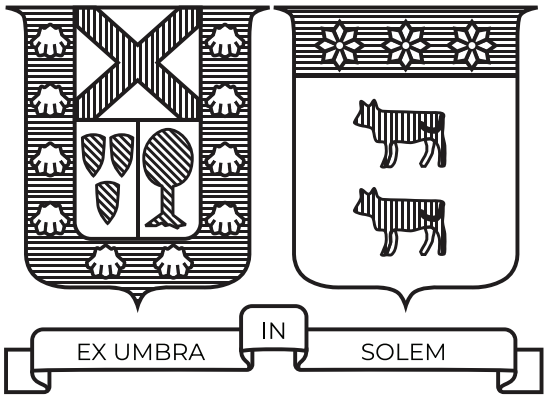
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso, Chile



Business process tasks prediction with Multi-Attribute Transformers

Gonzalo Rivera Lazo

Tesis para optar al Grado de
Magíster en Ciencias de la Ingeniería Informática

Profesor Guía: Dr. Hernán Astudillo
Profesor Co-Guía: Dr. Ricardo Ñanculef A.
Profesor Correferente Interno: Dra. Raquel Pezoa
Profesor Correferente Externo: Dr. Marcos Sepúlveda
Presidente Comisión:

24 de noviembre de 2023

Acknowledgements

I would like to express my deeply grateful to my parents, whose unwavering support and encouragement have been the cornerstone of my journey.
To my esteemed professors, your guidance and wisdom have been invaluable.
To my friends, your camaraderie and understanding have brightened the path.

Abstract

Predicting the evolution of an ongoing process is a challenging task in Business Process Management (BPM). An approach from Process Mining (PM) is to use event logs from information systems as a historical source of patterns. Recurrent neural networks have shown promise in sequence prediction but have limitations. State-of-the-art models with higher accuracy consider not only activity sequences but also other event attributes. Meanwhile, Transformer-based models have become the standard in sequence modeling, processing sequences in a single step. Some studies suggest using Transformer variants for this task, focusing solely on event activity information. In this work, we propose using Transformer-based models for predicting ongoing processes in the BPM domain, incorporating Multi-Attribute Transformers to leverage activity sequences as well as performer and timing information. We evaluate various architectures for encoding and integrating these attributes and explore multi-task variants for predicting the next activity, when it will occur, and which resource(s) will be assigned to it. Moreover, we propose an encoder-decoder architecture for sequence-to-sequence (seq2seq) tasks and apply it to predict the complete trace of events following the ongoing process. We evaluate two methods to merge the representations of the encoded attributes. Our experiments on three real datasets show that Multi-attribute Transformers perform better than Transformers using only previous process activities in two out of three datasets in terms of accuracy and Damerau-levenshtein similarity.

Resumen

Predecir la evolución de un proceso en curso es un desafío en la Gestión de Procesos de Negocio (BPM). Un enfoque de la Minería de Procesos (PM) es usar registros de eventos de sistemas de información como fuente histórica de patrones. Las redes neuronales recurrentes han mostrado promesa en la predicción de secuencias, pero tienen limitaciones. Los modelos de vanguardia con mayor precisión consideran no solo secuencias de actividades sino también otros atributos de eventos. Mientras tanto, los modelos basados en Transformadores se han convertido en el estándar en la modelización de secuencias, procesando secuencias en un solo paso. Algunos estudios sugieren usar variantes de Transformadores para esta tarea, centrándose únicamente en la información de actividad del evento. En este trabajo, proponemos usar modelos basados en Transformadores para predecir procesos en curso en el dominio de BPM, incorporando Transformadores Multi-Atributo para aprovechar secuencias de actividades, así como información sobre quién las realiza y cuándo. Evaluamos varias arquitecturas para codificar e integrar estos atributos y exploramos variantes multitarea para predecir la próxima actividad, cuándo ocurrirá y qué recurso(s) se asignará(n) a ella. Además, proponemos una arquitectura de codificador-decodificador para tareas de secuencia-a-secuencia (seq2seq) y la aplicamos para predecir la traza completa de eventos siguiendo el proceso en curso. Evaluamos dos métodos para fusionar las representaciones de los atributos codificados. Nuestros experimentos en tres conjuntos de datos reales muestran que los Transformadores Multi-Atributo tienen un mejor desempeño que los Transformadores que usan solo actividades de procesos anteriores en dos de los tres conjuntos de datos evaluados con accuracy y similitud de Damerau-levenshtein.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last few decades, there has been a noticeable surge in the adoption of digital transformation across various industries. This phenomenon has become almost ubiquitous, with organizations seeking to leverage the benefits of modern technology and tools to enhance their operations. One crucial element that plays a pivotal role in driving this transformation within an organization is the implementation of information systems that supports and streamlines both internal and external processes. Within the information systems, user interactions take center stage in the form of various events, as they encompass a wide array of activities performed by individuals as they engage with the system. These events, when systematically captured and recorded, give rise to what is commonly referred to as *event logs*. The event logs essentially serve as a digital trail, meticulously documenting the real-time execution of various business processes within the organization. The significance of event logs extends beyond mere documentation. They hold immense potential for knowledge discovery and can be harnessed for engineering applications in the realm of business process management (BPM). In essence, the data encapsulated within the logs can be analyzed and leveraged to gain valuable insights into the intricacies of organizational processes [3] as decision-making, process optimization and overall business strategies. This thesis proposed to use Transformer based models for two predictive business process tasks: *Next event prediction* and *Trace prediction*. These models can exploit multiple event log attributes such as: activities, resources, and time stamps to make better and more informed predictions. Further, an exploration of encoding and fusion methods is perform in order to effectively execute a robust aggregation of these attributes.

Additionally, in the *Next event* prediction, multi-task variants which address the task of predicting not just the next activity but other attributes attached to it, including the resource it could trigger and its expected timestamp are assessed. Through experiments on two of three public benchmark datasets it can be demonstrated that multi-attribute Transformers are competitive or better than existing recurrent models. Further, they inherit the intrinsic advantages of Transformers, i.e., they allow significantly more parallelism during inference and lead to more in-

terpretable predictions thanks to attention weights matrices that experts can inspect and visualize. Moreover, the proposed multi-attribute models outperform literature Transformers-based methods that only use information about previous activities.

Then, this thesis provide an extension of the next event prediction methods to address the problem of predicting the entire trace of activities that follows the ongoing process. This sequence-to-sequence problem is solved by using Transformer based models that fuse attribute information through cross-attention layers at the decoding stage. To address this fusion, two methods by which a decoder can attend to the features computed by different encoders are evaluated. For this task, experiments reveal that our sequence-to-sequence architecture improves four previous methods based on recurrent networks. Lastly, visualizations of the learned attention weights that illustrate the model's potential to obtain more interpretable predictions that are valuable to the business are presented and set as future research.

1.1 Problem Context

Numerous process mining tools are available to extract valuable insights from log data, enabling tasks like pinpointing bottlenecks and decision points within processes. However, some of these tools do not provide *operational support* which means they don't have the capability to offer real-time recommendations to users as processes unfold. This limitation is highlighted in a study by Fernandes et al. [12]. To address this gap, researchers have turned to statistical and machine learning methods which aim to predict the next step within an ongoing or partially completed process. By doing so, they empower organizations to make more informed decisions and potentially optimize their operations in real-time. Still, the tasks of predicting the time until the next event and the resources associated with the following activities have received little attention. Predicting the time it will take for the next event to occur is crucial in dynamic processes where timing is essential. It can help organizations ensure that processes stay on track and meet their performance targets. Furthermore, understanding the resources needed for upcoming activities is vital for effective resource allocation, cost control, and overall process efficiency. Despite these advantages, there has been limited research in these areas, and they present opportunities for further investigation.

While deep learning methods like recurrent neural networks have outperformed traditional machine learning and statistical methods in ongoing process monitoring tasks, they also involve a higher computational complexity. One such approach gaining prominence is the use of Transformers, that offer more parallelism during training, thereby potentially enhancing efficiency. Regrettably, the methods that use Transformer-based architectures in process prediction tasks, primarily focus on attributes related only to the activities information, neglecting other attributes. While, prior research in the area has indicated that attributes such as the resources employed and timestamps are valuable pieces of information for enhance prediction, i.e. pre-

dictive models that incorporate this contextual information tend to outperform their counterparts that do not consider such attributes [27]. This finding underscores the significance of including a broader set of process-related features in the modeling process. Several other domains, such as time-series forecasting and neural machine translation, have also witnessed significant improvements by leveraging context-rich models [8, 28].

1.2 Hypothesis

The hypothesis explored in this study is as follows:

The incorporation of multiple attributes into the Transformer-based model provides it with valuable process information, thereby enhancing its performance for both the prediction of the subsequent event and the entire sequence.

1.3 Objectives

1.3.1 General Objective

1. *"Design, implement and evaluate Transformer-based model for predicting business processes that incorporates multiple attributes across the sequence of events, competitive with state-of-the-art methods"*

1.3.2 Specific Objectives

1. To develop a summary of the state of the art.
2. To create model variations designs based on Transformers for the prediction of business processes integrating multiple attributes.
3. To implement model variations based on Transformers for the prediction of business processes integrating multiple attributes.
4. To evaluate the model variations based on Transformers for performing prediction tasks according to specific metrics.
5. To Conduct a comprehensive comparative analysis against state-of-the-art methods using three real datasets.

1.4 Structure

The remainder of the thesis is structured as follows. First, Section 1.1 provides basic definitions related to the domain problem. While the technical background requires

to formulate our architectures is presented in Section 2. Then, Section 3 presents related work relevant to the proposal. Section 4 details both proposed methods, the next event prediction and the trace prediction tasks. Section 5 describes the experimental setup, a baselines comparison, and a discussion of the results in Presented in Section 6. Finally, Section 7 summarizes the findings and contributions, concluding this thesis and outlining future work.

Chapter 2

Background

This section presents essential BPM concepts, formalizes the problem(s) addressed in this thesis, and provides the technical background required to formulate our methods.

2.1 Business Process Management Concepts

Business Process Management (BPM) is a holistic approach to improving and optimizing an organization's business processes to achieve enhanced efficiency, effectiveness, and agility. It encompasses a set of methodologies, techniques, and tools aimed at designing, modeling, implementing, monitoring, and continually improving business processes [42].

At its core, BPM focuses on streamlining workflows, reducing operational inefficiencies, and aligning processes with the organization's strategic goals [9]. It seeks to enhance the overall performance of an organization by providing a systematic framework for:

1. *Process Design and Modeling*: The goal is to identify and document existing processes, often using graphical representations like flowcharts or process diagrams. This step allows stakeholders to visualize the current state and identify areas for improvement.

2. *Process Automation*: BPM advocates for the automation of repetitive and rule-based tasks within processes. This automation can be achieved through Business Process Management Systems (BPMS) or workflow automation tools, reducing errors and processing times.

3. *Process Monitoring*: Continuous monitoring of processes is a crucial aspect of BPM. Organizations employ Key Performance Indicators (KPIs) and process analytics to track the performance of processes in real-time. This enables proactive intervention in case of deviations from expected outcomes.

4. *Process Optimization*: Based on the data collected during monitoring, BPM enables organizations to identify bottlenecks, inefficiencies, and areas for improvement. Process redesign and optimization efforts aim to enhance the efficiency and effectiveness of processes.

5. *Process Governance*: As it emphasizes the importance of clear ownership, accountability, and compliance within processes, it helps in ensuring that processes adhere to legal and regulatory requirements.

BPM plays a pivotal role in enhancing an organization's adaptability to changes in the business environment. It enables agility by facilitating rapid adjustments to processes in response to evolving market dynamics, customer demands, or regulatory changes.

2.1.1 Process Mining

Process Mining is a specialized discipline within BPM that leverages event data to gain insights into actual process execution, rather than relying solely on modeled or assumed processes [38]. It provides a data-driven approach to process analysis and improvement. Process Mining encompasses three primary steps:

1. *Data Extraction*: Event logs, generated by information systems during process execution, are extracted and prepared for analysis. These logs typically contain information about activities, timestamps, resources, and the order of events.

2. *Process Discovery*: Process Mining algorithms analyze event logs to automatically discover and visualize the real process flows. Process discovery techniques, such as Petri nets or process trees, provide insights into how processes are actually executed.

3. *Conformance Checking and Enhancement*: Once the actual process is discovered, it can be compared to the intended or modeled process. Discrepancies are identified, allowing organizations to address deviations and optimize processes for better alignment with organizational goals.

By integrating Process Mining with BPM, organizations gain a deeper understanding of their operational processes and can make data-driven decisions to enhance efficiency, compliance, and customer satisfaction. Further, this thesis is proposed to be within the scope of the *Process Monitoring* goal and include deep learning methods as part of the support approach to predictive monitoring. Subsequently, the concepts associated with the problem being addressed and applied within this domain will be elaborated upon in greater detail.

A business process is a series of coordinated activities carried out at a particular moment in a technical or organizational context [35]. These operations could employ one or more organizational resources and involve internal and external entities.

We define an *event* e_i as the execution of an activity related to a business process in the organization. An event can be characterized by m attribute values as such as activity ID, resource, and completion timestamp

$$e_i = \{a_{i1}, a_{i2}, \dots a_{im}\} \tag{2.1}$$

Besides, we define a *case* as an entire instance of a business process, that is, a sequence of l_j events sorted chronologically and linked by a unique identifier usually called *Case ID* as

$$c^{(j)} = \langle e_1^{(j)}, e_2^{(j)}, \dots, e_{l_j}^{(j)} \rangle \tag{2.2}$$

Finally, we define an *event log* E as a collection of p cases produced by the execution of different business processes instances and recorded into an information system.

$$E = \{c^{(1)}, c^{(2)}, \dots c^{(p)}\} \tag{2.3}$$

2.2 Next Event Prediction

Giving an incomplete case $c_{1:h} = \langle e_1, e_2, \dots, e_h \rangle$ representing an ongoing process from which we know only h events, *Next event prediction* is the task of estimating the event e_{h+1} following $c_{1:h}$, with all its m attribute values as illustrated in Fig 2.1.

Length	Case ID	Activity	Start Time	Name	Age	Resource
1	2314	Arrival	2020-12-03 05:07	-	-	-
2	2314	Registration	2020-12-03 05:27	F. Taylor	25	I.Son
3	2314	Diagnosis	2020-12-03 05:40	F. Taylor	25	J. Yun
4	2314	Blood Test	2020-12-03 06:26	F. Taylor	25	F. Paz
h+1	2314	Blood Result	2020-12-03 07:04	F. Taylor	25	F. Paz

Figure 2.1: Incomplete Case 2314 from an Urgency process event log.

2.3 Trace Prediction

Similarly, we could define *Trace prediction* (also refer as *Suffix prediction*) as the task of predicting the sequence of events $\langle e_{h+1}, e_{h+2}, \dots e_{l(c)} \rangle$ following $c_{1:h}$. However, in this thesis, only the prediction of activity IDs ($e_i = \{a_{i1}\}$) based on information from past events is addressed. A detailed explanation is presented in section 4.2.

