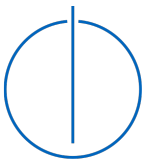
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Evaluating Approaches to Overcome the Input Size Limitations of Transformer-Based Language Models on Patent Documents

Jan Robin Geibel



DEPARTMENT OF INFORMATICS

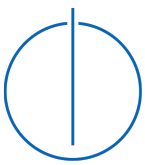
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Evaluating Approaches to Overcome the Input Size Limitations of Transformer-Based Language Models on Patent Documents

Evaluierung von Ansätzen zur Überwindung der Beschränkungen der Eingabegröße von Transformer-basierten Sprachmodellen für Patentdokumente

Author:	Jan Robin Geibel
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Anum Afzal
Submission Date:	10.10.2022



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 10.10.2022 Jan Robin Geibel

Acknowledgments

I would like to thank Professor Matthes for giving me the opportunity to conduct this project at the Chair for Software Engineering for Business Information Systems. I would further like to thank Anum Afzal for her continued support and advice during this project.

Abstract

Large pretrained language models have made substantial progress in their ability to encode a text sequence's syntactic and semantic information. Such language models thereby enable the construction of powerful machine learning models with little additional training data for downstream tasks ranging from question answering systems to text classification. The key to their pervasive success is the Transformer architecture they are based on. The Transformer and its self-attention mechanism are able to capture long-range dependencies in sequential data. Furthermore, self-attention allows the tokens of an input sequence to be processed in parallel. This parallelism enables the degree of pretraining necessary to achieve state of art results on downstream tasks. However, the Transformer's memory and compute requirements grow quadratically with regard to the input sequence's length. This renders processing long sequences prohibitively expensive. This paper examines a selection of the models created to overcome these limitations and evaluates different aspects of their performance on downstream machine learning tasks. The BigPatent corpus, a collection of of U.S. patent documents, is used to set up benchmark task in which a variety of model configurations is tested on classifying patents according to their subject matter. The project's findings indicate that the encodings produced by efficient Transformers retain more information, as they are able to process longer sequences at a time. Moreover, significant differences in the ability of various attention mechanism's to absorb information during training on a downstream task are found.

Contents

1. Introduction

Large pretrained language models have enabled tremendous progress in the field of Natural Language Processing (NLP) [38]. In essence, these language models process text sequences and yield numeric encodings thereof. These numeric vector representations can be used as input for a multitude of downstream machine learning tasks such as text classification. Models like BERT (Bidirectional Encoder Representations from Transformers) are able to capture complex contextual information in their encodings and thus enable the construction of powerful machine learning models with little additional training data [11].

A key enabler in the development of pretrained language models was the introduction of the Transformer architecture and the so-called self-attention mechanism. The Transformer is able to capture long-range dependencies within sequential data [51]. It further allows the tokens of an input sequence to be processed in parallel. It can thus fully utilize modern hardware accelerators such as GPUs or TPUs for the training of models on vast amounts of data [56]. This parallelism, therefore, enables the degree of pretraining necessary to achieve state of art results on downstream tasks with little additional data [56].
The Transformer's memory and compute requirements grow quadratically with regard to the input sequence's length. This, however, renders processing long sequences prohibitively expensive [2]. The resulting limitations of Transformer based language models have sparked a keen interest in the development of less resource intensive Transformer architectures [49]. This project examines a selection of the approaches followed with these models and evaluates their implications for downstream machine learning tasks. One question investigated is which models for encoding long text sequences are most suited for downstream machine learning tasks. The project, thus, intends to answer whether the encodings produced by these models retain enough of the information codified in the original sequences, so that another machine learning model can be successfully trained on downstream tasks with these encodings as inputs. The second question posed is how adapting a model's attention mechanism effects performance on downstream tasks. The corresponding experiments, consequently, aim to explore the models' ability to incorporate information while being finetuned on a downstream task. The last phase is concerned with finding the most performant

machine learning model with regard to the downstream task set up for this project. The BigPatent corpus, a collection of 1.3 million records of U.S. patent documents, is used to set up benchmark task in which a variety of model configurations is tested on classifying patents according to their subject matter.

The first chapter of this paper outlines prior efforts in surveying efficient Transformer architectures as well as attempts at testing their performance on downstream tasks and machine learning benchmarks.
The second chapter is concerned with describing the Transformer's architecture and limitations. It further gives a comprehensive overview of the approaches followed with various efficient Transformer models according to the taxonomy developed by Tay et al. in 2020 [49]. Moreover, detailed descriptions of the models used in this project's experiments, i.e., the Reformer [21], the Linformer [53], the Performer [9], the Longformer and Longformer-Encoder-Decoder [2] as well as BigBird [56], are given. Apart from that, the chapter provides an outline of the theoretical bases of the classification models used.
The third chapter describes the BigPatent corpus as well as the categorization scheme according to which patents are classified. Several samples of various sizes are drawn from the BigPatent corpus for development and experimentation purposes. As the length of the input sequences is of paramount importance for the experiments conducted in this project, an overview of the average number of tokens per document in the various samples is given. Furthermore, the distribution of the number of tokens per document within different classes is illustrated.
Subsequently, the methodology followed throughout the conducted experiments and the research questions they intent to answer are described. This section also includes information about the hardware accelerators used in the project.
Lastly, the experiments' exact configurations and findings are discussed. This section ends with some concluding remarks and thoughts regarding possible future work going beyond the scope of this project.

2. Prior Related Work

Several attempts at surveying and categorizing the ever-growing number of Transformer variants have been made. Tay et al. provide a survey of model architectures that address the shortcomings of the original Transformer [49]. The authors develop a taxonomy of the approaches followed with these models and further provide a detailed description of a variety of efficient Transformer architectures. Fournier et al. similarly explore widely used methods to reduce the memory and compute requirements of the Transformer. Apart from that, they discuss several of their strengths, limitations, and primary assumptions [12]. Moreover, Tianyang et al. compare various efficient Transformers with regard to their architecture, pretraining, and application [29].

The *Long Range Arena* was introduced by Tay et al. in 2021 in order to benchmark a variety of efficient Transformer architectures on different tasks involving the processing of long sequences [48]. The authors compare the performance of the Sparse Transformer [7], the Sinkhorn Transformer [47], the Synthesizer [46], the Linear Transformer [19], the Reformer [21], the Linformer [53], the Performer [9], the Longformer [2] as well as BigBird [56]. The latter five models are also evaluated during this project. The authors state the following six desired characteristics of the Long Range Arena [48]:

- **Generality**: The tasks considered should be suitable for all evaluated models.
- **Simplicity**: The tasks should be designed in a simple manner in order to ensure comparability between models.
- **Challenging**: The tasks should be challenging enough to allow for a certain margin for future advancement of the models.
- **Long inputs**: The length of the input should be long enough to sufficiently assess the models' ability to process long sequences.
- **Probing diverse aspects**: The tasks considered should be diverse enough to assess different aspects of the models' performance.
- **Non-resource intensive and accessible**: The tasks should be non-resource intensive so that they can be easily repeated.

Tay et al. introduce the following tasks to evaluate the models' ability to grasp long-range dependencies within the input sequence [48]:

- **Long listops**: This task is an extension of the listops task introduced by Nangia and Bowman in 2018 [34]. It is intended to test a model's ability to process data possessing a hierarchical structure. The dataset is comprised of pairs of sequences of numbers and operators, such as the *maximum*, *minimum*, or *median*, as well as the corresponding output numbers.
- **Byte-level text classification**: Tay et al. use the IMDb reviews dataset [32] to test the considered model on a text classification task. The authors use byte-level classification to simulate longer sequences. The maximum length of an input sequence is set to 4k positions.
- **Byte-level document retrieval**: The authors use the ACL Anthology Network dataset [41] to asses a model's ability to compress long inputs in order to determine the similarity between two sequences. The maximum length of one input sequence is set to 4k positions and two sequences are presented to the model at once.
- **Image classification on sequences of pixels**: In this task the CIFAR-10 dataset [24] is used to assess a model's classification capabilities when being presented with the image as a sequence of pixels.
- **Pathfinder (long-range spatial dependency)**: The authors evaluate each model on the path finder challenge [30] in which it needs to evaluate whether two dots in an image are connected by a dashed line. The image is presented as a sequence of pixels to a model.
- **Pathfinder-X (long-range spatial dependency with extreme lengths)**: In this variation of the pathfinder challenge the sequence length is increased from 1,024 to 16k.

The creators of BigBird assess their model's performance on question answering, classification, and summarization in the context of natural language processing [56]. Moreover, the authors conduct several experiments related to Genomics. For question answering, the Natural Questions [25], HotpotQA-distractor [55], TriviaQA-wiki [18], and WikiHop [54] datasets are used. Furthermore, BigBird is being evaluated on classification tasks using the IMDb [32], Yelp-5 [58], Arxiv [15], BigPatent [26], and Hyperpartisan [20] datasets. For summarization, the Arxiv [10], PubMed [10], BigPatent [44], BBC XSum [35], and CNN/DailyMail [17] datasets are used. Lastly, the authors test their model

on the GLUE (General Language Understanding Evaluation) [52] benchmark which comprises various natural language understanding tasks [56].

Beltagy et al. evaluate the Longformer on question answering, coreference resolution, and document classification [2]. For question answering, the authors make use of the WikiHop [54], TriviaQA [18], and HotpotQA [55] corpora. The OntoNotes dataset [39] is used to test the model's coreference resolution capabilities. Both the IMDb [32] as well as the Hyperpartisan [20] datasets are used to evaluate the Longformer's performance on a classification task. Furthermore, the authors test the Longformer-Encoder-Decoder's performance on text summarization using the ArXiv dataset [10].

Kitaev et al. run several experiments with the Reformer and the imagenet [43] as well as the enwik8[1] datasets [21]. The authors further test the Reformer on the WMT 2014 English-to-German translation task [4].

Choromanski et al. conduct various pretraining experiments with the Performer and apply their model to protein sequences [9].

Apart from examining the Linformer's pretraining perplexities, Wang et al. finetune the model on a variety of downstream tasks [53]. The the authors use the IMDb [32] and SST-2 [45] datasets to evaluate the Linformer's performance on sentiment classification. They further use QNLI [42] to test its natural language inference capabilities and QQP [6] to investigate its ability to judge text similarities.

[1] https://huggingface.co/datasets/enwik8

3. Background

3.1. The Transformer and Its Limitations

Before the Transformer was introduced by Vaswani et al. in 2017 [51], machine learning models for processing sequences were predominantly relying on a recurrent or convolutional neural network architecture. Recurrent models, such as gated recurrent neural networks and long short-term memory models, process their input sequentially, with the hidden state h_t at position t being computed from the respective input as well as the prior hidden state h_{t-1} [51]. Notably, these sequential computations cannot be parallelized [51]. A variety of such models do already incorporate an attention mechanism in combination with a recurrent network to account for the dependencies between tokens irrespective of their distance within the sequence [51]. The architecture proposed by the Transformer, however, entirely foregoes recurrent and convolutional networks and wholly relies on self-attention. The Transformer, therefore, allows a substantially higher degree of parallelization [51].

