UHASSELT

KNOWLEDGE IN ACTION

Faculteit Bedrijfseconomische Wetenschappen

master handelsingenieur in de beleidsinformatica

Masterthesis

Transferring Learned Representations for Process Outcome Prediction with Transformer-Based Models

Dmitri Beloshitskiy

Scriptie ingediend tot het behalen van de graad van master handelsingenieur in de beleidsinformatica

PROMOTOR :

dr. Gert JANSSENSWILLEN

2021

2022

UHASSELT

KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

UHASSELT

KNOWLEDGE IN ACTION

Faculteit Bedrijfseconomische Wetenschappen

master handelsingenieur in de beleidsinformatica

Masterthesis

Transferring Learned Representations for Process Outcome Prediction with Transformer-Based Models

Dmitri Beloshitskiy

Scriptie ingediend tot het behalen van de graad van master handelsingenieur in de beleidsinformatica

PROMOTOR :

dr. Gert JANSSENSWILLEN

Transferring Learned Representations for Process Outcome Prediction with Transformer-based Models

Dmitri Beloshitskiy, *Faculty of Business Economics, Hasselt University.*

Abstract—With digitalisation, the amount of business process execution data is ever increasing. Wielding this data in order to improve business processes could be beneficial for the performance of businesses. In similar vein, the field of predictive business process monitoring aims to improve the execution of processes by analyzing event logs and foreseeing future events and process-outcomes. Furthermore, significant progress has been made on deep learning techniques over the recent years. Pre-trained transformer models have been proven to be effective in dealing with sequential data in natural language processing. The same architecture and pre-training techniques are now being used in other research fields, such as recommender systems. This paper aims to study the relevance of these techniques in predictive business process monitoring by applying the transformer architecture to the process-outcome prediction task. To the best of our knowledge, transformers are not yet applied to process-outcome prediction. In addition, the significance of transfer learning is examined by reattempting the prediction task on a model that was pre-trained on the next-activity prediction task. This way, we will examine whether learned representations from one task can be easily transferred onto another task. Results show that transformers perform competitively against established models such as LSTMs and CNNs, even showing slight advantages in terms of accuracy and speed, while transfer learning yields no hypothesized benefit.

Index Terms—Predictive Business Process Monitoring, Process Outcome Prediction, Transfer Learning, Transformers

1 Introduction

A clear trend towards digital transformation dominates the current business landscape [1]. With digitalisation, the amount of business process execution data is increasing at a rapid rate [1]. Businesses could benefit from the increasing amount of execution data by wielding it to monitor and predict process behaviour [2]. Predictive business process monitoring (PBPM) originates from process mining, which is a field aiming to discover, control, and improve business processes by analyzing process logs [3].

Meanwhile, significant progress has been made on deep learning techniques during the past years. Much of that progress happened in the fields of natural language processing (NLP) and computer vision (CV) [4], [5]. NLP problems are characterized by the sequential nature of language. Pre-trained transformer models proved to be very effective in dealing with sequential data [6], [7]. In fact, transformer model architectures and pre-training techniques derived from the field of NLP are considered to be the driving force of the recent advancement in recommendation systems [8].

The purpose of this paper is to further study whether transformers and transfer learning are relevant in the field of PBPM. Some work already exists on the use of transformers in predicting process behaviour. ProcessTransformer [1] for example, is an open source library that enables a few PBPM tasks such as next activity prediction, event time prediction and remaining time prediction. In its concluding remarks, it is mentioned that future work could focus on other tasks of interest such as similar trace retrieval, activity recommendation and process-outcome prediction. Furthermore, little emphasis has been put on the role of transfer learning in PBPM. It is unclear how learned representations from one task could be used in order to fine tune for another task. Transfer learning through pre-training a model and fine tuning it on specific tasks is common practice for language models in NLP tasks. It often yields advantages such a more efficient training process, requiring less time, less computational resources and less labeled data [9].

The contribution of this text is twofold. In the first place, the text aims to apply the transformer architecture to another PBPM task: process-outcome prediction. Transformers are, to the best of our knowledge, not yet applied to process-outcome prediction. The performance is compared to the performance of other deep learning techniques such as recurrent neural networks (RNN) and convolutional neural networks (CNN). Secondly, the experiment is repeated on a pre-trained model that is only fine tuned on the process-outcome prediction task. With this, the aim is to get insight whether learned representations can be transferred from one task to another and whether this is beneficial to do.

The remainder of the text is structured as follows. In the next section, the text elaborates on related work regarding the use of transformers in PBPM and regarding the process-outcome prediction task in particular. Afterwards, representation learning and sequence modeling will be discussed in Section 3. Section 3 also covers the memory issue of learning representations from long input sequences. RNNs, LSTMs and transformers are discussed as possible solutions to this problem. Finally, the topic of transfer learning will close the section. Section 4 clarifies the datasets used in the experiments, as well as the deep learning models that were utilized in achieving the results. The findings of the experiment are discussed in Section 5. Finally, Section 6 closes the text with a conclusion along with implications for future research work.

2 Related Work

Transformers got popular due to their successful application in the field of NLP, which is mainly concerned with dealing with unstructured language data. Yet a considerable amount of present-day research focuses on the use of transformers for structured sequential data. In the next subsection, we highlight some of that work that is related to PBPM. Afterwards, we take a closer look at research concerning the process-outcome prediction task in particular. Finally, we mention some work in related fields that may be worthwhile to pay attention to as research progresses. Table 1 contains an overview of related research work and highlights which deep learning models and process prediction tasks are discussed.

TABLE 1
Overview of related research work concerning the use of deep learning techniques in process prediction tasks. This list is not exhaustive and contains only the papers addressed in this text.

Paper	Transf.	LSTM	CNN	Outcome	Other
Ketyko [10]	x	x	-	-	x
Wickra. [11]	-	x	-	-	x
Teinemaa [12]	-	-	-	x	-
Bukhsh [1]	x	-	-	-	x
Weytjens [2]	-	x	x	x	-
This paper	x	x	x	x	-

2.1 Predictive Business Process Monitoring

As mentioned previously, Bukhsh et al. [1] developed the ProcessTransformer library that enables easy application of transformers on event logs to predict future characteristics of a running process. In their work, they managed to perform competitively on two PBPM tasks: event time prediction and remaining time prediction. Even more so, they outperformed current baselines on the next activity prediction task. The authors only used the minimum amount of attributes (i.e. case id, activity and timestamp) as input for their model and did not consider any additional case or event attributes. With this, they emphasise that ProcessTransformer is able to bring optimal performance even if additional attributes are not available. In their concluding remarks, they underscore that research could scope out by including other PBPM tasks such as similar trace retrieval, activity recommendation and process-outcome prediction.

Ketyko et al. [10] apply the transformer architecture to yet another PBPM task: suffix prediction. In their work, they compare the performance of seven different deep learning architectures, of which four have never been applied to suffix prediction. During evaluation, they compared the performance of each architecture on different prefix lengths only to conclude that none of the architectures is a one-size-fits-all type of solution. They argue that average measures fail to capture these type of differences and that more research is needed to correctly assess the suitability of deep learning architectures for a given task. Furthermore, the authors discuss how event log data differs from natural language sentences or images, which are more often studied in deep learning research. That is to say that event log traces are multidimensional sequences, unlike language or visual sequences, containing both label and time instead of continuous corpus. These kind of distinctions may require different processing. Currently, little attention is paid on differences between event logs and how they are processed.

