

# Transformer-Based Partial Order Resolution of Event Logs

Master Thesis

presented by  
Daniel Häffner  
Matriculation Number 1520618

submitted to the  
Data and Web Science Group  
Prof. Dr. Han van der Aa  
University of Mannheim

April 2022

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	2
1.2	Contribution . . . . .	2
<b>2</b>	<b>Theoretical Framework</b>	<b>4</b>
2.1	Process Mining . . . . .	4
2.1.1	Event Data Uncertainty . . . . .	6
2.1.2	Partial Order Resolution . . . . .	7
2.2	Machine Learning . . . . .	10
2.2.1	Embeddings . . . . .	10
2.2.2	Sequence Models . . . . .	11
<b>3</b>	<b>Transformer-Based Architecture</b>	<b>13</b>
3.1	The Transformer Encoder . . . . .	13
3.1.1	Self-Attention . . . . .	14
3.1.2	Positional Encoding . . . . .	16
3.2	Data Preprocessing . . . . .	17
3.2.1	Masking . . . . .	17
3.2.2	Sampling . . . . .	18
3.3	Training and Evaluation . . . . .	19
3.3.1	Next Activity Prediction . . . . .	19
3.3.2	Decoding . . . . .	21
<b>4</b>	<b>Experimental Study</b>	<b>24</b>
4.1	Data . . . . .	24
4.1.1	Real-World Event Logs . . . . .	24
4.1.2	Synthetic Data . . . . .	25
4.1.3	Dataset Structure in the Framework . . . . .	26
4.2	Hyperparameter Searches and Experimental Setup . . . . .	27

4.2.1	Hyperparameter Searches . . . . .	27
4.2.2	Learning Rate . . . . .	28
4.2.3	Label Smoothing . . . . .	29
4.3	Probabilistic Baseline Models . . . . .	30
4.3.1	Trace Equivalence Model . . . . .	30
4.3.2	N-Gram Model . . . . .	31
4.3.3	Weak Order Model . . . . .	32
4.4	Results and Comparison with Probabilistic Models . . . . .	33
4.4.1	Real-World Event Logs . . . . .	33
4.4.2	Synthetic Event Logs . . . . .	34
4.5	Discussion . . . . .	36
<b>5</b>	<b>Conclusion</b>	<b>39</b>
5.1	Limitations . . . . .	40
5.2	Future Work . . . . .	40

# List of Algorithms

1	Decoding of an uncertain trace . . . . .	22
---	--	----

# List of Figures

2.1	Relation of the main sub-fields of process mining to the data source (software system) and the real-world actions [1]. . . . .	5
2.2	Example of a process model using Business Process Model Notation (BPMN). . . . .	6
3.1	Illustration of the transformer encoder architecture based on <i>The Illustrated Transformer</i> [2] . . . . .	15
3.2	Illustration of how the self-attention output for <i>Input 1</i> is calculated.	16
3.3	Visualization of the positional encoding embeddings of dimension 64 for an input of length 15. . . . .	17
3.4	Sampling of training examples from a certain trace. . . . .	19
4.1	Learning Rate Schedule . . . . .	29
4.2	Label Smoothing . . . . .	30
4.3	Illustration of the probabilistic models by van der Aa et al [3]. . .	31
4.4	Average of the results of the examined models on the synthetic data collections grouped by order of uncertainty. . . . .	37
4.5	Average of the results of the examined models on the synthetic data collections grouped by the ground truth event logs. . . . .	38

# List of Tables

2.1	Event data of one specific case. . . . .	8
4.1	Characteristics of the real-world event logs. . . . .	25
4.2	Characteristics of the synthetic event logs. . . . .	26
4.3	Accuracy Results of the different partial order resolution models. The best score for each dataset is highlighted. . . . .	33
4.4	Accuracy Results of the event logs with 20 % uncertain traces and 25 % events in uncertain event sets. The best score for each dataset is highlighted. . . . .	34
4.5	Accuracy Results of the event logs with 50 % uncertain traces and 50 % events in uncertain event sets. The best score for each dataset is highlighted. . . . .	35
4.6	Accuracy Results of the event logs with 99.5 % uncertain traces and 30 % events in uncertain event sets. The best score for each dataset is highlighted. . . . .	35
4.7	Accuracy Results of the event logs with 99.5 % uncertain traces and 60 % events in uncertain event sets. The best score for each dataset is highlighted. . . . .	36

# Abstract

Process mining is a fairly new computer science discipline that uses data mining techniques in the context of process analysis. The basis for any process mining project is the underlying event data which consists of the recorded event logs. For many process mining techniques one key attribute of the individual events in an event log is a precise timestamp to infer a total order of events. In practice the timestamps in event logs may not be precise enough, thus only allowing to infer a partial order. In this thesis a transformer-based model to predict the total order of a partially ordered event log is proposed. The proposed model acquires information from the totally ordered as well as the partially ordered traces in an event log and learns high level representations of the dependencies between events. The performance of the model is compared to three state-of-the-art probabilistic partial order resolution models on three real-world event logs as well as various synthetic event logs. The proposed model achieved the best accuracy on two of the three real-world logs and ranked second on the third.

# Chapter 1

## Introduction

The interest of companies to include more process oriented decision making became more prevalent in recent history. Traditionally the modelling of business processes and the process analysis, to achieve a more process oriented workflow, was done manually by humans through conducting interviews, workshops or similar methods [4]. With modern information systems in fields such as finance, production, and health care, data about the specific processes, supported by these systems can easily be gathered. These data can be used to automate parts or even the entirety of the previously manually handled process modelling and analysis tasks by applying data science techniques.

The field of techniques working on this is called process mining, which is an emerging data science discipline using process data to learn how a process is truly executed, to identify performance and conformance issues, and to leverage process information to optimize the considered process. The ubiquitous trend towards availability of cheaper storage solutions makes it more affordable to store large amounts of process data which is the basis for any process mining project. The data used for process mining are called event logs which are collections of recorded process executions called traces. Such a trace can be described as an ordered set of events. To generate these traces, each recorded activity execution must be labeled with a timestamp, so the correct order of the activity executions in a specific trace, called event, can be ensured.

However, the quality of the event log data is not always high enough to ensure that all the events have precise enough timestamps to infer a total order [5], making them only partially ordered. Contrary to this observation a key assumption of most process mining techniques is that all events of an event log have timestamps that allow to infer a total order which is often not the case.

As event logs can be seen as collections of sequences, numerous sequence



models used in other domains such as natural language processing (NLP) and image processing, have been experimented with in the process mining domain [6, 7]. These sequence models include RNNs, LSTMs and transformers. The transformer model with its self-attention mechanism, has become the dominant architecture for neural machine translation and natural language understanding [8]. It has thus aroused interest in trying to exploit its capabilities in sequence modeling for process mining tasks [9].

## 1.1 Problem Statement

As inferring a total order of events is crucial for several process mining techniques, van der Aa et. al [3] introduced the problem of partial order resolution. They argue that it is often possible to derive the total order of traces in an event log even when they are only partially ordered. A partially ordered trace hereby is a trace where two or more events cannot be definitively placed in a total order based on the gathered event data. This leads to multiple possibilities existing for the actual execution order called resolutions. They did this by applying probabilistic behavioural models on the event logs to correlate events.

The transformer architecture has shown to perform well on various sequence-to-sequence tasks [10, 11]. Since the goal is to find the most probable resolution (output sequence) of a partially ordered trace (input sequence), the partial order resolution can be seen as such a sequence-to sequence task. Therefore, the main goal of this work was to design a transformer-based approach for the task of partial order resolution and to evaluate the capabilities of the said model on event logs.

## 1.2 Contribution

This thesis provides a new approach for partial order resolution based on the deep learning transformer architecture. The implemented framework was specifically designed to handle this task and try to exploit the transformers capability of sequence modelling. This is done by learning high-level generic representations of the dependencies between events from an event log. The self-attention mechanism of the transformer model hereby allows to asses global dependencies between the events with its access to all relevant parts of the given input sequences.

An environment was created to compare the performance of the developed approach on the task of predicting the correct resolutions of the uncertain traces in an event log to the previously mentioned probabilistic models by van der Aa et. al [3]. An experimental study on three real-world event logs and 16 synthetically created event logs was then conducted and carefully documented for reproducibility. The

results and the framework of this experimental study can be used as a baseline for further experimentation with deep learning and partial order resolution.

The structure of thesis is as follows. Chapter 2 introduces the theoretical background relevant for understanding the problem of partial order resolution as well as the important aspects related to machine learning. Chapter 3 explains the general transformer-based architecture that is used for the partial order resolution. Then Chapter 4 illustrates and discusses the results and the setup of the experimental study, comparing the new transformer based approach to the probabilistic models. Chapter 5 concludes and points to the limitations of this work as well as giving potential directions for future research.

## Chapter 2

# Theoretical Framework

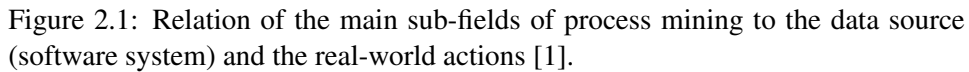
In this thesis the disciplines of process mining and machine learning are combined through the targeted use of deep learning techniques to solve the process mining problem of partial order resolution. The following chapter will introduce the necessary theoretical background knowledge to understand the transformer-based approach proposed in Chapter 3. The first part is an overview of the process mining related background in Section 2.1, followed by the machine learning related in Section 2.2.

### 2.1 Process Mining

The discipline of process mining focuses on the analysis of execution data of operational processes. It is comprised of a family of techniques which aim to learn how a process is truly executed, to identify performance and conformance issues and to leverage process information to optimize the considered process. Namely, these techniques can be grouped into the sub-fields of process discovery, conformance checking and process enhancement [1] as shown in Figure 2.1.

**Process Discovery** has the goal to create a process model from a record of events which depicts the flow of activities in a process, e. g. by visualizing the execution order of specific activities. The difficulty hereby lies in finding a representative model that captures the most common behaviour of the process and avoids over-fitting. The goal of the discovered model is to give insight into how activities relate to each other.

**Conformance Checking** aims to find deviations between the actual executions of a process to the according process model to improve the process, the process model or both. This can be done by deriving alignments, which indicate the magnitude of deviation between the model and the recorded execution.



The basis for the data driven techniques used in all the mentioned sub-fields are the recorded process data in the form of event logs. As with most data science applications, the quality of the data with which new insights on the underlying processes shall be obtained, is one of the key deciding factors for achieving the desired process mining goals. However, most event log data contain uncertainties making it very difficult to gain valuable insights.

It is essential for example in conformance checking, where the goal is to analyze deviations between the recorded and the desired behaviour of a process, that an event log consists of a set of events each of which carries at least a timestamp, a reference to an activity, and an identifier of the case for which an activity was executed for. With this information one can derive the traces that represent the different executions of the process that were recorded in the event log. These traces can then be used to check their conformance with a process model. An example

for such a model is depicted in Figure 2.2, henceforth referred to as model  $M$ .