Note that $l(c)$, the length of c after completion, is unknown. Therefore, models for Next event prediction and Trace prediction should be able to trigger a particular

Length	Case ID	Activity	Start Time	Name	Age	Resource
1	2314	Arrival	2020-12-03 05:07	-	-	-
2	2314	Registration	2020-12-03 05:27	F. Taylor	25	I.Son
3	2314	Diagnosis	2020-12-03 05:40	F. Taylor	25	J. Yun
4	2314	Blood Test	2020-12-03 06:26	F. Taylor	25	F. Paz
h+1	2314	Blood Result	2020-12-03 07:04	F. Taylor	25	F. Paz
h+2	2314	Presciption	2020-12-03 09:23	F. Taylor	25	J. Yun
c	2314	Departure	2020-12-03 09:37	F. Taylor	25	-

Figure 2.2: Incomplete Case 2314 from an Urgency process event log.

state or symbol if the current event is the last. In addition, the *prefix length* h can vary from case to case from a minimum of h_* to $l(c) - 1$. To address the next event and trace prediction problems, we assume we have an event log E representing the past execution of processes at the organization.

2.4 Deep Learning

Deep Learning is a family of machine learning models that uses multiple interconnected layers of computation nodes to compute non-linear transformation on data [21]. By training its layers' parameters, this architecture can learn to detect features and patterns that relate input data x with some desired output y and thus allow making predictions on novel data (e.g., classification or regression).

Applications where the input or output data are sequences (e.g., natural language processing) require specialized architectures. Recurrent neural networks (RNNs) have been long used in these cases [15]. RNNs process an input sequence $\langle x_T \rangle = x_1, x_2, \ldots, x_T$ one element at a time, updating an internal state h_t that keeps a trace of all the sequence information up to time step t. A recurrence relation determines the next state h_{t+1} as function of the previous state h_t and the data x_t. The hidden state $h_T = \text{RNN}(\langle x_T \rangle)$ obtained after processing the last input token x_T allows the model to make predictions about the sequence.

However, as Bengio et al. detailed in [4], training these models on large sequences can lead to exploding and vanishing gradients, which causes the model to lose information about the input as it gets processed. Then, a limitation of RNNs (trained through backpropagation) is that only recent inputs and hidden states significantly influence the model's predictions. Therefore, learning long-term dependency patterns becomes difficult or impossible [31]. Additionally, recurrent computations can

be inefficient on long sequences.

2.5 Transformers

To circumvent RNNs limitations, Transformers [40] substitute recurrence by *attention mechanisms*, as shown in Fig **??**. In a nutshell, attention mechanisms allow a model to select and combine pieces of data (e.g., words in a sentence) to solve a prediction task, focusing on a part of the available information while ignoring the rest [29]. In sequence learning, an attention-based model can dynamically (and sensibly) choose elements from the past and the future of the current time step, shortening the effective time period between two sequence elements. This ability allows the model to discover complex patterns in long sequences with limited computational resources.

To support sequence-to-sequence transformations in which the input and output data are sequences such as neural machine translation [2], the Transformer architecture includes two different subnets: an *encoder* that maps an input sequence to a sequence of internal states, and a *decoder* that maps that sequence to the desired output sequence. Attention mechanisms based on Eq. (2.6) are included in both components. By stacking attention mechanisms of this type, encoder and decoder can generate high-level representations of sequences without recurrent connections. Indeed, the mechanism can access any subset of positions of the input sequence simultaneously, without a preferred direction of information flow, allowing encoder and decoder to capture wide-range dependencies among their input elements.

Transformer's Multi-head Attention

Given a sequence of feature vectors $\mathcal{V} = \{v_1, v_2, \ldots, v_{N_v}\}$ computed at a given layer of the model, an attention mechanism computes a set of context vectors $\mathcal{H} = \{h_1, h_2, \ldots, h_{N_h}\}$ by recombining the elements of $\mathcal{V}$ as

$$h_n = \sum\nolimits_{m=1}^{M} \alpha(q_n, k_m) v_m \,, \tag{2.4}$$

where k_m is a vector referred to as *a key*, q_n is a vector referred to as *a query*, and $\alpha(q_n, k_m) \in [0, 1]$ is the *attention weight* assigned by the attention mechanism to v_m. In this abstraction, the query encodes an information need, while the key k_m is a descriptor of the information contained in v_m. By arranging the values, keys, and queries as rows of matrices $V \in \mathbb{R}^{N_v \times d_v}$, $K \in \mathbb{R}^{N_v \times d_k}$, and $Q \in \mathbb{R}^{N_h \times d_q}$, respectively, the Transformer can compute all the contextualized representations $\mathcal{H}$ in a single step, as the rows of the matrix

$$H = \mathit{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \,, \tag{2.5}$$

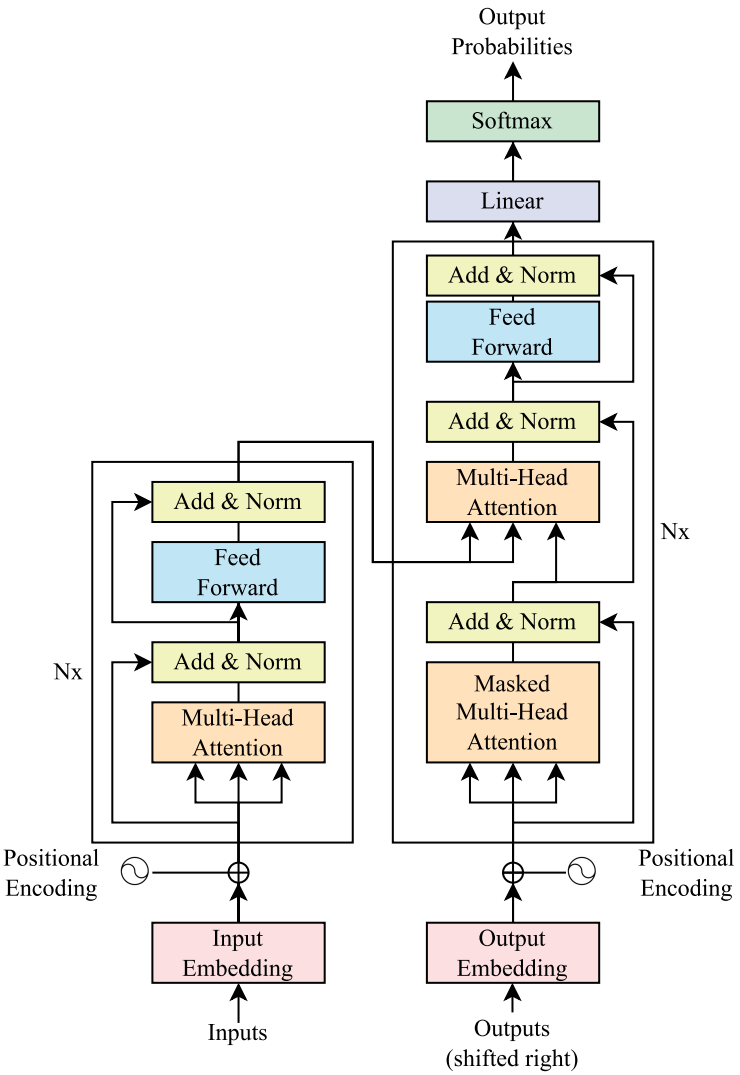


Figure 2.3: Transformer architecture presented by Vaswani et al. [40]

where d_k is the column dimension of K. Typically, values, keys, and queries are first projected into a common subspace using embedding matrices W^Q, W^K, W^V so that we can set d_k and d_h without constraints. In addition, to increase flexibility and support more parallelism, the attention mechanism is allowed to use multiple heads, i.e., M_h independent sets of attention weights led to N_H different sets of context vectors for the same query q_j. The Transformer concatenates all these representations into a single position-wise representation and makes a final projection (parametrized by W^O) that allows controlling the dimension d_m of the fused context vectors. Putting all together, the final output of a multi-head attention mechanism

is given by

$$H^{\oplus} = \text{Multi-Head}(Q, K, V) = \text{Concat}(H^{(1)}, ..., H^{(M_h)})W^O$$
$$\text{where } H^{(h)} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\,, \tag{2.6}$$

is the output of the h-th head as is illustrated in Fig. 2.4.

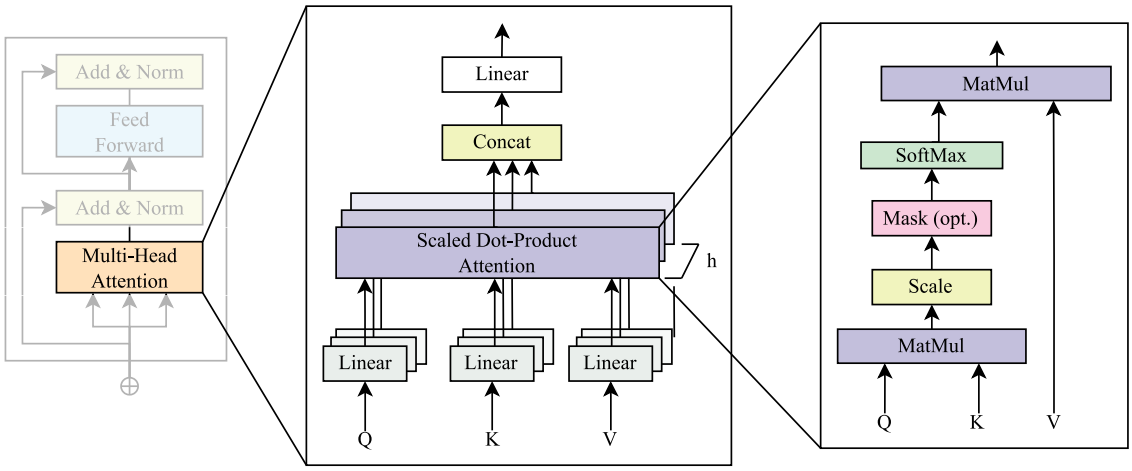


Figure 2.4: Overview of the Transformer's Multi-Head Attention and the Scaled dot-product attention within each *head* [40].

Positional Encoding

Positional encoding is a fundamental concept in the architecture of Transformer models, which have revolutionized the field of natural language processing (NLP) and achieved remarkable success in various sequence-to-sequence tasks such as language translation and text generation. This encoding method addresses a critical challenge in these tasks: preserving the order of elements in a sequence, like words in a sentence, when there is no inherent notion of order in the model's self-attention mechanism.

Positional encoding is typically represented as a fixed-length vector that is added to the embeddings of input elements. The positional encoding scheme applied in the trace prediction task utilizes a combination of trigonometric functions, such as sine and cosine, to encode positional information. For a given position in the sequence, the positional encoding vector is calculated as follows:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_\text{m}}}\right)$$
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_\text{m}}}\right) \tag{2.7}$$

Here, pos represents the position of the element, i represents the index within the positional encoding vector, and d_m is the dimensionality of the model's embeddings.

This scheme ensures that each position has a unique positional encoding, and the encoding values for different positions are dispersed across the dimensions of the embeddings.

Additionally, in the *Next event* prediction tasks, a simple encoding is evaluated that include learnable parameters to make the representation of each token sensitive to its position in the input sequence. The fixed sinusoidal encoding is also evaluated in the initial experiments, but due to the low performance difference with respect the learnable encoding it is not considered for the rest of the experiments for this task. Nevertheless, in order to implement the original sequence-to-sequence Transformer architecture, the sinusoidal encoding is considered in the *Trace Prediction* task.

Transformer Encoder

This component first process the input sequence $x_{1:N_x} = \langle x_1, x_2, \ldots, x_{N_x} \rangle$ through positional encoding [40]. In the case of categorical inputs, the encoder's first layer also includes an embedding layer that maps each token x_i to a continuous feature vector of dimension d_e. The outputs of these initial transformations are arranged into a matrix $E_0 \in \mathbb{R}^{N_x \times d_e}$. Starting from E_0, the encoder then recursively computes $M_e = 6$ additional transformations of the form $E_k = \text{EBlock}(E_{k-1})$, where $\text{EBlock}(\cdot)$ will be referred to as an *encoder's block*. Each block encompasses two transformations referred to as layers: a *multi-head self-attention mechanism* followed by a position-wise feed-forward net. The self-attention mechanism is an attention mechanism based on Eq. (2.6), where $Q = K = V = E_{k-1}$. In addition, each block's layer includes residual connections [14] and a layer normalization [1]. For convenience, we denote the final transformation implemented by the Transformer encoder as

$$\begin{aligned} \text{TEnc}(x_{1:N_x}) = E_{M_e} \text{ where } E_k = \text{EBlock}(E_{k-1}) \ \forall k = 1, \ldots, M_e \\ \text{and } E_0 = \text{EInit}(x_{1:N_x}) \end{aligned} \tag{2.8}$$

Therefore, the output of an encoder's block is given by

$$\begin{aligned} E_k = \text{EBlock}(E_{k-1}) = \text{LayerNorm}\left(\text{FeedForward}(\tilde{E}_k) + \tilde{E}_k\right) \\ \text{where } \tilde{E}_k = \text{LayerNorm}\left(\text{Multi-Head}(E_{k-1}, E_{k-1}, E_{k-1}) + E_{k-1}\right) . \end{aligned} \tag{2.9}$$

Transformer Decoder

This component starts by encoding the desired output sequence $y_{1:N_y} = \langle y_1, \ldots, y_{N_y} \rangle$ through an embedding layer and positional encoding, just as the encoder does with $x_{1:N_x}$. The resulting vectors are arranged as rows of a matrix $D_0 \in \mathbb{R}^{N_y \times d_e}$. Starting from D_0, the decoder then computes a cascade of $M_d = 6$ transformations of the form $D_k = \text{DBlock}(D_{k-1}, E_{M_e})$, where E_{M_e} is the final encoder's output and $\text{DBlock}(\cdot)$ will be referred to as a *decoder's block*. In contrast to the encoder's blocks, the

decoder's blocks include three layers. The first is a *multi-head self-attention mechanism* analogous to the encoder's self-attention mechanism. A significant difference here is that the self-attention mechanism is modified to preserve the auto-regressive property, i.e., to prevent row j from attending subsequent positions in the decoder's input sequence $j+1, \ldots, N_y$. The second layer is *multi-head cross-attention mechanism* where the queries come from the previous layer, and the keys and values come from the output of the encoder E_{M_e}. The last layer is a position-wise feed-forward net [40]. As in the encoder, all the block's layers include residual connections [14] and a layer normalization [1]. For convenience, we denote the final transformation implemented by the Transformer decoder as

$$\begin{aligned} \text{TDec}(y_{1:N_y}, E_{M_e}) = D_{M_d} \text{ where } D_k &= \text{DBlock}(D_{k-1}, E_{M_e}) \ \forall k = 1, \ldots, M_d \\ \text{and } D_0 &= \text{EInit}(y_{1:N_y}) \end{aligned} \tag{2.10}$$

To summarize, the output of a decoder's block is given by

$$\begin{aligned} D_k = \text{DBlock}(D_{k-1}) &= \text{LayerNorm}\left(\text{FeedForward}(\widetilde{D}_k) + \widetilde{D}_k\right) \\ \text{where } \widetilde{D}_k &= \text{LayerNorm}\left(\text{Multi-Head}(\tilde{D}_{k-1}, E_{M_e}, E_{M_e}) + \tilde{D}_{k-1}\right), \\ \text{and } \tilde{D}_{k-1} &= M \odot \text{LayerNorm}\left(\text{Multi-Head}(D_{k-1}, D_{k-1}, D_{k-1}) + D_{k-1}\right), \end{aligned} \tag{2.11}$$

where M denotes the masking matrix.