Deep learning models are often critiqued to be ”black boxes”. It other words, it is difficult to understand why a certain prediction has been made. Wickramanayake et al. [11] attempt to tackle this problem by creating interpretable LSTM models for the next activity prediction task. They introduce two types of attentions, event attention and attribute attention, that should respectively show which event(s) or which attribute(s) contributed to the prediction of the next activity label. They finally evaluated the generated explanations by consulting with domain experts.

Rama-Maneiro et al. [13] published a literature review discussing a wide array of deep learning techniques for PBPM. These deep learning techniques include different kinds of transformers as well as RNNs. The authors provide additional evidence to support the fact that transformers are yet to be applied to certain PBPM tasks such as process-outcome prediction.

2.2 Process-outcome prediction

On the topic of process-outcome prediction in particular, Teinemaa et al. [12] studied the relevant research up to 2019. It becomes clear that only traditional machine learning techniques were considered and no deep learning techniques were present. These traditional techniques include decision trees, support vector machines, random forests and gradient boosting machines. They emphasize the fact that process outcome prediction research differs in terms of input data, learning technique, bucketing technique and the type of sequence encoding.

Kratsch et al. [14] verified that deep learning is generally superior to traditional machine learning in predicting process-outcomes. While the performance gain is modest for well-balanced and highly standard event logs, they identified three types of event logs in which deep neural networks significantly outperform traditional machine learning techniques: event logs with (1) many non-standard cases containing different prefixes, (2) imbalanced outcome labels and (3) many event attributes (relative to the amount of case attributes).

Weytjens et al. [2] compared different kinds of deep learning techniques for the process-outcome prediction task. In their research, they specifically compared RNNs to CNNs. The RNNs made use of long short-term memory (LSTM) gates with and without attention mechanisms. They concluded that despite having similar results, CNNs were the preferred model of choice due to the fact that they required less computation time than RNNs. The authors also noted that the attention mechanisms did not significantly impact the performance of the RNNs.

2.3 Recommendation systems and model size

Other related work stems mainly from the field of recommendation systems. Session-based recommender systems (SBRS) have recently emerged as a new paradigm within the

field [15]. Different from collaborative filtering and content based recommendation, SBRS aims to capture short-term but dynamic user preferences. As a result, SBRS is able to provide more timely and context-sensitive recommendations [15]. SBRS share similarities with PBPM. For example, both process multimodal, sequential and tabular input data. As such, it might be beneficial to keep track of research in the field.

Drawing its inspiration from NLP, BERT4Rec [16] aims to use the popular language model BERT as a sequential recommendation model for tabular data. In a similar vein, De Souza et al. [8] provided the Transformer4Rec library in order to make the use of transformers more accessible for recommendation problems involving tabular data. The library builds directly on top of HuggingFace transformers, which is a popular open source library containing many state-of-the-art pre-trained transformer models for NLP [5].

Despite the constant emphasis on the transformer architecture, breakthroughs in NLP may actually be attributed to the training of enormous language models [17], [18]. After pre-training on large amount of data, language models are usually finetuned on a specific task [7]. Little research has been put on the potential of transfer learning in PBPM.

3 Fundamentals

In the following subsections, we discuss representation learning and sequence modeling. Traditional feed-forward neural networks struggle to learn representations of sequential data due to a lack of memory mechanisms. We show how RNNs and LSTMs attempt to solve this issue. The transformer architecture provides another solution to the memory problem using attention mechanisms. Finally we discuss the concept of transfer learning.

3.1 Representation learning

Representation or feature learning is the act of extracting high level features from raw data. The most common example is that of computer vision, in which an object recognition model first learns distinct lines and shapes of an object (e.g. the nose or tail of a dog), in order to finally create an internal representation of that object. Representation learning can also be demonstrated in NLP. For instance, language models create internal representations of words called word embeddings. The learned word embeddings allow to model relations and similarities between different words as illustrated in Figure 1. There we can clearly see that words like *cat* and *kitten* are closely related, but less related to *dog* and even less to *houses*. Going even further, the representation of the word *queen* can be mathematically described using arithmetic with other representations:

$$\text{“}queen\text{”} = \text{“}king\text{”} - \text{“}man\text{”} + \text{“}woman\text{”}$$

The same process of representation learning happens with other kinds of data such as event log data. In order to enable representation learning, input data must first be transformed in numbers that can be fed to a deep learning model. The process of encoding raw sequential data into sequences of numbers is called sequence encoding or sequence modeling.

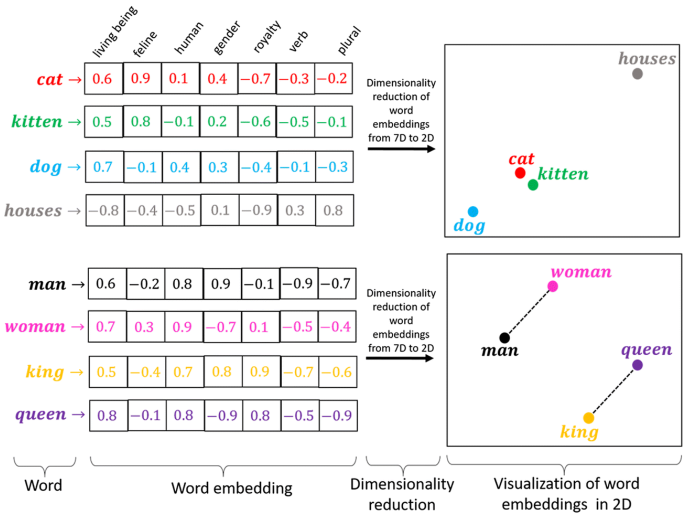

Fig. 1. Representation learning in natural language processing [19].

For sequential data such as language and event logs, the data needs to modeled into a sequence. Elements in a sequence are dependent of each other and must adhere to their order in the sequence. Traditional feed forward neural networks are unable to deal with the sequential nature of data and struggle to keep memory of items that happened early on in the sequence. This led to the creation of recurrent neural networks (RNNs) [20].

3.2 RNNs and LSTMs

RNNs are different from traditional feed forward neural networks, in the sense that all neuron inputs and outputs are interconnected. This design allows for recursion which is necessary to process sequential data [20]. A network is said to have a memory if it is able to look back at previously encountered input tokens [21]. Traditional feed forward neural networks fail to have a memory because there is no remembering mechanism. Each token simply gets forgotten after each traversal through the network. RNNs instead maintain a hidden state in which information of previous tokens is retained [22]. For example, when token n passes through the network, the model merges the state representing all tokens up to token $n-1$ with the new information of token n, to create a new state. Theoretically, the information from one token can propagate arbitrarily far as newer tokens are traversing the network. In practice, however, RNNs fail to achieve this ability [23]. The vanishing/exploding gradient problem is one important reason for the lack of long-term memory in RNNs [24].

In order to still achieve the desired property of having a long-term memory, the long short-term memory (LSTM) network was adopted [25]. A LSTM network is a particular type of RNN, where the neurons are not only interconnected, but also designed in such a way as to create a mechanism specifically for retaining or forgetting certain information [25]. The mechanism consists of a cell state through which information can flow. Information inside the cell state can either be removed or added by the LSTM through the use of gate structures [25].

Note that the cell state is additional to the hidden state that RNNs and LSTMs normally possess. In fact, it might be useful to consider the hidden state as working memory, while viewing the cell state as long-term memory. With the cell state, it is now possible for the network to remember relevant information from the past, no matter how long ago the input tokens were processed. At the same time, it will not pay any exceptional attention to information that might have been encountered more recently, but was in fact less relevant [26].