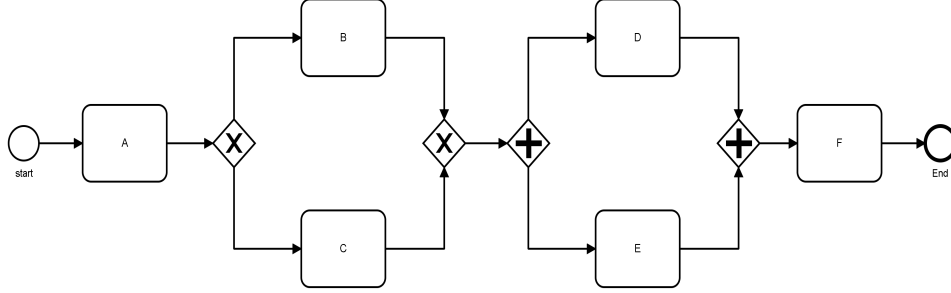


Figure 2.2: Example of a process model using Business Process Model Notation (BPMN).

This example of a process model consists of six distinct activities  $A, B, C, D, E, F$ . Activity  $A$  always comes first, followed by either  $B$  or  $C$  but not both. Then  $D$  and  $E$  are executed in parallel, meaning both will be executed eventually but the order may vary.  $F$  is the final activity of the process.

Assuming an event log obtained from the process represented by  $M$  consisting of the traces  $\pi_0 = \langle a, b, e, d, f \rangle$ ,  $\pi_1 = \langle a, c, d, e, f \rangle$ ,  $\pi_2 = \langle a, b, d \rangle$  and  $\pi_3 = \langle a, b, c, e, d, f \rangle$ , it is noticeable that traces  $\pi_0$  and  $\pi_1$  conform to  $M$  while  $\pi_2$  and  $\pi_3$  do not. The *non-conformance* in  $\pi_2$  manifests itself with the activities  $E$  and  $F$  missing and in  $\pi_3$  both activities  $B$  and  $C$  occur even though  $M$  specifies that only one of the two can occur in one process iteration. This *non-conformance* can be detected and analyzed by state-of-the art techniques [12].

Process discovery techniques on the other hand would take the traces  $\pi_0 - \pi_3$  to automatically derive an according process model. An example for such algorithm is the inductive miner [13], that creates a sound model that allows all behaviour recorded in an event log. The quality of the discovered model and its ability to represent the analyzed process is thus only as good as the quality of the event data that it used.

### 2.1.1 Event Data Uncertainty

For the success of both, the state-of-the-art conformance checking as well as process discovery and enhancement techniques it is crucial that the underlying event data quality along its dimensions such as accuracy, completeness, consistency, and currency, etc. [14] is sufficient.

This work focuses on the data quality issues of imprecise and missing timestamps

of events. The precision of the timestamps, especially the ability to create a total order for each trace from them is necessary to apply them and thus assumed to be present by various approaches [15]. However, this is often not the case and there are various reasons for the lack of data quality. Some of these reasons identified by van der Aa et. al [3] include:

- *Lack of synchronization*: process data may be recorded from different sources e.g. different information systems, which can lead to incomparable timestamps if the clocks are unsynchronized [16].
- *Manual recording*: this refers to timestamps that are not directly observed by information systems but rather manually recorded by people. If the specific recording task is not done thoroughly or an error occurs by accident the resulting data is faulty.
- *Data sensing*: data recorded by real-time locating systems cannot be assumed to be completely accurate since the construction of concrete events includes probabilistic inference [17, 18].

The faults mentioned above in the recording mechanisms lead to imprecise timestamps. Another reason for imprecise timestamps can also be that the granularity of the recording is too coarse to construct a total order. An example for such a coarse-granular trace is given in Table 2.1. The events of the depicted case are listed with their respective Event ID and timestamp. As event 2 and event 3 have the exact same timestamp it is not possible to infer the total order of the trace, without further insight, and it is thus only partially ordered. This results in the two traces  $\pi_4 = \langle a, e, b, d, f \rangle$  and  $\pi_5 = \langle a, b, e, d, f \rangle$  as possible resolutions for the case. Only  $\pi_5$  conforms to model  $M$ . While previous works, related to conformance checking with uncertain event logs either considered all possible resolutions equally likely or neglected the order uncertainty by only checking if one of the possible resolutions is conforming [19], van der Aa et. al [3] significantly improved the accuracy over the state-of-the-art by assessing a fine-granular total event order. Beside the benefits for conformance checking techniques, such a total order might also benefit process discovery techniques [20].

### 2.1.2 Partial Order Resolution

Before advancing to the proposed approach, the formal model to capture normative and recorded process behaviour for the task of partial order resolution proposed by van der Aa et. al [3] is introduced.

Event ID	Activity	Timestamp
1	<i>A</i>	9:00
2	<i>B</i>	10:00
3	<i>E</i>	10:00
4	<i>D</i>	11:00
5	<i>F</i>	12:00

Table 2.1: Event data of one specific case.

This example case of a specific execution of  $M$  has one uncertain event sets  $\{2, 3\}$  due to the imprecise timestamps of Event 2 and 3.

The foundation of recording process behaviour are recorded events. Each recorded event captures the execution of a distinct activity from the set of all possible activities  $A$ . The set of activities  $A$  must be defined according to the task at hand for the events to be correctly mapped. In the context of the previously introduced process model  $M$ ,  $A$  would be defined as  $\{A, B, C, D, E, F\}$

**Definition 1 (Event)** *Let  $A$  be the set of all activities,  $C$  the set of cases,  $T$  the time domain. An event is a tuple  $E = (a, c, t)$ , with  $a \in A, c \in C$  and  $t \in T$ . Events represent the recorded execution of a specific activity at a specific point in time.*

It must be noted that events can have more attributes, but the mentioned ones are indispensable and are the only relevant ones for this work. The case attribute refers to the specific process instance that the event was executed in. Analogues to the depiction in Table 2.1, events are also given a unique identifier, the *EventID* so they can be distinguished in case of missing or inaccurate timestamps.

**Definition 2 (Trace)** *In this work a trace  $t$  is defined as a sequence of disjoint sets of events,  $\sigma = \langle E_1, \dots, E_n \rangle$ , with  $E_\sigma = \bigcup_{1 \leq i \leq n} E_i$  as the set of all events of  $\sigma$*

While traces are usually defined as sequences of events that belong to the same case, the definition must be broadened to sets of events instead of single events as formalized in Definition 2, while the requirement of belonging to the same case stands for this work as well. Resulting from this definition an event set of  $\sigma$  denotes:

**Definition 3 (Event Set)**  $E_i, 1 \leq i \leq n$  for  $\sigma = \langle E_1, \dots, E_n \rangle$  and  $\forall e_j, e_k \in E_i : t_{e_j} = t_{e_k}$ .

an uncertain event set then is:

**Definition 4 (Uncertain Event Set)** *An event set  $E_i$  with a magnitude of  $|E_i| > 1$ .*

Resulting from definitions 2, 3, and 4 an uncertain trace is a trace that contains at least one uncertain event set and a certain trace is a trace that does not contain any uncertain event sets. Thus, a total order in a certain trace can be derived from the timestamps of all the individual events. In uncertain traces a total order can only be established between the different event sets but not within them. This results in uncertain traces only being partially ordered. A collection of all the uncertain and certain traces recorded for a specific process is an event log:

**Definition 5 (Event Log)** *An event log is a tuple  $L = (\Sigma, \lambda)$  where  $\Sigma$  is a set of traces and  $\lambda : \bigcup_{\sigma \in \Sigma} E_\sigma \rightarrow A$  assigns activities to all events of all traces.*

Following this, traces can be written as sequences of sets of activities instead of sequences of events. The example trace for the case in Table 2.1  $\sigma_0 = \langle \{1\}, \{2, 3\}, \{4\}, \{5\} \rangle$  can therefore also be written as  $\sigma_0 = \langle \{a\}, \{b, e\}, \{d\}, \{f\} \rangle$ .

To get the desired data quality for a downstream process mining tasks such as conformance checking or process discovery, the order uncertainty needs to be resolved. This can be done by predicting the most probable possible resolution for each trace, the total order of which will then be used for the task at hand.

**Definition 6 (Possible Resolutions)** *Given a trace  $\sigma = \langle E_1, \dots, E_n \rangle$ , possible resolutions for event sets and traces are defined as follows:*

- *any total order of an event set  $E_i$  over its events, i.e.,  $\Phi(E_i) = \{ \langle e_1, \dots, e_{|E_i|} \rangle \mid \forall 1 \leq j, k \leq |E_i| : e_j \in E_i \wedge e_k \in E_i \Rightarrow j = k \}$ .*
- *any total order of a trace  $\sigma$  over events of its event sets, i.e.,  $\Phi(\sigma) = \{ \langle e_1^1, \dots, e_1^{m_1}, \dots, e_n^1, \dots, e_n^{m_n} \rangle \mid \forall 1 \leq i \leq n : \langle e_i^1, \dots, e_i^{m_i} \rangle \in \Phi(E_i) \}$ .*

From now on possible resolutions will be written as the assigned activities for each resolution. For the example trace  $\sigma_0 = \langle \{a\}, \{b, e\}, \{d\}, \{f\} \rangle$  the two possible resolutions would be  $r_1 = \langle a, b, e, d, f \rangle$  and  $r_2 = \langle a, e, b, d, f \rangle$ . The approach proposed in this thesis aims to find the most probable of the possible resolutions using a transformer based deep learning architecture.

**Problem 1 (Partial Order Resolution)** *Let  $L = (\Sigma, \lambda)$  be an event log. The problem of partial order resolution is to derive for each trace  $\sigma \in \Sigma$  the most probable resolution  $\phi$ .*



It shall be noticed that the originally proposed problem of partial order resolution by van der Aa et. al [3] slightly differs from this one. They defined it as deriving the whole probability distribution over all possible resolutions for each trace because this was necessary to solve their proposed conformance checking problem. Since the main goal of this work is to assess the pure accuracy of predicting the original total order with a deep learning architecture, and there is no particular downstream task evaluated in this thesis it was refrained from deriving such a probability distribution.

## 2.2 Machine Learning

The foundation of the architecture used to tackle the problem of partial order resolution in this thesis lies within the field of machine learning. More specifically, it is the field of deep learning which uses artificial neural networks in combination with representation learning in supervised as well as unsupervised training settings. Supervised learning hereby refers to training models with labeled data and unsupervised learning without annotated data.

In the following the aspects of representation learning and deep artificial neural network architectures, which are relevant for this thesis, will be introduced in the form of embeddings in Section 2.2.1 and sequence models in Section 2.2.2.

### 2.2.1 Embeddings

As event logs consist of traces as a sequence of events which represent activities, it becomes relevant to obtain embeddings for these activities to train deep learning models on them. Embeddings are the foundation to represent each activity of the set of activities from the event log in an n-dimensional vector space instead of using another form of representation such as one-hot-encoding. Compared to one-hot encoding, the vectors are denser and can thus encode more information. While such embeddings can be pre-trained through machine learning tasks as De Koninck et. al have shown [21] it was chosen to initialize them randomly using Glorot initialization [22]. By using a large corpus of traces, training embeddings aims to place activities in a vector space in relation to each other depending on their context. The embeddings are trained together with the machine learning model they are used with.