3.1.1. Attention

The attention module allows the model to incorporate contextual information from every position of the sequence [2]. In a first step, query, key, and value vectors are derived from each position of the input sequence by multiplying each input vector with three individual weight matrices W_q, W_k, W_v $\in \mathrm{R}^{d_{model} \times d_k}$ [51]:

$$Q = XW_q \tag{3.1}$$

$$K = XW_k \tag{3.2}$$

$$V = XW_v \tag{3.3}$$

where X $\in \mathrm{R}^{n \times d_{model}}$ [49].

In general, an attention function computes an updated value vector from a set of value, key, and query vectors. Said updated value vector equals the weighted sum of value vectors, with the weights being derived from the respective query and key vectors using a compatibility function [51]. The dot-product attention used by Vaswani et

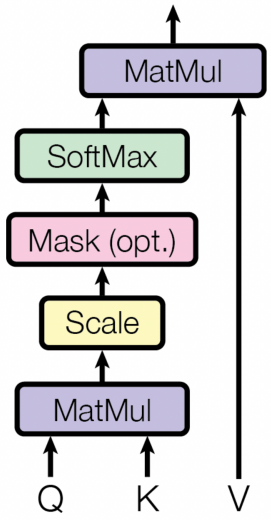


Figure 3.1.: Illustration of scaled dot-product attention (taken from [51]).

al. in the original Transformer derives weighting factors assigned to individual value vectors by calculating the dot product between a key and query vector, multiplying with $\frac{1}{\sqrt{d_k}}$ and applying a softmax function. For all positions of the sequence the described computation can be expressed as a matrix multiplication as follows [51]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.4}$$

where the softmax function is applied row-wise [49].

3.1.2. Architecture

A Transformer is composed of multiple layers of Transformer blocks, with each of these blocks consisting of multiple self-attention heads, a feed-forward network, layer normalization as well as residual connectors [49]. Transformers take varying shapes depending on the use case the model is built for. Transformers can be used as encoder-only modules, e.g, for a classification task, decoder-only modules, e.g., for language modelling purposes, or in an encoder-decoder way for sequence to sequence tasks [49]. While the original Transformer was designed in an encoder-decoder manner [51], this project focuses on encoder architectures.

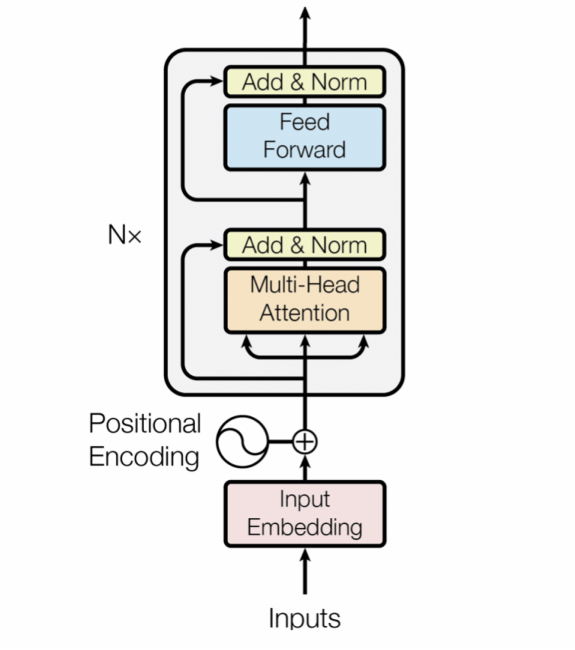


Figure 3.2.: Architecture of an encoder block (taken from [51]).

Embeddings and Positional Encodings

In a first step, each token of the input sequence is numerically encoded by an embedding layer. In order to incorporate information about the order of the sequence, so-called *positional encodings* are added to the resulting embeddings [51]. These postional encodings can themselves be produced by a trainable embedding layer [49]. Yet, the orginal Transformer uses sine and cosine functions of different frequencies to generate the encodings. Thus, each dimension conforms to a sinusoid [51].

Multi-Head Attention

The Transformer model uses multiple attention heads. The concatenated outputs of these heads serve as input to a further dense layer resulting in the final values [49]. Each head thus incorporates information from a different representation subspace at a different position [51].

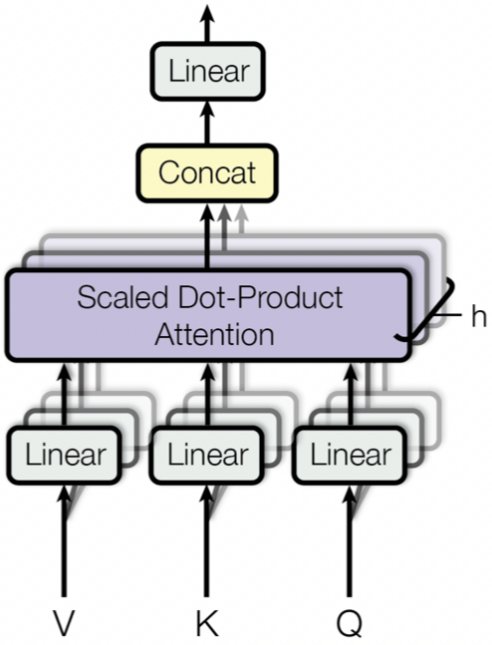


Figure 3.3.: Illustration of multi-head attention (taken from [51]).

Position-wise Feed-forward Network

Each position of the output of the multi-head self-attention mechanism is processed independently yet identically by a two-layered fully connected feed-forward network with a ReLU activation function [51]. The feed-forward network thus gives the following equation:

$$X_F = F_2(ReLU(F_1(X_A))) \tag{3.5}$$

where F_1 and F_2 represent the respective layers of the network and X_A is the output of the attention module [49].

Residual Connectors and Layer Normalization

The outputs of both the attention mechanism as well as the feed-forward network are passed to a layer normalization layer before being additively connected to the respective module's input [49]. One Transformer block can consequently be described as:

$$X_A = LayerNorm(MultiheadAttention(Q_x, K_x)) + X \tag{3.6}$$

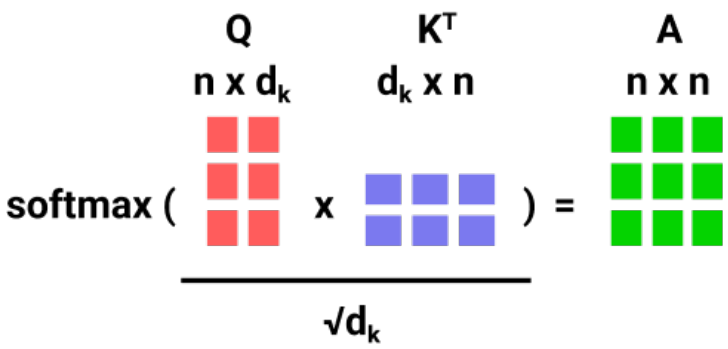


Figure 3.4.: Illustration of the computation of the attention matrix (A) by multiplying the query (Q) and the key matrices (K) with d_k denominating the dimensions of key and query vectors and n representing the sequence length (based on [51]).

$$X_B = LayerNorm(PositionwiseFNN(X_A)) + X_A \tag{3.7}$$

with X being the block's input, Q_x the query values, K_x the key values and X_B the output of the entire block [49].

3.1.3. Limitations

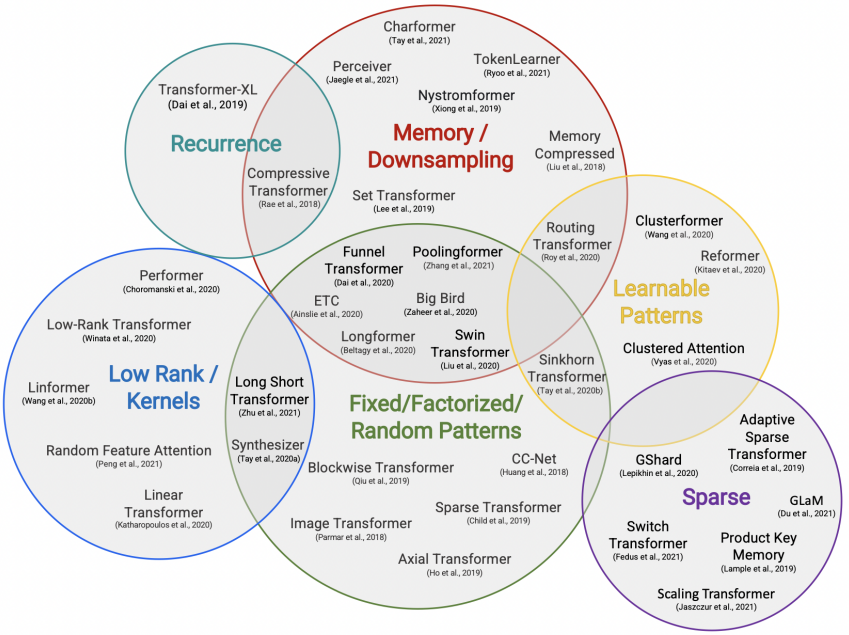
The scaled dot-product attention used in the attention heads of a Transformer block has quadratic complexity with regard to the sequence length [49]. This quadratic complexity results from the multiplication of the query and the key matrices (see Figure 3.4.) and is expressed both in terms of memory as well as compute requirements. Transformers in their original form are thus unequipped to process longer sequences without the need of vast amounts of resources [2]. In the majority of cases, memory requirements have larger implications during training than during inference, as the former requires gradient updates while the latter does not [2]. The quadratic complexity, however, limits the calculation speed during training as well as inference [2]. Apart from the computation of the attention matrix, the feed-forward network contributes substantially to the compute cost of each Transformer block [2]. The described limitations with regard to the input sequence's length sparked the introduction of a large variety of Transformer variants with the ability to process longer sequences while requiring fewer resources [49].

3.2. Efficient Transformers

3.2.1. Overview

Tay et al. provide a taxonomy of common approaches to overcoming the shortcomings of the vanilla Transformer [49]. See Figure 3.5. for an overview and the corresponding classification of essential Transformer architectures. The authors distil the following methods frequently used to improve the Transformer's efficiency:

- **Fixed Patterns**: One way to adapt the attention mechanism is to sparsify the attention matrix, thereby effectively limiting the number of attention operations that are performed. This is achieved by processing only parts of the sequence at a time. Blockwise patterns chunk the sequence into blocks of fixed size. Strided patterns process the input in overlapping intervals using a strided window that is being slid over the sequence. Compressed patterns employ a pooling operation to compress the sequence to a shorter length.
- **Combination of Patterns**: The attention mechanism can also be adapted by using multiple of the above described patterns.
- **Learnable Patterns**: In some models the way attention patterns are being applied is learned from data rather than being defined ex ante. The model's architecture incorporates a way to determine the relevance of tokens to each other according to which the attention pattern is used.
- **Neural Memory**: Another method, termed neural memory by Tay et al., defines certain global tokens which attend the entire sequence. According to the authors, this can be seen as similar to a pooling operation which allows the model to form a compressed representation of the input sequence.
- **Low-Rank Methods**: Other model architectures use the assumption that the attention matrix (A) is low-rank. A is consequently approximated with a smaller matrix.
- **Kernels**: The use of kernel functions allows the mathematical reformulation of calculations performed in the attention module in order to circumvent the explicit computation of the attention matrix. Kernel methods, therefore, approximate the attention matrix and can thus also be seen as low-rank methods [8].
- **Recurrence**: Approaches using recurrence extend the fixed blockwise patterns described above. The chunks of the original input sequence are connected via recurrent operations rather than merely being processed independently.



