Predictive Analysis of Business Processes Using Neural Networks with Attention Mechanism

Patrick Philipp*, Ruben Jacob*, Sebastian Robert† and Jürgen Beyerer*†
*Vision and Fusion Laboratory IES, Karlsruhe Institute of Technology KIT, Karlsruhe, Germany
†Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB,
Fraunhofer Center for Machine Learning, Karlsruhe, Germany
Email: p.philipp@kit.edu

***Abstract*—The analysis of ongoing processes is an important task in business process management. This is not surprising, since (e.g.) being able to predict future events in processes enables companies to intervene at an early stage if deviations from a desired workflow are likely to occur. Subsequently, errors and associated financial losses can be prevented or avoided. A common basis for being able to predict future events is a sequence of previous events that are typically stored in a so-called event log. In this work we present a neural network with attention mechanism, which is trained using publicly available event logs (e.g. BPI Challenge 2013). Furthermore, we elaborate the proposed model with an extensive dataset of events of a worldwide known German software company. In addition to promising results, the training time of the proposed model is shorter than that of typical reference models such as neural networks with a long short-term memory architecture (LSTM).**

Keywords: Process Prediction, Deep Learning, Neural Networks, Attention Mechanism, Transformer.

I. Introduction

The ability to monitor ongoing business processes and to predict their future progress marks an important competitive advantage for a lot of companies [1]. It allows (e.g.) for intervening in time if deviations from the desired flow of events are likely to occur and can influence decisions on the distribution of strategic resources [2].

As a consequence, errors and associated financial losses can be prevented or avoided. Possible target groups are case managers identifying ongoing compliance violations to mitigate business risks or production managers for whom the future progress of events is important for planning resources.

The underlying problem to all of these tasks is the prediction of events occurring next. Data gathered during former process executions is commonly stored in so called event logs. These logs serve as valuable source for training predictive models, whereby the underlying assumption is that past events are a meaningful indicator for the future course of a process.

There have been several studies conducted in this field evaluating different approaches on relatively small event logs with a low level of complexity [3], [4]. In this work we use these logs as basis but also use larger event logs of complex processes for evaluation. Our approach is an adaption of the highly popular self-attention-based transformer model [5] used in natural language processing. To the best of our knowledge this is the first time transformer models are used for the prediction of future events in the given application case.

This paper is structured as follows: In Section II a short overview of the used terminology and the attention mechanism is given. In Section III we elaborate related approaches, whereas in Sections IV and V the used datasetes and the architecture of the proposed approach as well as further details of implementation are provided. Section VI shows the experimental settings and the results of evaluation. Finally, Section VII gives a summary and outlook.

II. Fundamentals

A. Event Log

An event log consists of several cases [6]. Each case represents the execution of a business process instance and is thus a trace of events (cf. Figure 1). Events contain attributes that represent their characteristics – typical attributes are the name of the executing activity, the timestamp of the event and organizational resources or roles. The problem of predicting the next event is typically understood as predicting the executing activity of the next event in an ongoing trace considering the sequence of past events from that particular trace.

B. Attention Mechanism

The neural attention mechanism first described in [9] is a construct commonly used in neural architectures for natural language processing. Its purpose in sequence prediction tasks is to learn how much attention has to be payed to each previous element in a sequence of elements for a best possible prediction of future elements.

While in early works the attention mechanisms have only be used in conjunction with recurrent neural networks, further works showed that these models can learn long-term dependencies within sequences on its own. Approaches based on these architectures have become known as transformer models [5].

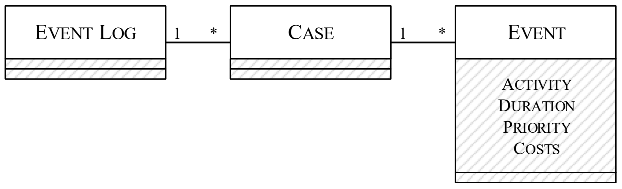


Figure 1: This figure depicts a simplified (UML-) class diagram to illustrate the interdependencies between event log, case and event in context of process analysis [7], [8]. An event log consists of multiple cases but one case is always assigned to exactly one event log. The same applies to the relationship between case and event; Typical attributes of an event are activities, duration, priority or costs [8].