Embeddings have shown to be an effective way of representing the meaning of words in relation to each other in NLP [23] and are also used frequently in recent deep learning architectures with process mining context [21, 7]. As state-of-the-art sequence models such as transformers use embeddings as their input in

the fields of NLP [24], image processing [25] and even some recent process mining related works [9]. This thesis aims to evaluate whether the transformer architecture can solve the problem of partial order resolution. It was chosen to use activity embeddings as the encoding method.

### 2.2.2 Sequence Models

Traditional machine learning techniques, generic feed forward neural networks and even deep neural networks consisting of only linear or convolution layers perform well in tasks such as speech recognition [26] and visual object recognition [27]. Nevertheless they are often not the best fit for modelling sequential data, since they can only be applied to inputs and targets with a fixed dimensionality. Meaning they neither keep the order of their inputs in memory nor typically have any other mechanisms to relate samples seen in the past to the current ones [28].

In contrast to those models, sequence models can deal with a variable length sequence as input and targets which makes them prevalent in fields that deal with sequential data, most prominently NLP [29] where sequences of words are often encountered. As process data is also of sequential nature a lot of the architectures that are used successfully in NLP are recently also applied to various process mining tasks.

**Recurrent Neural Networks (RNNs)** as proposed by Elman [30] introduced a way to retain the memory of past inputs by using an internal memory of these. They aim to encode dependencies over an entire sequence. An RNN takes as input a sequence  $(x_1, \dots, x_n)$  and outputs a corresponding sequence  $(y_1, \dots, y_n)$ . A vector representation of the sequence can be derived by taking the last output  $y_n$ . However, RNNs suffer from a problem known as vanishing gradients, which describes how the influence of a token on the final representation of the sequence fades over the length of the sequence.

**Long Short-Term Memory networks (LSTMs)**, a variation of RNNs, are a gated architecture that addresses this problem [31]. Although they usually perform better than RNNs due to avoiding the vanishing gradient problem, they still face a lot of the same limitations and are computationally expensive because of the long gradient paths and thus the difficulty of parallelization [32].

An architecture specifically designed to overcome these long gradient paths by avoiding recursion and thus making parallel computation a lot more efficient is the transformer architecture.

**Transformers** primarily distinguish themselves from RNNs and LSTMs through their non-sequentiality, self-attention and positional encoding. First introduced by Vaswani et al. [24] for neural machine translation the transformer was intended to be a more time and resource-effective alternative to the existing recursive ap-

proaches. This could be achieved through the self-attention mechanism that allows the transformer to derive relational attributes over long-range dependencies and thereby create input-output representations of sequential data. Hereby the relative importance of each token of the input sequence is assessed to all the other tokens in the input sequence according to their position.

Due to the success of the transformer in various NLP tasks [10, 11] as well as other domains [25], it was chosen as the focal point of this thesis. The exact architecture of the transformer-based approach to solve partial order resolution as well as a more detailed overview of the transformer architecture will be described in the following chapter.

## Chapter 3

# Transformer-Based Architecture

As event logs consist of sequential data and as transformers have shown to outperform RNNs, LSTMs and other sequence models in various domains [8, 25], this thesis addresses the problem of partial order resolution using a transformer encoder-based architecture which will be described in Section 3.1. The task of partial order as tackled in this work can be seen as a mix of supervised and unsupervised learning, hence it could be defined as a semi-supervised learning task.

This semi-supervised nature originates from the way that the input data for the transformer is generated. The uncertain traces, for which the partial order resolution is performed, of the event log account for the training data that could be described as unsupervised. This is due to the uncertain parts of the uncertain traces at hand staying uncertain and thus unlabeled during training via a masking technique specifically designed for this task. The certain traces of the event log on the other hand are used to create labeled example traces through sampling. These concepts will be further described in Section 3.2.

The generated input data is then embedded, and positional encoding is added to it before being passed through a transformer encoder which is trained on the custom task of next event prediction which later enables to assess the most probable resolution for an uncertain trace during validation and testing. This process and details about the training and validation of the proposed transformer-encoder architecture will be discussed in the following.

### 3.1 The Transformer Encoder

The originally introduced transformer architecture consists of a transformer encoder as well as a transformer decoder. It was decided to only use the transformer encoder architecture for this work, because the concept of *masked self-attention*,

which is the main differentiating attribute between transformer encoders and decoders, as well as the encoder-decoder architecture in general is not needed to train the model on the previously specified task.

The transformer encoder consists of one or several structural identical and connected encoder layers, followed by a linear and a softmax layer. Figure 3.1 depicts an example of a transformer encoder consisting of two encoder layer blocks. The input is a sequence of event embeddings. Before getting passed to the encoder layers, positional encoding vectors are added to each of the event embeddings. In each of the encoder layers, the inputs are passed through a self-attention layer simultaneously and then through a feed-forward layer individually. Next, to each self-attention and feed-forward layer, normalization is applied. After passing through one encoder layer the outputs, which are a sequence of embeddings, are passed to the next encoder layer as its inputs. Following the last encoder layer, the outputs are passed through a final linear and softmax layer to obtain the respective probability distribution for the output sequence.

As hinted at before a transformer decoder architecture would include a *masked self-attention* module before the self-attention module. The reason behind this is to prevent attention from tokens from the input sequence of future tokens (tokens to the right of the current one) during training to prevent bias during validation. This is not necessary for this task as future tokens are known during validation. In the following a deeper insight on the mechanisms of self-attention and the positional encodings will be given, as they are the defining modules of the transformer encoder.

### 3.1.1 Self-Attention

Self-attention is the foundation of all transformer-based models. It places the relevance of the individual items in a given sequence to each other without being restricted by their positional distance. Each token in a sequence has an attention score for every other token in the sequence, with higher attention scores indicating semantic similarity and thus enabling the model to capture the manifested dependencies in the input data. How these self-attention scores are obtained is illustrated in Figure 3.2. It illustrates how the output of one self-attention layer is generated for *input1*. For each of the three inputs, the so called query  $q$ , key  $k$  and value  $v$  vectors are created and their weights are initialized randomly using Glorot initialization [22] which will be updated later during the backpropagation step of the training phase. By then multiplying the inputs with  $q$ ,  $k$  and  $v$  the query, key and value representations for each of them are obtained. The attention scores for *input1* are generated by taking the scaled-dot product between the query of *input1* and the key of all the inputs respectively. The next step is taking the softmax of the

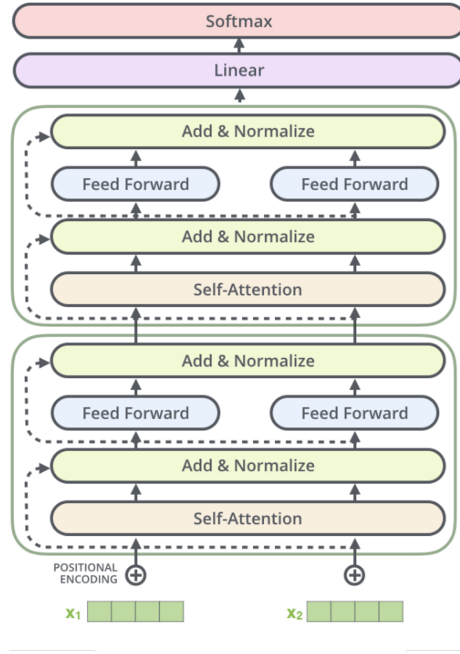


Figure 3.1: Illustration of the transformer encoder architecture based on *The Illustrated Transformer* [2]

scaled-dot products, which results in three attention scores for *input* 1. These are then multiplied with the respective value scores and then added together to obtain the output vector for *input* : 1. The whole process is repeated for all the inputs to obtain an output vector for each input vector.

The self-attention layers in the transformer encoder are further refined by adding a concept called **multi-headed attention**. What this does is, applying multiple of the described self-attention functions, called attention heads, to the same input by dividing the input into a fixed number of sub-vectors and applying self-attention to each. In practice this means that when for example using four attention heads, an input of the dimensionality 64 that each input vector would be split up into four 16-dimensional vectors and self-attention would be applied to each of them. The outputs are then concatenated by a separate weight matrix that is also randomly initialized and learned during training.

The multi-head attention allows the model to attend information from different information sub spaces improving its effectiveness for many state-of-the-art use cases [33].

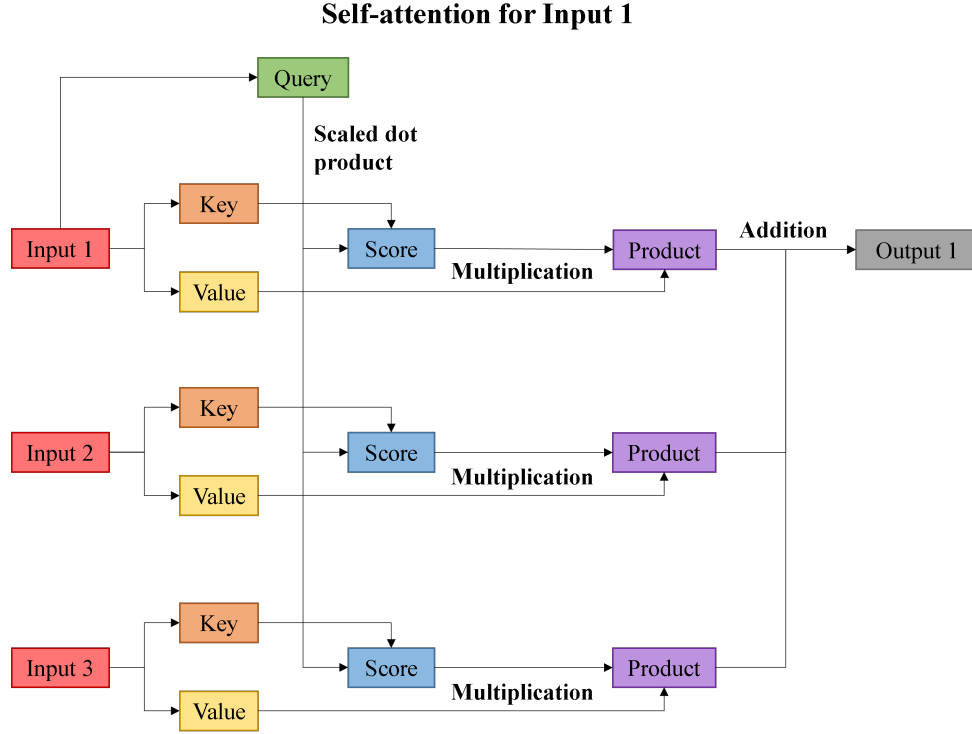


Figure 3.2: Illustration of how the self-attention output for *Input 1* is calculated.