.

Figure 3.5.: Illustration of the taxonomy of efficient transformers developed by Tay et al. (taken from [49]).

- **Downsampling**: Downsampling the original input sequence is another way to reduce the costs of the attention mechanism.
- **Sparse Models and Conditional Computation**: Tay et al. consider models in which a subset of parameters are only sparsely used as examples of this class. This approach is closely linked to fixed patterns methods described above. In this case, however, the approach is not limited to the attention mechanism.

Of the models used in this project Tay et al. classify the Reformer as a model using learnable patterns, the Performer as one using kernel methods, the Linformer as one using a low-rank approach, and the Longformer and BigBird as models using a combination of fixed patterns and neural memory [49]. For an overview see Table 3.1.

3.2.2. BigBird

BigBird is a long sequence Transformer that was introduced by Zaheer et al. in 2020 and can process up to 4,096 tokens at a time. The attention mechanism of BigBird

Model	Complexity	Class
Longformer [2]	$O(N(k+m))$	fixed patterns and neural memory
BigBird [56]	$O(N)$	fixed patterns and neural memory
Performer [9]	$O(N)$	kernel method
Reformer [21]	$O(N \log N)$	learnable patterns
Linformer [53]	$O(N)$	low-rank method

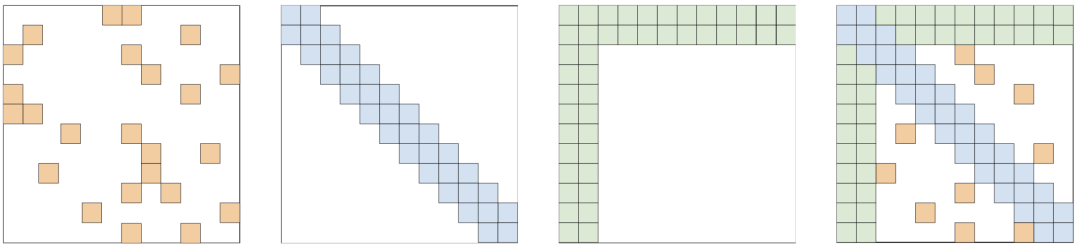
Table 3.1.: Overview of efficient transformer approaches examined in this project (the Longformer Encoder-Decoder is a variation of the Longformer) with N refering to the sequence length (taken from [49]).

essentially consists of three parts in which all tokens attend to a set of global tokens, a set of randomly chosen tokens, and all tokens in direct adjacency [56]. The set of global tokens attending to the entire sequence are artificially introduced ones, such as the *[CLS]* token which was already employed in BERT to derive a compressed representation of a sentence that can be used for classification purposes [11]. The local attention is implemented in form of a sliding window of width w in which a token attends to the $w/2$ preceding and following tokens in the sequence. See Figure 3.6. for an illustration of BigBird's attention mechanism. The BigBird model's memory complexity is linear with regard to the length of the input sequence, i.e., it is $O(N)$ [49].

Zaheer et al. implement BigBird's attention mechanism as a computation on blocks of tokens. Infrequent retrievals of small size cannot be efficiently performed on GPUs or TPUs. On GPUs computations are performed on a large number of cores in parallel which makes sparse matrix multiplication inefficient [56]. Computing attention on blocks of tokens, thus, optimizes BigBird for modern hardware accelerators [56]. In effect, this means that queries and keys are combined to blocks. In random attention, for instance, each such query block then attends to a predefined number of randomly chosen key blocks [56]. See Figure 3.7. for illustration of this block-wise attention mechanism. By default BigBirds uses a sliding window size of three blocks, three randomly chosen blocks in each attention operation, and a block size of 64 tokens. See here[1] for more information on BigBirds default configurations.

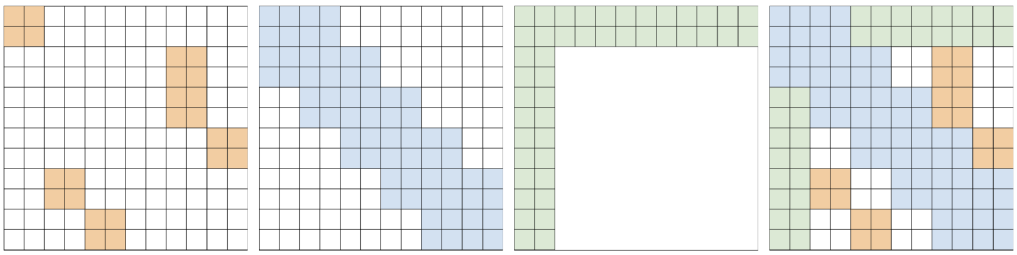
BigBird is available as a base as well as a large version, both having been pretrained on masked language modeling starting from roBERTa checkpoints [56]. roBERTa is a version of BERT that has been pretrained for a longer time using bigger batch sizes and

[1] https://huggingface.co/transformers/v4.7.0/model_doc/bigbird.html



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 3.6.: Illustration of Big Bird's attention mechanism with white spaces signaling that no attention is computed (a) showing random attention with 2 tokens, (b) showing sliding window attention with a width of 3 and (c) showing the pattern created by 2 global tokens (taken from [56]).



(a) Random Attention (b) Window Attention (c) Global Attention (d) BIGBIRD

.

Figure 3.7.: Illustration of Big Bird's block-wise attention mechanism with a block size of 2, effectively breaking up the attention matrix into 2 x 2 blocks (taken from [56]).

more data [31]. Furthermore, roBERTa is not trained on next sentence prediction, but only on masked token prediction with dynamically changing masking patterns [31]. Zaheer et al. use the BookCorpus [59] as well as the CC-News [14], Stories [50], and English Wikipedia datasets to further pretrain BigBird.

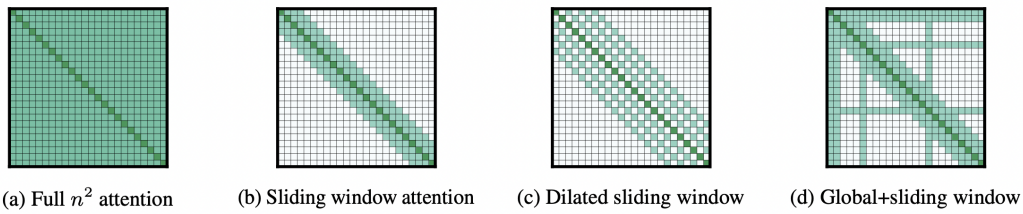
3.2.3. Longformer

The Longformer was introduced by Beltagy et al. in 2020. The model can process up to 4,096 tokens at once. Similar to the BigBird model, the Longformer relies on a sliding window attention of width w with each token attending to the $w/2$ preceding and following tokens in the sequence. Stacking multiple layers, each using sliding window attention, ensures that a large amount of contextual information is embedded in each token's encoding [2]. This resembles the way convolutional neural networks (CNNs) function [40]. Apart from sliding window attention, the authors use what they term dilated sliding window attention. Similar to dilated CNNs [36], the sliding window in this attention mechanism possess gaps [2]. This in effect reduces the resolution of the sequence and allows the model to include more contextual information with fixed computational costs [2]. Moreover, the authors find that varying the degree to which the window attention is dilated across attention heads further increases performance. The Longformer model also incorporates global attention. Similar to BigBird's global attention, a set of predefined positions in the input sequence attend to the entire sequence and all tokens in the sequence attend to said global tokens [2]. See Figure 3.8. for an illustration of the Longformer's attention mechanism. The memory complexity of the attention mechanism is $O(Nw)$ with w being the window size. The default window size of the Longformer is 512 tokens. For more information of the model's default configuration see here[2].

Beltagy et al. use TVM [5] to implement their own custom CUDA kernel in order to realize dilated window attention, as pervasive deep learning frameworks do not support the banded matrix multiplications required in this case [2]. Due to said custom kernel, the Longformer model, however, cannot be trained on a TPU accelerator. The authors provide a pytorch implementation which can be used on a TPU, but does not support dilation and requires twice the amount of memory. See here[3] for more information.

The authors further conduct experiments in autoregressive language modeling. In autoregressive language modeling the model is used to estimate the probability distribution of the respective token or character occurring at a particular position given the

[2]https://huggingface.co/transformers/v2.11.0/model_doc/longformer.html
[3]https://github.com/allenai/longformer



.

Figure 3.8.: Illustration of the pattern created by the Longformer's attention mechanism compared to one of full self-attention (taken from [2]).

preceding tokens or characters of the sequence [2]. Beltagy et al. increase the window size from lower to higher layers and refrain from using dilation in lower layers. In fact, dilation is only used in two attention heads and only in higher layers. The authors perform several experiments in character-level language modeling [2]. Autoregressive language modeling is, however, of lesser relevance to this project.

Similar to BigBird, the Longformer model is pretrained on masked language modeling starting from publically available roBERTa checkpoints. The authors merely make minor adjustments to support the Longformer's attention mechanism [2]. Said attention mechanism can, however, be used with any pretrained language model to extend it for longer sequences without adapting its core architecture [2]. In this case, the roBERTa's absolute postion embeddings are adapted to accommodate 4,096 rather than 512 tokens by copying them multiple times. The Longformer is available as a base as well as a large version. Beltagy et al. use the fairseq toolkit [37] to pretrain the Longformer on the BookCorpus [59], English Wikipedia, as well as the Realnews [57], and Stories [50] corpora.

3.2.4. Longformer Encoder-Decoder

Apart from the Longformer, an encoder-only model, Beltagy et al. further introduce the Longformer Encoder-Decoder (LED) model for sequence-to-sequence tasks such as summarization or translation [2]. This project only considers downstream tasks that focus on the encoder part of the Transformer. The LED model, however, is able to process up to 16,384 tokens at a time and is thus also examined. Nonetheless, solely the LED's encoder is used during the experiments.

The authors initialize the LED's weights from BART checkpoints and further maintain

BART's architecture. BART is a sequence-to-sequence model that was introcuded by Lewis et al. in 2019 [28]. BART is designed as a denoising autoencoder that incorporates a biderectional encoder as well as an autoregressive decoder. The model is pretrained using corrupted documents as inputs and the respective original documents as outputs [28]. The model is thus trained to reconstruct the original documents. Lewis et al. use the original Transformer architecture introduced by Vaswani et al. [51] while replacing its ReLU activation with a GeLU function [16]. The encoder and decoder are comprised of six and twelve layers in the base and the large versions respectively [28]. BART allows any document corruption during pretraining. The authors, therefore, use the following methods [28]:

- **Token masking**: Randomly chosen tokens are replaced with an artificially introduced token leaving the model to fill the gaps.
- **Token deletion**: Tokens are deleted from the sequence, leaving the model to decide where gaps that need to be filled exist.
- **Text Infilling**: Not merely tokens, but entire text spans are masked, leaving the model to also decide how many tokens have been removed.
- **Sentence Permutation**: The document's sentences are randomly shuffled.
- **Document Rotation**: The document is rotated along the axis of a randomly chosen token. The corrupted document consequently starts with that token.

