. Wang, "Mm-pred: A deep predictive model for multi-attribute event sequence," in *Proceedings of the 2019 SIAM international conference on data mining*. SIAM, 2019, pp. 118–126.

[5] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Proceedings of the 29th International Conference on Advanced Information Systems Engineering*. Springer, 2017, pp. 477–492.

[6] B. R. Gunnarsson, S. v. Broucke, and J. De Weerdt, "A direct data aware lstm neural network architecture for complete remaining trace and runtime prediction," *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 2330–2342, 2023.

[7] I. Ketykó, F. Mannhardt, M. Hassani, and B. F. van Dongen, "What averages do not tell: predicting real life processes with sequential deep learning," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1128–1131. [Online]. Available: https://doi.org/10.1145/3477314.3507179

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

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[12] Z. A. Bukhsh, A. Saeed, and R. M. Dijkman, "Processtransformer: Predictive business process monitoring with transformer network," 2021.

[13] P. Philipp, R. Jacob, S. Robert, and J. Beyerer, "Predictive analysis of business processes using neural networks with attention mechanism," in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2020, pp. 225–230.

[14] W. Kratsch, J. Manderscheid, M. Röglinger, and J. Seyfried, "Machine learning in business process monitoring: A comparison of deep learning and classical approaches used for outcome prediction," *Business & Information Systems Engineering*, vol. 63, no. 3, pp. 261–276, Jun 2021. [Online]. Available: https://doi.org/10.1007/s12599-020-00645-0

[15] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, and I. Teinemaa, "Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, jul 2019. [Online]. Available: https://doi.org/10.1145/3331449

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

[17] J. Moon, G. Park, and J. Jeong, "Pop-on: Prediction of process using one-way language model based on nlp approach," *Applied Sciences*, vol. 11, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/2/864

[18] H. Weytjens and J. De Weerdt, "Creating unbiased public benchmark datasets with data leakage prevention for predictive process monitoring," in *Business Process Management Workshops*, A. Marrella and B. Weber, Eds. Cham: Springer International Publishing, 2022, pp. 18–29.

[19] B. Wuyts, H. Weytjens, S. vanden Broucke, and J. De Weerdt, "DyLoPro: Profiling the dynamics of event logs," in *Business Process Management*, C. Di Francescomarino, A. Burattin, C. Janiesch, and S. Sadiq, Eds. Cham: Springer Nature Switzerland, 2023, pp. 146–162.