### 3.1.2 Positional Encoding

During self-attention all the calculations are done simultaneously, and it does not account for the order of the input tokens. This is addressed by adding positional encoding vectors of the same dimensions as the input vectors to each input vector. While these positional vectors could also be initialized randomly and learned from scratch, state-of-the-art transformer models [24, 10] use sine and cosine functions of different frequencies:

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

This function is applied for each dimension  $i$  of each position  $pos$  of the input sequence generating a positional embeddings for each input token. This allows the model to determine the distance between each token of the sequence. Figure 3.3

shows a visualization of the positional encodings for a sequence of length 15 and embedding dimensionality 64.

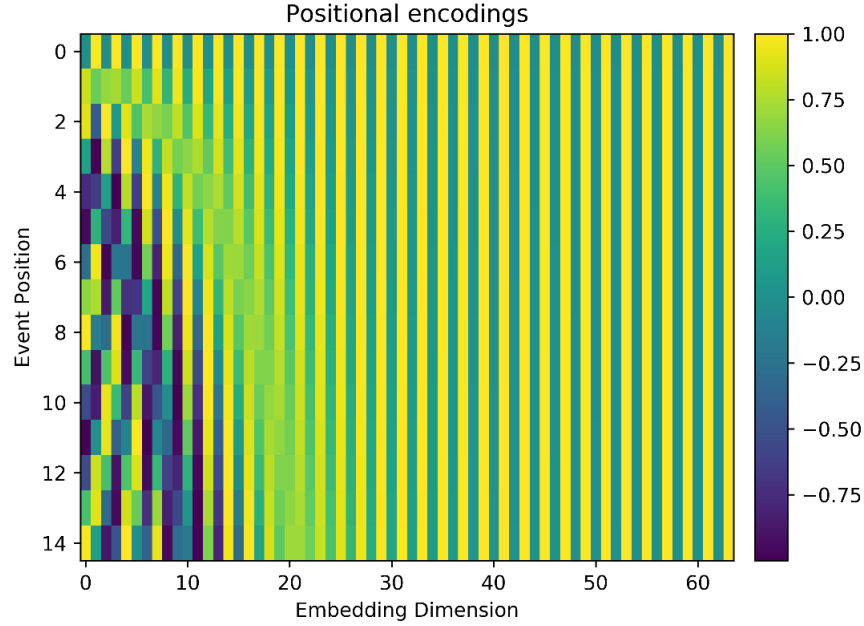


Figure 3.3: Visualization of the positional encoding embeddings of dimension 64 for an input of length 15.

## 3.2 Data Preprocessing

Data Preprocessing is the task of bringing the input data in the right format, so that the transformer architecture could work with it. The input data for the partial order resolution consists of recorded event data. From this event data, which can be provided in various forms (e.g xes, csv, etc.), the event log for which the partial order resolution model shall be created, was extracted. This was done by gathering all the certain traces and uncertain traces with their uncertain event sets as described in Section 2.1.2. The following Section will show how the certain and uncertain traces were modified to be ready for training and testing.

### 3.2.1 Masking

As one objective of this thesis was to incorporate as much information about the log at hand as possible, a way had to be found to include the information avail-



able from the uncertain traces, to help the model better grasp the behavioural regularities mentioned above while avoiding the introduction of bias. This was accomplished by masking the events in uncertain event sets, with a  $\langle MASK \rangle$  token. This masking process enables the model to learn dependencies between the certain parts and teaches it to recognize the common places where uncertainty occurs. During training, these masked traces were trained on the next event prediction task introduced above. For the example trace introduced in Section 2.1,  $\sigma_0 = \langle \{a\}, \{b, e\}, \{d\}, \{f\} \rangle$ , the masked trace would look as follows:

$$\sigma_0 = \langle \{a\}, \{ \langle MASK \rangle \}, \{ \langle MASK \rangle \}, \{d\}, \{f\} \rangle$$

and  $\sigma_0 = \langle a, \langle MASK \rangle, \langle MASK \rangle, d, f \rangle$  in short-hand notation. It was also experimented with ways to incorporate information about the specific events in the uncertain event sets by, for example, introducing specific tokens for all the combinations of events in uncertain event sets. This was discarded because of the large number of possible combinations of event sets and the resulting difficulty for the model to categorize these tokens.

### 3.2.2 Sampling

In contrast to the information carried by the uncertain traces, the information carried by the certain traces could directly be used for training. As these certain traces are the most reliable source for extracting high quality behavioural regularities of the event log, sampling was used to generate training examples from them that also incorporate the masking previously described in Section 3.2.1.

The idea was to create "*uncertain*" example traces that can be used for training in addition to the uncertain traces from the event log. The advantage of the sampled uncertain traces is that the ground truth of their order is known. This was done by creating all possible combinations of uncertain event sets for a certain trace and then masking them. Figure 3.4 illustrates how sampled training examples are generated from a certain trace. It shows that a large amount of training samples can be generated from each certain trace. Through this method the model learns how to resolve the masked parts of uncertain traces and internalize the dependencies of the events from the certain traces.

The idea for this sampling was inspired by Smith and Eisner [34] who introduce negative sampling as a method to create negative examples for training tasks with a high ratio of positive training examples.

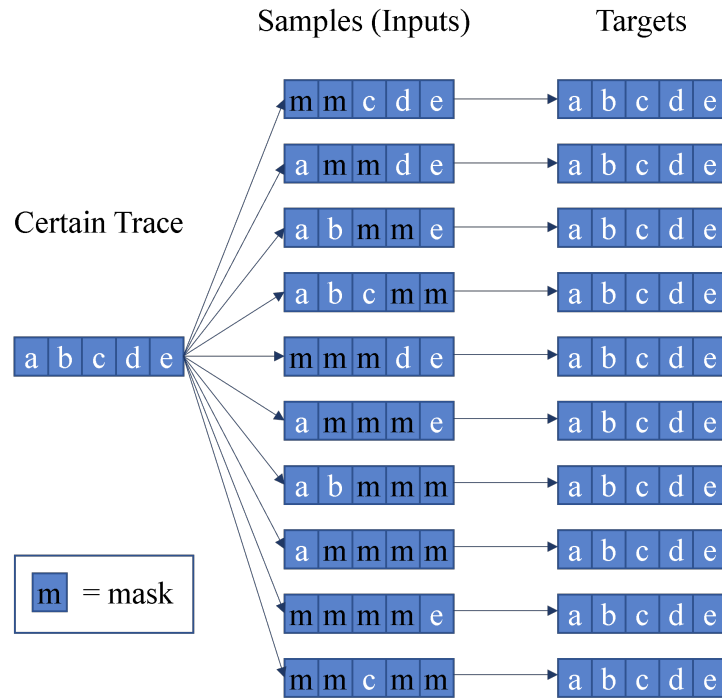


Figure 3.4: Sampling of training examples from a certain trace.

Shown is a certain trace consisting of the activities a,b,c,d and e. Through the sampling of all of the possible combinations of uncertain event sets and masking them accordingly, 10 distinct training examples can be generated.

### 3.3 Training and Evaluation

In this section the training task that the transformer-encoder model was trained on to then predict the most probable resolutions, for the uncertain traces in the validation and training phase will be explained and then the decoding technique for actually getting the most probable resolution will be demonstrated.

#### 3.3.1 Next Activity Prediction

One of the main questions when designing an architecture for a deep learning model is, how to structure the training so the model can solve the desired task. A sequence model can be trained to convert an input sequence to an output sequence through gradient decent. As the proposed model is a sequence model, a way had to

be found to enable the model to convert an uncertain trace into a prediction for the most probable resolution. The training task the model was chosen to learn is next event prediction. This means that for every event  $e_n$  in the input trace  $\sigma$  the target would be the event  $e_{n+1}$  from either  $\sigma$  itself, if the current input trace is a masked uncertain trace, or  $e_{n+1}$  from  $\sigma_{certain.trace}$  the certain trace, which the sample was generated from, if the current input trace is one of the sampled masked traces. In order for the model to also learn the first and last tokens of each trace without running into trouble because the input and target traces have to be the same length, each input and target trace is surrounded with a start of trace token ( $< SOT >$ ) and an end of trace token ( $< EOT >$ ) before feeding it to the transformer.

Here is an example for an uncertain trace from the event log:

$$\sigma_0 = \langle \{a\}, \{b, c\}, \{d\} \rangle$$

To generate an input sequence for training from this, first the  $< SOT >$  and  $< EOT >$  would be added and the masking of the uncertain event sets (in this case just  $\{b, c\}$ ) would be applied resulting in:

$$input_0 = \langle < SOT >, a, < MASK >, < MASK >, d, < EOT > \rangle$$

From this the corresponding target can be generated by shifting the whole sequence to the left by one token. Since this leaves the target sequence with one token less as the input sequence, the  $< EOT >$  is removed from the input sequence so that input and target are of the same length and no information from the trace is lost.  $Input_0$  and  $target_0$  then are:

$$input_0 = \langle < SOT >, a, < MASK >, < MASK >, d \rangle$$

$$target_0 = \langle a, < MASK >, < MASK >, d, < EOT > \rangle$$

This procedure is applied to each of the uncertain traces in the event log, resulting in the first part of the input, target tuples used for training.

The second part of the input, target tuples are generated from the remaining certain traces from the event log. An example for the generation of the input, target tuple for training from a certain trace  $\sigma_1$  would look as follows:

$$\sigma_1 = \langle \{a\}, \{b\}, \{c\}, \{d\} \rangle$$

Six different samples of masked traces can be generated from  $\sigma_1$  as described in Section 3.2.2. One of them is presented here:

$$input_1 = \langle < SOT >, < MASK >, < MASK >, c, d, < EOT > \rangle$$

after adding  $\langle SOT \rangle$  and  $\langle EOT \rangle$ . For this sample as well as each sample generated from  $\sigma_1, \sigma_1$  shifted by one token to the left would be the target, after also adding  $\langle SOT \rangle$  and  $\langle EOT \rangle$ . For this example, the input target tuple would look as follows:

$$input_0 = \langle \langle SOT \rangle, \langle MASK \rangle, \langle MASK \rangle, c, d \rangle$$

$$target_0 = \langle \langle MASK \rangle, \langle MASK \rangle, c, d, \langle EOT \rangle \rangle$$

Additionally to the masked samples the identity of the certain trace was also included as an input and target sequence. The target being identical to the input in this case, but shifted one token to the left as the other targets. Also these identities were included in the training data as many times as the variant of the certain trace occurred, while the masked samples were only included once per variant, with a variant of a trace being one distinct sequence of activities. After this process has been executed for all generated samples from all certain traces, and combining these with the training inputs from the uncertain traces, the training data is complete, and the transformer can be trained. It now learns to predict the next token for each of the input tokens of each sequence and the self-attention assesses the dependencies between the tokens to each other manifested in the event log. How the trained model will then be used to solve the partial order resolution is described in the next section.

### 3.3.2 Decoding

Decoding is the final step in the pipeline of this deep learning framework and the point at which the actual partial order resolution occurs. It is used during the evaluation of the uncertain traces against the *ground truth*. Further details on the acquisition of this *ground truth* will be discussed in Chapter 4 in connection with the data used in the conducted experiments.