2.6 Adam Optimizer

According to Kingma et al. [19] Adam is an optimization algorithm designed for efficiently handling the optimization of stochastic objective functions based on first-order gradients. It employs adaptive estimates of lower-order moments to adaptively adjust the learning rates for each parameter. Adam is known for its simplicity in implementation, computational efficiency, low memory requirements, and its suitability for large-scale problems involving extensive data and parameters. Additionally, it is robust when dealing with non-stationary objectives and problems featuring noisy or sparse gradients. The hyperparameters in Adam have intuitive interpretations and typically require minimal tuning. It has demonstrated its practical effectiveness and competitiveness in comparison to other stochastic optimization methods.

In this thesis, a modification is applied to the Adam Optimizer in order to used the same setting of the original Transformer by [40].

Chapter 3

Literature Review

Traditional statistical methods as Hidden Markov models (HMM) [20], Bayesian techniques [22], Annotated transition systems (ATS) [39], among others [33], have been proposed to solve predictive process monitoring. Classical machine learning classification methods such as Support Vector Machines (SVM) and Random Forest have also been used in process prediction [37] [18]. Unfortunately, these models often have failures supporting long sequences.

A systematic literature review by [41], which tested various models, concluded that Recurrent Neural Networks outperformed other methods on at least 13 of 16 datasets. Recently, another systematic literature review [27] has shown that deep models have higher accuracy than traditional machine learning methods in sequence modelling.

This study also emphasized the importance of attribute encoding when supplying models with diverse attribute types. In essence, the *next event* prediction task can be modeled using the language modeling formula $P(W_t|W_{\text{t-k}}, ..., W_{\text{t-1}})$, where the word W is substituted with an activity A. Consequently, it's not unexpected that certain methodologies previously proven effective in the natural language processing domain have found application in process prediction as well. These include techniques such as onehot encoding, ngrams, word embedding, and recurrent neural networks.

In predictive process monitoring, [10] used an embedding technique for categorical attributes and a shallow LSTM. The method of [36] shares a similar architecture but applies directly one-hot encoding. [6] also evaluated the benefit of including event attributes (as a triplet of activities, resources, and timestamps) into the model through different concatenation strategies.

Lin et al. [24] combined only categorical attributes with an LSTM encoder-decoder architecture and proposed aggregating a Modulator layer between them, outperforming the other methods that predict the next activity as depicted in Fig. 3.1.

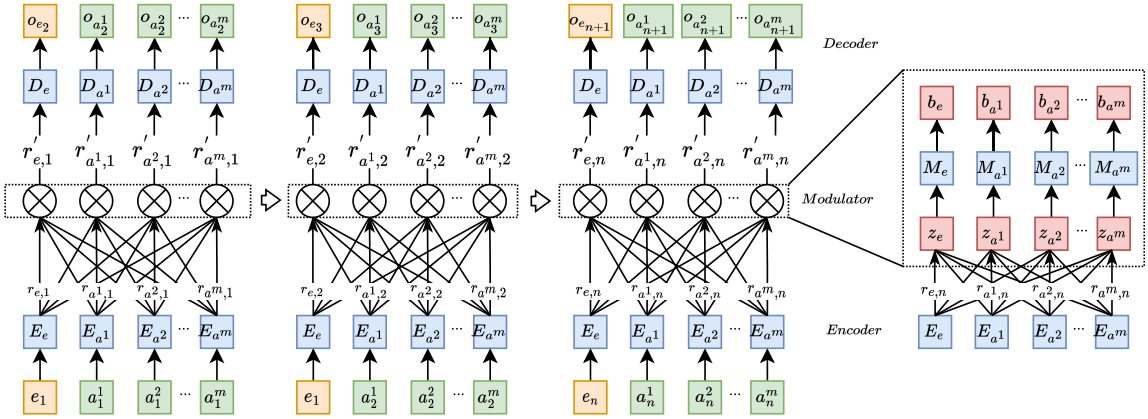


Figure 3.1: MM-Pred architecture proposed by Lin et al. al.

The Modulator layer receives a vector $\mathbf{r} = [r_{a1}, r_{a2}, ..., r_{am}]$ of hidden representations of categorical attributes and then applies element-wise operations to extract inter-dependencies across attributes as shown in Equation 3.1. In contrast to the original paper which uses an LSTM encoder for each attribute to produce the hidden representations, we propose to use the Transformer encoder to feed with representations to the Modulator layer to take advantage of the inter-dependencies identification feature.

$$\begin{aligned} \boldsymbol{z}_e &= [\boldsymbol{r}_{e,n} \circ \boldsymbol{r}_{a^1,n}, \cdots, \boldsymbol{r}_{e,n} \circ \boldsymbol{r}_{a^m,n}, \boldsymbol{r}_{e,n}, \boldsymbol{r}_{a^1,n}, \cdots] \\ \boldsymbol{z}_{a^1} &= [\boldsymbol{r}_{a^1,n} \circ \boldsymbol{r}_{e,n}, \cdots, \boldsymbol{r}_{a^1,n} \circ \boldsymbol{r}_{a^m,n}, \boldsymbol{r}_{e,n}, \boldsymbol{r}_{a^1,n}, \cdots] \\ &\vdots \\ \boldsymbol{z}_{a^m} &= [\boldsymbol{r}_{a^m,n} \circ \boldsymbol{r}_{e,n}, \cdots, \boldsymbol{r}_{a^m,n} \circ \boldsymbol{r}_{a^{m-1},n}, \boldsymbol{r}_{e,n}, \boldsymbol{r}_{a1,n}, \cdots] \end{aligned} \tag{3.1}$$

The attention mechanism proposed by [16] generates a context vector by weighting information pieces from the input sequence. Later, [17] expanded this method to accept multiple categorical variables using a hierarchical attention approach as illustrate Fig 3.2. Wickramanayake et al. [43] proposed to extract explanations from this model, visualizing the attention weights corresponding to a given output.

Lastly, the ProcessTransformer presented in [5] uses a minimal pre-processing step and encodes only the activity sequence into a Transformer to perform three tasks separately: predicting the next activity, the next timestamp, and the remaining time of an incomplete trace. The POP-ON method proposed in [26] only uses the activity sequence in the Multi-head Attention layer and concatenates all the other attributes after the encoder representation is generated. Besides, it applies an embedding technique to encode the attributes. The method by [32] encodes activity sequences using only sin and cos positional embedding. Therefore, attention-based models with attributes other than the activities do have been explored. However, previous works have not presented results on predicting the resource or the triplet (activity, resource & timestamp) following an incomplete sequence.

Altought a plethora of methods have been proposed for sequence-to-sequence problems in Deep Learning, just a few works have addressed this problem in business process monitoring. [10] uses a simple LSTM to encode activity sequences. While, [36] proposed a two-layer LSTM that combine the activity and timestamp sequences using a shared layer strategy. Further, [24] reuse the same architecture in the next activity task explained above. All works mentioned use an explicit iterative approach that aggregates the predicted tokens to the input at each step in the inference mode. [6] also explored the random choice in the activity selection in order to generate an heterogerous event log besides of *argmax*, i.e. selecting the highest probability. Despite [16] proposed to used Bi-LSTM to generate sequences suffixes, its evaluation method correspond rather to the generation of event logs than to predict the rest of the traces. Nevertheless, these recurrent architectures suffer when predicting long sequences and implies a lot of effort to parallelize their computation.

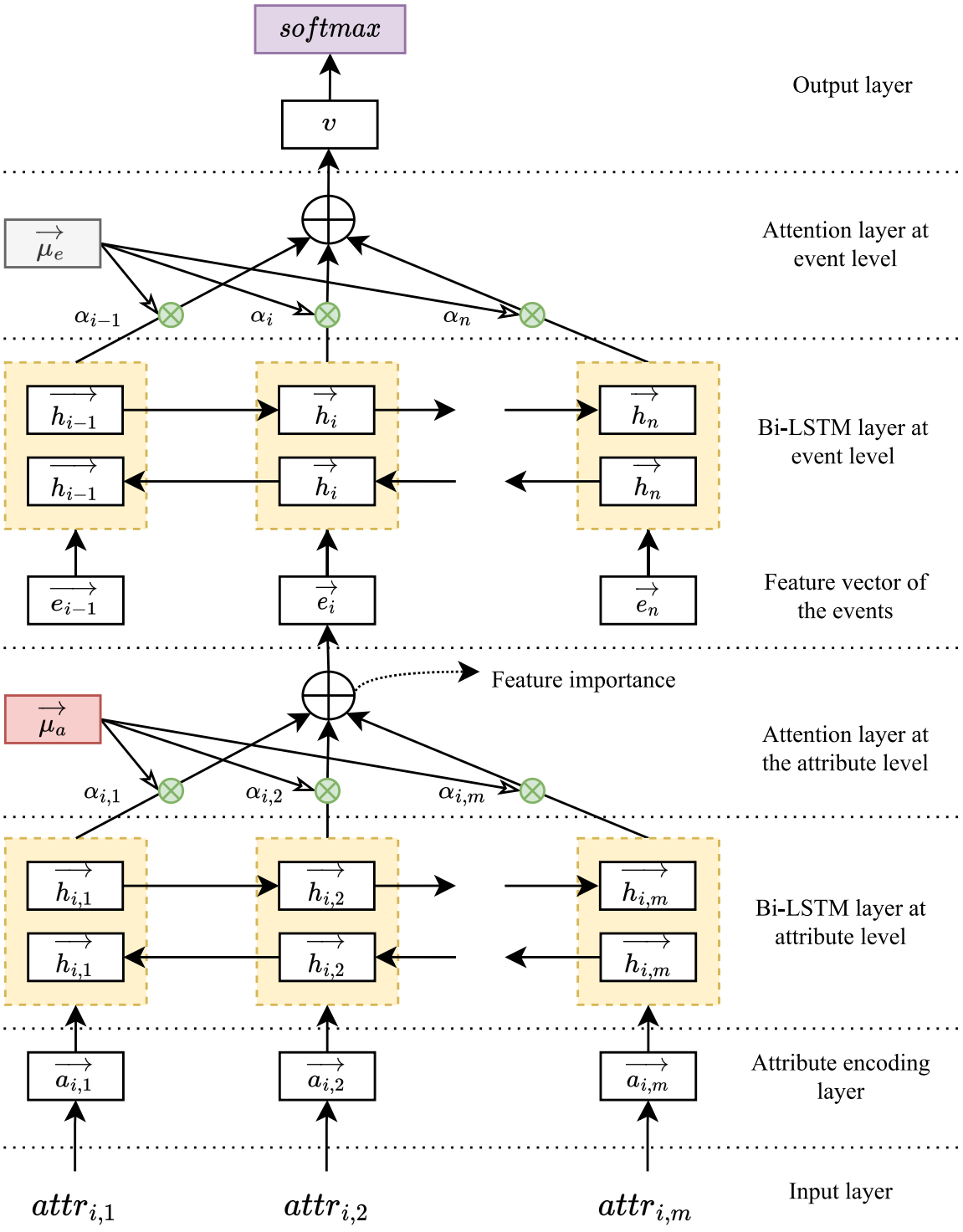


Figure 3.2: HAM-Net architecture proposed by Jalayer et al.

Chapter 4

Proposed Model

We first present the architectures to handle the Next event prediction task. Then, we formulate sequence-to-sequence extensions to handle the Trace prediction task. For both tasks, we predict the process's evolution using a neural net with two main components: a *process encoder* and *process decoder*. The process encoder uses one or more Transformer encoders to map an incomplete multi-attribute case $c_{1:h} = \langle e_1, e_2, e_3, ..., e_h \rangle$ into a high-level representation $E(c_{1:h})$. The decoder transforms this latent representation into the desired prediction. As the hyper-parameters of the multiple models refer to the adjustable settings that govern the behavior and performance of the models, different representations can be generated by each of these hyper-parameters.

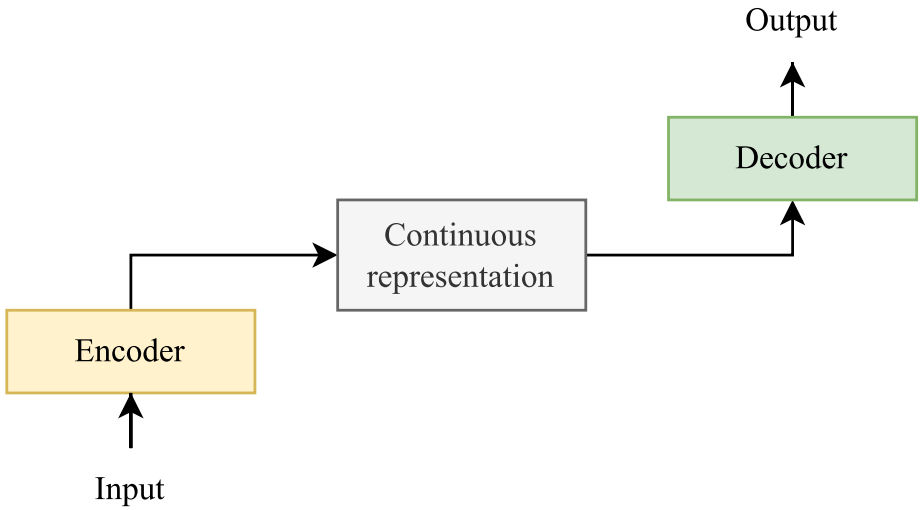


Figure 4.1: Encoder-Decoder general architecture.

4.1 Architectures for Next Event Prediction

For this task, the process encoder produces a vector representation of $c_{1:h}$, i.e., $E(c_{1:h}) \in \mathbb{R}^{d_o}$. The decoder has to transform $E(c_{1:h})$ into a prediction for each attribute associated with the next event e_{h+1}. For each event in the input case, we can have categorical and continuous attributes. We assume the existence of at least three raw attributes: activity ID (categorical), resource ID (categorical), and (continuous) time base features handcrafted from timestamps. Below we present encoder architectures that incrementally integrate these (and possibly other) attributes to obtain $E(c_{1:h})$. We also present decoder architectures that differ in how they share parameters to predict the different attributes.