3.3 Transformers

The transformer consists of an encoder-decoder structure. When first introduced, it was common practice to employ the encoder-decoder structure in various deep learning problems [4]. Figure 2 illustrates a simplified version of the transformer architecture with a stack of encoder blocks on the left and a stack of decoder blocks on the right.

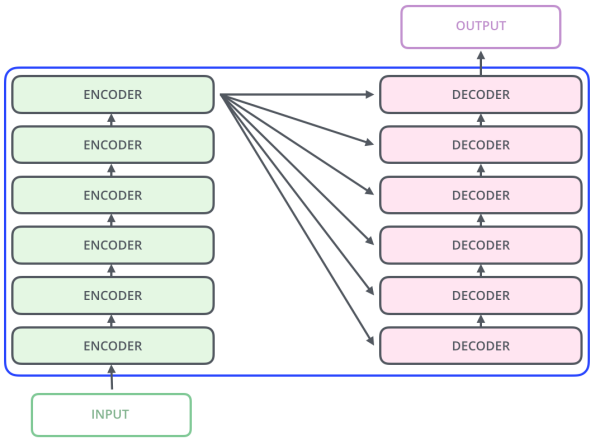

Fig. 2. A simplified illustration of the transformer architecture, highlighting the encoder-decoder structure [27]. Notice how each stack contains six blocks, as originally proposed by Vaswani et al. [4]. In our case, the input will be a sequence of events.

The encoder and decoder stacks serve different purposes. The purpose of the encoder is to take input sequences and to learn high level representations from those input sequences. In turn, these learned representations can be used for tasks such as classification. When classification is the end goal, decoders are not necessarily required. However, when the task is to generate an output that is of similar form as the input sequence (e.g. you input a sentence and you expect a sentence as output), it is imperative to use decoders. It is for that reason that encoder-decoder models are also referred to as sequence-to-sequence models.

Before the introduction of the transformer, sequence-to-sequence models often used LSTM cells inside the encoder and decoder blocks [28]. With transformers, it turned out that LSTM cells were not needed. Instead, their function got replaced by attention mechanisms. This is significant, as the recursive nature of the LSTM cell allowed for the processing of sequential data. Transformer models are still able to process sequential data despite not having a recursive design [4]. Even more so, the hidden states and cell states of the LSTM were essential to produce a context vector and to use the attention mechanism. Instead, the transformer makes use of self-attention [4].

Self-attention was first proposed as an addition to the sequence to sequence model [29], [30]. With the creation of transformers, it turned out that the self-attention alone was enough to complete NLP related tasks [4]. Instead of the encoder layer being a LSTM cell, it now consists of two main layers: a self-attention layer and a feed forward layer. The layout of a single encoder block is illustrated in Figure 3. The decoder block only differs by having an additional attention layer that works in a similar fashion as the attention mechanism in the sequence to sequence model [4].

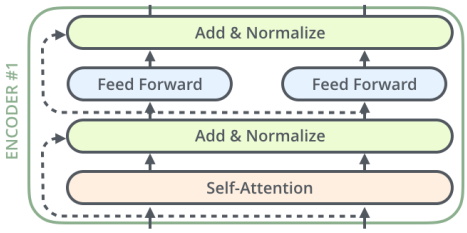

Fig. 3. The layout of a single encoder block [27]. It consists of two primary parts: the self-attention layer and a feed forward layer. Supplementary, each of these parts has a residual "addition" and "normalization" operation. These operations are alterations to solve the vanishing gradient problem [31] and to improve training [32].

Before entering the self-attention layer of the encoder, each input token is modified with its positional encoding (illustrated in Figure 4) [33]. This is done to remember each tokens position in the original input sequence. In the self-attention layer, a unique attention score is calculated for each input token [4]. The attention score provides additional information to the input token, by relating it to other tokens in the input sequence. The purpose of this is to eventually provide a better encoding for the input token [4]. (To illustrate this, suppose we have a NLP related problem in which the word "sleep", i.e. an input token, is passing through the self-attention layer. The encoding of the word "sleep" will be quite different if it is preceded by the word "no".) Remember how a RNN was able to incorporate the representations of previously processed tokens with the tokens that are currently being processed by maintaining a hidden state. With transformers, this mechanism is replaced by self-attention.

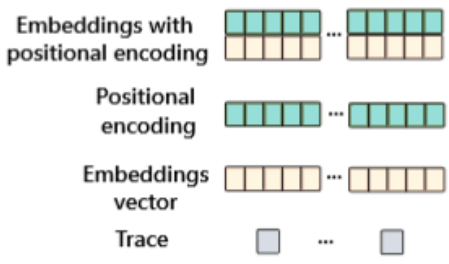

Fig. 4. Each trace (sequence of events) will be embedded into a vector before entering the encoder. Additionally, the order of events gets encoded into a second vector. Both vectors will then be concatenated [1].

A big advantage of transformers over RNNs such as LSTM, is the ability to process data in parallel [4]. This parallelization is possible because transformers do not need to process data in order. The knowledge of the order of the input tokens is replaced by the attention mechanism that learns the context that confers meaning to each word in the sentence. The fact that parallelization is possible, allows the model to process larger sequences of input tokens while memory constraints would usually limit the ability to batch across examples [4]. Parallelization also reduces the time needed to train such a language model. In turn, this allows for training on larger datasets which improves performance. These developments lead to the creation of current state-of-the-art pre-trained systems such as Google's BERT model (Bidirectional Encoder Representations from Transformers) [6] and OpenAI's GPT-3 model (Generative Pre-trained Transformer) [7]. Both BERT and GPT-3 are, as their full name suggests, variations of the transformer architecture.

3.4 Transfer Learning

Transfer learning is about applying knowledge and skills learned in previous tasks to new tasks [34]. The task from which knowledge originates, is called the source task. The new task that takes advantage of the learned knowledge is referred to as the target task. In practice, large pre-trained models are developed. These pre-trained models are proficient in solving a certain source task. Afterwards, the pre-trained models are fine-tuned by being exposed to examples of a target task.

A necessary requirement to transfer knowledge between tasks, is that the source and target domain share a common feature space [34]. For example, in NLP tasks, a large part of the model is dedicated to learning word embeddings. These word embeddings are used in top layers to perform different language related tasks. While the tasks may be different, the underlying word embeddings do not differ significantly between tasks. Non-language models do not have word embeddings, but they do extract other high level representations from raw data. These high-level representations or features could be common across different tasks. In transfer learning, the learned representations from the source task are derived as a starting point to solve a target task.

While there are various reasons as to why this approach is desirable, the main reason is that it allows to train new models faster [9]. The pre-trained model would have spent a considerable amount of time on learning representations from the source task. Since the source model already spent the effort to learn representations, it will require less time to train new models as the new model would not have to start learning from scratch. This is significant as training deep learning models can be costly in terms of computational resources [9]. To illustrate this with an example, it has cost up to twenty hours for some of the models to be trained in this paper due to hardware constraints. The hardware and experimental setup are described further in Section 4.

Besides enabling a faster learning process, transfer learning requires less labelled data for training on the target task [9]. The reason for this lies again in the fact that much of the representation learning already happened during pre-training. Requiring less labeled may be advantageous when the labeling of cases is not straightforward and costly (e.g. when the exact same prefixes have different outcomes, which may even require manual labeling). Furthermore, transfer learning helps prevent the issue of overfitting. Overfitting is prevented by freezing layers during the finetuning of a pre-trained model. Essentially, frozen layers have a zero learning rate which prevents model weights from being updated during finetuning.