This section will focus on the algorithm to obtain a prediction for the most probable resolution for an uncertain trace using the previously trained transformer-encoder. As the actual goal of the transformer model is not next event prediction but rather to show whether it is possible to predict the right resolution for uncertain traces, a method is needed to extract the important information for the actual task from the transformer while maintaining its capability of recognizing the learned dependencies between activity executions from the trace.

In this work this process is referred to as decoding the uncertain traces. The method used for decoding is presented in Algorithm 1. It is an iterative procedure that predicts the most probable next token for each uncertain event set until all the uncertain event sets are completely resolved.

**Algorithm 1** Decoding of an uncertain trace

**Data:**  $\sigma$ , an uncertain trace,  $T$ , the trained transformer model,  $\Omega_\sigma$ , set of uncertain event sets,  $P$ , positions of uncertain event sets

**Result:** the predicted resolution

```

while not all uncertain events predicted do
  for  $\forall \omega \in \Omega_\sigma$  do
    while  $\omega$  not empty do
      predict event  $e \in \omega$  at  $p_\omega \in P$  with  $T$ 
      replace  $\langle MASK \rangle$  at  $p$  in  $\sigma$ 
      delete  $e$  from  $\omega$ 
    end
  end
end

```

The algorithm takes as input the uncertain trace for which the most probable resolution shall be predicted  $\sigma$ , the trained transformer model  $T$ , the uncertain event sets  $\Omega_\sigma$  of the uncertain trace and the respective positions of the uncertain event sets  $P$ .

Now  $\sigma$  is passed through  $T$  and the first  $\langle MASK \rangle$  token of each uncertain event set in the input sequence is replaced with the event from the respective event set that has the highest probability of occurring at that position according to  $T$ . The predicted events are now deleted from the uncertain event sets and the procedure is repeated with the updated uncertain trace  $\sigma$  and the updated uncertain event sets  $\Omega_\sigma$  and their updated positions  $P$ . The following example will illustrate this procedure in detail. Let's take  $\sigma_0$  as an example for an uncertain trace from the event log:

$$\sigma_0 = \langle \{a\}, \{b, c\}, \{d\}, \{e, f\} \rangle$$

with the uncertain event sets:

$$\omega_0 \in \Omega_\sigma = \{b, c\} \text{ and } \omega_1 \in \Omega_\sigma = \{e, f\}$$

The masked uncertain trace that would be fed to the transformer model would be:

$$input_0 = \langle \langle SOT \rangle, a, \langle MASK \rangle, \langle MASK \rangle, d, \langle MASK \rangle, \langle MASK \rangle \rangle$$

Under the assumption that the transformer would predict  $b$  to be more likely than  $c$  to follow  $a$  and  $f$  to be more likely than  $e$  to follow  $d$ , the resulting updated input  $input_0$  and uncertain event sets  $\omega_0 \in \Omega_\sigma$  and  $\omega_1 \in \Omega_\sigma$  would be:

$$input_0 = \langle \langle SOT \rangle, a, b, \langle MASK \rangle, d, f, \langle MASK \rangle \rangle$$

$$\omega_0 \in \Omega_\sigma = \{c\} \text{ and } \omega_1 \in \Omega_\sigma = \{e\}$$

As there is only one event left in each of the uncertain event sets, the predicted output would be:

$$output_0 = \langle a, b, c, d, f, e \rangle$$

after removing the  $\langle SOT \rangle$  and it could be compared to the corresponding ground truth trace. The uncertain event sets  $\omega_0$  and  $\omega_1$  would be empty and removed from  $\Omega_\sigma$ .  $\Omega_\sigma$  being empty is the stopping condition for the algorithm. It can happen, of course, that not all the uncertain event sets are of the same magnitude. In this case the event sets of smaller magnitude would simply be removed from  $\Omega_\sigma$  before the ones of larger magnitude and the algorithm would continue with only the remaining uncertain event sets.

## Chapter 4

# Experimental Study

This chapter describes the experiments that were conducted to evaluate the accuracy of the transformer-based partial order resolution framework and to compare against the probabilistic partial order resolution models proposed by van der Aa et al [3]. This aims to validate the proposed approach, as well as creating a reproducible baseline for deep learning approaches to the partial order resolution task.

The implementation of the architecture used for the experiments described below is based on pytorch and enables training, reproducible evaluation, and hyperparameter optimization of the proposed approach as well as the comparison with the probabilistic baseline models.

### 4.1 Data

In this section the datasets, which included real-world as well as synthetic data collections, used in the conducted experiments will be introduced and a brief overview of the implemented dataset structure will be given.

#### 4.1.1 Real-World Event Logs

Three real-world event logs were chosen, their characteristics are shown in Table 4.1. This selection was inspired by the data set selection in the work of van der Aa et al. [3], who also chose the same three logs for the evaluation of their partial order resolution of event logs for conformance checking. They were also chosen because they differ considerably in essentially all their characteristics and thus cover a broad range of event log characteristics.

Characteristics	Traffic Fines [35]	BPI 12 [36]	BPI 14 [37]
$ A $	12	25	10
Traces	150,370	13,087	41,353
Uncertain Traces	9,166 (6.1%)	5,006(38.1%)	38,649(93.5%)
Variants	231	4,366	14,948
avg. trace length	3.7	20.0	8.9
max. trace length	20	175	167
Events	561,470	262,200	369,485
Events in uncertain event sets	21,060	10,280	83,766

Table 4.1: Characteristics of the real-world event logs.

Especially the number of uncertain traces that ranges from 6.1% to 93.5% is an important attribute that highly influences the difficulty of the partial order resolution, and it shows that coarse-grained timestamps exist in the real world, thus making the applicability of partial order resolution more relevant.

#### 4.1.2 Synthetic Data

In order to test different aspects of the transformer model during development as well as to get a wider range of event data for the comparison of the developed deep learning and the probabilistic baseline models, four synthetic sets of event data with varying characteristics were generated and will be referred to as SD 1, SD 2, SD 3 and SD 4. They were generated using the event log generator of van der Aa et al. [3] who implemented the generator using the state-of-the-art process model generation technique of Jouck Depaire [38] due to its ability to generate non-structured models, e.g., models that include non-local decisions. This technique allows to introduce aspects such as loops, arbitrary skips, and non-free choice constructs to the generated models. The parameters of the model generation were set as the default values in the technique of the developers. These models were then used to generate the synthetic event data. The main characteristics of the four generated sets of event data are shown in Table 4.2.



Characterisitcs	SD 1	SD 2	SD 3	SD 4
$ A $	11	17	13	13
Traces	20,000	20,000	25,000	20,000
Variants	2,024	6,761	3,251	6
avg. Trace length	5.4	7.1	5.3	6.0
max. Trace length	10	14	10	7
Events	190,212	141,130	133,557	120,236

Table 4.2: Characteristics of the synthetic event logs.

These sets of event data were generated using varying levels of noise, random inserts, swaps, and event removal. The order uncertainty of these sets of event data, as they were fully ordered by default, was added after their generation in four different degrees of magnitude resulting in four distinct event logs per set of event data resulting in a total of 16 synthetic event logs. This was done by first, randomly choosing the traces that should contain uncertain event sets according to the desired ration of uncertain traces, and secondly by randomly choosing the events which should belong to uncertain event sets from the previously chosen uncertain traces also according to the desired ratio. For clarification the ratio of events in uncertain event sets relates to the total number of events in uncertain traces, not to the total amount of events. The same four ratios for uncertain traces and events in uncertain event sets were applied for comparison to each of the synthetic data collection. The four chosen "*uncertain traces / events in uncertain event sets* " ratio tuples were 25%/20%, 50%/50%, 99.5%/30% and 99.5%/60%. The first tuple was chosen to simulate an event log where order uncertainties do not occur very often, the second one as one where exactly half of the traces as well as events are uncertain, and the last two as cases of very high and extreme uncertainty. The order of the event sets as it was before applying the randomization was used as the *ground truth* log for validation and testing. Even though the mentioned ratios of uncertain traces and events in uncertain event sets and were the only ones that experiments were conducted on in this work, the implemented framework allows to create uncertainty of any magnitude given a *ground truth* log.

#### 4.1.3 Dataset Structure in the Framework

This section will give some insight into the handling of data structures of the event logs used for the experiments the implementation. The storing of the event logs is handled for each training, validation and test set by storing the input, the target sequences and in addition the uncertain event sets of each uncertain trace.

After generating the event log data for further processing, i.e., dividing it into certain traces, uncertain traces, the ground truth, and the positions and length of the uncertain event sets, it is stored and can be reloaded at any time to reproduce the experimental results. This is of particular significance for the synthetically created event logs, since their order uncertainty is generated by the framework. The uncertain traces were additionally splitted into a validation set for the hyperparameter searches and a test set for the evaluation. Both of these sets were of equal size. As described earlier the uncertain traces as a whole were also used for training after masking them. During the creation of the data structures required to further process the event data, a vocabulary of all of the occurring activities represented by the events in the event log is built, which is used to numericalize these activities for further processing. Each activity receives an index which is then used to access the according embedding.

As not all the input traces have the same length, padding is added before each batch passes the transformer as it is common practice for sequence models. The padding tokens are excluded from the attention calculation and thus also the back propagation.

## 4.2 Hyperparameter Searches and Experimental Setup

As hyperparameters are set parameters that control the learning process of a machine learning model and thus highly influences the outcomes of machine learning tasks, this section focuses on explaining how the hyperparameter searches were conducted as well as illustrating the used learning rate schedule and the concept of label smoothing.

### 4.2.1 Hyperparameter Searches

For each of the introduced datasets a random hyperparameter search was conducted. It was chosen to do a random search instead of e.g., a grid search since that way a higher hyperparameter space could be explored with a lot less resources. Moreover random hyperparameter searches have been shown to be an effective method for deep learning in other domains [39].

For each dataset 15 hyperparameter searches were performed with each model being trained for a maximum of 150 epochs but stopping early if the validation accuracy did not improve for the last 15 epochs. The models were evaluated after each epoch on the validation sets. The best model was selected its performance on the test set. If the best model did not reach the early stopping point during the hyperparameter search, training for this model was continued for another 150

epochs.

The following hyperparameters were included in the random searches: embedding size selected from  $\{64, 128, 256\}$ , the dimension of the feed forward layers in the transformer encoder layers selected from  $\{1, 024, 2, 048\}$ , the number of transformer encoder layer selected from  $\{1, 2, 3\}$ , the number of attention heads of the multi-head attention from  $\{4, 8, 16\}$ , the dropout probability for the transformer encoder itself as well as the positional embeddings from the range  $[0.1, 0.8]$  at first but early experiments showed better results with lower dropout values so the range was changed to  $[0.1, 0.4]$ .

The batch sizes varied between 64, 128 and 256 depending on the event log and the machine that the experiments were conducted on as different machines with different capabilities were used.

The specific hyperparameters of the learning rate are discussed in the next Chapter.