4.1.1 Encoder Architectures

Baseline Encoder

Our *baseline encoder* obtains $E(c_{1:h})$ using activity IDs only. The input to this model is thus a sequence of h categorical values $s_a = \langle a_1, a_2, \ldots a_h \rangle$. We use a Transformer encoder to process this sequence and obtain a matrix of contextualized representations for each activity (see Sec. 2.5 for details)

$$E^{(\mathrm{A})} = \mathrm{TEnc}(s_a) \in \mathbb{R}^{h \times d_o} . \tag{4.1}$$

We then use Global Average Pooling (GAP) to compress the feature map into a global semantic representation of the input sequence. For any feature map $E \in \mathbb{R}^{H \times d_o}$ this operation is defined as

$$\mathrm{GAP}(E) = \sum_j E_{j,:} \in \mathbb{R}^{d_o} . \tag{4.2}$$

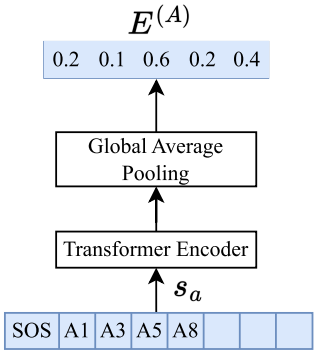


Figure 4.2: Baseline Encoder.

Combining Categorical Attributes

We evaluate two methods to integrate categorical attributes such as resources and activities: an *early fusion* approach and a *late fusion* approach. We implement early fusion through an architecture called *One-Tr* that concatenates the raw token sequences and feeds the resulting sequence into a Transformer encoder. Formally, if $s_a = \langle a_1, a_2, \dots a_h \rangle$ denotes the sequence of activity IDs and $s_r = \langle r_1, r_2, \dots, r_h \rangle$ denotes the sequence of resource IDs, the Transformer encoder is fed with $s_a \oplus s_r := \langle a_1, a_2, \dots a_h, r_1, r_2, \dots, r_h \rangle$, and produces the feature map

$$E^{(\text{A+R})}_{\text{One-Tr}} = \text{TEnc}(s_a \oplus s_r) \in \mathbb{R}^{2h \times d_o}\,, \tag{4.3}$$

on which we finally apply GAP to obtain a compressed representation $E(c_{1:h}) \in \mathbb{R}^{d_o}$. Categorical attributes other than resources and activities can be integrated similarly. The Transformer's embedding layer first learns continuous representations for all the categorical tokens. Then, as the self-attention blocks have access to any subset of positions of their input sequence, they can learn relationships between different event attributes (e.g., activities and resources) in a flexible and data-driven way.

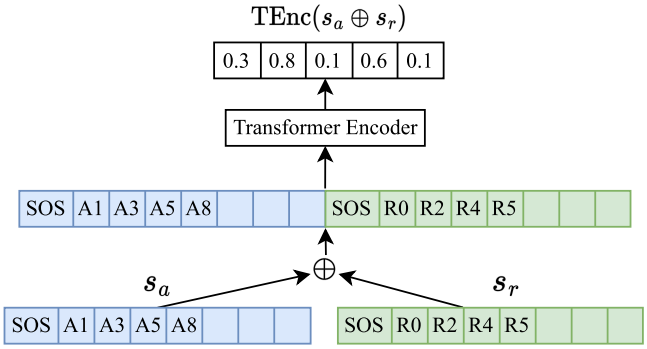


Figure 4.3: *Early Fusion* method to combine categorical attributes (OneTr) to combine categorical features in the encoder.

For *Late fusion*, we use an architecture called *Multi-Tr* that combines categorical attributes using a different Transformer encoder to process different token sequences. Formally, in the case of resources and activities, the feature map is given by

$$E^{(\text{A+R})}_{\text{Multi-Tr}} = \text{TEnc}(s_a) \oplus \text{TEnc}(s_r) \in \mathbb{R}^{2h \times d_o}\,, \tag{4.4}$$

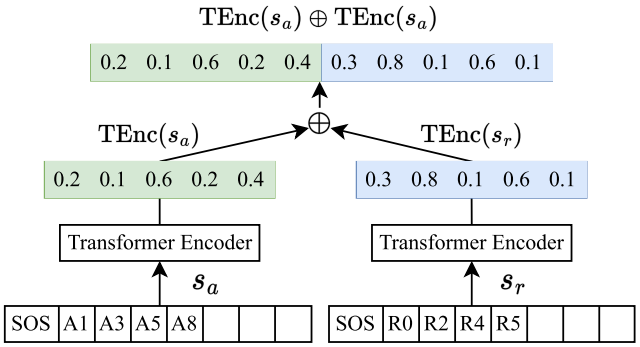


Figure 4.4: *Late Fusion* method to combine categorical attributes (MultiTr) to combine categorical features in the encoder.

In this method, the Transformer's self-attention mechanism cannot combine features corresponding to different attribute types. We thus evaluate the benefit of using a *Modulator layer* to alleviate this limitation. A Modulator layer [24] combines different feature vectors of an event assigning high weights to the more critical features and low weights to the unimportant features for a given predictive task. In our model, we apply the Modulator to the outputs of each Transformer. For instance, for predicting the next activity, the feature maps $\text{TEnc}(s_r)$ obtained from the resource sequence and $\text{TEnc}(s_a)$ obtained from the activity sequence are combined as

$$E^{(\text{A+R})}_{\text{Multi-Tr-Mod}} = \alpha_{aa} \cdot \text{TEnc}(s_a) + \alpha_{ar} \cdot \text{TEnc}(s_r) \in \mathbb{R}^{h \times d_o}\,, \tag{4.5}$$

where α_{aa}, α_{ar} are, respectively, the weights of the activity features and the resource features for predicting the next activity. In the case of multiple predictive tasks, we compute different sets of weights for each task. To conclude the encoding process as in the other methods, a compressed representation $E(c_{1:h}) \in \mathbb{R}^{d_o}$ is obtained through GAP.

Integrating Continuous Attributes

To integrate continuous attributes into the model, such as features computed from timestamps, we adopt a *late fusion* approach in which we concatenate the feature vectors derived from the categorical attributes with the response of a feed-forward net that encodes the continuous attributes. We found this method to be the simplest and most effective for combining time features with categorical attributes such as activities and resources. In addition, we found it helpful to pre-process timestamps to extract more meaningful features, such as the time elapsed from the start of the process. To formalize this method, we can denote by $s_t = \langle t_1, t_2, \ldots, t_h \rangle$ the sequence of continuous attributes, and by $f_t \in \mathbb{R}^T$ the feature vector hand-crafted from s_t. We combine f_t with the features obtained from the late fusion approach to combine

activity and resource information as

$$E^{(\text{A+R+T})}_{\text{Multi-Tr}} = \text{GAP}\left(\text{TEnc}(s_a) \oplus \text{TEnc}(s_r)\right) \oplus \text{FF}(f_t) \in \mathbb{R}^{d_o+h'}\,, \tag{4.6}$$

where h' is the output dimensionality of the feed-forward net. Similarly, we combine f_t with the features obtained from the early fusion approach to combine activity and resource information as

$$E^{(\text{A+R+T})}_{\text{One-Tr}} = \text{GAP}\left(\text{TEnc}(s_a \oplus s_r)\right) \oplus \text{FF}(f_t) \in \mathbb{R}^{d_e+h'}\,. \tag{4.7}$$

4.1.2 Decoder Architectures

We explore two methods to predict the next activity and other event attributes, such as resources and expected timestamps.

Specialized Layers

The most straightforward approach consists in using different sub-nets to predict different attributes. We use a classic two-layer neural net with Softmax activation for categorical attributes and linear activation for continuous attributes. In particular, for a categorical attribute a_{h+1}, the corresponding net learns a probability distribution over the possible values of a_{h+1} conditioned to the feature vector extracted by the encoder $E(c_{(1:h)})$. In Fig. 4.5, we illustrate this architecture combined with the One-Tr architecture on the encoder's side.

Multi-task approach

The second method we explore uses a shared bottleneck layer [13] to fed the output layer corresponding to each task. This hidden layer receives the feature map computed by the encoder $E(c_{(1:h)})$ and outputs a compressed representation z_h. Although $E(c_{(1:h)})$ already contains fused information from all the attributes in $c_{(1:h)}$, features learned in layers close to the output are often more specific to the corresponding task. Thus we hypothesize that a shared representation z_h at the decoding stage could enhance knowledge transfer across the different prediction problems. This architecture combined with the early approach to combine categorical attributes is illustrated in Fig. 4.5.

4.2 Architectures for Trace Prediction

In this section, we present two architectures to predict not only the next event's information but the entire trace of activity IDs $\langle a_{h+1}, a_{h+2}, \ldots a_{l(c)}\rangle$ that follows the current event e_h. We formulate these methods to extend a previous contribution presented at Discovery Science 2022 [30]. Given the results of that study, we restrict

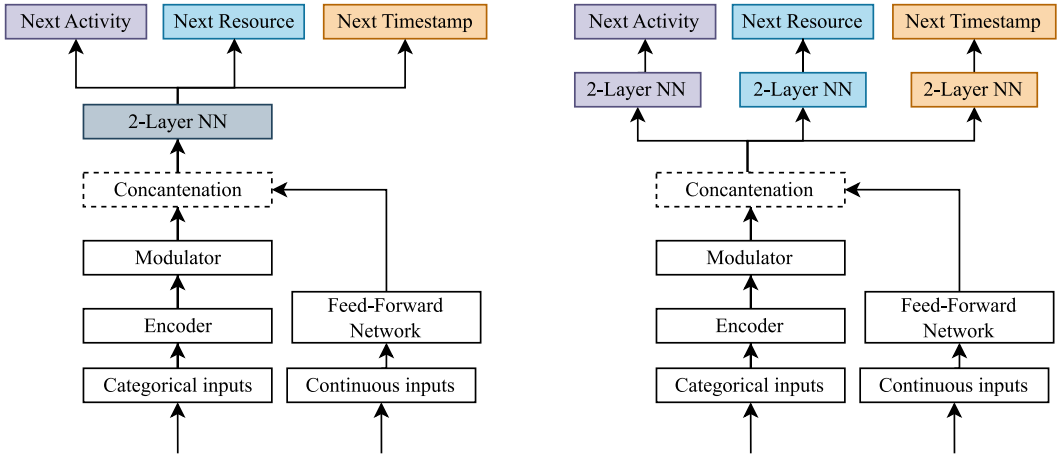


Figure 4.5: Decoder architectures for Next event prediction. At left, a decoder with independent output layers. At right, a decoder with a shared bottleneck close to the output. In both cases, the figure illustrates the late fusion approach (Multi-Tr) to combine categorical features in the encoder to enter the modulator or an early approach without considering the modulator.

our attention to models that combine categorical attributes such as activity and resource IDs.

As trace prediction is a sequence-to-sequence problem, we need to adapt both, the encoder and the decoder used previously such that they can predict multiple output tokens in the proper order with an architecture that fuse the two attribute representations (activity IDs and resource IDs).

Inspired by the work of [23], our architecture use different Transformer encoders to process different attribute sequences and fuse attribute information through cross-attention layers at the decoding stage. The encoders transform their corresponding input sequence s into a sequence of continuous representations E_s. In the case of activity and resource sequences s_a, s_r, we use two Transformer encoder and obtaimm,m,l,m,l,,l,..n the two matrices

$$\begin{aligned} E_{s_a} &= \text{TEnc}(s_a) = \text{TEnc}(\langle a_1, a_2, \dots a_h \rangle) \in \mathbb{R}^{h \times d_o} \\ E_{s_r} &= \text{TEnc}(s_r) = \text{TEnc}(\langle r_1, r_2, \dots, r_h \rangle) \in \mathbb{R}^{h \times d_o} . \end{aligned} \tag{4.8}$$

Then, the decoder simultaneously attend E_{s_a} and E_{s_r} to obtain predictions that are sensitive to both contexts, that provided by s_a and that provided by s_r. Below we present two methods based on [23] to implement this idea.

4.2.1 Decoder Architectures

Although [23] evaluated several strategies for implementing cross-attention mechanisms with multiple encoders, here we focus on the best two: the *parallel* and the

serial architectures. Both methods modify the cross-attention mechanism used in each block of the Transformer [40]. Formally, this mechanism is a multi-head attention mechanism where the queries come from the output of the previous decoder's layer D_{k-1}, and the keys and values come from the output E_e of the unique encoder present in the model (see Sec. 2.5 for details). Fig. 4.6 and Fig. 4.7 illustrate how the serial and the parallel strategies modify this operator respectively, where $D_k = \text{Multi-Head}(D_{k-1}, E_e, E_e)$. Below we summarize the technical details.

Parallel Architecture

In this strategy, the decoder attends to each encoder separately and then adds the obtained feature maps. In the case of two encoders with outputs $E_e^{(1)}$ and $E_e^{(2)}$, the parallel cross-attention mechanism computes $D_k = D_k^{(1)} \oplus D_k^{(2)}$

$$\begin{aligned} \text{where } D_k^{(1)} &= \text{Multi-Head}(D_{k-1}, E_e^{(1)}, E_e^{(1)}) \\ \text{and } D_k^{(2)} &= \text{Multi-Head}(D_{k-1}, E_e^{(2)}, E_e^{(2)})\,. \end{aligned}$$

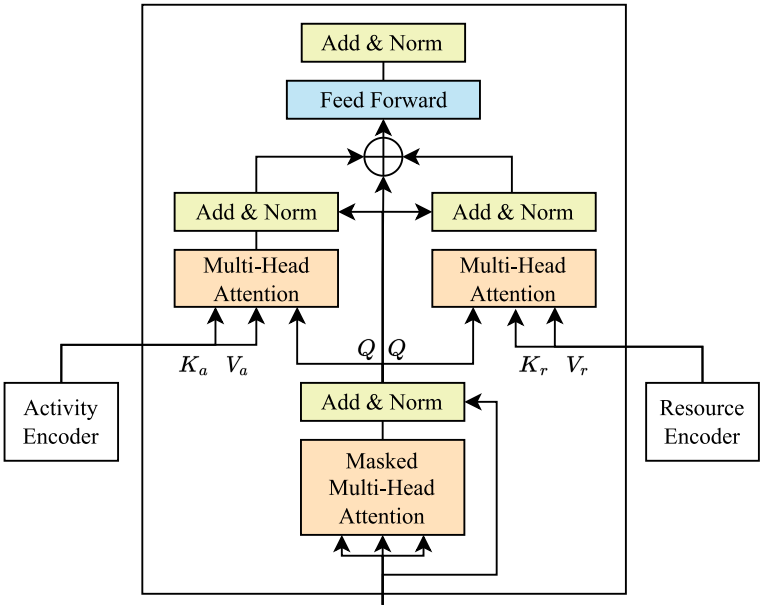


Figure 4.6: Parallel architecture.