Similar to the Longformer, the position embeddings of the LED are intialized by copying the ones of BART multiple times [2]. The encoder part of the LED, which is used during this project, incorporates local attention with a window size of 1,024 tokens. The artificially introduced <*s*> token marking the beginning of a document is used as a global token for computing global attention [2]. Unlike the Longformer model, the LED is merely initialized from BART's weights without further pretraining.

3.2.5. Reformer

The Reformer was introduced by Kitaev et al. in 2020. The Reformer model makes use of reversible residual networks (RevNet) which allow the model to store only one instance of the activations rather than having to store activations for every layer to be able to use back-propagation [21]. In RevNets any layer's activations can be restored from the ones of the following layer and the model's parameters [13]. Consequently, activations do not need to be stored, but can be inferred sequentially during the backward pass which reduces the model's memory requirements.

Apart from that, the authors process the activations of feed-forward networks in chunks. Kitaev et al. argue that these activations substantially contribute to the model's memory requirements, as the dimensions of feed-forward networks' intermediate layers are usually much larger than the ones of the attention activations. Chunking the feed-forward networks' activations thus further reduces the amount of memory needed [21]. These activations can be split into pieces because the computations of different positions are independent of each other in the layers of feed-forward networks. See section 3.3.1. for an explanation of feed-forward networks. While this independence would allow these activations to be computed in parallel, chunking and processing them sequentially does reduce memory requirements [21]. For models with a large vocabulary size the authors go one step further and also compute the loss based on the log-probabilities produced by the model in chunks [21]. In contrast to the model's activations, the amount of memory needed to store its parameters is not independent of the number of layers used in its architecture. In order to further recude memory requirements, Kitaev et al. therefore repeatedly swap parameters between the CPU and the particular hardware accelerator used [21].

To address the fundamental limitations of the Transformer, i.e. its quadratic complexity with regard to the input sequence's length, the authors use locality-sensitive hashing to approximate the attention matrix. The attention mechanism's outsized memory requirements result from the computation of the attention matrix, i.e., $softmax(\frac{QK^T}{\sqrt{d_k}})$, and in that mainly the computation of QK^T [21]. The authors point out that applying the softmax function implies that the attention matrix is dominated by the largest elements of QK^T. These largest elements result from the dot-product of the query and key vectors that are most similar to each other. Kitaev et al. note that the attention matrix can, consequently, be efficiently approximated by only computing the dot-product of those query and key vectors with the closest distance to each other. The Reformer uses locality sensitive hashing to determine the closest neighbors of each query vector. It is important to note that the authors use the same linear projection to create Q and K from the input sequence's embeddings. Queries and keys are therefore identical [21]. A hashing function that assigns vectors in close proximity to each other the same value, and different ones to those that are not, with high probability is called locality-sensitive [21]. Kitaev et al. implement locality-sensitive hashing with the same expected size for all hash-buckets by using the LSH scheme [1]. In a first step, a random matrix R of size $[d_k, b/2]$ is created with the aim to generate b hash values. The hash value of x is then defined as $h(x) = argmax([xR; -xR])$ with $[u; v]$ describing the concatination of vectors u and v [21]. See Figure 3.9. for an illustration of locality-sensitive hashing. The authors rewrite the attention computation for a single token at position i as follows:

$$o_i = \sum_{j \in P_i} exp(q_i * k_j - z(i, P_i))v_j \tag{3.8}$$

with $P_i = \{j : i \geq j\}$ denoting the set of positions that query i attends to, z describing the normalizing term in the softmax, and the scaling factor $\sqrt{d_k}$ being omitted for simplicity [21]. The attention computation, however, is usually performed for the larger set $\widetilde{P}_i = \{0, 1, ..., l\} \supseteq P_i$ as inputs are batched. The elements not in P_i are masked out during the computation [21]. This results in:

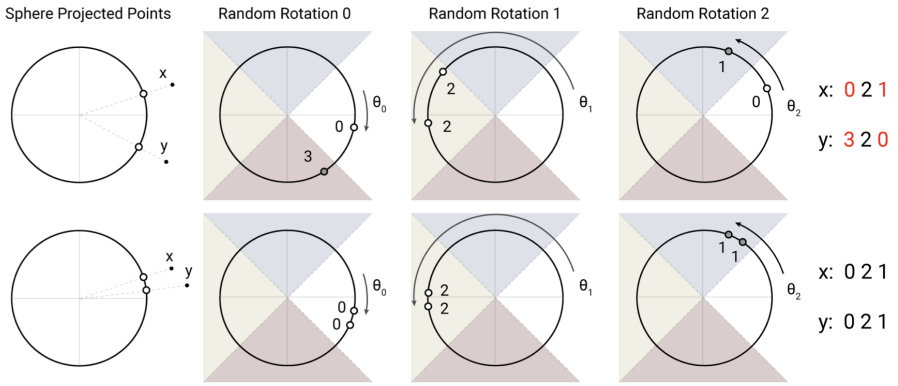
$$o_i = \sum_{j \in \widetilde{P}_i} exp(q_i * k_j - m(j, P_i) - z(i, P_i))v_j \tag{3.9}$$

where $m(j, P_i)$ is $\propto$ if $j \notin P_i$ and 0 otherwise. LSH attention then uses this reformulation to restrict the attention to be computed to those positions that are in the same hash-bucket, i.e., defining $P_i = \{j : h(q_i) = h(k_j)\}$. Kitaev et al. further set $k_j = \frac{q_j}{\|q_j\|}$ and thus $h(q_j) = h(k_j)$. This is necessary because buckets otherwise both tend to have different sizes and possibly contain an unequal number of query and key vectors [21]. Moreover, query vectors are sorted within each bucket according to their sequence position and overall by their bucket number. Query i will consequently be locates at position s_i. The sorted vectors are then chunked into blocks of size m. All vectors within one such chunk attend each other and are attended to by vectors of the same bucket located in the preceding chunk [21]. In the implementation of the Reformer m is set to $\frac{2N}{n_{buckets}}$ with N being the input sequence's length. The average size of one hash-bucket is thus $\frac{N}{n_{buckets}}$ [21]. See Figure 3.10. for an illustration of LSH attention. The authors note that the hashing operation can be performed multiple times to limit the probability of similar vectors being placed in different hash-buckets.

The memory complexity of the LSH attention mechanism is $O(N \log N)$ with N being the length of the input sequence [49].

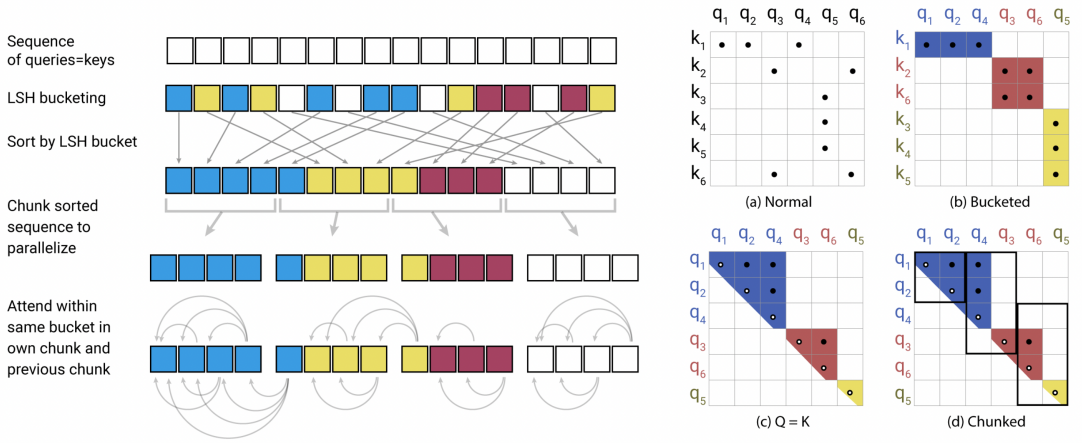
3.2.6. Linformer

The Linformer was introduced by Wang et al. in 2020 [53]. The key assumption of the Linformer's attention mechanism is that the attention matrix is low-rank [49]. The authors apply the base and large versions of the pretrained roBERTa model [31] on masked language modeling and sequence classification using the Wiki103 [33] and IMDb [32] datasets respectively. Wang et al. then apply singular value decomposition on the attention matrices of different layers as well as different heads and plot the



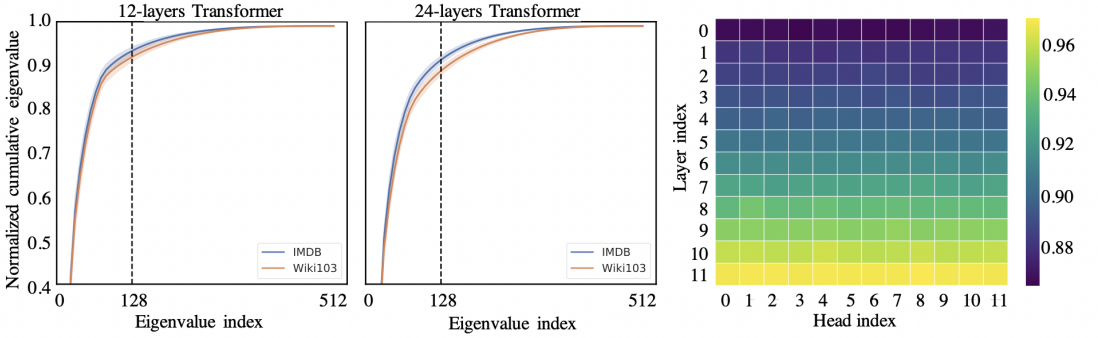
.

Figure 3.9.: Simplified illustration of angular locality sensitive hashing employing random rotations of spherically projected points to determine buckets with an argmax over signed axes projections. There is a low probability of points x and y being assigned to the same bucket when their respective spherical projections are not close to each other (taken from [21]).



.

Figure 3.10.: Illustration of the Reformer's attention mechanism, i.e., the hash-bucketing, sorting, chunking, the resulting causal attentions as well as the corresponding attention matrices (taken from [21]).



.

Figure 3.11.: The first two plots show the spectrum analyses of the attention matrices in pretrained roBERTa models [31] with $n = 512$, the y-axis showing the normalized cumulative singular value of the attention matrix, and the x-axis indicating the largest eigenvalue. The third plot shows a heatmap of normalized cumulative eigenvalues at the 128th largest eigenvalue of layers and heads using Wiki103 [33] data (taken from [53]).