The calculation of attention takes place in several steps: First, for each element x_i, the three vectors q_i (query), k_i (key) and v_i (value) are calculated by the use of learned linear transformations represented by weight matrices W_q, W_k, and W_v :

$$q_i, k_i, v_i = x_i W_q, x_i W_k, x_i W_v \ . \tag{1}$$

In a second step, the compatibility between the query vector and the key vectors of each element is calculated through a dot product. The result is normalized by a softmax function and multiplied with the corresponding value vector. Finally, the weighted value vectors are summed up to the final result. This calculation is given by:

$$\mathrm{Attention}(q_i, K, V) = \sum_{j=0}^{n} \mathrm{softmax}_i(q_i^T k_j) v_j \ , \tag{2}$$

where K and V are matrices of all key and value vectors. Generally, each position in a sequence can attend to any other position within a sequence. Since this would allow the prediction model to look into the future (and thereby considering the whole sequence of events), positions after the current one are masked by setting their values to negative infinity which causes the softmax function to not take them into account.

In practice, attention values for all positions are calculated simultaneously and the input to the softmax function is additionally scaled by $\frac{1}{\sqrt{d_k}}$ where d_k is the dimension of the key matrix. This prevents cancellation in the softmax for large dot products. Furthermore, the query, key and value vectors are projected by learned linear transformations into h different subspaces on each of which attention values are calculated in parallel. The results are concatenated and projected back into the feature space which allows the model to jointly attend to information from different representation subspaces at different positions [5]:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V, \tag{3}$$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{concat}\left(\mathrm{head}_1, ..., \mathrm{head}_h\right) W^O, \tag{4}$$

whereby a head is defined as

$$\mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V), \tag{5}$$

using the trainable weight matrices $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{model} \times d_k}$, and $W^O \in \mathbb{R}^{hd_k \times d_{model}}$. Please note, that d_{model} represents the size of the embedding.

III. Related Work

Classical approaches of process event prediction focused on using markov models [3] in combination with clustering algorithms such as k-means or k-NN [10]. Somewhat similar to a markov models, [4] used a probabilistic finite automaton based on bayesian concepts which uses the expectation maximization approach to estimate the relevant process parameters. The evaluation was carried out using event logs such as the publicly available BPI Challenge 2012 and BPI Challenge 2013.

Recently, as in many other fields, neural networks approaches have gain popularity. In an early study, the authors applied recurrent neural networks with a long short-term memory (LSTM) architecture and word embeddings for event prediction [11]. BPI Challenge 2012 and 2013 datasets were used to validate the prediction results.

Other works [12] shows that one-hot encodings may be used as an alternative to word embeddings. Also applying the LSTM approach, the authors transformed the event feature using one-hot encoding and concatenating it with the generated time features to a single feature vector. Additionally, the latter study [12] also conducted experiments on another dataset from an Italian software company.

Another approach treated event prediction as a classical multi-class classification problem and used stacked autoencoders to extract features which are then classified using a deep feed-forward network [13]. The study brought a new concept to the preprocessing of input data using a combination of representation through n-grams and feature hashing. However, this approach is only feasible for simple datasets since the amount of distinct representations grows polynomially with the number of unique event classes.

IV. Dataset

In this work, we examine the publicly available dataset from BPI Challenge 2013 [14] and an additional dataset supplied by a German software company. The event logs contain entries of the usage of a special software tool utilized by lawyers, accountants and auditors to conduct their financial accounting, manage their payment transactions and prepare their annual financial statements. The dataset consists of roughly 208 million events, each identified by a session

Dataset	Unique Event Classes	Number of Events	Number of Cases	Longest Case	Average Length
BPI '12 W	6	72,413	9,658	74	7.4977
BPI '12 A	10	60,849	13,987	8	4.6496
BPI '12 O	7	31,244	5,015	30	6.2301
BPI '13 I	13	65,533	7,554	123	8.6753
BPI '13 P	7	9,011	2,306	35	3.9076
DATSET	16,003	208,344,613	4,433,452	31,727	48.4384

Table I: Comparison of important characteristics (e.g. number of event classes) between BPI Challenge 2012, 2013 datasets and the extensive dataset additionally considered in this contribution (DATSET): On the one hand, the large amount of available data favors the training of neural networks. On the other hand, many unique event classes increase the difficulty of the prediction task.

id indicating the case they belong to, an event type and a timestamp. Each user interaction (usually the click of a button) is considered an event and a case lasts from startup of the application until it is shut down. Table I shows the most important characteristics of the dataset in comparison to datasets from BPI Challenge 2012 [15] and 2013 [14] used in previous works.

It clearly shows the differences in size and complexity. On the one hand, the large amount of available data helps in training neural network models that require a lot of training examples. On the other hand, the many unique event classes increase the difficulty of prediction.

It should be noted that as in smaller datasets the majority of cases are very short. About a quarter of all cases consist of five events or less and only one percent of all cases has a length of 500 or more. Similarly, their are a few event classes that occur very frequently while most others only occur a couple of times throughout the entire dataset. The five most common event classes make up almost half of the entire dataset.