In order to transfer representations from a source model, it is necessary to change the final layer(s) of the model because the output of the source model and the target model should be different [35]. In the example of this paper, the source model involves predicting the next-activity while the target model predicts the outcome of a process. Besides changing the final classification layers, the weights of the newly created model need to be updated by processing examples of the target task. This process of changing weights according to examples is called fine-tuning [35].

In updating the weights of the new model, the first layers of the pre-trained model are usually frozen. In other words, the weights of these first layers cannot be changed while the other weights get optimized for the examples. There is a motive for freezing some of the weights. The frozen layers generally serve as generic feature extractors that derive meaningful features that are common between source task and target model. The weights of the frozen layers are the direct evidence of how knowledge learned from a source task is transferred onto a target task.

In designing a deep learning model with transferred knowledge, the designer would have to find the optimal balance between freezing and fine-tuning by adjusting the learning rates. Freezing the starting layers from a trained source model is important to avoid that the target model relearns basic features. If the starting layers are not frozen, the learning of the source task will be lost and no knowledge would be transferred. Freezing more layers could be beneficial to avoid overfitting the model on the target task labels. This is especially important if the target task labels are scarce and many possible examples are unseen. If the target task labels are abundantly available, it could be beneficial to fine-tune more of the weights.

4 Methodology

In the following subsections, we will describe the datasets that were used in conducting the experiments. Likewise, we will highlight the necessary data preprocessing (i.e. the labeling of cases with process-outcomes) for each dataset. Following that, we will discuss the experimental setup and we clarify the models that were used to obtain the results and how they contribute to our research questions.

4.1 Preliminaries

Before introducing the datasets and the experimental setup, a number of key concepts will be defined first. These are commonly applied in the field of process mining and will be referred to in the remainder of the text.

Definition 1 (**Event**). Let A be the set of activities, C the set of cases, T the time domain and $D_1, .., D_m$ the set of related attributes where $m > 0$. An event is a tuple $e = (a, c, t, d_1, ..., d_m)$, where $a \in A$, $c \in C$, $t \in T$ and $d_i \in \{D_i\}$ with $i \in [1, m]$ [1].

Definition 2 (**Mapping Functions**). Let π_A, π_C , and π_T be functions that map an event $e = (a, c, t, d_1, ..., d_m)$ to an activity, to a unique case id, or to a timestamp. For example: $\pi_A(e) = a$, $\pi_C(e) = c$ or $\pi_T(e) = t$ [1].

Definition 3 (**Trace, Prefix**). A trace is defined as a finite non-empty sequence of events $\sigma = \langle e_1, ..., e_n \rangle$, such that $\forall e_i, e_j \in \sigma$, it must hold that: the events within a trace σ must have same case id, i.e. $\pi_C(e_i) = \pi_C(e_j)$ and time should be non-decreasing, i.e. $\pi_T(e_j) \geq \pi_T(e_i)$ for $j > i$. We say that a trace $\sigma = \langle e_1, ..., e_n \rangle$ has length n, denoted by $|\sigma|$. Given a trace $\sigma = \langle e_1, ..., e_n \rangle$ and a positive integer $k \leq n$, the prefix of a trace is the sequence of events up to the kth event, such that $prefix(\sigma, k) = \langle e_1, ..., e_k \rangle$ [1].

Definition 4 (**Event Log**). An event log is collection of traces $L = \{\sigma_1, \sigma_2, ..., \sigma_l\}$. We say that an event log $L = \{\sigma_1, \sigma_2, ..., \sigma_l\}$ has size l, denoted $|L|$ [1].

The process-outcome prediction task aims at predicting the class label (the process-outcome) of a given trace [12]. In order to train such a model, we need a set of completed cases with their known class labels. In other words, we need a labeling function that labels completed cases to their outcomes. The outcomes of a process can be stored differently across event logs. On some occasions, outcomes are simply defined by the last event in a completed case. On other occasions, the derivation of an outcome is more complex and custom rules need to be defined to label a case.

Definition 5 (**Labeling Function**). A labeling function λ aims to label a given trace σ to its class label (its process outcome) O, such that $\lambda(\sigma) \in O$. For process-outcome predictions, O is a finite set of categorical outcomes. For example, for a binary outcome $O = \{0, 1\}$.

Definition 6 (**Outcome Prediction**). Process-outcome prediction is the definition of a function ϕ that takes a prefix $prefix(\sigma, k)$ where $k \in [1, n-1]$, and predicts the most likely outcome O of that prefix, i.e.:

$$\phi(prefix(\sigma, k)) = O$$

4.2 Datasets

The experiments are based on four real-life event logs. Table 2 provides an overview of the event logs and their descriptive statistics. The events logs are publicly available at the 4TU research data repository [1]. For all event logs, different labeling functions λ were needed to create the class labels. In the following paragraphs, we describe each of these event logs and their labeling methods in more detail.

BPIC2012 [36]. The dataset was originally published with regard to the Business Process Intelligence Challenge (BPIC) in 2012. It contains the execution history of a loan application process for a Dutch financial institution. Each case in this event log records the events related to a singular loan application. We define a labeling function λ for the purpose of predicting the process-outcome. In this particular dataset the labeling function λ is simply taking the last event of a completed case. This results in three possible process-outcomes: application accepted, rejected or cancelled.

1. Data repository available at https://data.4tu.nl/

TABLE 2
Datasets used in the experiments of this text.
Descriptive statistics are given for each dataset.

Datasets	Cases	Events	Unique	Min.	Max.
BPIC2012 [36]	13 087	262 200	24	3	175
BPIC2017 [37]	31 509	1 202 267	26	10	180
Traffic [38]	150 370	561 470	11	2	20

BPIC2017 [37]. This dataset shares similarities with BPIC2012. Particularly, it stems from the same financial institution and it is still concerned with the loan application process. However, the procedure for data collection has improved which resulted in a cleaner and richer event log. The data itself is more recent as is indicated by the year of the dataset. The labeling function λ still takes the last event of a completed case. The resulting process-outcome labels (accepted, rejected and cancelled) also remain unchanged.

Traffic [38]. The dataset is an event log of an information system managing road traffic fines. The log comes from an Italian police station. The dataset contains fine notification events as well as repayments. Additional information includes the total fine amount and the amount of repayments for each fine. The labeling is based on [12], where the labeling function λ assigns the two possible process-outcomes: repaid in full or sent for credit collection.

In Table 3 we present the datasets and their corresponding outcome labels. Note that for the *Hospital* dataset, we have in fact four different process-outcome labels.

TABLE 3
Datasets and their corresponding process-outcome labels

Datasets	Process-outcome labels
BPIC2012	accepted, cancelled, declined
BPIC2017	accepted, cancelled, declined
Traffic	repaid in full, credit collection

4.3 Experimental setup

The experiments are based on the ProcessTransformer library [1]. The ProcessTransformer library enables easy application of transformers on event logs to predict future characteristics of a running process. ProcessTransformer relies on Google's TensorFlow library [39]. TensorFlow is an open source library created to develop machine learning models. In the next subsections, additional information on the experimental setup is provided.

4.3.1 Data preprocessing

The original datasets do not contain any outcome-labels. As mentioned above, labeling functions were defined to first append an outcome to each observation.