Serial Architecture

In the serial strategy, the decoder attends to multiple encoders one by one, stacking multi-head attention layers. In the first cross-attention mechanism, the queries come from the decoder's self-attention mechanism (as usual), and the keys and values come from the first encoder. However, in the second cross-attention mechanism, the queries

come from the previous self-attention mechanism, and the keys and values come from the second encoder. More than two encoders can be integrated similarly. Formally and in the case of two encoders, it computes

$$D_k = \text{Multi-Head}(\tilde{D}_k, E_e^{(2)}, E_e^{(2)}) \tag{4.9}$$
$$\text{where } \tilde{D}_k = \text{Multi-Head}(D_{k-1}, E_e^{(1)}, E_e^{(1)})\,.$$

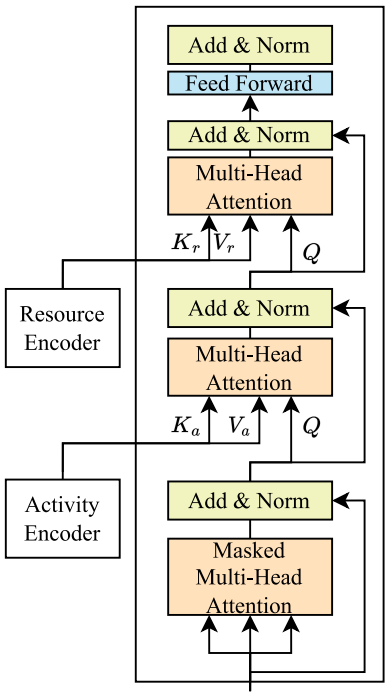


Figure 4.7: Serie architecture.

To conclude, the serial and the parallel strategy architecture equip each attention layer with residual connections and layer normalization.

4.3 Training and Inference

It is worth mentioning that, in the training phase, the decoder can process an output sequence in parallel, i.e., it can predict all the output tokens in a single step. This is possible thanks to the auto-regressive masking technique described in Sec. 2.5 and a training technique known as *teacher forcing*. Teacher forcing trains the decoder to predict the value of a_{h+s} using the ground-truth values of $a_{h+1}, a_{h+2}, \ldots, a_{h+s-1}$. As this is infeasible during inference, the decoder often predicts out-of-sample sequences in an auto-regressive way, i.e., it generates the outputs one by one, conditioning the decoding process on its previous predictions until a special end-of-sequence token

is generated. Although preliminary results on non-autoregressive methods currently exist, we adopt the teacher forcing approach for training and the autoregressive approach for inference.

Chapter 5

Experiments

This section presents the experiments we conducted to evaluate the proposed methods. First, we briefly describe the datasets and some implementation details. Then we discuss the metrics used for evaluation. Then, we discuss the results obtained in predicting the next event and its attributes, comparing them with those of other applicable methods. Finally, we present the results obtained with the extension formulated to predict the trace of activities following an incomplete case.

5.1 Datasets

We assess the proposed method on three real-life event logs widely used in predictive process monitoring literature:

Helpdesk[1]: The dataset comprises log data originating from the management process of an Italian ticketing system spanning the time frame of 2010 to 2014. This rich dataset provides valuable insights into the operational dynamics and historical performance of the ticketing management process during this five-year period.

BPIC 2012[2]: This dataset encapsulates the log records pertaining to online loan applications within a prominent Dutch financial institution. The data spans from October 2011 to March 2012, offering a comprehensive view of the intricacies involved in the loan application process during this specific time frame. Notably, this dataset is structured into three distinct subprocesses, each of which contributes to the holistic understanding of the loan application process within the institution:

- *Application Status:* This subprocess encompasses the tracking of the application's progress through various stages, providing critical insights into the lifecycle of each application.
- *Status of Tasks Involved in the Application:* Within this subprocess, meticulous records are maintained regarding the completion status of individual tasks

[1]https://data.4tu.nl/repository/uuid:0c60edf1-6f83-4e75-9367- 4c63b3e9d5bb
[2]https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

associated with each application, enabling a granular analysis of task-level performance.

- *Offer's Status:* This component monitors the status of loan offers extended to applicants, shedding light on the final outcome of each application and the institution's response.

BPIC 2017[3]: The dataset under consideration is a continuation of the log data generously provided by the same company as part of the Business Process Intelligence Challenge (BPIC) 2012, offering a broader and more extensive collection of samples. This extended dataset significantly augments the volume of available data compared to its predecessor, thereby facilitating more comprehensive and detailed analyses of the company's business processes.

Table 5.1 provides an encapsulated overview of the primary statistical characteristics and attributes of the datasets under consideration.

Table 5.1: Statistics of datasets used for evaluation. Case length/duration in days.

				Case Length		Case Duration	
Dataset	Cases	Events	Activities	Max.	Avg.	Max.	Avg.
Helpdesk	4,580	21,348	14	15	4.6	60	40.69
BPI 2012	13,087	262,200	36	175	20.0	137.5	8.6
BPI 2017	31,509	1,202,267	26	180	38.1	286	21.9

5.2 Pre-processing

For each dataset, we sorted the cases (process instances) chronologically according to the first event's timestamp and kept the first 80% for training and the remainder 20% for testing. Categorical values such as resources and activities are converted to unique integer tokens. Then, we added an end-of-sequence (EOS) token to each sequence and augmented the number of samples by applying a *prefix generator function* to each case. This operator maps a sequence to the list of all its sub-sequences, starting from length ml $= 1$ to the maximum case length $l(c)$. This procedure attempts to replicate a real-world monitoring environment in which we have to predict the evolution of ongoing processes, i.e., incomplete sequences.

For the Next event prediction experiments, each sequence of length $k \in [1, l(c)]$ leads to a prediction instance where the input is a sequence of length $k-1$, and the target is the last event. Indeed, when the first event is executed, then the model can predicts the following events. Note that for sequences of length $k = l(c)$, the target is the EOS token, i.e., the model has to predict the process termination. For the

[3]https://data.4tu.nl/articles/BPI_Challenge_2017/12696884

Next trace prediction experiments, we use each sequence s_k of length $k \in [1, l(c)]$ originated from a case s to define a sequence-to-sequence instance where the input is s_k and the target is the sequence of symbols t_k such that $s_k \oplus t_k = s$. Then, we added a start-of-sequence (SOS) token to each input sequence. Now, the method consider this start token as the first event of the case. Finally, as required by Transformer encoders, we padded all the input sequences to a fixed length. In sequence-to-sequence experiments, we also padded all the target sequences with a special symbol that allows us to skip these cases when computing the loss.

To end our pre-processing pipeline and as explained in Sec. 4.1.1, we need to process timestamps to extract continuous features. Indeed, following [6], we transformed timestamps into three time-related features: the time between the previous and current event, the time between the next-to-last event and the current event, and the time passed since the process started.

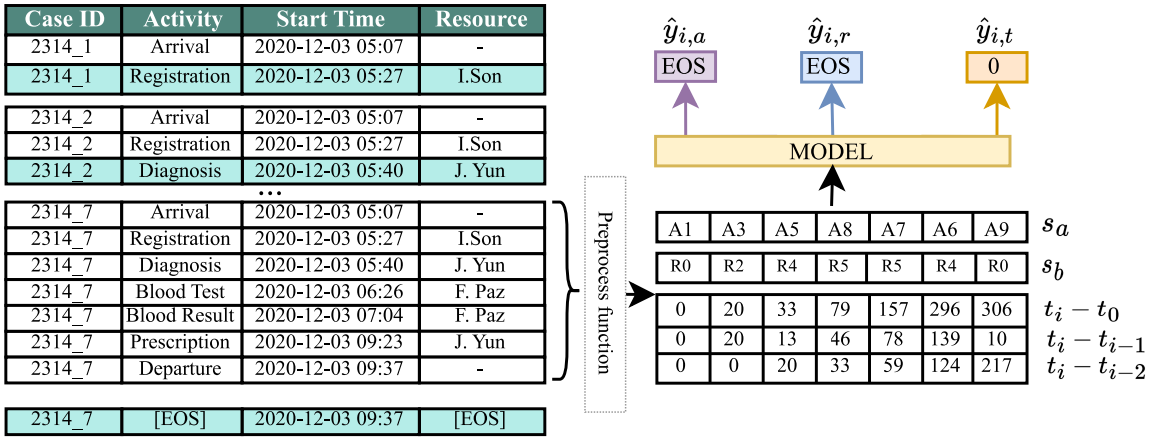


Figure 5.1: General preprocessing steps

5.3 Prefix Length

It is worth noting that some authors exclude short training and evaluation sequences from the datasets. For instance, Lin et al. [24] only retains sequences s_k containing at least $k = 5$ steps of events in *BPIC 2017* and *BPIC 2012*. And for *Helpdesk*, only works with sequences of length larger than $k = 3$. We preferred to preserve all the sequences regardless of length to favor a direct comparison with most previous studies.

As mentioned before, few previous works impose a minimum prefix length $\text{ml} > 1$ to accept a case for evaluation, i.e., incomplete cases of length $k < \text{ml} = 1$ are not considered for assessing the performance of the method. In our experiments, we use $\text{ml} = 1$ to replicate the needs of real-world scenarios. Nevertheless, when we report results from other works, we also include the minimum prefix length used by the

authors. Fig. 5.2 depicts an example of the same case with four different minimum trace lengths and illustrates their respective impacts.

Table 5.2: Example of 4 different minimum trace lengths (*ml*). This condition illustrates the shorter the length of the sequence (prefix), the greater the number of examples but with less information about the process itself.

ml	Activity n-gram	Resource n-gram	Activity target
1	[0 0 0 0 0 1]	[0 0 0 0 0 16]	2
1	[0 0 0 0 1 2]	[0 0 0 0 16 18]	4
1	[0 0 0 1 2 4]	[0 0 0 16 18 18]	8
1	[0 0 1 2 4 8]	[0 0 16 18 18 20]	15
2	[0 0 0 0 1 2]	[0 0 0 0 16 18]	4
2	[0 0 0 1 2 4]	[0 0 0 16 18 18]	8
2	[0 0 1 2 4 8]	[0 0 16 18 18 20]	15
3	[0 0 0 1 2 4]	[0 0 0 16 18 18]	8
3	[0 0 1 2 4 8]	[0 0 16 18 18 20]	15
4	[0 0 1 2 4 8]	[0 0 16 18 18 20]	15

5.4 Implementation

All the models were implemented using Python with the functional Deep Learning API Keras. To set the hyper-parameters, a small grid search with cross-validation on the *Helpdesk* dataset's training set to identify the optimal hyper-parameters. Once determined, these settings were extrapolated to the other two datasets to ensure consistency and comparability across the models. We trained all sequence-to-one models with Stochastic Gradient Descent (SGD) using batch sizes of 16, 32, and 128 in *Helpdesk*, *BPIC 2012* and *BPIC 2017* respectively. In the initial experiments, various learning rates and epochs were evaluated; however, no significant improvements were identified in comparison with the prolonged processing time upon which they rely. Therefore, they were fixed at a learning rate of 0.001 and 10 epochs across all datasets.

Meanwhile, for the sequence-to-sequence models the custom Adam optimizer from the original Transformer [40] is used. The number of encoder-decoder layers is incremented to 6 with 8 heads each of the multi-head attention layer. The dimensionality of the model is set to 512 and for the feed-forward net to 2048.

5.5 Metrics

As in most previous studies regarding *Next event* prediction, the accuracy metric is used to assess the performance of the different methods in predicting categorical attributes, i.e., next activity and resource. Then, the Mean Average Error (MAE)

is proposed to assess the ability of the methods to predict continuous (time-related) attributes. Both accuracy and MAE were used as the baseline comparative metrics in the literature, while other metrics such as precision, recall, or MAPE were set aside.

5.5.1 Accuracy

Accuracy is a metric that quantify how closely the prediction of a model align with the actual phenomena it aims to represent, as explained by Sammut and Webb [34] that can be calculated by the Eq. 5.1 and outputs a value between 0 and 1 or represented as percentage. In this classification model scenario, accuracy can be defined as $P(\lambda(X) = Y)$, where $P(X, Y)$ is a joint distribution and the classification model λ: $X \to Y$.

$$\text{Accuracy} = \frac{\text{Number of correctly classified objects}}{\text{Total number of objects}} \in [0, 1] \tag{5.1}$$

5.5.2 Mean Absolute Error

The Mean Absolute Error score is measured as the average of the absolute error values. The error values are calculated as the difference between the expected values and the predicted values, as they can be positive or negative, an absolute operator is added to force positive results. The MAE value can be calculated as follows:

$$\text{MAE} = \frac{1}{n} \sum_{1}^{n} |y_1 - \hat{y}_i|^2 \in [0, 1] \tag{5.2}$$

5.5.3 Damerau-Levenshtein similarity

In the sequence-to-sequence experiments, we assess performance by computing the *normed Damerau-Levenshtein distance* $DL(v_1, v_2)$ [7] between a predicted trace v_1 and the ground-truth trace v_2. This metric is widely used in Deep learning [25] to measure the minimum number of characters we need to edit (restricted to four operations: insertion, deletion, substitution and transposition) to transform one string a (e.g., a predicted sentence) into other b (e.g., a ground-truth sentence). The distance from string a and string b can be defined recursively as:

$$d_{a,b}(i,j) = \min \begin{cases} 0 & \text{if } i = j = 0, \\ d_{a,b}(i-1,j) + 1 & \text{if } i > 0, \\ d_{a,b}(i,j-1) + 1 & \text{if } j > 0, \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} & \text{if } i,j > 0, \\ d_{a,b}(i-2,j-2) + 1_{(a_i \neq b_j)} & \text{if } i,j > 1 \wedge a_i = b_{j-1} \wedge a_{i-1} = b_j, \end{cases} \tag{5.3}$$

Where i-symbol is the prefix of string a and j-symbol is the prefix of string b. Consequently, based on $DL(v_1, v_2)$, we can define the *Normed Damerau-Levenshtein similarity* $s(v_1, v_2)$ between the two sequences as:

$$s(v_1, v_2) = 1 - \frac{DL(v_1, v_2)}{\max(\text{len}(v1), \text{len}(v2))} \in [0, 1]\,. \tag{5.4}$$

Furthermore, Evermann et al. [11] previously used this metric to compare traces in predictive process monitoring.

5.5.4 Evaluation specifications

Finally, in the process of assessing a method's performance on a specific task, a comprehensive report is generated, summarizing the average values of the relevant metrics across all test sequences. Moreover, to enhance the robustness of performance estimation, the mean values of each metric are calculated from the results of five independent runs for each model.

Chapter 6

Results

6.1 Next Event

6.1.1 Next Activity

Table 6.1 presents a comprehensive summary of the performance of various architectural configurations employed in our study for the prediction of subsequent activities within the context of the business process. Our findings underscore the superiority of models that leverage a fusion of information from both activities and resources, surpassing the performance achieved by models reliant solely on activity-based data.

Interestingly, the results also shed light on the impact of incorporating time-based features into the predictive models. Contrary to our initial expectations, the inclusion of temporal information has shown a slight reduction in performance. This phenomenon is further elucidated in Fig. 6.2, where we observe a potential correlation between the drop in performance and model instability. These insights encourage us to delve deeper into the dynamics of our models to address this apparent instability, potentially leading to improvements in predictive accuracy. But due to time constraints and/or the extent of the thesis topic, it is left as an open question for future research as well, question whether adding more attributes really provides information or bias in each business process.