#### 4.2.2 Learning Rate

The learning rate schedule used to train the in this work introduced model was proposed by Vaswani et al. and is calculated for each training step as follows:

$$lr = lr\_factor \times (d_{model}^{-0.5} \times \min(step\_num^{-0.5}, step\_num \times warmup\_steps))$$

and was used with the Adam optimizer [40] with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.98$  and  $\epsilon = 10^{-9}$ . The  $lr\_factor$  was not included in the original proposition for this learning rate scheduler, but was added so more variations of different learning rate growths could be experimented with. Figure 4.1 depicts the bounds of using this scheduler with the embedding dimensions 64 and 128 with  $lr\_factor = 1.0$ . As mentioned before the warm-up steps were randomly selected between 4,000 and 16,000 during the hyperparameter searches. As can be seen, for a given dimensionality the learning rates approach the same value after the set amount of warm-up steps. The  $lr\_factor$  helps overcome this limitation. The learning rate rises linearly until the appointed number of warm-up steps is reached and then declines inverse to the square root of the step number. One step being one iteration of the back-propagation mechanism which occurred once per epoch in this experimental study

It was chosen to use this learning rate schedule because it achieved good results for the original transformer and early experiments with fixed learning rates suggested that it also performs better for the approach developed in this work.

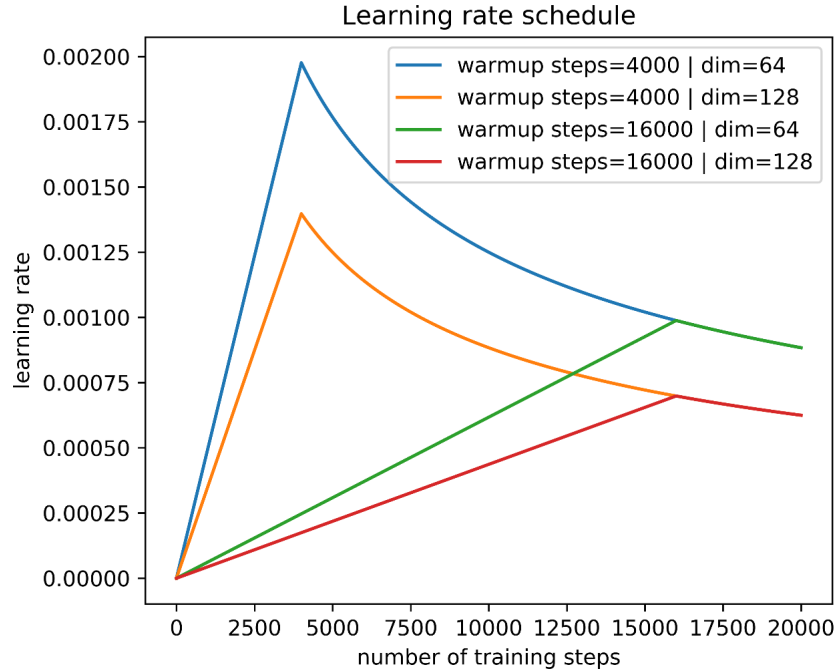


Figure 4.1: Learning Rate Schedule

Depicted are the developments of the learning rates, using the described learning rate schedule. It shows the impact of the chosen number of *warmup\_steps* on the maximum learning rate as well as the slope. For a given dimensionality the learning rate converges to the same value for any number of *warmup\_steps* a given dimension.

### 4.2.3 Label Smoothing

Label smoothing is a regularization technique [41] which helps preventing the problems of overfitting and overconfidence. It replaces the one-hot encoded vector of the softmax function during loss calculation with a *smoothed* one. This means the value of the correct label, in the context of this work the next token, is set to a confidence value of usually 0.9 instead of 1 and the rest of the mass (usually 0.1) is distributed over the rest of the vocabulary. This makes the model more adaptive as otherwise it would get "too confident" of its predictions and overfit the training data and results in better model calibration [42]. Label smoothing hurts perplexity, as the transformer model learns to be more unsure, but improves accuracy.

A visualization of a label smoothed vs. a one-hot encoded distribution is given in Figure 4.2. It depicts the confidence values of a numericalized target sequence

$target = 1, 2, 3, 2, < PAD >$ . 0 hereby denotes the padding token  $< PAD >$  which is excluded from the label smoothing and thus denoted with a confidence value of 0. The non-target values have a slightly higher value than 0 in the label smoothed distribution while being 0 in the one-hot encoded distribution.

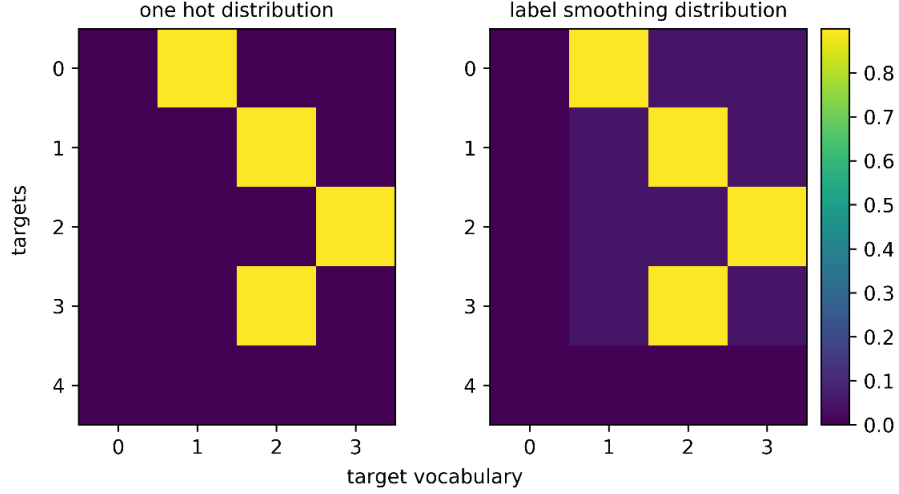


Figure 4.2: Label Smoothing

Compared are the target value distributions of the one-hot and the label smoothing distribution. The label smoothing distributes a small amount of the probability mass over the whole target vocabulary except the padding token while the one-hot distribution assigns 0 to all of the non-true labels.

### 4.3 Probabilistic Baseline Models

To assess the quality of the transformer based partial order resolution it will be compared against the three probabilistic partial order resolution approaches introduced by van der Aa et. al [3], illustrated in figure 4.3, will be outlined in this section.

#### 4.3.1 Trace Equivalence Model

This model estimates the probability of a resolution by observing how often the respective sequence of activity executions can be found in the event log, in traces without order uncertainty. Two resolutions are equivalent, if they represent the same sequence of executed activities.

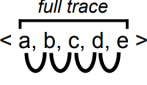
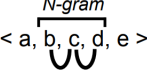
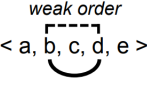
Model	Illustration	Basis
Trace equivalence		Equal, certain traces
N-gram		Equal sub-sequences of length $N$
Weak order		Indirectly follows relation of events

Figure 4.3: Illustration of the probabilistic models by van der Aa et al [3].

The certain log is defined as  $\Sigma_{certain} = \{\sigma \in \Sigma \mid |\Phi(\sigma)| = 1\}$  with  $L = (\Sigma, \lambda)$  as an event log and  $\sigma, \sigma' \in \Sigma$  as two traces of the same length. Let  $\phi \in \Phi(\sigma)$  and  $\phi' \in \Phi(\sigma')$  be two of their resolutions, the probability of a resolution  $\phi \in \Phi(\sigma)$  of a trace  $\sigma$  is then quantified as:

$$P_{trace}(\phi) = \frac{|\{\sigma \in \Sigma_{certain} \mid \exists \phi' \in \Phi(\sigma) : \phi' \equiv \phi\}|}{|\Sigma_{certain}|}$$

For a given uncertain trace, this approach measures how often a particular resolution occurs in the certain log and enables direct assessment of the most probable resolution. It is the simplest of the three probabilistic models and may only have limited applicability as it only considers the traces without uncertain event sets.

### 4.3.2 N-Gram Model

N-Gram models consider adjacent activity executions with a sliding window size of  $n$ . Consequently, a trace can be split into its n-Grams.

Given a resolution  $\phi = \langle a_1, \dots, a_n \rangle$  firstly a probability distribution for each activity respective position is estimated. For  $N = 4$  and the possible resolution  $\phi_1 = \langle a, b, c, d, f, g \rangle$  the probability of  $f$  being at the fifth position as it is in  $\phi_1$  would be estimated by the probability of  $f$  following the sequence  $\langle b, c, d \rangle$ . The predicate *certain* in this case for an activity sequence  $A = \langle a_1, \dots, a_m \rangle$  and a trace  $\sigma = \langle E_1, \dots, E_n \rangle$  is defined as the trace containing the respective activity sequence without order uncertainty ( $m \leq n$ ):

$$certain(A, \sigma) \iff \exists i \in \{0, \dots, n-m\}, \forall j \in \{1, \dots, m\} : E_{i+j} = \{e_{i+j}\} \wedge \lambda(e_{i+j}) = a_j$$

The probability of an activity  $a$  to follow an activity sequence  $\langle a_1, \dots, a_m \rangle$  is then dependent on the number of times this activity sequence is followed by  $a$  in the observed event log versus the number of times it is not:

$$P(a | \langle a_1, \dots, a_m \rangle) = \frac{|\{\sigma \in \Sigma \mid \text{certain}(\langle a_1, \dots, a_m, a \rangle, \sigma)\}|}{|\{\sigma \in \Sigma \mid \text{certain}(\langle a_1, \dots, a_m \rangle, \sigma)\}|}$$

Based thereon, the N-Gram probability for a possible resolution  $\phi = \langle e_1, \dots, e_n \rangle$  aggregates the probabilities of all its events:

$$P_{N-Gram}(\phi) = \prod_{k=2}^n P(\lambda(e_k) \mid \langle \lambda(e_{\max(1, k-N+1)}), \dots, \lambda(e_{k-1}) \rangle)$$

This approach addresses the limitation of establishing the probabilities of resolutions solely on the certain traces in the event log. The experiments in this work were conducted with an  $N$  value of 2.

### 4.3.3 Weak Order Model

Lastly the weak order model was introduced which exploits indirect order (weak order) dependencies among activity executions. Formally the predicate *order* assesses whether or not a trace  $\sigma = \langle E_1, \dots, E_n \rangle$  contains two activities  $a$  and  $b$  in such an indirect order or not:

$$\text{order}(a, b, \sigma) \iff \exists i, j : i < j, \quad e_i \in E_i \wedge e_j \in E_j \wedge \lambda(e_i) = a \wedge \lambda(e_j) = b \quad i, j \in \{1, \dots, n\}$$

*order* allows for a probability estimation for specific activity executions in weak order by considering the ratio of traces that contain the respective events:

$$P(a, b) = \frac{|\{\sigma \in \Sigma \mid \text{order}(a, b, \sigma)\}|}{|\{\sigma \in \Sigma \mid \exists e, e' \in E_\sigma : \lambda(e) = a \wedge \lambda(e') = b\}|}$$

The probability of a possible resolution  $\phi = \langle e_1, \dots, e_n \rangle$  then is determined by aggregating the probabilities of all pairs of events to occur in the particular weak order:

$$P_{WO}(\phi) = \prod_{\substack{1 \leq i < n \\ i < j \leq n}} P(\lambda(e_i), \lambda(e_j))$$

The weak order model has the highest abstraction level of all of three proposed probabilistic models and can therefore exploit information from many traces of an event log.