V. Modeling Approach

The model was implemented in Tensorflow 2.0 [16] using the tf.data-API as input pipeline and the tf.keras-API to define our model.

The data is preprocessed in multiple steps. First, the event types are tokenized to integers and all events belonging to a case are put together in a tensor. We add an end-of-sequence token to each case and generate training labels by shifting to the left by one position. As shown in figure **??**, case lengths vary greatly. For naive batching where every case is padded to the length of the longest one, this results in a large amount of unnecessary padding. Instead, we group cases into buckets according to their length and only batch together cases from the same buckets. Each bucket contains cases of lengths l_{i-1} to l_i where

$$l_0 = 8, \tag{6}$$

$$l_i = \max(\alpha l_{i-1}, l_{i-1} + 1) \ . \tag{7}$$

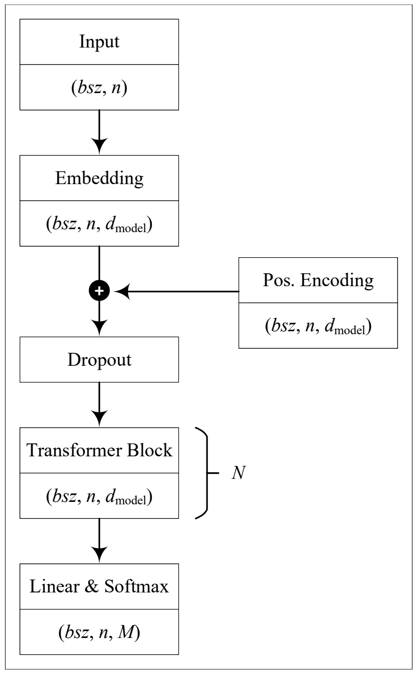
We chose $\alpha = 1.1$. The exponential growth of bucket sizes is motivated by the frequency distribution of trace lengths. To ensure similar processing times for each batch, we don't set a fixed batch size. Instead, we only define an upper bound for event per batch and vary batch sizes according to the trace length of the batch.

Figure 2 shows an overview of the proposed model. N denotes the number of Transformer blocks, d_{model} the size of the embedding, h the number of attention heads and d_k the dimensions of query, key and value vectors. M is the size of the vocabulary, n the case length of the batch and bsz the batch size.

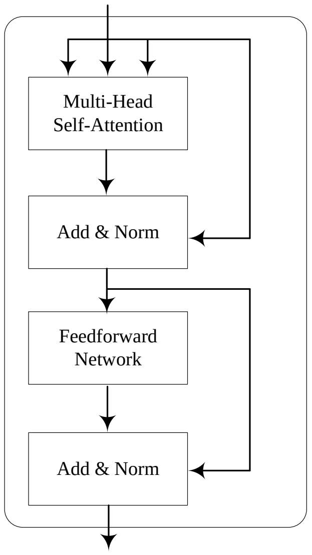
Inputs are mapped to an d_{model}-dimensional feature space by an embedding layer which is realized by a trainable lookup matrix $W^E \in \mathbb{R}^{M \times d_{model}}$ with random normally distributed initial weights. The same lookup matrix is used in the pre-softmax linear transformation to calculate the M-dimensional vector of logits from the output of the last transformer block.

Since transformer architectures don't process the cases sequentially, they need additional information about the position of the current element within the case. To achieve this, we add a positional encoding onto the embedding vector which is the same combination of sin and cosine functions as in [5]. The Transformer blocks themselves are also largely implemented as in the original paper consisting of a multi-head self-attention layer and a feed-forward network, each surrounded by a residual connection [17] and normalized through layer normalization [18] as well as dropout [19].

We used categorical cross entropy with label smoothing [20] as loss during training. That means that the training labels are smoothed by a parameter ϵ^{ls} such that the correct event type is represented as $1 - \epsilon^{ls}$ instead of 1 and all other event types as ϵ^{ls} instead of 0. This penalizes over-confidence in the model and thus increases its robustness. Furthermore, positions where the event type of the label is padding are ignored. We use the Adam optimizer [21] with parameters $\beta_1 = 0,9$, $\beta_2 = 0,997$ and $\epsilon = 10^{-9}$ and the learning rate schedule from [5] that first increases the learning rate for a number warmup steps before decreasing it thereafter. During prediction, the results of attention calculations are cached and reused for later calculations as suggested in [22].