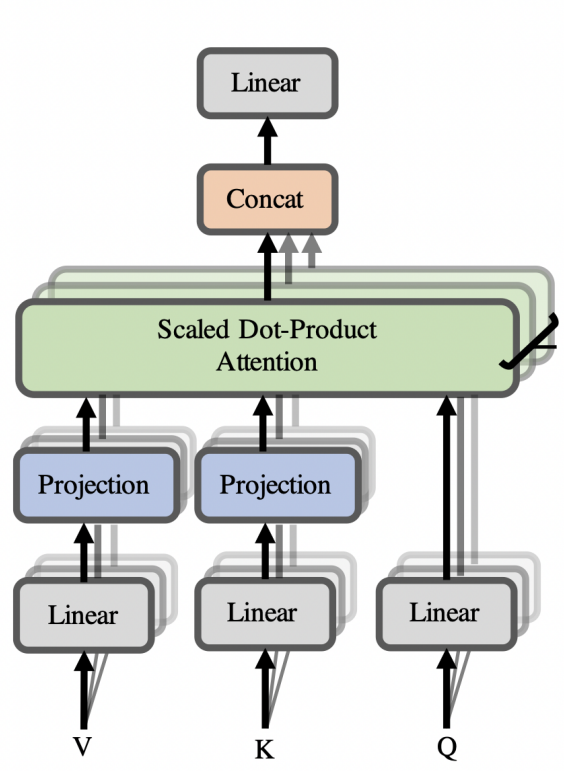
resulting normalized cumulative singular values averaged over 10k sentences [53]. See Figure 3.11. for an illustration.
The authors argue that the long-tail spectrum distribution in each layer clearly indicates that the attention matrices' information can be largely maintained by merely considering the first few largest singular values [53]. For a theoretical analysis of these spectrum results see [53].

The Linformer model incorporates two further linear projections $E_i, F_i \in \mathbb{R}^{nxk}$ when computing the keys and values [53]. The original keys and values with dimension $n\ x\ d$ are projected into $k\ x\ d$ dimensional ones. The Linformer then computes an $n\ x\ k$ attention matrix, termed $\overline{P}$ by the authors, using scaled-dot product attention. Lastly, the context embeddings for each attention head are computed by multiplying $\overline{P}$ with $F_i V W_i^V$. This results in:

$$head_i = Attention(QW_i^Q, E_i K W_i^K, F_i V W_i^V) \tag{3.10}$$

which can be rewritten to:

$$head_i = softmax(\frac{QW_i^Q (E_i K W_i^K)^T}{\sqrt{d_k}}) \cdot F_i V W_i^V \tag{3.11}$$



.

Figure 3.12.: Illustration of the Linformer's multi-head attention mechanism (taken from [53]).

See Figures 3.12. and 3.13. for an illustration of the Linformers attention mechanism. Figure 3.10. illustrates the relationship between the input sequence's length and the consequent inference time of multiple Linformer models.
Wang et al. argue that the Linformer significantly recudes time and space complexity if a projected dimension $k \ll n$ is chosen. The authors theoretically determine that choosing a $k = O(\frac{d}{\epsilon^2})$ independent of the sequence length yields and approximation of the original attention mechanism with ϵ error [53].

To further improve performance and efficiency, the Linformer shares parameters between projections [53]. The authors conduct experiments in which the projection matrices E and F are shared across all heads, head-wise sharing and sharing the same projection matrix for keys and values are used, and lastly layerwise sharing in which one projection matrix is shared among all layers of the model is deployed [53]. More-

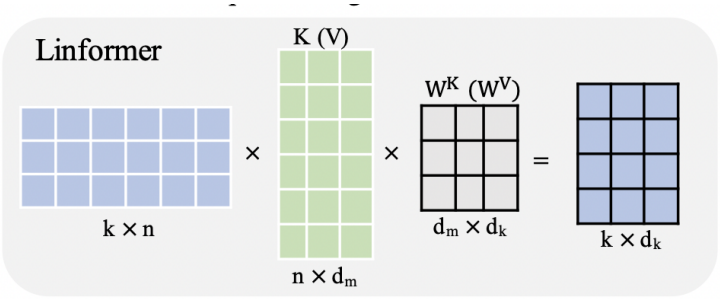


Figure 3.13.: Illustration of the Linformer's attention computation (taken from [53]).

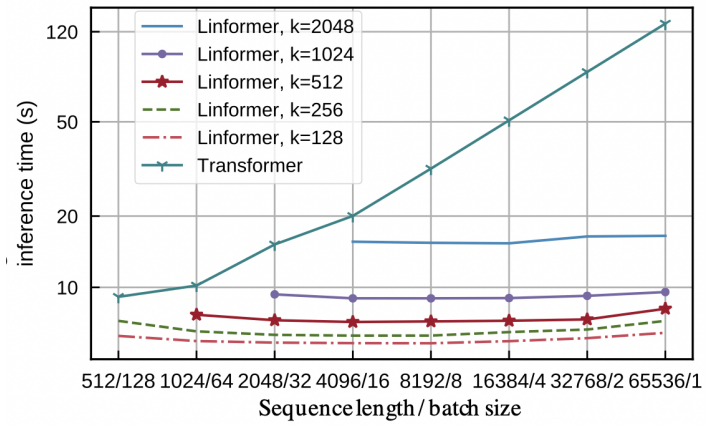
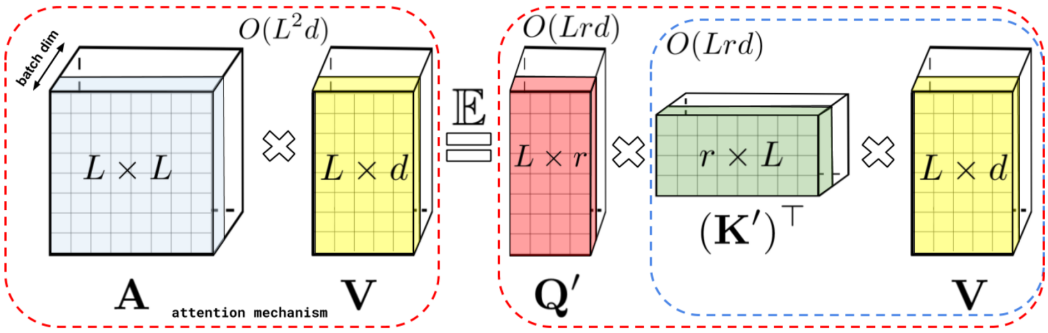


Figure 3.14.: Illustration of the inference time in relation to the input sequence's length for multiple Linformer architectures (taken from [53]).



.

Figure 3.15.: Illustration of the Performer's approximation of the attention matrix before renormalization with dashed lines indicating the order in which computations are performed (taken from [9]).

over, Wang et al. argue that one can choose a different projected dimension k across heads and layers and suggest a smaller dimension for higher layers. Apart from that, the authors suggest experimenting with various projection methods and name mean or max pooling as well as convolution as examples [53].

The Linformer model's complexity with regard to memory consumption is $O(N)$ [49].

3.2.7. Performer

The Performer was introduced by Choromanski et al. in 2020 [9]. The Performer model relies on a mechanism termed *Fast Attention Via positive Orthogonal Random features* (FAVOR+) to approximate the attention matrix. The authors define a Kernel K: $\mathbb{R}^d x \mathbb{R}^d \longrightarrow \mathbb{R}_+$ such that K(x,y) $= \mathbb{E}[\phi(x)^T \phi(y)]$ with $\phi : \mathbb{R}^d \longrightarrow \mathbb{R}_+$ and $\phi(u)$ termed a *random feature map* for some $r > 0$ and $u \in \mathbb{R}^d$. FAVOR+ is thus defined for a matrix $A \in \mathbb{R}^{LxL}$ as $A(i,j) = K(q_i^T, k_j^T)$ with q_i and k_j being the the *ith* and *jth* row vector of the query and key matrices respectively [9]. Choromanski et al. define their approximation of the attention matrix $\widehat{Att}_{\longleftrightarrow}$ thus as follows:

$$\widehat{Att}_{\longleftrightarrow}(Q,K,V) = \hat{D^{-1}}(Q'((K')^T V)) \tag{3.12}$$

for the query, key, and vector matrices (Q,K,V) with $\hat{D} = diag((Q'((K')^T 1_L)))$ and $Q', K' \in \mathbb{R}^{Lxr}$ and their rows defined as $\phi(q_i^T)^T$ and $\phi(k_i^T)^T$. See figure Figure 3.15. for an illustration.

The authors define the softmax-kernel describing the regular attention matrix as $SM(x,y) = exp(x^T y)$ omitting $\frac{1}{\sqrt{d}}$ as $x, y \in \mathbb{R}^d$ can also be renormalized [9]. Choromanski et al. further define the approximation of $SM(x,y)$ as follows:

$$SM(x,y) = \mathbb{E}_{\omega \sim \mathcal{N}(0,\mathbf{I}_d)}[exp(\omega^T x - \frac{\|x\|^2}{2}) exp(\omega^T y - \frac{\|y\|^2}{2})] \tag{3.13}$$

According to the authors, this can be reformulated to:

$$SM(x,y) = \Lambda \mathbb{E}_{\omega \sim \mathcal{N}(0,\mathbf{I}_d)} cosh(\omega^T z) \tag{3.14}$$

with Λ being defined as $exp(-\frac{\|x\|^2 + \|y\|^2}{2})$, z being $x + y$, and $cosh$ describing the hyperbolic cosine function. The variance of this estimator can be further reduced by ensuring that ω_1,...,ω_m are exactly orthogonal to each other [9]. See [9] for a theoretical analysis of the described mechanism.

Choromanski et al. argue that the attention mechanism used in the Performer provides an unbiased or nearly-unbiased estimator that also does not have any prior assumptions such as sparsity or low-rankness of the attention matrix. The authors further state that the Performer is compatible with existing pretrained language models and requires little further finetuning [9].

The Performer's complexity is $O(N)$ [49].

3.3. Classification Heads

3.3.1. Feed-forward Neural Network

The feed-forward neural network (FNN) was developed as a machine learning algorithm for data which cannot be separated linearly [22]. The architecture of such a neural network consists of an input layer, an output layer, as well as so-called hidden layers located between the two. The individual units, so-called neurons, of each of these layers are connected to the ones of the directly preceding and the directly succeeding layer (see Figure 3.16. for an illustration) [22]. The input only passes from one direction through the network. In the process, a number of computations using the weights associated with the connections between units as parameters are performed [22].

Firstly, M linear combinations are computed using the input variables x_i, weights w_{ij}, and biases w_{j0}, with $i \in [1, D]$ and $j \in [1, M]$ (equation 3.15) [3].