## 4.4 Results and Comparison with Probabilistic Models

Before presenting the results that the transformer-based approach to partial order resolution achieved compared to the probabilistic models, the performance measure will be clarified.

To determine the performance of the models on any given event log, the accuracy of predicting the exact resolution found in the ground truth logs was used. It must be noted that, to avoid bias from the hyperparameter searches and make the evaluation setting more realistic, the transformer models were chosen based on their performance on the test sets. The accuracy values are the ones the chosen models achieved on the whole set of uncertain traces.

### 4.4.1 Real-World Event Logs

The accuracy results of the experiments done on the real-world event logs are summarized in Table 4.3. It is noticeable that there are high discrepancies between the accuracy results for the different datasets, while the accuracy values of the different models are in a fairly narrow range.

Model	Traffic Fines [35]	BPI 12 [36]	BPI 14 [37]
Transformer	<b>99.41%</b>	<b>19.78%</b>	30.83%
Trace Equivalence	98.57%	18.08%	28.16%
2-Gram	97.54%	17.60%	<b>31.98 %</b>
Weak Order	98.48%	17.54%	30.01%

Table 4.3: Accuracy Results of the different partial order resolution models. The best score for each dataset is highlighted.

The low accuracy results for the BPI-12 data set can be explained with having over two uncertain event sets per uncertain trace, with some even having eight uncertain event sets. These uncertain traces with a high amount of uncertain event sets or very long uncertain event sets make it very hard for any model to get a correct prediction.

For the traffic fines dataset on the other hand the models score such high accuracy scores, first of all because the event log has a far larger number of certain traces, which makes it easier for the models to internalize the regularities encoded in the event logs, and secondly 98.6% of the uncertain traces from this dataset only have one uncertain event set with the maximum length of the uncertain event sets being three.



The BPI 14 dataset has by far the largest number of uncertain traces and has an average of 2.54 uncertain event sets per uncertain trace. The highest amount of uncertain event sets in an uncertain subtrace is 46 which is very high and makes it nearly impossible to predict the correct resolution for such a trace.

The proposed transformer-based approach could achieve better accuracy scores for the traffic fines being the only model achieving over 99% ,and the BPI 12 datasets being the only model achieving over 19%. Therefore, it outperformed the probabilistic models in two of the three real-world event logs. In the third event log, the BPI 14, the 2-Gram model achieved the highest score followed by the transformer-based approach as the second highest. For the other two event logs the 2-Gram ranked fourth and third, making the transformer the overall most reliable option.

#### 4.4.2 Synthetic Event Logs

As mentioned earlier 16 synthetic event logs were generated from four synthetic event data collections by introducing different degrees of order uncertainty. In the following the performance of the evaluated models will be illustrated for all 16 of these event logs grouped by the degree of order uncertainty in the Tables 4.4 - 4.7.

Model	SD 1	SD 2	SD 3	SD 4
Transformer	82.11%	54.63%	67.91%	78.64%
Trace Equivalence	<b>83.83 %</b>	<b>56.72 %</b>	<b>70.82%</b>	81.82%
2-Gram	78.68%	55.55%	67.98%	<b>82.31 %</b>
Weak Order	75.27%	48.22%	61.61%	73.11%

Table 4.4: Accuracy Results of the event logs with 20 % uncertain traces and 25 % events in uncertain event sets. The best score for each dataset is highlighted.

Table 4.4 shows the results for the lowest amount of order uncertainty introduced in the synthetic event logs with 20 % uncertain traces and 25 % events in uncertain event sets. It can be seen that the trace equivalence model performs best on three of the four event logs on this uncertainty setting while the weak order model performs worst on all of them. The transformer-based and 2-Gram model perform similar on SD 2 and SD 3 and are also close to the best performing trace equivalence model. While the transformer outperforms the 2-Gram on SD 1, the 2-Gram model outperforms all models on SD 4.

Model	SD 1	SD 2	SD 3	SD 4
Transformer	70.50%	34.05%	53.51%	72.24%
Trace Equivalence	<b>74.75 %</b>	<b>37.49 %</b>	<b>54.58%</b>	<b>72.51 %</b>
2-Gram	68.93%	35.35%	51.32%	72.13%
Weak Order	65.59%	28.42%	43.19%	58.18%

Table 4.5: Accuracy Results of the event logs with 50 % uncertain traces and 50 % events in uncertain event sets. The best score for each dataset is highlighted.

For the second degree of order uncertainty introduced, the uncertainty parameters are 50 % uncertain traces and 50 % events in uncertain event sets. The resulting accuracy values are depicted in Table 4.5. As for the former uncertainty setting, the trace equivalence model achieves the best results again, outperforming the other three models on all the event logs. The transformer performs the second best on the three event logs SD 1, SD 3, and SD 4 having similar results to the 2-Gram model again. The weak order model has still the worst accuracy scores over all the data sets.

Model	SD 1	SD 2	SD 3	SD 4
Transformer	<b>79.34%</b>	48.27%	<b>65.03%</b>	<b>80.29%</b>
Trace Equivalence	71.58%	29.84%	52.36%	80.20%
2-Gram	77.41%	<b>52.43%</b>	63.75%	80.20%
Weak Order	73.65%	44.94%	56.91%	70.22%

Table 4.6: Accuracy Results of the event logs with 99.5 % uncertain traces and 30 % events in uncertain event sets. The best score for each dataset is highlighted.

The remaining two orders of uncertainty applied both have a ratio of 99.5 % uncertain traces to show the edge cases of order uncertainty. They differ in their ratio of events in uncertain event sets. The first one has a value of 30 % for the number of events in uncertain event sets and the results for this parameter setting and the four respective event logs are shown in Table 4.6. In contrast to the previous two order uncertainty settings, the trace equivalence model is no longer the best performing model. It even performs the worst on three of the four datasets, the only exception being SD 4, which is the event log with only 6 different variants. The transformer model on the other hand achieves the best results for SD 1, SD 3 and SD 4, only falling short to the 2-Gram model on SD 2. The weak order model is still outperformed on all the datasets by the transformer and the 2-Gram model.

Model	SD 1	SD 2	SD 3	SD 4
Transformer	<b>66.58%</b>	24.76%	44.86%	66.78%
Trace Equivalence	59.91%	12.56%	38.88%	<b>67.19%</b>
2-Gram	64.92%	<b>28.63%</b>	<b>45.41%</b>	66.93%
Weak Order	61.63%	22.08%	36.95%	50.18%

Table 4.7: Accuracy Results of the event logs with 99.5 % uncertain traces and 60 % events in uncertain event sets. The best score for each dataset is highlighted.

The highest amount of order uncertainty applied to the synthetic data collections is shown in Table 4.6. The uncertainty parameters are 99.5 % uncertain traces and 60 % events in uncertain event sets having the highest values in both categories compared to the other three experimental settings. Compared to the previously discussed results, there is no clearly dominating model. The 2-Gram model achieves the highest accuracy for SD 2 and SD 3, the transformer model for SD 1 and the trace equivalence model for SD 4.

## 4.5 Discussion

Overall, it can be seen that for SD 4, which only has 6 trace variants, the transformer, trace equivalence and 2-Gram model have very similar accuracies over all uncertainty distributions. Also, the number of events in uncertain event sets seems to worsen the accuracies of all the models more than the ratio of uncertain traces in the event log as the numbers for the 50%/50% distribution are lower for each single model and each single dataset than the numbers of the 30%/99.5% ditribution.

The transformer model and the 2-Gram model seem to be the most consistent ones as their results are within the reach of the best performing model for all the tested configurations. Additionally, the accuracy of the predictions correlates with the number of variants that the data collection from which the event logs are generated have, for SD 1, SD 2, and SD 3. The accuracies of all models decline for these data collections with rising trace variants.

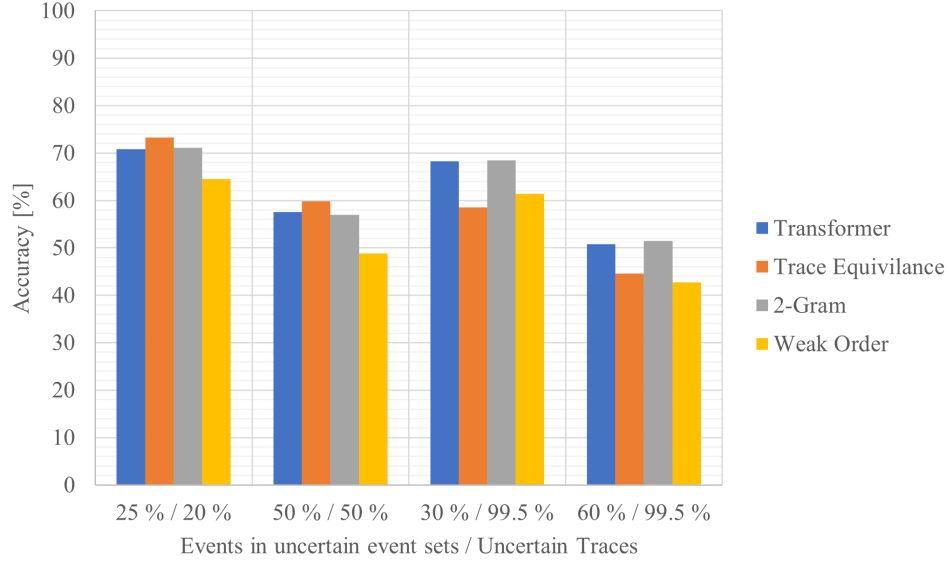


Figure 4.4: Average of the results of the examined models on the synthetic data collections grouped by order of uncertainty.

To further illustrate these tendencies of the used models and discuss whether the proposed model could achieve any significant improvement over the probabilistic models, Figures 4.4 and 4.5 were included. They show the averages of the model performances grouped by data collection and magnitude of order uncertainty. The averages were calculated using the values from Tables 4.4 - 4.7. Both of these tables suggest that the transformer model could not significantly outperform all of the probabilistic models. Nevertheless it seems to be the most consistent model on the synthetic data collections together with the 2-Gram.

A possible reason for the similar results of the 2-Gram model and the transformer could be that the 2-Gram already encodes the dependencies that can be derived from the event log very well and that there is simply not enough relevant information that can be extracted from the event logs to achieve significantly better accuracy results.