(a)



(b)

Figure 2: Overview over the proposed model; Architecture is depicted on the left hand side and components of the transformer blocks are shown on the right hand side of the figure. d_{model} denotes the size of the embedding, h the number of attention heads and d_k the dimensions of query, key and value vectors. M is the size of the vocabulary, n the case length of the batch and bsz the batch size. N denotes the number of Transformer blocks.

VI. Verification and Validation

We ran experiments on two datasets: BPI '13 I and DATSET (cf. Table I, last row). Of the several datasets from the BPI Challenges 2012 and 2013, the incidents part of BPI '13 (BPI '13 I) is the most complex with the largest number of event types as well as the longest average length of cases. That's why we chose it to compare our model against previous approaches. Since there are no evaluations of other approaches for the DATSET dataset, we compare our model to a baseline with LSTM layers.

We split each dataset into three parts: training, validation and test. For each of the datasets we conducted an evaluation on the validation set of different sets of hyperparameters and number of transformer layers to find an optimal combination of the two. This configuration was then used to obtain the final results on the test set.

For BPI '13 I we chose a ratio of 80%, 10%, 10% between training, validation and test sets, for DATSET we chose 96%, 2%, 2% due to the much larger amount of available data. Models for BPI '13 I were trained for 30,000, models for DATSET for 100,000 training steps with one epoch consisting of 1,000 steps.

For BPI '13 I we evaluated four configurations of hyperparameters:

- $K_1 = \{d_{model} = 16, h = 2, d_{ff} = 128\}$
- $K_2 = \{d_{model} = 32, h = 4, d_{ff} = 256\}$
- $K_3 = \{d_{model} = 64, h = 4, d_{ff} = 512\}$
- $K_4 = \{d_{model} = 128, h = 8, d_{ff} = 1024\}$

d_{model}	h	d_{ff}	N	Accuracy
16	2	128	1	0,7151
16	2	128	2	0,7160
16	2	128	4	**0,7166**
32	4	256	1	0,7159
32	4	256	2	0,7153
32	4	256	4	0,7122
64	4	512	1	0,7122
64	4	512	2	0,7140
64	4	512	4	0,7119
128	8	1024	1	0,7148
128	8	1024	2	0,7099
128	8	1024	4	0,7099

Table II: Results of the evaluation of different sets of hyperparameters for BPI '13 I.

Each of these configurations was tested in combinations with one, two and four transformer layers, respectively. All models were trained on a single NVIDIA V100 GPU with training taking between 60 and 70 seconds per epoch.

Table II shows the results of the evaluation – the smallest set of hyperparameters in combination with four layers achieves the highest accuracy. Configurations with larger dimensions in general seem to perform worse. Since train-

	[4]	[11]	[13]	Proposed Model
Accuracy	0,714	**0,735**	0,655	0,707

Table III: Comparison of final result on BPI '13 I with related work (cf. Section III).

d_{model}	h	d_{ff}	N	Accuracy
128	8	1024	4	0,5658
128	8	1024	6	0,5561
256	8	1024	4	**0,6218**
256	8	1024	6	0,6212
512	8	2048	4	0,6198
512	8	2048	6	0,5808

Table IV: Results of the evaluation of different sets of hyperparameters for DATSET.

ing accuracy keeps increasing with size, this could be an indicator of overfitting. Table III compares the achieved results on the test set to those of other approaches. While the performance is clearly better than the n-gram based approach of [13] and comparable to the PFA of [4] it does not match the accuracy reported in [11]. However, the latter approach uses additional process information other than the previous events in the trace which could improve their performance. This is in line with our expectations. However, the experiment demonstrates that our attention-based approach is generally suited to the task of process event prediction and can perform comparably to the state of the art.

Due to the much larger size of the DATSET dataset, we used configuration K_4 as well as two additional sets of hyperparameters and evaluated each in combination with four and six transformer layers:

- $K_5 = \{d_{model} = 256, h = 8, d_{ff} = 1024\}$
- $K_6 = \{d_{model} = 512, h = 8, d_{ff} = 2048\}$

All models were also trained on a single NVIDIA V100 GPU with training taking between 180 and 230 seconds per epoch. Very long cases with lengths greater than 500 were filtered out due to memory limitations. As described in section IV, these cases only make up about one percent of the entire dataset. Table IV shows the results of the evaluation on the validation set. Configuration K_5 in combination with 4 transformer layers achieves the highest accuracy. Models with smaller dimensions perform significantly worse which suggests that they are unable to fully model the complexity of the data.