4.3.2 Training hardware

Training was performed using consumer grade hardware. In particular, a single NVIDIA RTX3060TI GPU was used. While Tensorflow is compatible with CUDA enabled GPUs such as the RTX3060TI, training remains a lengthy process. The quickest model was able to perform an epoch in five minutes, while the slowest model had a total training time of almost thirty hours. Total training times are provided for each model in Section 4.5

4.3.3 Data split

For evaluation purposes, each dataset was split with a 20% test set being reserved to calculate evaluation metrics after training. Of the remaining data, another 20% was used as a validation set during training. The validation set was used to calculate the loss and accuracy measures after every epoch. These measures were used to implement an early stopping mechanism. More precisely, if the loss of the validation set did not improve in five epochs, the training was stopped.

4.3.4 Overfitting

The early stopping mechanism prevents overfitting by stopping training as soon as the validation set seizes to improve for five epochs. Besides the early stopping mechanism, standard regularization techniques were employed to reduce overfitting. Regularization techniques include the use of normalization and the use of dropout layers to set arbitrary input units to zero while training. Additionally, transfer learning reduces the amount of overfitting due to the freezing of early layers.

4.3.5 Hyperparameters

In conducting the experiments, no effort was spent on optimizing the hyperparameters. Instead, the hyperparameters were based on values used in similar models in the literature. While it is not practical to discuss all available hyperparameters, some of them are listed in Table 4. The number of epochs is determined by the early stopping mechanism as discussed above. The only exception to this is the second experiment with transfer learning, where each model was trained for ten epochs. For more details on hyperparameters, please refer to the source code.

4.3.6 Reproducibility

For reproducibility, the source code is made available on Github at http://github.com/dmitri-bel/thesis. Additionally, training information on each model can be viewed at https://tensorboard.dev/experiments/[EXPERIMENT ID]. Experiment IDs and hyperlinks are listed in Table 8.

4.4 Experiments

In the next sections, two experiments are discussed. The first experiment involves applying the transformer architecture to process-outcome prediction. The second experiment studies whether learned representations can be transferred between different process prediction tasks.

TABLE 4
Overview of used hyperparameters.

Hyperparameter	Value
Loss function	Sparse categorical cross entropy
Optimizer	Adam
Learning rate	0.001
Dropout rate	0.1
Batch size	12

TABLE 5
The amount of trainable parameters for each model. Notice how the CNN model contains significantly more trainable parameters.

Model	Amount of trainable parameters
Transformer	29706
LSTM	31298
CNN	93080

4.4.1 Experiment 1: Process outcome prediction

The first goal is to apply the transformer architecture to process-outcome prediction. In order to do this, the ProcessTransformer library was expanded in three steps. In the first step, a new data processor has been added to prepare the datasets. The processor outputs a table in which the trace of each case is linked to the related process-outcome label. A labeling function λ assigns the process-outcome label to each case as described in Section 4.2. Furthermore, for each trace, all possible prefixes are added as additional examples. The dataset is then split into a train set and a test set. In the second step, the data loader takes the train and test set, collects the necessary metadata and transformers the labeled examples into padded sequences of numbers that can serve as input for the transformer model. In the third step, the actual transformer model is created using TensorFlow with the Keras [35] interface.

Figures 5 and 6 display a high level view on the architectures of the transformer, LSTM and CNN models. The components in the figures represent the layers as available in Keras. Despite their visual size difference, the transformer model and the LSTM model actually share a number of similarities. First of all, both models start with an embedding layer, highlighted in darker gray). The embedding layer, as described in Figure 4, embeds each trace into a vector and adds a positional encoding using a second vector. Secondly, both the transformer and LSTM model enter their characteristic building block. For the transformer, it is the encoder block with self-attention as described in figure 3. For the LSTM model, the characteristic building block is a LSTM layer as defined in Keras. Thirdly, both models close off with two fully connected ("dense"), feed forward neural network layers with dropout.

The architecture of the CNN model is based on the top performing model of a Kaggle[2] competition for tabular data. In studying the CNN model, notice how there is no embedding layer compared to the previous two models. The CNN model does not require the same positional embedding and thus does not have the same embedding layer. The

2. CNN model inspired by Kaggle submission

CNN model also stands out due to the amount of trainable parameters as presented in Table 5.

4.4.2 *Experiment 2: Transferring learned representations*

In this experiment, we create another transformer model to predict process-outcomes. This new model is based on the next-activity prediction task of the ProcessTransformer. After training for the next-activity task, the resulting weights are saved and the output layers of the model are altered to fit the process-outcome prediction task. The model is then retrained on process-outcome examples. The architecture remains unchanged and is identical to the transformer architecture in Figure 5.

4.5 Results

The results of the first experiment are displayed in Table 6. In this experiment, three different outcome-prediction models were compared for each dataset. Note that the BPIC2012 and BPIC2017 dataset are included multiple times for each possible outcome label. The three models include the transformer, LSTM and CNN architectures as discussed above. For each dataset, the highest accuracy score is marked in bold, highlighting the top performing model. The results show, however, that the measures differ only slightly between the different models. This can be especially seen in the traffic dataset, where all three models achieve the same scores after rounding for two digits.

The results of the second experiment are displayed in Table 7. The goal of the second experiment was to study the significance of transfer learning in process prediction tasks. Hypothesizing a quicker training process, we ran two process outcome prediction models for ten epochs each. The first model ("pre-trained") loaded weights from a previous training process for a the next-activity prediction task. The second model served as a benchmark to compare to the performance of the pre-trained model. Both models have been exposed to ten epochs of training on the same training data. In examining the results of Table 7, we notice that the scores of the pre-trained model are a few percentage points below the scores of the benchmark models.

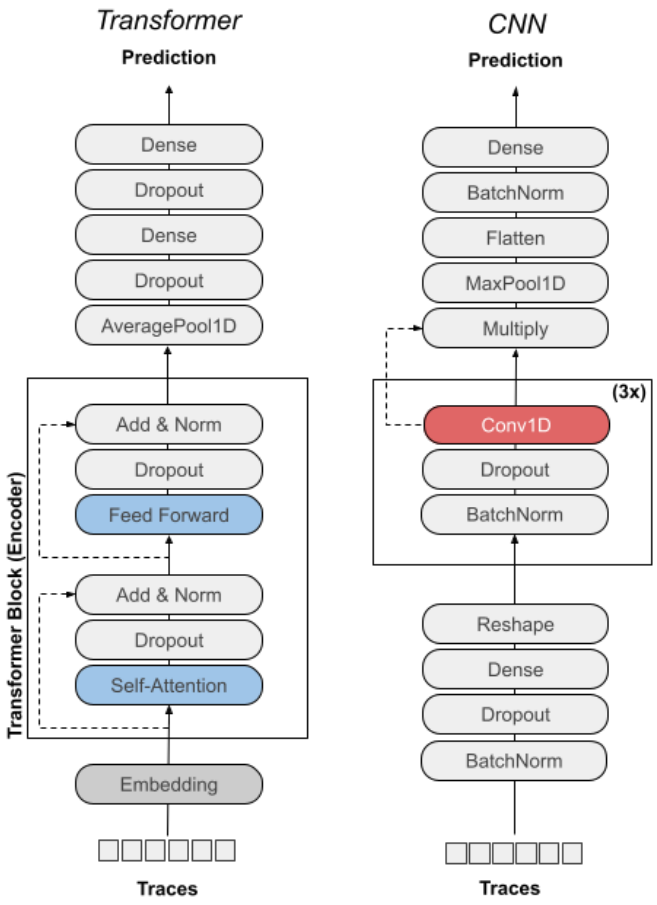

Fig. 5. Architectures used for the transformer model (left) and CNN model (right). Compared to the transformer model and LSTM model, the CNN model does not use an embedding layer (marked in darker gray).