This assumption is strengthened by the results of the models on the real-world event logs in the sense that the performances of the different models were in a fairly narrow range. Without them it could be speculated that the random introduction of the uncertainty makes the partial order resolution unfounded. Their inclusion meanwhile strengthens the assumption that with increasing order uncertainty and

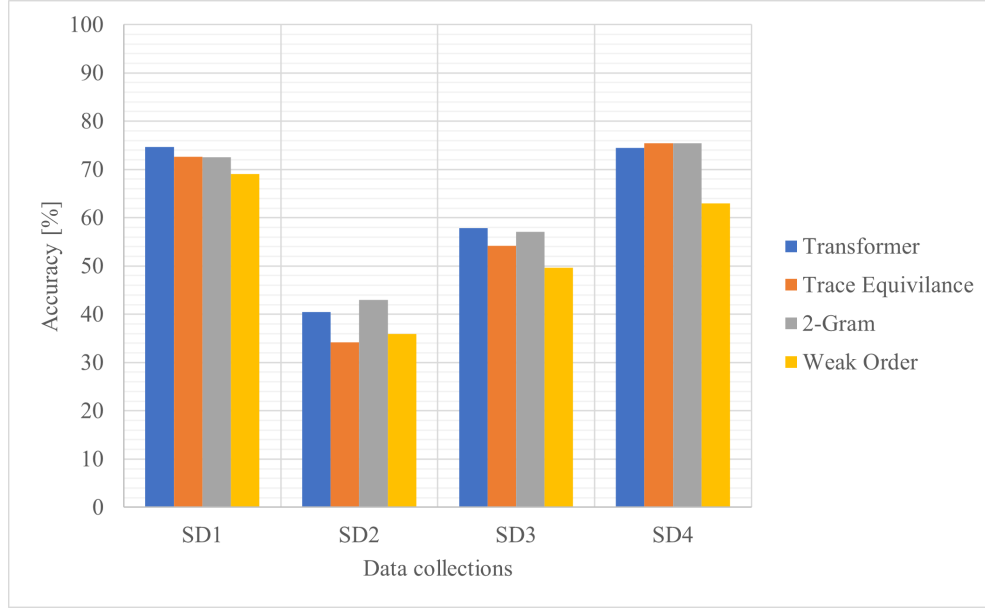


Figure 4.5: Average of the results of the examined models on the synthetic data collections grouped by the ground truth event logs.

trace length, the partial order resolution becomes significantly harder.

On the other hand the transformer did perform slightly better than the probabilistic models on the real-world event logs which could mean that the order uncertainty of them might contain some kind of pattern, in contrast to completely random uncertainty, that the transformer could internalize while the probabilistic models could not.

## Chapter 5

# Conclusion

This thesis aimed to develop a transformer-based approach to solve the task of partial order resolution and compare its accuracy against the presented probabilistic baseline models. The partial order resolution problem had to be translated to the deep learning space, meaning a method needed to be designed for the transformer model to learn the necessary attributes of an event log to perform this task.

The techniques implemented to accomplish this goal were first the ones helping to design the input and target data that the model was trained on, specifically the introduced masking and sampling method. The next step was the development of a decoding algorithm for actually performing the partial order resolution after training the model. Lastly an environment was created that allows to compare the accuracy scores of the transformer-based model to the probabilistic models developed by van der Aa et. al [3] and an experimental study on three real-world and 16 synthetic event logs was conducted.

The proposed transformer model hereby had the best accuracy scores in two of the three real-world event logs and the second highest on the third and thus seems to be the most reliable option for these data sets.

The results on the synthetic event logs were not as one-sided. The trace equivalence model here scored high on the event logs with fewer uncertain traces while the 2-Gram and transformer model had the best scores for the event logs with higher ratios of uncertain traces. Nevertheless, the transformer and 2-Gram achieved accuracy scores close to the highest ones in all of the experiments while the other two models fell far of in some cases.

Concluding it can be said that the developed transformer architecture is fit to solve the task of partial order resolution and already performs the task on the same or even slightly higher level as the existing probabilistic approaches.

## 5.1 Limitations

It must be pointed out that more thorough hyperparameter searches can potentially improve the investigated evaluation metric for the transformer model as the setup for this master thesis was limited in time and computational resources.

Additionally, the performance can be potentially improved, and a more thorough understanding of the potential capabilities of a transformer or deep learning based partial order resolution approach could be achieved by investigating some of the ideas presented in the following.

## 5.2 Future Work

Firstly, this work only focused on the partial order resolution itself which in a real-world project would be an upstream task to another process mining technique such as process discovery or conformance checking as in the work of van der Aa et. al [3]. It would be interesting to evaluate the proposed approach on such a task.

For the partial order resolution with deep learning models itself, future projects could also investigate other sequence-to-sequence architectures such as RNNs and LSTMs.

Also, the use of pretrained activity embeddings as proposed by De Koninck et. al [21], instead of randomly initializing them could improve the partial order resolution as they aim to encode the event logs characteristics.

Another way of potential improvement of the results achieved in this work, would be to combine the knowledge from the probabilistic models and the deep learning approach through a voting system.

The decoding strategy used could potentially also be improved by introducing beam-search to the decoding which has been shown to be an effective decoding strategy in machine translation [43].

# Bibliography

- [1] W. M. Van der Aalst, Process mining: data science in action, Springer, 2016.
- [2] J. Alammar, The illustrated transformer, <http://jalammar.github.io/illustrated-transformer/>, accessed: 2022-03-20.
- [3] H. Van der Aa, H. Leopold, M. Weidlich, Partial order resolution of event logs for process conformance checking, *Decision Support Systems* 136 (2020) 113347.
- [4] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, et al., Fundamentals of business process management, Vol. 1, Springer, 2013.
- [5] R. Jagadeesh Chandra Bose, R. Mans, W. M. van der Aalst, Wanna improve process mining results?: it's high time we consider data quality issues seriously, *BPM reports* 1302 (2013).
- [6] Y. Lu, Q. Chen, S. K. Poon, A deep learning approach for repairing missing activity labels in event logs for process mining, *arXiv preprint arXiv:2202.10326* (2022).
- [7] T. Nolle, S. Luetzgen, A. Seeliger, M. Mühlhäuser, Binet: Multi-perspective business process anomaly classification, *Information Systems* 103 (2022) 101458.
- [8] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., Transformers: State-of-the-art natural language processing, in: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.
- [9] Z. A. Bukhsh, A. Saeed, R. M. Dijkman, Processtransformer: Predictive business process monitoring with transformer network, *arXiv preprint arXiv:2104.00721* (2021).



- [10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv:1810.04805 [cs]ArXiv: 1810.04805 (May 2019).
- [11] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners (2020). doi:10.48550/ARXIV.2005.14165.
- [12] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, Conformance checking, Switzerland: Springer.[Google Scholar] (2018).
- [13] S. J. Leemans, D. Fahland, W. M. Van Der Aalst, Discovering block-structured process models from event logs-a constructive approach, in: International conference on applications and theory of Petri nets and concurrency, Springer, 2013, pp. 311–329.
- [14] C. Batini, M. Scannapieca, Introduction to data quality, Data Quality: Concepts, Methodologies and Techniques (2006) 1–18.
- [15] A. Adriansyah, B. F. van Dongen, W. M. van der Aalst, Conformance checking using cost-based fitness analysis, in: 2011 IEEE 15th international enterprise distributed object computing conference, IEEE, 2011, pp. 55–64.
- [16] E. Koskinen, J. Jannotti, Borderpatrol: Isolating events for black-box tracing, in: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008, Eurosys '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 191–203. doi:10.1145/1352592.1352613.
- [17] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, P. Shenoy, Probabilistic inference over rfid streams in mobile environments, in: 2009 IEEE 25th International Conference on Data Engineering, IEEE, 2009, pp. 1096–1107.
- [18] N. Busany, H. v. d. Aa, A. Senderovich, A. Gal, M. Weidlich, Interval-based queries over lossy iot event streams, ACM Transactions on Data Science 1 (4) (2020) 1–27.
- [19] X. Lu, R. S. Mans, D. Fahland, W. M. van der Aalst, Conformance checking in healthcare based on partially ordered event data, in: Proceedings of the

- 2014 IEEE Emerging Technology and Factory Automation (ETFA), 2014, pp. 1–8. doi:10.1109/ETFA.2014.7005060.
- [20] M. Pegoraro, M. S. Uysal, W. M. P. van der Aalst, Efficient time and space representation of uncertain event data, CoRR abs/2010.00334 (2020).
  - [21] P. De Koninck, S. vanden Broucke, J. De Weerd, act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes, in: International Conference on Business Process Management, Springer, 2018, pp. 305–321.
  - [22] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
  - [23] A. Neelakantan, J. Shankar, A. Passos, A. McCallum, Efficient non-parametric estimation of multiple embeddings per word in vector space, arXiv preprint arXiv:1504.06654 (2015).
  - [24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, CoRR abs/1706.03762 (2017).
  - [25] N. Parmar, A. Vaswani, J. Uszkoreit, Kaiser, N. Shazeer, A. Ku, D. Tran, Image transformer (2018). doi:10.48550/ARXIV.1802.05751. URL <https://arxiv.org/abs/1802.05751>
  - [26] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal processing magazine 29 (6) (2012) 82–97.
  - [27] D. Ciregan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, in: 2012 IEEE conference on computer vision and pattern recognition, IEEE, 2012, pp. 3642–3649.
  - [28] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, CoRR abs/1409.3215 (2014).
  - [29] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, Journal of machine learning research 12 (ARTICLE) (2011) 2493–2537.

- [30] J. L. Elman, Finding structure in time, *Cognitive science* 14 (2) (1990) 179–211.
- [31] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural computation* 9 (8) (1997) 1735–1780.
- [32] A. Sherstinsky, Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network, *Physica D: Nonlinear Phenomena* 404 (2020) 132306.
- [33] L. Liu, J. Liu, J. Han, Multi-head or single-head? an empirical comparison for transformer training, *CoRR abs/2106.09650* (2021).
- [34] N. A. Smith, J. Eisner, Contrastive estimation: Training log-linear models on unlabeled data, in: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, 2005, pp. 354–362.
- [35] M. M. de Leoni, F. Mannhardt, Road Traffic Fine Management Process (2 2015). doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.
- [36] B. van Dongen, BPI Challenge 2012 (4 2012). doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
- [37] B. van Dongen, BPI Challenge 2014: Activity log for incidents (4 2014). doi:10.4121/uuid:86977bac-f874-49cf-8337-80f26bf5d2ef.
- [38] T. Jouck, B. Depaire, Generating artificial data for empirical analysis of control-flow discovery algorithms, *Business & Information Systems Engineering* 61 (6) (2019) 695–712.
- [39] D. Ruffinelli, S. Broscheit, R. Gemulla, You can teach an old dog new tricks! on training knowledge graph embeddings, in: *International Conference on Learning Representations*, 2019.
- [40] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014). doi:10.48550/ARXIV.1412.6980.
- [41] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, *CoRR abs/1512.00567* (2015).

- [42] R. Müller, S. Kornblith, G. E. Hinton, When does label smoothing help?, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Vol. 32, Curran Associates, Inc., 2019.
- [43] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).

The source code, a documentation, some usage examples, and additional test results as well as the hyperparameter settings of all of the conducted experiments are available at <https://github.com/dhaeff/Transformer-Based-Partial-Order-Resolution.git>

## **Ehrenwörtliche Erklärung**

Ich versichere, dass ich die beiliegende Masterarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 12.04.2022

Unterschrift

A handwritten signature in black ink, consisting of a large, stylized 'D' followed by a series of overlapping, cursive loops and strokes.