In order to judge this performance we developed a baseline model and trained it under the same terms as the attention-based model. The baseline model follows the same architecture as the proposed approach, only the transformer blocks have been swapped for LSTM layers and the positional encoding has been removed since it is unnecessary. We used the same embedding size $d_{model} = 256$ and number of layers $N = 4$ as well as the same loss function and training setup described in section V. This allows for a ceteris paribus comparison of the performance of transformer layer versus LSTM layer.

	LSTM-Baseline	Proposed Model
Accuracy	0,5403	**0,6219**

Table V: Comparison of results on DATSET between the proposed model and the LSTM-Baseline.

Table V shows the results of the experiment on the test set. Our proposed attention-based model seems to perform better than the LSTM-based baseline model. Moreover, training of the baseline model took approximately twice as long per epoch as the training of our proposed model of similar size. This shows the advantages of the attention mechanism for long trace lengths which is able to process an entire trace at once instead of one element at a time.

VII. Conclusion

In this work we presented a neural attention-based model for the task of business process event prediction. We demonstrated that attention-based models are suited to this task and can achieve results comparable to the state of the art on smaller datasets such as BPI. We also elaborated the proposed attention-based model on a more complex dataset, resulting in good accuracy and training time compared to a baseline LSTM model.

Currently, there is a lot of research activity for transformer models and it's going to be interesting to see the technology develop. Future work in their application to business process analysis will focus on expanding the context length for very long traces. Two possible approaches to this could be either the segmenting of traces and introducing a segment-level recurrence as in [22] or only partially calculating the attention matrix as in [23]. Furthermore, we plan to expand our architecture to include other process attributes in the prediction.

Acknowledgment

The presented study is part of the project "KIRA – Artificial Intelligence for the Review of Workflows" which is funded by the Federal Ministry of Education and Research (BMBF), Germany. Additionally, this contribution is supported by the Fraunhofer Cluster of Excellence "Cognitive Internet Technologies" Center for Machine Learning. The authors Ruben Jacob and Patrick Philipp contributed equally to this publication.

References

[1] W. M. Van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Information systems*, vol. 36, no. 2, pp. 450–475, 2011.

[2] B. F. van Dongen, R. A. Crooy, and W. M. van der Aalst, "Cycle time prediction: When will this case finally be finished?" in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2008, pp. 319–336.

[3] M. Le, B. Gabrys, and D. Nauck, "A hybrid model for business process event prediction," in *Research and Development in Intelligent Systems XXIX*, M. Bramer and M. Petridis, Eds. London: Springer London, 2012, pp. 179–192.

[4] D. Breuker, M. Matzner, P. Delfmann, and J. Becker, "Comprehensible predictive models for business processes," *MIS Q.*, vol. 40, no. 4, pp. 1009–1034, Dec. 2016. [Online]. Available: https://doi.org/10.25300/MISQ/2016/40.4.10

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[6] X. W. Group, "Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams," *IEEE Std 1849-2016*, pp. 1–50, 11 2016.

[7] W. van der Aalst, *Process Mining*. Springer Berlin Heidelberg, 2016, pp. 3–23.

[8] P. Philipp, R. Morales Georgi, S. Robert, and J. Beyerer, "Analysis of control flow graphs using graph convolutional neural networks," *IEEE Intl. Conference on Soft Computing an Machine Intelligence, Johannesburg, South Africa*, 2019.

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

[10] M. Le, D. Nauck, B. Gabrys, and T. Martin, "Sequential clustering for event sequences and its impact on next process step prediction," in *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, A. Laurent, O. Strauss, B. Bouchon-Meunier, and R. R. Yager, Eds. Cham: Springer International Publishing, 2014, pp. 168–178.

[11] J. Evermann, J. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *CoRR*, vol. abs/1612.04600, 2016. [Online]. Available: http://arxiv.org/abs/1612.04600

[12] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with lstm neural networks," in *Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Cham: Springer International Publishing, 2017, pp. 477–492.

[13] N. Mehdiyev, J. Evermann, and P. Fettke, "A multi-stage deep learning approach for business process event prediction," in *2017 IEEE 19th Conference on Business Informatics (CBI)*, vol. 01, 07 2017, pp. 119–128.

[14] W. Steeman, "Bpi challenge 2013," 2013. [Online]. Available: https://data.4tu.nl/repository/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07

[15] Van Dongen, B.F., "Bpi challenge 2012," 2012. [Online]. Available: https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] J. L. Ba, J. R. Kiros, and G. Hinton, "Layer normalization," *arXiv preprint arXiv:1607:06450*, 2016.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salkutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, 2014.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architeture for computer vision," *arXiv preprint arXiv:1512.00567*, 2015.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] Z. Dai, Z. Yang, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[23] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.