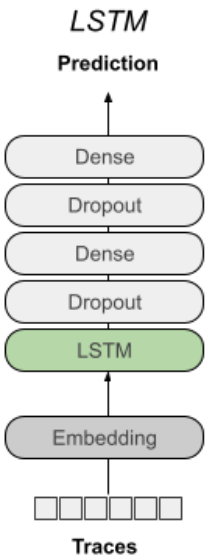

Fig. 6. Architecture used for the LSTM model. Despite the visual difference in size, the LSTM architecture is in fact very similar to the transformer architecture in figure 5

5 Discussion

Process outcome prediction

The results of the first experiment showed that the transformer model is indeed capable of completing the process outcome prediction task. While in some cases the transformer model seems to have a slight edge on the LSTM and CNN models, the differences are small. It appears that different deep learning models perform similarly compared to each other, based on the average measures used in the experiment. This is in line with the findings of Weytjens et al. [2].

However, as noted by Ketyko et al. [10], average measures fail to capture certain differences between models. One of these differences is the speed at which models can be trained. In their paper, Weytjens et al. suggested that CNN models were a good alternative to LSTM models as

TABLE 6
Results of the process outcome prediction task. Three types of models were compared to: transformers, LSTMs and CNNs. Four average measures were computed: accuracy, f1-score, recall and precision.

Dataset	Label	Model	Acc.	F-Sc.	Rec.	Prec.	Time
BPIC2012	Acc.	Transf.	**0.76**	0.75	0.75	0.76	101
	Acc.	LSTM	0.74	0.73	0.73	0.74	202
	Acc.	CNN	0.73	0.72	0.72	0.73	253
	Canc.	Transf.	**0.80**	0.78	0.77	0.80	63
	Canc.	LSTM	0.78	0.77	0.77	0.78	175
	Canc.	CNN	0.77	0.76	0.74	0.77	116
	Decl.	Transf.	**0.82**	0.81	0.80	0.82	45
	Decl.	LSTM	0.80	0.80	0.79	0.80	154
	Decl.	CNN	0.80	0.77	0.78	0.80	54
BPIC2017	Acc.	Transf.	0.69	0.65	0.68	0.69	241
	Acc.	LSTM	**0.70**	0.67	0.68	0.70	326
	Acc.	CNN	0.68	0.63	0.66	0.68	320
	Canc.	Transf.	0.81	0.74	0.85	0.81	314
	Canc.	LSTM	0.81	0.74	0.84	0.81	480
	Canc.	CNN	0.81	0.73	0.83	0.81	1714
	Decl.	Transf.	**0.88**	0.83	0.88	0.88	322
	Decl.	LSTM	0.88	0.83	0.86	0.88	416
	Decl.	CNN	0.87	0.81	0.83	0.87	486
Traffic	Paym.	Transf.	0.68	0.68	0.68	0.68	26
	Paym.	LSTM	0.68	0.68	0.68	0.68	32
	Paym.	CNN	0.68	0.68	0.68	0.68	61

TABLE 7
Results of the experiment on transfer learning. For each dataset, two models were run for 10 epochs.

Dataset	Label	Model	Acc.	F-Sc.	Recall	Prec.	Time
BPIC2012	Acc.	Pre-trained	0.73	0.73	0.73	0.73	55
	Acc.	Benchmark	**0.76**	0.75	0.75	0.76	59
	Canc.	Pre-trained	0.77	0.77	0.76	0.77	54
	Canc.	Benchmark	**0.80**	0.78	0.77	0.80	58
	Decl.	Pre-trained	0.81	0.80	0.80	0.81	62
	Decl.	Benchmark	**0.82**	0.81	0.80	0.82	61
BPIC2017	Acc.	Pre-trained	0.64	0.59	0.74	0.68	240
	Acc.	Benchmark	**0.69**	0.67	0.67	0.69	254
	Canc.	Pre-trained	0.79	0.70	0.62	0.79	242
	Canc.	Benchmark	**0.81**	0.74	0.85	0.81	252
	Decl.	Pre-trained	0.88	0.83	0.88	0.88	254
	Decl.	Benchmark	0.88	0.83	0.88	0.88	247
Traffic	Paym.	Pre-trained	0.68	0.68	0.68	0.68	39
	Paym.	Benchmark	**0.68**	0.68	0.68	0.68	76

they reached similar results but required less time to be trained. In conducting these experiments however, the CNN models required significantly more time to be trained. The reason for this is that the CNN model of this experiment has more trainable parameters than the transformer models and LSTM models.

In similar vein, the performance of each model was examined across different prefix lengths. The resulting accuracy scores are presented in Figure 7. The results of the traffic dataset were not included as the prefix lengths are substantially shorter compared to other datasets (the largest prefix length of the traffic test set was only nine events). When examining the graphs, notice the differences between the models in the BPIC2012 datasets. For instance, the CNN model in BPIC2012c performs significantly worse on longer prefixes, while the CNN model in BPIC2012a actually outperforms the other models on longer prefixes. In BPIC2012d, the LSTM model seems to perform poor when the prefix length is larger than 40. Nevertheless, there is little evidence that one model would be preferred over another. In fact, if we examine the graphs of the BPIC2017 datasets, we notice that models behave even more competitively to each other as in BPIC2012. This is remarkable as both datasets originate from the same underlying process. The larger competitiveness of the BPIC2017 models may be related to the larger size of the BPIC2017 datasets. Another explanation could be that process improvements over the years have lead to less noise in the newer BPIC2017 dataset.

Transfer learning

The results of the second experiment showed that pre-trained models consistently perform below the benchmark. To make sure no errors were made during development, an additional test was conducted. In this test, we attempted to transfer representations without changing the prediction task. This was done by pre-training a model and only replacing the classification layer with a copy without trained weights. In this case, pre-training did reach high accuracy scores with only a fraction of the training time, as one would expect in a model that is essentially unchanged.

It becomes clear that pre-trained models do not perform as well after transfer learning as newly trained models in process prediction tasks. The poor performance is likely due to an increase in noise. Noise could have increased as a consequence of training for two different tasks. With this, the model essentially experienced twice the amount of training data. A larger amount of training data could, in turn, introduce more ambiguity as higher level representations might start to become contradicting. Usually a model would adjust its weights to fit the training data. In the case of transfer learning, however, a large set of weights is frozen and cannot be updated to fit the new data.

To study this, we created three additional transformer models for each dataset. The first model is not pre-trained. The second and third model are both pre-trained, but differ in the fact that one has all weights unfrozen. The accuracy scores for each of these models is displayed in Figure 8. Contrary to the previous experiment, we did not set a maximum number of epochs and allowed the models to train until the validation loss stopped improving. As expected,

the pre-trained model without freezing scores better than the model with freezing. In fact, the scores of the freshly trained model and the model without freezing seem to converge. However, not freezing any weights goes against the purpose of transfer learning, as no knowledge is actually transferred. Furthermore, the speed advantages of transfer learning will likely seize to exist as the amount of trainable weights increases.

Pre-trained models in domains such as NLP and CV do not seem to suffer from this phenomena, as indicated by their popularity [7]. A possible explanation lies in the "expressive power" of data. Presumably, text and image data contain more expressive power than process data. That is, a natural language has hundreds of thousands of words to express a certain thing. Meanwhile the datasets used in these experiments only contain tens of unique activities (Table 2).