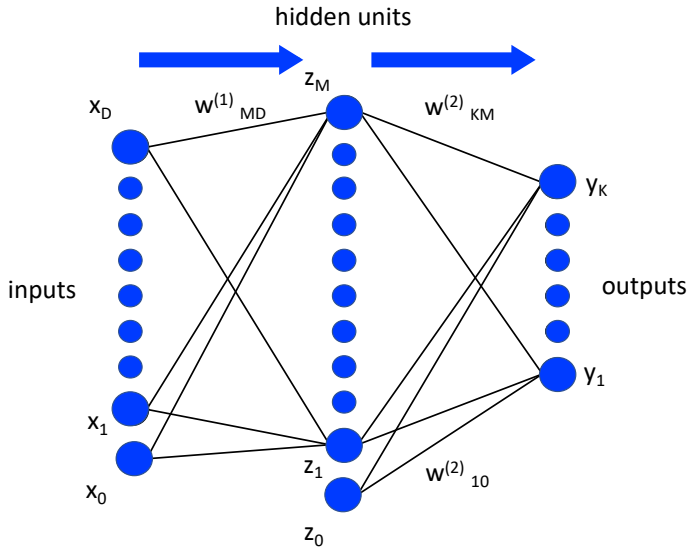


Figure 3.16.: Illustration of a feed-forward neural network (based on [3]).

$$a_j = \sum_{i=1}^{D} w_{ij}^{(1)} x_i + w_{j0}^{(1)} \tag{3.15}$$

Secondly, the intermediate values a_j, termed *activations*, are passed through a non-linear and differentiable activation function [3].

$$z_j = h(a_j) \tag{3.16}$$

The values z_j resulting from equation 3.16 serve as input for a further linear transformation (equation 3.17) [3]. The described computations are performed for every hidden layer of the particular model with the output of one layer serving as input for the following one [3]. The last layer produces the model's output activation units a_k, with $k \in [1, K]$ and K being the number of output values [3].

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{3.17}$$

For classification problems, these output activation units are passed through an activation function like the sigmoid resulting in the output values y_k [3].

$$y_k = sigmoid(a_k) \tag{3.18}$$

3.3.2. Convolutional Neural Network

Convolutional Neural Networks (CNN) are deep learning algorithms that have originally been primarily introduced for the processing of image data but found broad application in text classification as well [23]. The CNN's design takes advantage of the fact that an image's pixels are usually stronger correlated the closer they are to each other. Local features are derived from small sections of the image independently with the resulting information being merged at a later stage of the algorithm [3].
In each convolutional layer units form a number of planes, termed feature maps, which independently process small subsets of the input data. Each feature map possesses its own set of weights and biases. Consequently, one such feature map, serves as a filter on the input data [3].

The output values produced by convolutional layers are pooled which reduces the number of elements passed on to the next layer and thus also reduces the model's complexity [23]. For instance, one commonly used pooling technique is max pooling in which the largest value of the particular window is selected [23]. Moreover, the dimensionality of multiple stacked feature maps is reduced to one column by a flatten

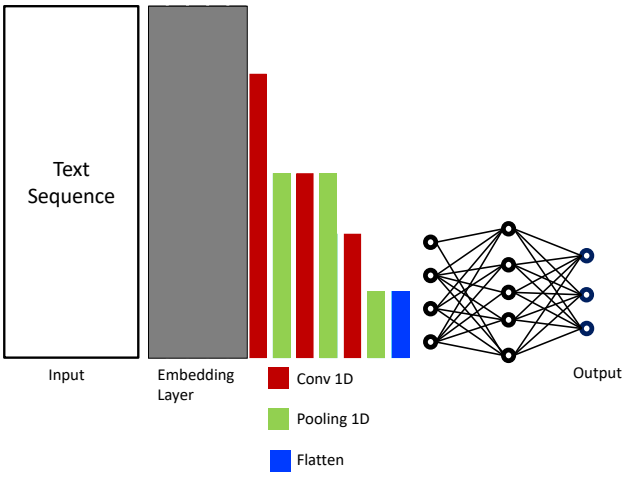


Figure 3.17.: Example of a CNN for text classification (based on [23]).

layer in order to allow further processing by the following layers [23].

The model's convolutional architecture is usually followed by fully connected layers such as those of the feed-forward network. For a multi-class classification problem the final layer's output is passed through a softmax nonlinearity. [3].

Figure 3.17. shows an example of a deep learning model architecture using convolutional layers for text classification.

4. Corpora

The BigPatent corpus [44] is used as a benchmark dataset during this project. The corpus[1] is obtained from Huggingface[2], an open source provider of natural language processing technologies. The BigPatent corpus is too large to be processed in its entirety within the bounds of this project. Multiple corpora of various sizes are consequently sampled from the dataset for development and experimentation purposes.

4.1. BigPatent Corpus

The BigPatent corpus was introduced by Sharma et al. in 2019 as a benchmark for abstractive summarization models [44]. The corpus consists of 1.3 million records of U.S. patent documents along with summaries written by a human. The patents are classified according to their subject matter following the Cooperative Patent Classification (CPC)[3] code. The documents are consequently classified according to the following nine categories:

- **A: Human Necessities**
- **B: Performing Operations; Transporting**
- **C: Chemistry; Metallurgy**
- **D: Textiles; Paper**
- **E: Fixed Constructions**
- **F: Mechanical Engineering; Lightning; Heating; Weapons; Blasting**
- **G: Physics**
- **H: Electricity**
- **Y: General tagging of new or cross-sectional technology**

[1] https://huggingface.co/datasets/big_patent
[2] https://huggingface.co/
[3] https://www.cooperativepatentclassification.org/home

Corpus	Training Size	Validation Size	Test Size
Small	1,250	54	81
Medium	12,150	540	810
Large	36,000	1,800	2,700

Table 4.1.: BigPatent sample corpora sizes.

4.2. Sample Corpora

For this project corpora of three sizes, each consisting of training, validation, and test splits, were sampled from the BigPatent corpus for development and experimentation purposes. While the medium and large corpora are used for several experiments, the small one merely serves prototyping and debugging purposes. The resulting sample corpora are balanced datasets in which every class is represented with an equal number of examples.

4.2.1. Medium Sample

Table 4.2. and Figure 4.1. show the distribution of the number of tokens among the patent documents of the medium sized corpus' train split yielded by the Longformer tokenizer. The average document consists of 3,768.8 tokens. The number of tokens, however, ranges from as little as 304 to as many as 157,819. The distribution within individual classes can differ significantly. The average document in class *c*, for instance, possesses 6,413.45 tokens while the average of class *e* is 2,953.06. See Figure 4.3. for an illustration of the distribution of the number of tokens per document within different classes in the train split.

Table 4.3. and Figure 4.2. show the distribution of the number of tokens among the patent documents within the test split. The average number of tokens per input sequence in the test split is 3,698.7. See Figure 4.4. for an illustration of the distribution of the number of tokens per document within different classes in the test split.

See appendix A.1. for an overview of the distribution resulting from the tokenizer that is used with the Reformer model during this project.

Category	Mean	Median	Mininum	Maximum
a	3,851.93	2,757	336	43,548
b	3,127.33	2,467	484	27,239
c	6,413.45	4,198.5	399	157,819
d	3,135	2,575.5	460	27,126
e	2,953.06	2,463	499	30,272
f	3,004.58	2,413	304	20,405
g	4,223.33	3,302.5	553	29,966
h	3,613.54	2,886.5	577	31,242
y	3,597.06	2,816.5	459	34,964
overall	3,768.8	2,786	304	157,819

Table 4.2.: Distribution of the number of tokens (Longformer tokenizer) of the medium sized sample corpus (train split).

Category	Mean	Median	Mininum	Maximum
a	4,271.36	2,731.5	667	35,959
b	2,910	2,492	593	8,329
c	5,304.61	3,658.5	697	26,092
d	3,046.88	2,684.5	470	12,512
e	2,650.69	2,222	936	7,080
f	2,815.33	2,394.5	938	15,513
g	4,441.28	3,723	933	17,607
h	3,899.14	2,940	831	16,185
y	3,949.02	3,229	681	13,087
overall	3,698.7	2,735	470	35,959

Table 4.3.: Distribution of the number of tokens (Longformer tokenizer) of the medium sized sample corpus (test split).

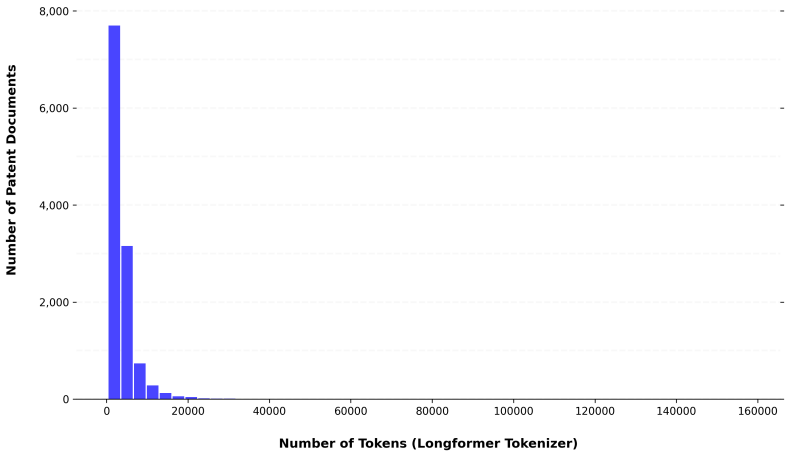


Figure 4.1.: Distribution of the number of tokens (Longformer tokenizer) of the medium sized sample corpus (train split).

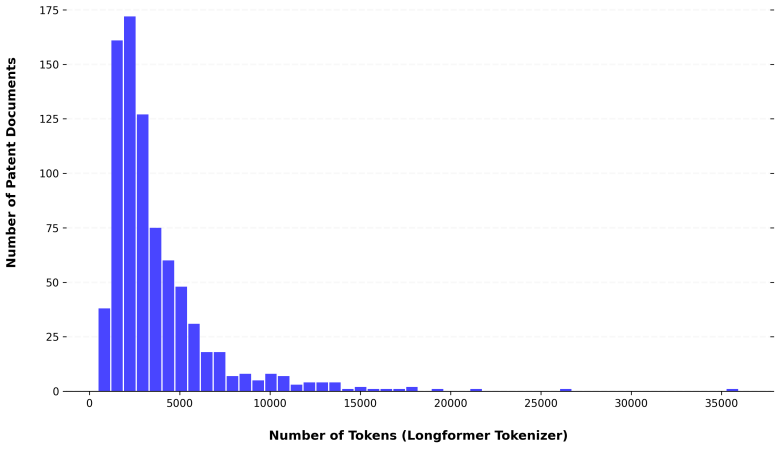


Figure 4.2.: Distribution of the number of tokens (Longformer tokenizer) of the medium sized sample corpus (test split).