Additionally, our investigation scrutinized the role of the Modulator layer within our architecture. Contrary to our anticipation, the results reveal that the Modulator layer did not yield a significant enhancement in prediction performance. This finding suggests that the core Multi-Attribute Transformers exhibit a commendable capability to accurately predict subsequent activities on their own, reinforcing their utility in our predictive framework. Further research may explore whether fine-tuning the Modulator layer or exploring alternative enhancements could provide additional benefits to our predictive models.

In Table 6.2, we compare our best two strategies with other results reported in recent literature for the next activity prediction task. Results on *Helpdesk* and

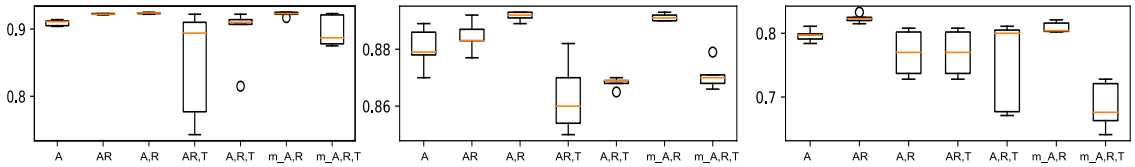


Figure 6.1: Accuracies of next activity task by model on each dataset

BPIC 2012 show that using Transformer encoders to combine activity and resource attributes outperforms other approaches if we set a minimum prefix length of ml = 1 for evaluation. The advantage of [17] in *BPIC 2017* is likely due to the aggregation of other attributes or data processing methods.Table 6.2 also shows that the proposed methods outperform scores reported for LSTM-based methods in [36], and [16], even if these studies restrict evaluation to ml = 2. As a minimum prefix length increases, the input sequences carry more information about the process. Therefore, we should expect an increase in the model's performance.

Table 6.2: Comparison of the proposed methods with other results available in the literature for the task of predicting the next activity.

Method	Input	Data split	Prefix (ml)	Helpdesk	BPIC 2012	BPIC 2017
LSTM [6]	A, R, T	70/30	1	0.789	0.786	-
ProcessTransformer [5]	A	60/20/20	1	0.856	-	-
HAM-net bi-LSTM [17]	C	80/20	1	0.844	0.868	**0.929**
One-Tr (proposed)	A, R	80/20	1	0.922	**0.884**	0.823
Multi-Tr (proposed)	A, R	80/20	1	**0.924**	0.892	0.769
LSTM [36]	A, T	66/33	2	0.712	-	-
Attention bi-LSTM [16]	A	80/20	2	0.833	0.816	-
MM-Pred LSTM [24]	C	70/20/10	4	0.916	0.974	0.974

Also, Fig. 6.2 shows the performance of our models as we vary the minimum prefix length ml. We can see that results are mixed. In *Helpdesk*, we observe the trend we expected: accuracy increases as ml increases. However, performance remains relatively stable in *BPIC 2012* while in *BPIC 2017*, there is no trend whatsoever. To understand these results, we must note that increasing ml decreases the total number of instances for evaluation because there are fewer suffixes to predict.

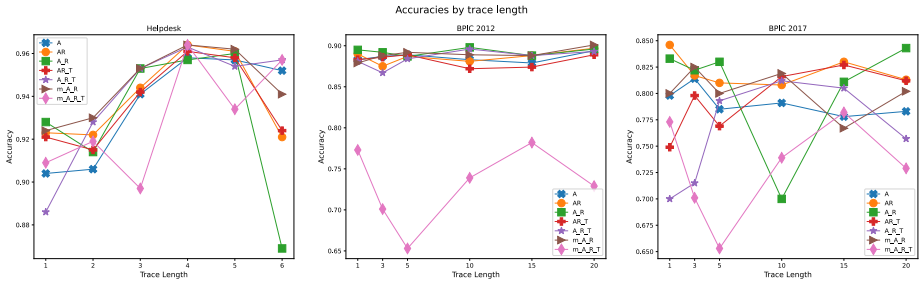


Figure 6.2: Accuracies of next activity task by trace length.

6.1.2 Next Activity and Resource

In Table 6.3, we compare different combinations of encoders, decoders, and input attributes, in the task of simultaneously predicting the activity (A) and resource (R) corresponding to the next event. The best results we achieve in predicting the next resource (R) are 88%, 78%, and 76% in *Helpdesk*, *BPIC 2012* and *BPIC 2017* respectively. To the best of our knowledge, these are the first results demonstrating the application of artificial intelligence on this task. Results on *Helpdesk* and *BPIC 2012* also suggest that training the models to predict the next activity and resource jointly can slightly improve their performance in predicting the next activity. Furthermore, the Multi-Tr encoder architecture benefits from combining information from activities, resources, and timestamps.

Table 6.3: Performance of the proposed methods in predicting the next activity (A) and the next resource (R) simultaneously. We compute the accuracy on each target (A,R) separately and compare the accuracy of different encoders (One-Tr, Multi-Tr), decoders (S: specialized layers, M: multi-task approach), and process attributes (A: activities, R: resources, T: time-based features).

			Helpdesk		BPIC 2012		BPIC 2017	
Encoder	Decoder	Input	A	R	A	R	A	R
One-Tr	M	A, R	91.9	87.9	86.3	76.7	76.9	73.5
One-Tr	M	A, R,T	76.8	74.9	82.9	75.5	71.3	62.8
One-Tr	S	A, R,T	90.8	87.9	80.6	74.2	64.1	61.6
Multi-Tr	M	A, R	92.3	87.9	88.7	77.2	72.2	75.0
Multi-Tr	M	A, R, T	**92.5**	87.6	**89.4**	77.1	57.7	68.6
Multi-Tr	S	A, R, T	92.4	87.6	88.7	**78.6**	**82.0**	**76.3**
Multi-Tr-Mod	M	A, R	**92.5**	87.6	88.6	78.2	81.9	75.3
Multi-Tr-Mod	M	A, R, T	91.5	**88.1**	88.3	77.2	80.9	74.2

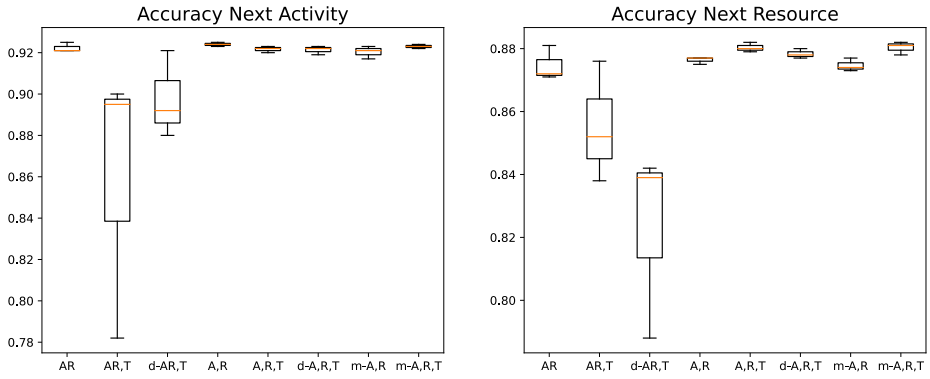


Figure 6.3: Accuracies predicting next activity and resource in Helpdesk

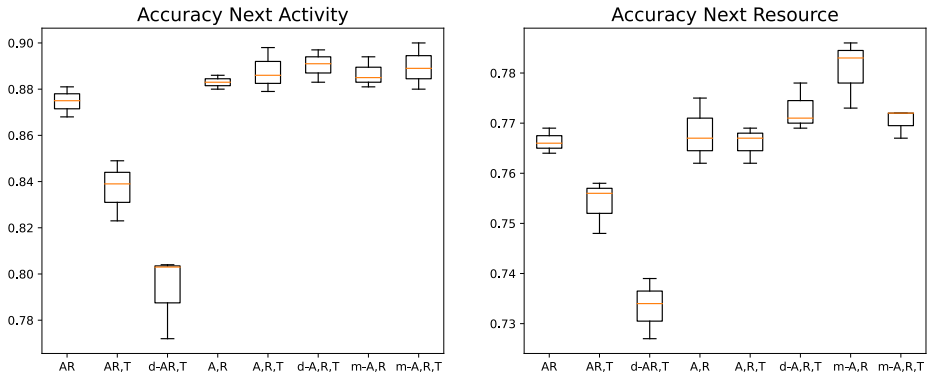


Figure 6.4: Accuracies predicting next activity and resource in BPIC 2012

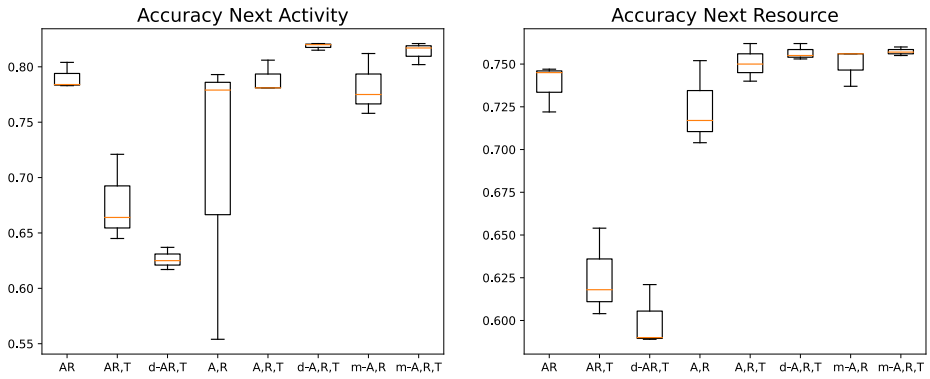


Figure 6.5: Accuracies predicting next activity and resource in BPIC 2017

6.1.3 Next Activity, Resource, and Time

Table 6.4 summarizes the performance of the different architectures in predicting the next event's activity (A), resource (R), and timestamp (T) simultaneously. Results show a slight decrease in the accuracies of the two categorical prediction tasks when we jointly train the model for the three tasks. On the other hand, results show that predicting the next timestamp does not always benefit from a specialized decoder combined with a Multi-Tr encoding approach. Overall, our results underperform those presented by [5] and [36] using specialized architectures for the task. Their methods achieved MAE metrics below 6 and 0.4, respectively, in the *Helpdesk* and *BPIC 2012* datasets.

Table 6.4: Performance of the proposed methods in predicting the next event's activity (A), resource (R), and timestamp (T) simultaneously. We compare different architectures, computing the evaluation metric for each target (A,R,T) separately. Note that in the case of timestamps, we use MAE (so, lower is better).

Dataset	Encoder	Decoder	Input	A	R	T
Helpdesk	One-Tr	M	A, R, T	88.4	72.4	86.5
Helpdesk	Multi-Tr	M	A, R, T	90.6	82.3	78.0
Helpdesk	Multi-Tr	S	A, R, T	**90.7**	**84.2**	**75.7**
BPIC 2012	One-Tr	M	A, R, T	84.7	74.7	**6.55**
BPIC 2012	Multi-Tr	M	A, R, T	88.3	**76.5**	6.68
BPIC 2012	Multi-Tr	S	A, R, T	**88.6**	76.3	6.67
BPIC 2017	One-Tr	M	A, R, T	**76.9**	**73.8**	**7.73**
BPIC 2017	Multi-Tr	M	A, R, T	55.9	55.0	8.02
BPIC 2017	Multi-Tr	S	A, R, T	36.5	39.0	7.92

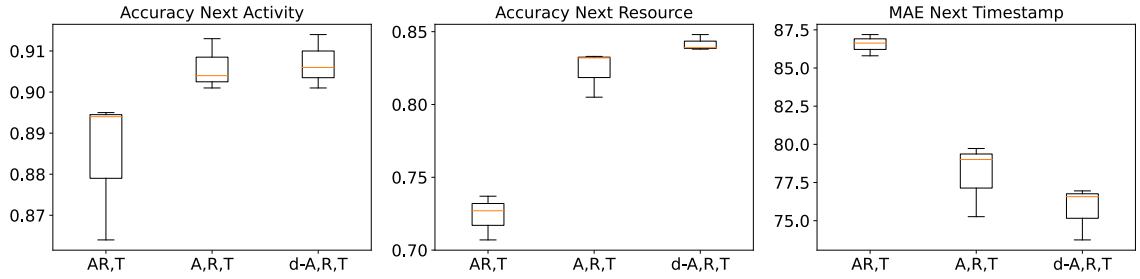


Figure 6.6: Next event in task act,res,time from Helpdesk

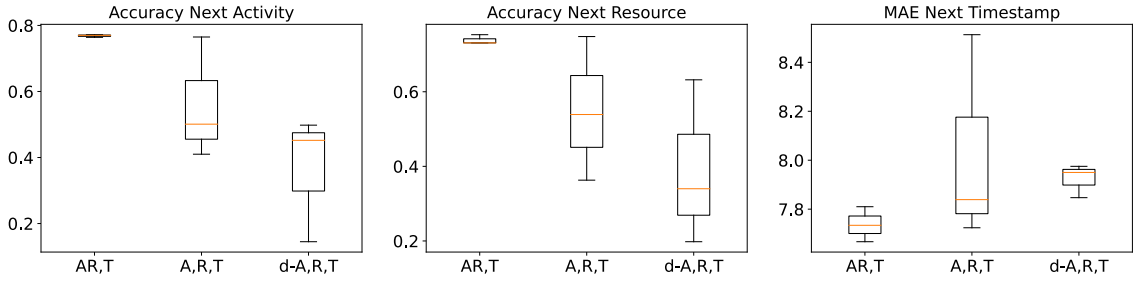


Figure 6.7: Next event in task act,res,time from BPIC 2012

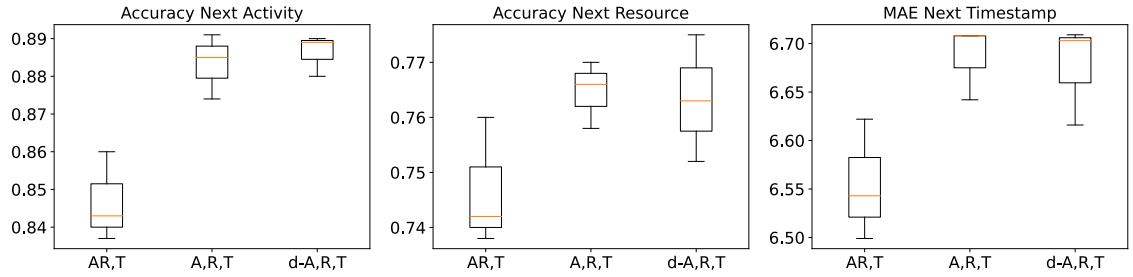


Figure 6.8: Next event in task act,res,time from BPIC 2017

6.2 Next Trace Prediction

Table 6.5 presents the results of our methods in the Next trace prediction task. Besides the Parallel and Serial cross-attentional mechanisms discussed in Sec. 4.2.1, we evaluate a baseline Transformer trained on activity sequences only. Results demonstrate that combining attributes about the input case (information about activity and resources) improves the baseline Transformer in all three datasets. Moreover, the Parallel architecture outperforms the serial architecture by significant margins. Also, we compare our best architecture for Next trace prediction with other results reported in recent literature for this task. Overall, our method improves known results except in *BPIC 2012* by [6] that apply a random method to select the next activity in the last layers, which avoids getting stuck in typical cases in long sequences. Furthermore, the comparison suggests that our method has its advantage by leveraging both multiple attribute fusion and state-of-the-art attention-based architectures.