6 Conclusion

In this paper, transformer-based neural networks with self-attention were used in predicting business process-outcomes. The resulting transformer models proved to perform competitively against established alternatives such as long short-term memory networks and convolutional neural networks. In fact, transformers proved to be significantly faster at training than LSTMs and CNNs. For some datasets, transformers even improved average measures with a few percentage points.

The role of transfer learning in process prediction tasks was studied next. To this objective, we compared the performance of two transformer-based models on the process-outcome prediction task. The first model was pre-trained on the next-activity prediction task and finetuned for outcome prediction. The second model was not pre-trained and merely served as benchmark for the first model. Results show that pre-trained models, despite being faster in training, consistently perform below the benchmark. The poor performance is likely due to an increase in noise as a consequence of being trained on two different tasks. Interestingly, pre-trained models in other domains do not seem to suffer from this. While a possible explanation was formulated, no additional research was done to verify the conjecture.

In developing the models, little attention was paid to hyperparameter tuning. Therefore the results obtained in these experiments are unlikely to be the best possible results. In similar vein, no effort was spent on creating the most optimal model architecture. As a result, the CNN model turned out to be relatively more complex by having more trainable parameters. In all likelihood, this impacted the speed at which the CNN model was trained, making the speed comparison between the models less accurate. In fact, and contrary to the findings in this paper, evidence exists that CNN models are preferred over LSTM models as they provide similar performance while requiring significantly less time to be trained.

Furthermore, all models were trained on trace input only. In other words, only activities and their respective order were taken into account. Additional input, such as event and case attributes, were not considered. It would seem

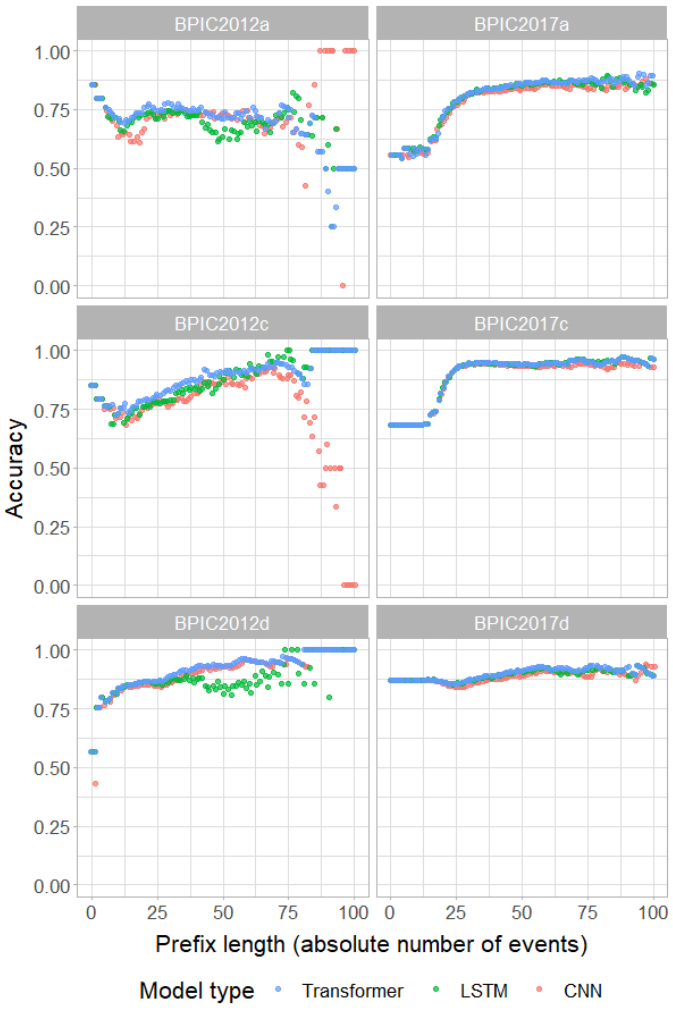

Fig. 7. Accuracy scores for different prefix lengths.

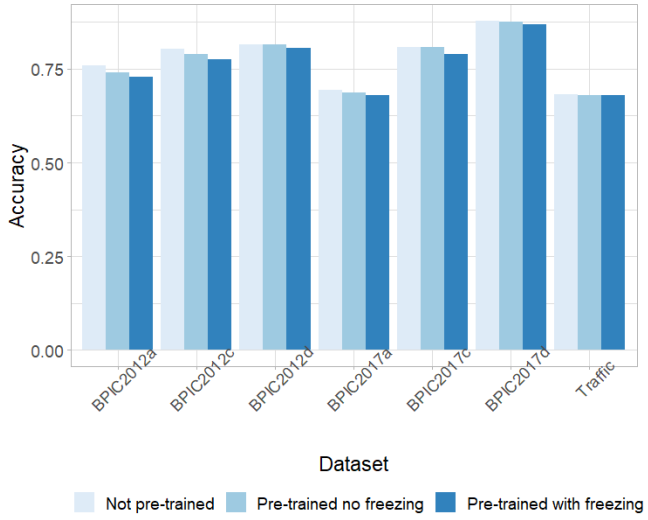

Fig. 8. Accuracy scores of pre-trained models with and without freezing, compared to a model with no pre-training. While not freezing improves accuracy, it defeats the purpose of transfer learning.

that including extra attributes could improve the expressive power of process data. Future research could thus focus on different techniques for incorporating additional attributes in deep learning models. Subsequently it could be beneficial to see whether adding extra attributes would impact the previously mentioned conclusions. Likewise, the datasets used in these experiments do not represent all possible process data. Therefore future research should attempt to include a broader variety of event logs to gauge the generalizability of the conclusions.

TABLE 8
Training information on each model is uploaded at https://tensorboard.dev/experiments/[EXPERIMENT ID].

Dataset	Model	Experiment ID
BPIC2012a	Transformer	tjH7ytA4SAqncKfznqNkQw
	LSTM	2HahLuoqSAyQsX5tvLWKsA
	CNN	vVeADqw2RXi9zT8yQAPv9Q
	Pre-trained	0U5L2najRBC6dWtPgEJPgw
	Benchmark	Gt8bhqPzRYGFEgQ6ZTlRug
BPIC2012c	Transformer	GMuewOwZTaaR8cv4Vd30rA
	LSTM	skztVHQuRC20jFvKZuNDyA
	CNN	teSr5eAcQEeVomAjHk0KvQ
	Pre-trained	Qfvz1ID5Txmk77vuGcKdhA
	Benchmark	jzwxR49fQhKdyf7eRKsXsg
BPIC2012d	Transformer	Ur8RGK9VRkuxUEUgR6kzUA
	LSTM	xMneB2WlTlu6GWZp5yy1ag
	CNN	k41zgYDvTXK5QUrV9LV7LA
	Pre-trained	cdAhzk82RWeYtzlEWPGNfg
	Benchmark	QorKdGyFR6exuna98qKZNg
BPIC2017a	Transformer	wXFLt4nZRjeJwsZpBrzCwQ
	LSTM	whJxlqa9QxmDb33Mhghxgg
	CNN	tOabNA0BS8CeMCDLrd65sg
	Pre-trained	LL1fLrEqTQ2lcbcN2fEVOg
	Benchmark	B3v4kRmDRserclX6LaHWWQ
BPIC2017c	Transformer	HAn9GxheQxWs2Te6RU9Vvg
	LSTM	j88uXwrZQDiR2U2OH6FqEQ
	CNN	4bXUjDnuRnaMqGumu4kAtg
	Pre-trained	pk0AFVixQYyQeZDYAr7L8A
	Benchmark	UqCMLro3Q5OUeP5VDkypUg
BPIC2017d	Transformer	lvC8hYHJSKef8sOw8O3gJg
	LSTM	700PQpEmQjGGoMNeHm8Tdg
	CNN	6OQOJD9JQOy0NvjIBdBWsw
	Pre-trained	7ylz0XIaQtmM2aPKYD8UGA
	Benchmark	qqAMj9O0Qhynru8pfZLJew
Traffic	Transformer	GXDqnpcaQnKJgwT0MGtqsw
	LSTM	WeAJcA4lT4yHq1sYcWJBQA
	CNN	WF8voDWsRnSdJiok9cAEpg
	Pre-trained	HQy8vW0ASkaf7MUZw30KRA
	Benchmark	brPrvjuPQ4uoDpYUpRhO0Q