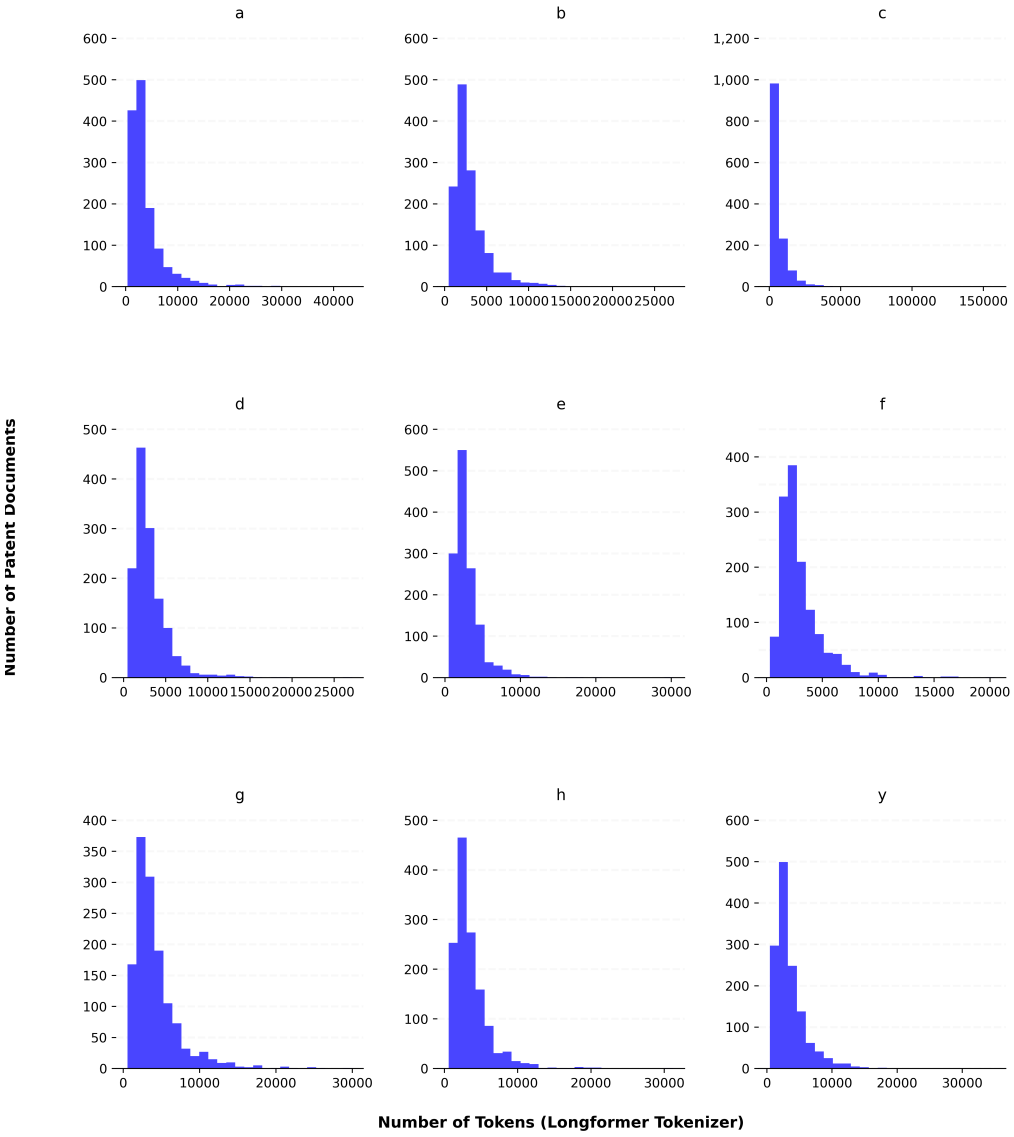


Figure 4.3.: Distribution of the number of tokens (Longformer tokenizer) of the medium sized sample corpus (train split) grouped by category.

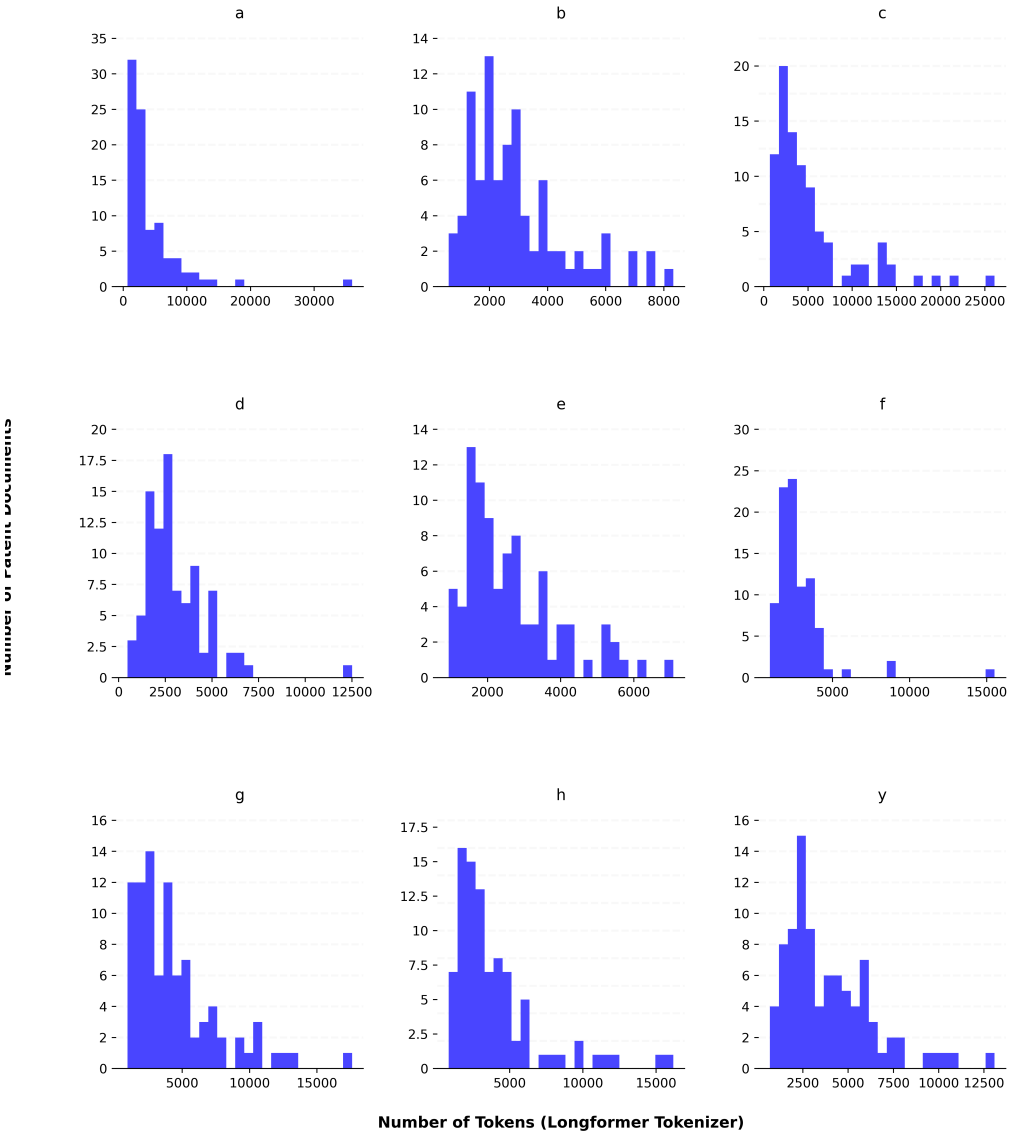


Figure 4.4.: Distribution of the number of tokens (Longformer tokenizer) of the medium sized sample corpus (test split) grouped by category.

Category	Mean	Median	Mininum	Maximum
a	4,084.23	2888	370	71,186
b	3,144.22	2556	375	63,053
c	6,261.42	4,059.5	399	74,282
d	3,209.56	2,582	287	35,937
e	2,980.91	2,458	303	25,551
f	2,913.53	2,395	483	27,779
g	4,250.13	3,264	553	52,287
h	3,829.83	3,059.5	346	49,546
y	3,662.96	2,801.5	449	43,181
overall	3,815.2	2,814	287	74,282

Table 4.4.: Distribution of the number of tokens (Longformer tokenizer) of the large sample corpus (train split).

4.2.2. Large Sample

The average length of the patents in the large sample's training set resulting from the use of the Longformer tokenizer is 3,815.2 tokens. Similar to the medium sized sample, the number of tokens can vary significantly between classes. The average length of a document in class *c* is 6,261.42 tokens, while the one of class *f* is 2,913.53. See Table 4.4. and Figure 4.5. for an overview. Figure 4.7. illustrates the distribution of the input sequence's length across different classes.

The average number of tokens per input sequence in the test split is 3,845.27. Table 4.5. and Figure 4.6. display the distribution of the number of tokens among the patent documents within the test split. See Figure 4.8. for an illustration of the distribution of the number of tokens per document within different classes in the test split.

See appendix A.2. for an overview of the distribution resulting from the tokenizer that is used with the Reformer model during this project.

Category	Mean	Median	Mininum	Maximum
a	4,022.34	2,857.5	623	31,353
b	3,080.83	2,479.5	674	13,506
c	6,955.37	4,338	660	53,640
d	3,045.03	2,485.5	440	14,558
e	3,186.90	2,463	653	21,276
f	2,889.45	2,377	611	14,818
g	4,353.25	3,268.5	377	56,928
h	3,617.12	2,976.5	773	13,201
y	3,457.11	2,851.5	534	15,898
overall	3,845.27	2,810.5	377	56,928

Table 4.5.: Distribution of the number of tokens (Longformer tokenizer) of the large sample corpus (test split).

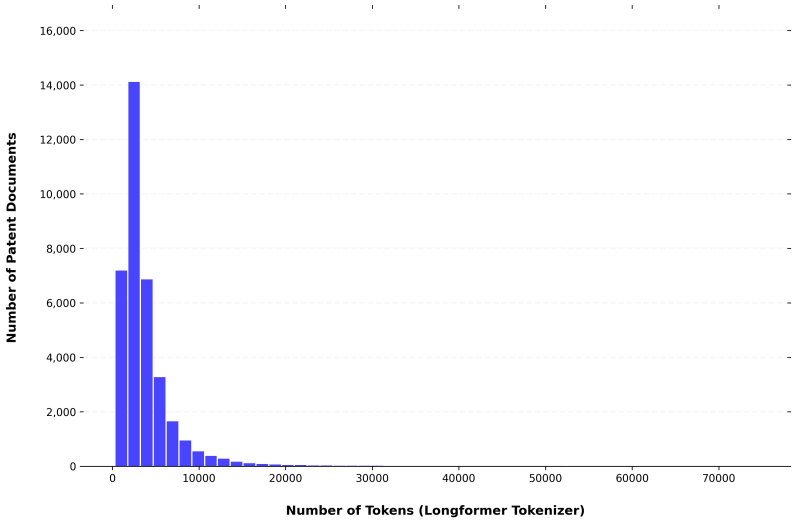


Figure 4.5.: Distribution of the number of tokens (Longformer tokenizer) of the large sample corpus (train split).

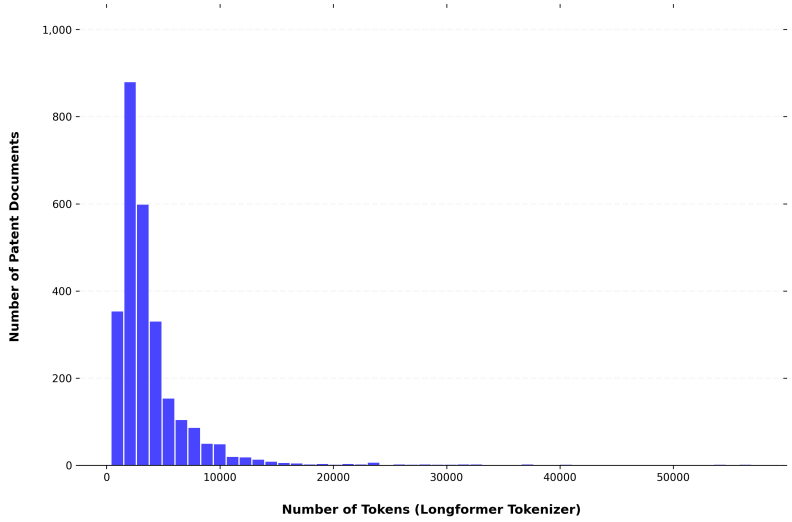


Figure 4.6.: Distribution of the number of tokens (Longformer tokenizer) of the large sample corpus (test split).