Table 6.5: Baselines comparison for the trace prediction based on similarity. [10] results were replicated in [24].

			Damerau-Levenshtein Similarity		
Architecture	Input	Prefix	Helpdesk	BPIC 2012	BPIC 2017
LSTM [10]	A	2	0.742	0.110	-
LSTM [36]	A, T	2	0.766	-	-
LSTM [6]	A, R, T	1	0.917	**0.623**	-
MM-Pred LSTM [24]	Cat. Attr.	1/5/5	0.874	0.281	0.301
Baseline Transformer (Proposed)	A	1	0.915	0.449	0.602
Parallel Fusion (Proposed)	A, R	1	**0.921**	0.516	0.561
Serial Fusion (Proposed)	A, R	1	0.919	0.544	**0.610**

Further, we represent attention weight matrices through heatmaps of each of the heads within each layer to identify inner insights of the business process within the model. To build these heatmaps, we firstly take an incomplete case from the evaluation dataset and feed the model with the sequence of activities occur at that time. Then, iteratively concatenate the input with the predicted activity (Inference mode). In each step, we only extract the attention weights from the last predicted token with respect to the input, in order to be able to see which activities are being attended to in that prediction. In Fig. 6.9 it can be illustrated that the closer the case is to the end, the more information can be obtained about the relationships between the past activities and the ouput. Furthermore, in this example, they are prone to pay attention to most of the past activities but also, to ignore unrelated activity loops, as it is in the 50th predicted activity (*W_ Nabellen offertes-START*). As these matrices are extracted from one layer of one head, we could expect that there are other layers that may provides different values related to each of the proposed architectures .

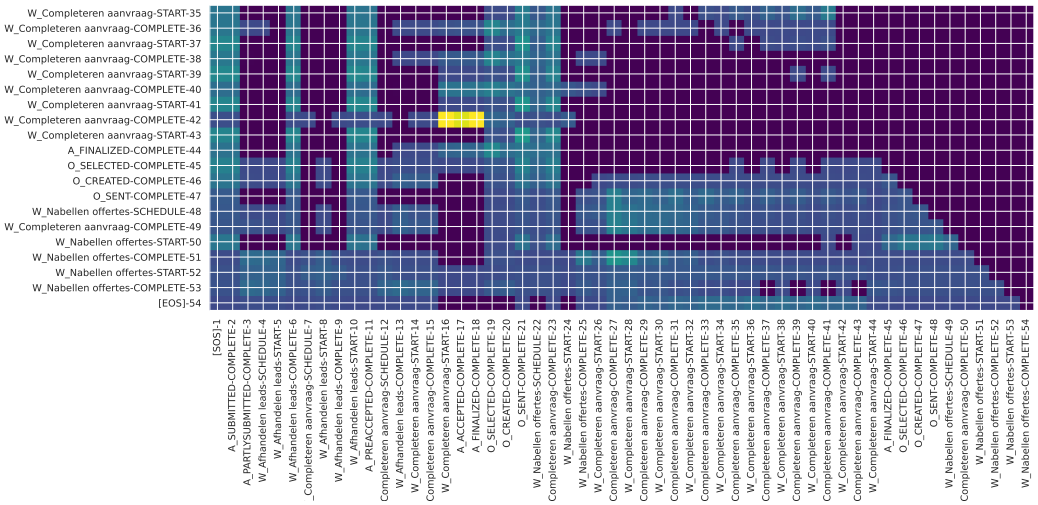


Figure 6.9: Example of attention weights in a incomplete case with prefix 42 by the input of the activity encoder to the parallel cross-attention layer in the decoder, where in both axis are the predicted tokens but in x-axis are one step before representing the input in the inference mode. Therefore, the decoder has access to the activities from the start token to the 42th activity (only the bottom part of the matrix is presented), and from there, it begins to predict until the end token is identified.

Table 6.1: Results of the proposed methods in the different datasets for predicting the next activity. We compare the accuracy of different strategies (One-Tr, Multi-Tr) to integrate process attributes (A: activities, R: resources, T: time-based features). As discussed in Sec. 4.1.1, the Multi-Tr-Mod architecture includes a modulator layer to assign a weight to different attribute types

Architecture	Input	Helpdesk	BPIC 2012	BPIC 2017
One-Tr	A	90.9	88.0	79.6
One-Tr	A, R	92.2	88.4	**82.3**
Multi-Tr	A, R	**92.4**	**89.2**	76.9
One-Tr	A, R, T	84.9	86.3	76.9
Multi-Tr	A, R, T	89.4	86.8	75.3
Multi-Tr-Mod	A, R	92.2	89.1	80.9
Multi-Tr-Mod	A, R, T	89.7	87.1	68.6

Chapter 7

Conclusions and future work

This thesis presented multi-attribute Transformers for predicting the next event and its attributes in a business monitoring environment. In contrast with other methods, we considered as input not only the activity sequences but the resource and timestamp sequences because they can provide valuable information to the model. Our methods first leverage the Transformer attention method to create continuous representations of the categorical attributes. Then, they can join these representations with continuous features extracted from timestamps. Finally, a simplified decoder (fully connected net) learns to perform one or multiple prediction tasks on the multi-attribute representation. Based on the results of this research, we also formulated an extension capable of predicting the entire trace (sequence of activities) that follows the current state of an ongoing process.

The experimental results revealed that our method outperforms current approaches in predicting the next activity on two of the three datasets using single-task and multi-task architectures. We obtained the best performance with the variants that considered multiple attributes. However, when integrating the timestamps, we observed more instability in the model's performance. In addition, when predicting the next timestamp, our method does not overpass current approaches that use dedicated architectures for the task. In contrast, the methods we formulate for next trace prediction consistently improved current results, including those based on multiple attribute.

In future work, an investigation of sequence-to-sequence methods to perform a multi-task approach to predict multiple attributes corresponding to the next event. We also plan to assess methods that circumvent the auto-regressive generation constraint that is currently present in our model for next-trace prediction. Finally, as research on explainable Intelligence Artificial progresses, studying methods that provide insights into the relevance of features for obtaining a prediction is an interesting direction to explore.

Bibliography

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv.1409.0473*, 2014.

[3] A. Baiyere, H. Salmela, and T. Tapanainen. Digital transformation and the new logics of business process management. volume 29, pages 238–259. Taylor & Francis, 2020.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[5] Z. A. Bukhsh, A. Saeed, and R. M. Dijkman. Processtransformer: Predictive business process monitoring with transformer network. *arXiv:2104.00721*, 2021.

[6] M. Camargo, M. Dumas, and O. González-Rojas. Learning accurate lstm models of business processes. In *Business Process Management: 17th International Conference, BPM 2019, Vienna, Austria, September 1–6, 2019, Proceedings*, page 286–302, Berlin, Heidelberg, 2019. Springer-Verlag.

[7] F. J. Damerau. A technique for computer detection and correction of spelling errors. volume 7, pages 171–176. ACM, 1964.

[8] S. Du, T. Li, and S.-J. Horng. Time series forecasting using sequence-to-sequence deep learning framework. In *2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, pages 171–176, 2018.

[9] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers. *Introduction to Business Process Management*, page 1–33. Springer Berlin Heidelberg, 2018.

[10] J. Evermann, J.-R. Rehse, and P. Fettke. A deep learning approach for predicting process behaviour at runtime. pages 327–338. Springer, 2017.

[11] J. Evermann, J.-R. Rehse, and P. Fettke. Predicting process behaviour using deep learning. volume 100, pages 129–140. Elsevier, 2017.

[12] J. Fernandes, J. Reis, N. Melão, L. Teixeira, and M. Amorim. The role of industry 4.0 and bpmn in the arise of condition-based and predictive maintenance: A case study in the automotive industry. volume 11, page 3438. MDPI, 2021.

[13] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *CVPR*, pages 3205–3214, 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778. IEEE, 2016.

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov. 1997.

[16] A. Jalayer, M. Kahani, A. Beheshti, A. Pourmasoumi, and H. R. Motahari-Nezhad. Attention mechanism in predictive business process monitoring. In *EDOC*. IEEE, 2020.

[17] A. Jalayer, M. Kahani, A. Pourmasoumi, and A. Beheshti. HAM-net: Predictive business process monitoring with a hierarchical attention mechanism. volume 236, page 107722. Elsevier, 2022.

[18] B. Kang, D. Kim, and S.-H. Kang. Periodic performance prediction for real-time business process monitoring. *Industrial Management &amp Data Systems*, 112(1):4–23, Jan. 2012.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2017.

[20] G. Lakshmanan, D. Shamsi, Y. Doganata, M. Unuvar, and R. Khalaf. A markov prediction model for data-driven semi-structured business processes. volume 42, pages 97–126. Springer, 2013.

[21] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. volume 521, pages 436–444. Nature, 2015.

[22] B. Letham, C. Rudin, T. McCormick, and D. Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. arXiv, 2015.

[23] J. Libovický, J. Helcl, and D. Mareček. Input combination strategies for multi-source transformer decoder. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 253–260, Brussels, Belgium, Oct. 2018. Association for Computational Linguistics.

[24] 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*, pages 118–126. SIAM, 2019.

[25] Y. Liu, P. Sun, N. Wergeles, and Y. Shang. A survey and performance evaluation of deep learning methods for small object detection. *Expert Systems with Applications*, 172:114602, 2021.

[26] J. Moon, G. Park, and J. Jeong. Pop-on: Prediction of process using one-way language model based on nlp approach. *Applied Sciences*, 11(2), 2021.

[27] D. A. Neu, J. Lahann, and P. Fettke. A systematic literature review on state-of-the-art deep learning methods for process prediction. volume 55, pages 801–827. Springer, 2021.

[28] Y. Nishimura, K. Sudoh, G. Neubig, and S. Nakamura. Multi-source neural machine translation with missing data. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 92–99, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[29] Z. Niu, G. Zhong, and H. Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, Sept. 2021.

[30] P. Pascal and D. Ienco. *Proc. 25th International Conference on Discovery Science (DS 2022)*, volume 13601. Springer, 2022.

[31] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318. Pmlr, 2013.

[32] 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)*, pages 225–230, 2020.

[33] A. Rogge-Solti and M. Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In *Service-Oriented Computing*, pages 389–403. Springer Berlin Heidelberg, 2013.

[34] C. Sammut and G. I. Webb, editors. *Accuracy*, pages 9–10. Springer US, Boston, MA, 2010.

[35] M. A. Sandhu and A. Gunasekaran. Business process development in project-based industry. *Business Process Management Journal*, 10(6):673–690, Dec. 2004.

[36] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas. Predictive business process monitoring with LSTM neural networks. pages 477–492. Springer, 2017.

[37] I. Teinemaa, M. Dumas, F. M. Maggi, and C. D. Francescomarino. Predictive business process monitoring with structured and unstructured data. volume 9850, pages 401–417. Springer, 2016.

[38] W. van der Aalst. *Process Mining.* Springer Berlin Heidelberg, 2016.

[39] W. van der Aalst, M. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, Apr. 2011.

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. volume 30, 2017.

[41] 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. volume 10, pages 1–34. ACM, 2019.

[42] M. Weske. *Business Process Management: Concepts, Languages, Architectures*, pages 3–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[43] B. Wickramanayake, Z. He, C. Ouyang, C. Moreira, Y. Xu, and R. Sindhgatta. *Building interpretable models for business process prediction using shared and specialised attention mechanisms*, volume 248. 2022.

Appendix A

Tools and Technical Specifications

Here are listed the several tools and environments for software developing and testing that were used during this work.

A.1 Environment

All the models were implemented using Python with the functional Deep Learning API Keras. Dataset and experiments available in repository [1].

A.2 Hardware

Specific hardware (CPU, RAM, HDD and VRAM) for each environment is listed at Table A.1.

Table A.1: Hardware specifications for work environment

Personal Computer	
CPU	AMD Ryzen 5 3500u @2.1GHZ
RAM	8 GB
SDD	256 GB
Google Colab Pro+	
CPU	-
RAM	52GB
SDD	-
GPU	Nvidia GPU Tesla P100
VRAM	-

[1] `https://drive.google.com/drive/folders/1AQbI2bHwj2QfYd88k3YrDMiArcI0ByuQ?usp=sharing`

Appendix B

Examples of attention weight Matrices

Here are listed several examples of attention weight matrices that suggest a future work as an extension of this thesis.

Example of attention weights in a incomplete case with prefix 42 by the input of the activity encoder to the parallel cross-attention layer in the decoder

Fig. B.1 Example of attention weights in a incomplete case with prefix 3 by the input of the activity encoder to the parallel cross-attention layer in the decoder, where in both axis are the predicted tokens but in x-axis are one step before representing the input in the inference mode. Therefore, the decoder has access to the activities from the start token to the 42th activity (only the bottom part of the matrix is presented), and from there, it begins to predict until the end token is identified.

Fig. B.2 Example of attention weights in a incomplete case with prefix 22 by the input of the activity encoder to the parallel cross-attention layer in the decoder, where in both axis are the predicted tokens but in x-axis are one step before representing the input in the inference mode. Therefore, the decoder has access to the activities from the start token to the 42th activity (only the bottom part of the matrix is presented), and from there, it begins to predict until the end token is identified.

Fig. B.3 Example of attention weights in a incomplete case with prefix 42 by the input of the activity encoder to the cross-attention layer in the decoder, where in both axis are the predicted tokens but in x-axis are one step before representing the input in the inference mode. Therefore, the decoder has access to the activities from the start token to the 42th activity (only the bottom part of the matrix is presented), and from there, it begins to predict until the end token is identified.

Fig. B.4 Example of attention weights in a incomplete case with prefix 1 by the input of the activity encoder to the cross-attention layer in the decoder, where in both axis are the predicted tokens but in x-axis are one step before representing the input in the inference mode. Therefore, the decoder has access to the activities from the start token to the 42th activity (only the bottom part of the matrix is presented), and from there, it begins to predict until the end token is identified.