REFERENCES

[1] Z. A. Bukhsh, A. Saeed, and R. M. Dijkman, "ProcessTransformer: Predictive Business Process Monitoring with Transformer Network," *arXiv:2104.00721 [cs]*, Apr. 2021, arXiv: 2104.00721. [Online]. Available: http://arxiv.org/abs/2104.00721

[2] H. Weytjens and J. De Weerdt, "Process Outcome Prediction: CNN vs. LSTM (with Attention)," *arXiv:2104.06934 [cs]*, vol. 397, pp. 321–333, 2020, arXiv: 2104.06934. [Online]. Available: http://arxiv.org/abs/2104.06934

[3] W. M. P. v. d. Aalst, *Process Mining: Data Science in Action*. Springer, Apr. 2016, google-Books-ID: hUEGDAAAQBAJ.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017, arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762

[5] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv:1810.04805 [cs]*, May 2019, arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805

[7] 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, and D. Amodei, "Language Models are Few-Shot Learners," *arXiv:2005.14165 [cs]*, Jul. 2020, arXiv: 2005.14165. [Online]. Available: http://arxiv.org/abs/2005.14165

[8] G. de Souza, S. Rabhi, J. M. Lee, R. Ak, and E. Oldridge, "Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation," *Fifteenth ACM Conference on Recommender Systems*, pp. 143–153, Sep. 2021. [Online]. Available: https://doi.org/10.1145/3460231.3474255

[9] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, conference Name: IEEE Transactions on Knowledge and Data Engineering.

[10] I. Ketykó, F. Mannhardt, M. Hassani, and B. F. van Dongen, "What averages do not tell: predicting real life processes with sequential deep learning," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. Virtual Event: ACM, Apr. 2022, pp. 1128–1131. [Online]. Available: https://dl.acm.org/doi/10.1145/3477314.3507179

[11] B. Wickramanayake, Z. He, C. Ouyang, C. Moreira, Y. Xu, and R. Sindhgatta, "Building interpretable models for business process prediction using shared and specialised attention mechanisms," *Knowledge-Based Systems*, vol. 248, p. 108773, Jul. 2022.

[12] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, "Outcome-Oriented Predictive Process Monitoring: Review and Benchmark," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 2, pp. 17:1–17:57, 2019. [Online]. Available: https://doi.org/10.1145/3301300

[13] E. Rama-Maneiro, J. Vidal, and M. Lama, "Deep Learning for Predictive Business Process Monitoring: Review and Benchmark," *IEEE Transactions on Services Computing*, pp. 1–1, 2021, conference Name: IEEE Transactions on Services Computing.

[14] W. Kratsch, J. Manderscheid, M. Röglinger, and J. Seyfried, "Machine Learning in Business Process Monitoring: A Comparison of Deep Learning and Classical Approaches Used for Outcome Prediction," *Business & Information Systems Engineering*, vol. 63, no. 3, pp. 261–276, Jun. 2021. [Online]. Available: https://link.springer.com/10.1007/s12599-020-00645-0

[15] S. Wang, L. Cao, Y. Wang, Q. Z. Sheng, M. A. Orgun, and D. Lian, "A Survey on Session-based Recommender Systems," *ACM Computing Surveys*, vol. 54, no. 7, pp. 154:1–154:38, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3465401

[16] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer," *arXiv:1904.06690 [cs]*, Aug. 2019, arXiv: 1904.06690. [Online]. Available: http://arxiv.org/abs/1904.06690

[17] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A

Robustly Optimized BERT Pretraining Approach," arXiv, Tech. Rep. arXiv:1907.11692, Jul. 2019, arXiv:1907.11692 [cs] type: article. [Online]. Available: http://arxiv.org/abs/1907.11692

[18] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[19] D. Rozado, "Using Word Embeddings to Analyze how Universities Conceptualize "Diversity" in their Online Institutional Presence," *Society*, vol. 56, no. 3, pp. 256–266, Jun. 2019. [Online]. Available: http://link.springer.com/10.1007/s12115-019-00362-9

[20] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent Neural Network Based Language Model," p. 4, 2010.

[21] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, "Learning Longer Memory in Recurrent Neural Networks," *arXiv:1412.7753 [cs]*, Apr. 2015, arXiv: 1412.7753. [Online]. Available: http://arxiv.org/abs/1412.7753

[22] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, "Understanding Hidden Memories of Recurrent Neural Networks," in *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Oct. 2017, pp. 13–24.

[23] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies," 2001.

[24] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[25] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997.

[26] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, Mar. 2020.

[27] J. Alammar, "The Illustrated Transformer." [Online]. Available: https://jalammar.github.io/illustrated-transformer/

[28] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *arXiv:1409.3215 [cs]*, Dec. 2014, arXiv: 1409.3215. [Online]. Available: http://arxiv.org/abs/1409.3215

[29] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *arXiv:1409.0473 [cs, stat]*, May 2016, arXiv: 1409.0473. [Online]. Available: http://arxiv.org/abs/1409.0473

[30] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *arXiv:1508.04025 [cs]*, Sep. 2015, arXiv: 1508.04025. [Online]. Available: http://arxiv.org/abs/1508.04025

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[32] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," Jul. 2016, arXiv:1607.06450 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1607.06450

[33] G. Ke, D. He, and T.-Y. Liu, "Rethinking Positional Encoding in Language Pre-training," *arXiv:2006.15595 [cs]*, Mar. 2021, arXiv: 2006.15595. [Online]. Available: http://arxiv.org/abs/2006.15595

[34] L. Torrey and J. Shavlik, "Transfer Learning," *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pp. 242–264, 2010, iSBN: 9781605667669 Publisher: IGI Global. [Online]. Available: https://www.igi-global.com/chapter/transfer-learning/www.igi-global.com/chapter/transfer-learning/36988

[35] F. Chollet, "Keras," 2015. [Online]. Available: https://keras.io

[36] B. van Dongen, "BPI Challenge 2012," Apr. 2012, medium: media types: application/x-gzip, text/xml Version Number: 1 Type: dataset.

[37] ——, "BPI Challenge 2017," Feb. 2017, medium: media types: application/x-gzip, text/xml Version Number: 1 Type: dataset.

[38] M. M. de Leoni and F. Mannhardt, "Road Traffic Fine Management Process," Feb. 2015, medium: media types: application/x-gzip, application/zip, text/xml Version Number: 1 Type: dataset.

[39] T. Developers, "TensorFlow," May 2022. [Online]. Available: https://zenodo.org/record/4724125