B.1 Prefix 3

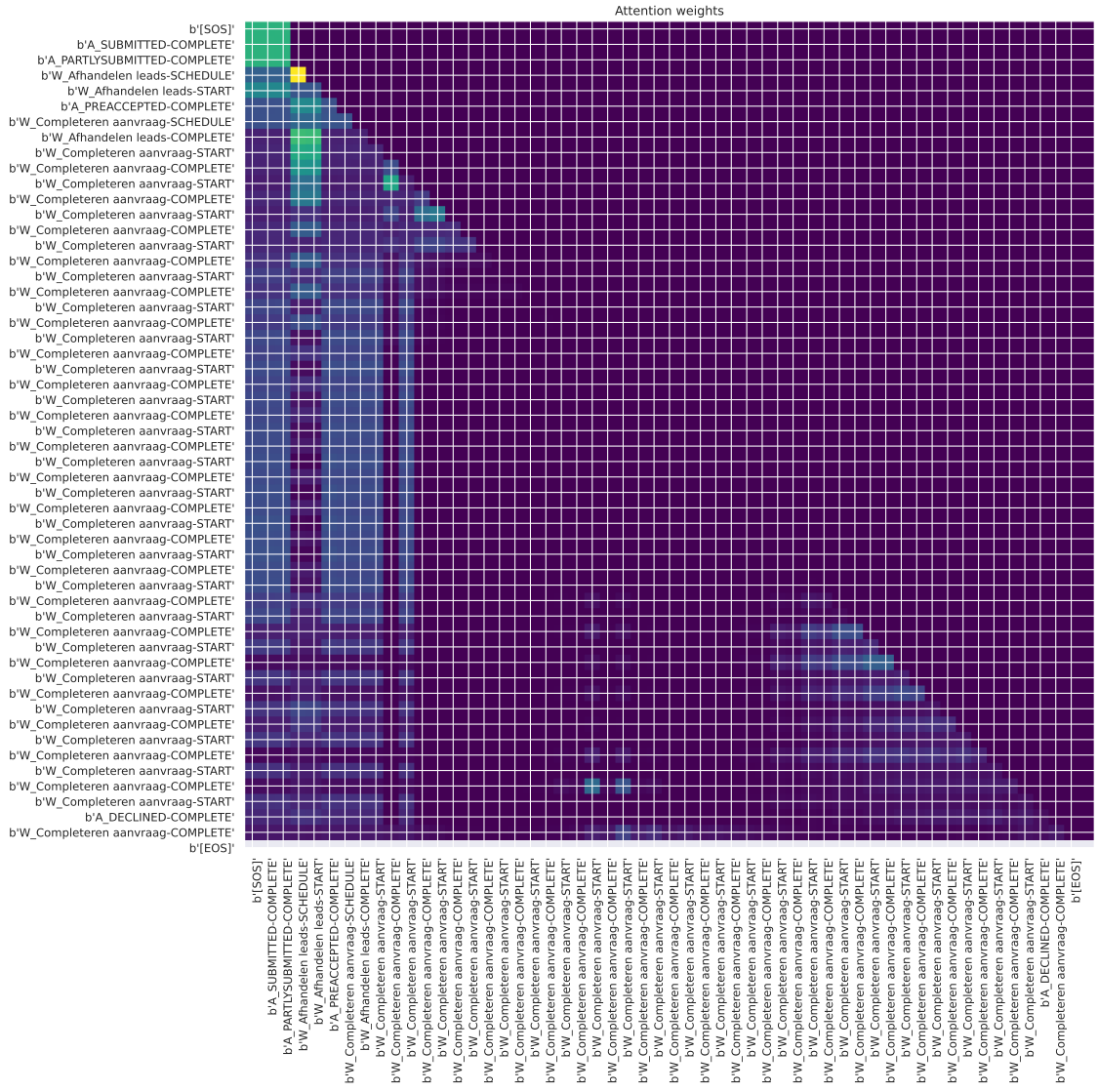


Figure B.1

B.2 Prefix 22

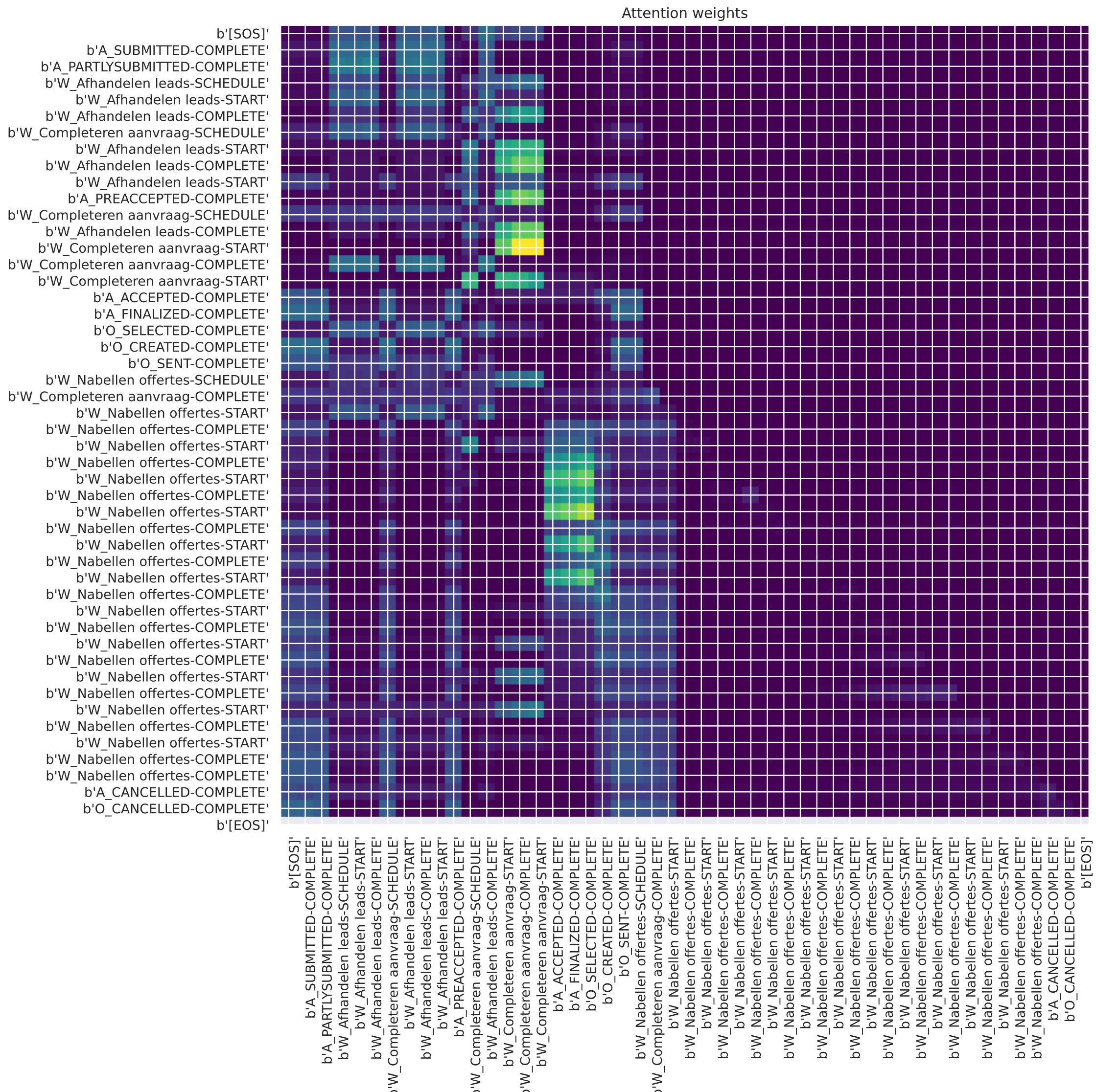


Figure B.2

B.3 Prefix 42 with only activity as input

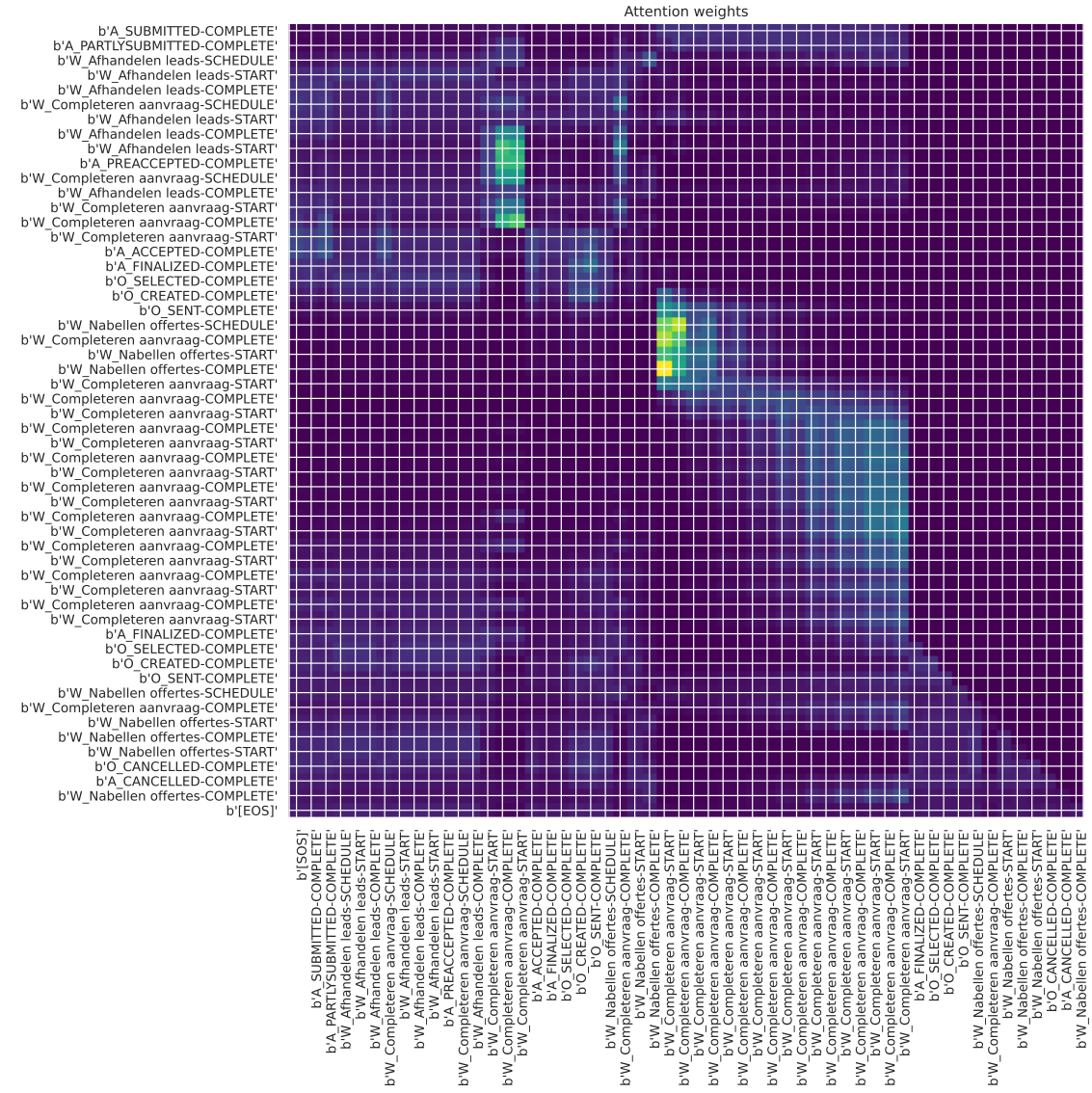


Figure B.3

B.4 Prefix 1 with only activity as input

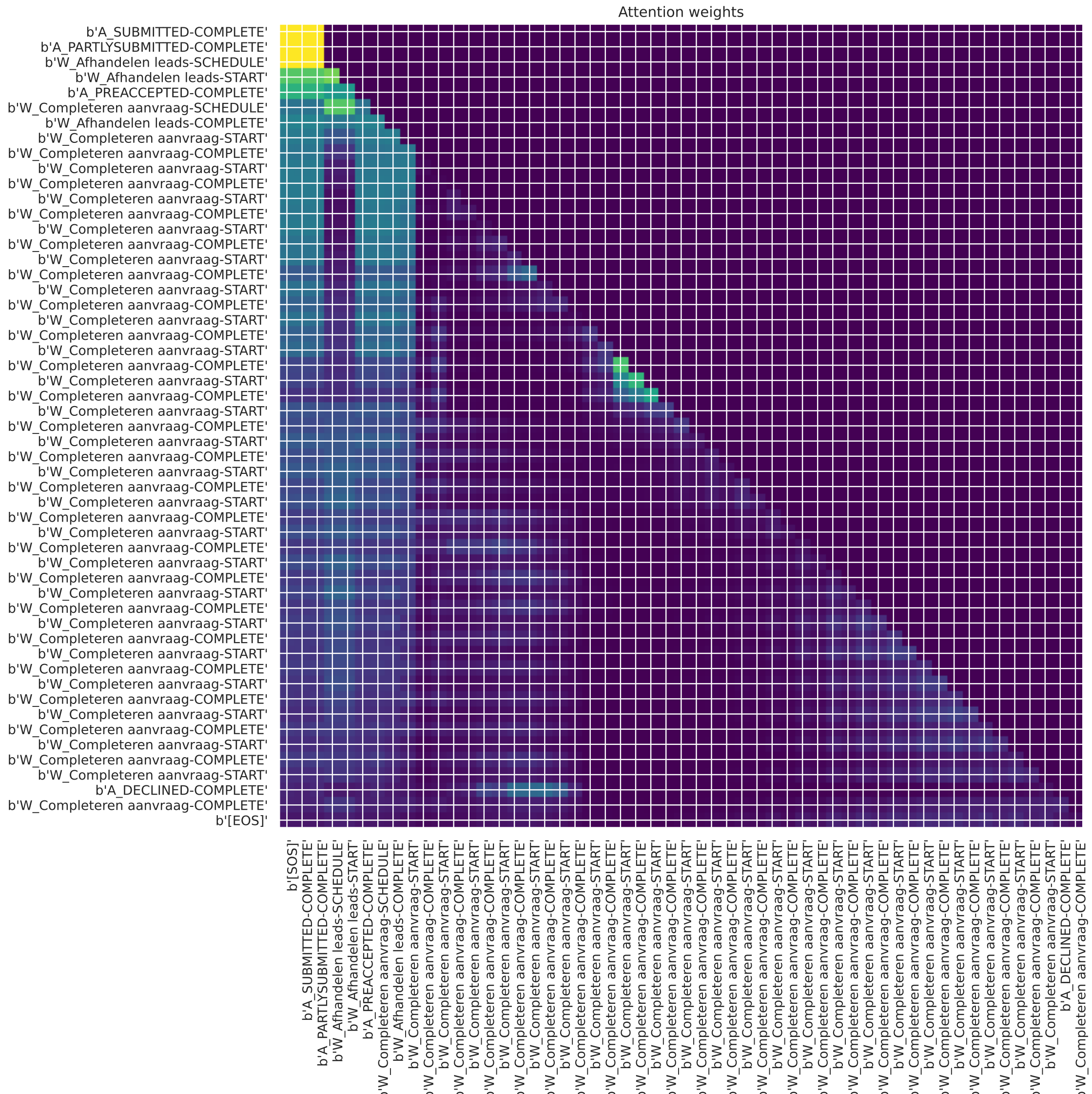


Figure B.4