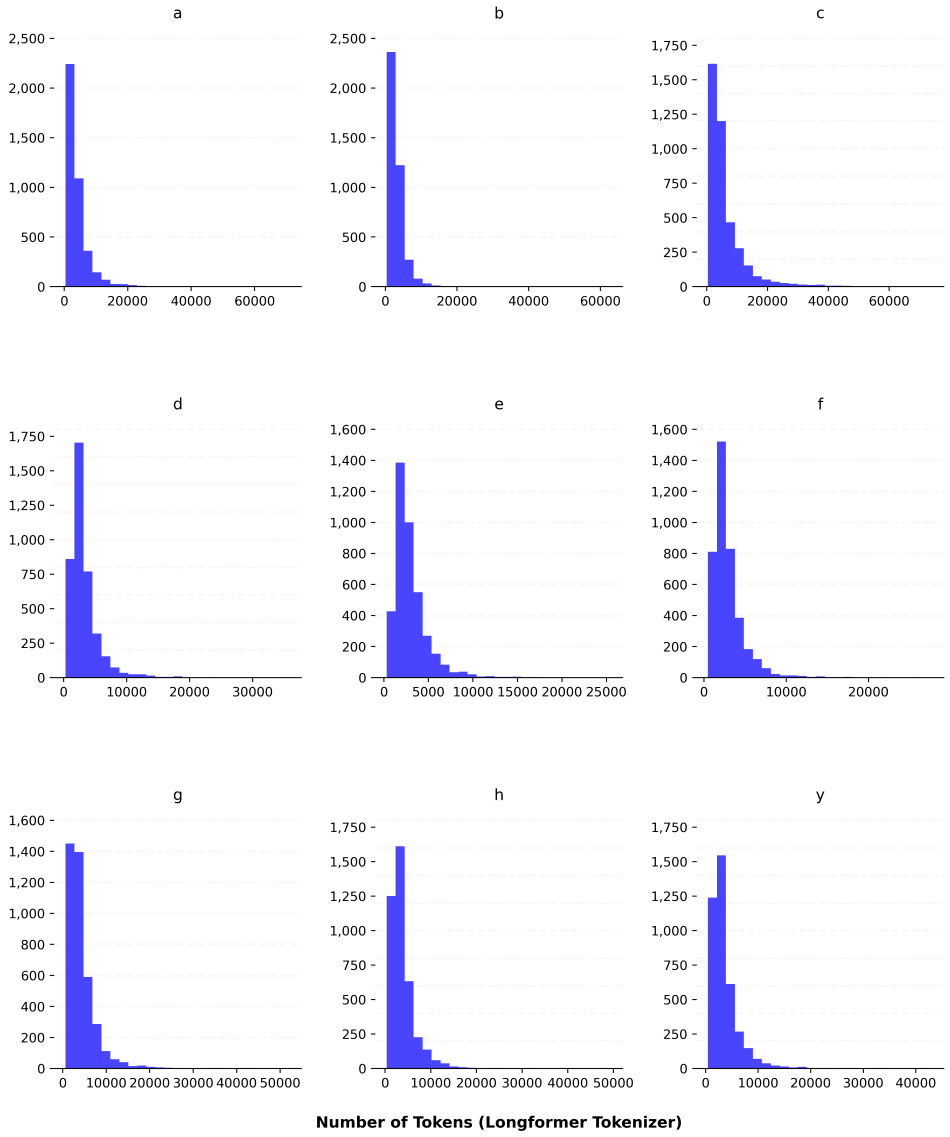


Figure 4.7.: Distribution of the number of tokens (Longformer tokenizer) of the large sample corpus (train split) grouped by category.

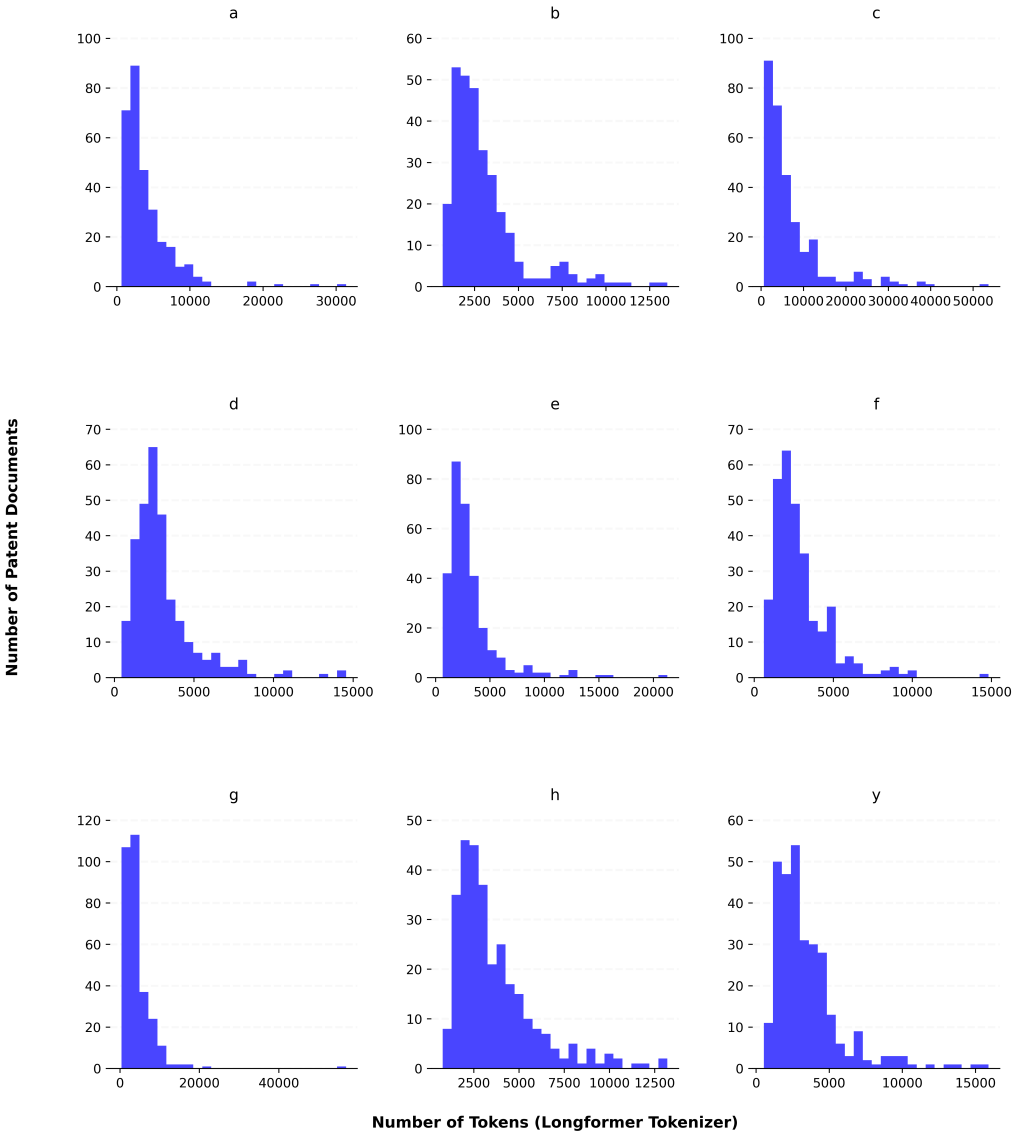


Figure 4.8.: Distribution of the number of tokens (Longformer tokenizer) of the large sample corpus (test split) grouped by category.

5. Methodology

5.1. Research Question 1: Which methods and models to encode long text sequences are most suited for downstream machine learning tasks?

The goal of this project's first phase is to evaluate the quality of the encodings produced by efficient Transformer models with regard to downstream tasks. The question, consequently, is whether the encodings produced by these models retain enough of the information codified in the original sequences, so that another machine learning model can be successfully trained on downstream tasks with these encodings as inputs. This project examines whether the models' encodings of patent documents retain sufficient information for a separate machine learning model to learn to infer the patent's subject matter. The question, however, is not how well the individual language models can incorporate information about a patent's subject matter during training. The weights of the particular language model are, thus, frozen during training. Hence, only language models for which pretrained checkpoints are available, i.e., BigBird, Longformer, Longformer Encoder-Decoder, and Reformer, are considered. Each efficient Transformer model is, therefore, paired with a classification head and trained on classifying patents using the medium sized sample of the BigPatent corpus. The models' performance is evaluated using the *accuracy* metric which is defined as the share of correctly classified examples in the dataset [27]:

$$accuracy = \frac{(TP + TN)}{(TP + FP + FN + TN)} \tag{5.1}$$

with TP, TN, FN, FP describing the number of true positives, true negatives, false negatives, and false positives respectively. Accuracy is an appropriate criterion in this case, as the dataset is balanced and all classes are represented with an equal number of examples. Yet, the models' performance on the test set in absolute terms is of lesser importance, as their results are evaluated relative to each other. All models are thus trained using the same classification head and hyperparameters to ensure comparability.

A convolutional neural network (CNN) head is used for a variety of reasons. As the

goal is to examine the amount of information retained in the produced encodings in general, the models' last hidden states, i.e., the embeddings for each individual token and not the condensed output representing the entire sequence, are used as input. Apart from that, not all models considered produce such a pooled output representing the entire sequence. The embeddings of individual tokens would need to be reduced to one column, in order for a feed-forward network head to be able to process them. This would, however, increase the number of parameters of the head's fully connected layers disproportionately. Alternatively, a recurrent neural network such as a Long Short-Term Memory (LSTM) could be used (see [23] for more information). However, due to the recurrent nature of its computations, the models' training time increases significantly. This renders LSTM heads impractical for the purposes of this project. A CNN head creates the possibility to train a model using the embeddings of each individual token as inputs in an admissible time frame.

For all experiments in this first phase the base versions of the language models' checkpoints are used. The models are obtained from Huggingface[1]. The Longformer and LED models are trained using the Tensorflow[2] framework, while Pytorch[3] is used to train the BigBird and Reformer models. See Figure 5.1. for an overview of the methodology followed in the project's first phase.

Experiment Q1-1: Investigating the amount of information retained in the models' embeddings

In a first step, roBERTa and BART are evaluated on the given classification task to serve as baselines for BigBird, Longformer, and Longformer Encoder-Decoder respectively. To accommodate roBERTa's limitations with regard to the input sequence's length, the model is trained both with truncated as well as with chunked and averaged encodings. In the same manner, the input sequences are truncated before being encoded by BART.

In a second step, the efficient Transformer language models are trained and evaluated with a classification head and a maximum sequence length of 4,096 tokens. The sequences are either truncated or padded to that maximum sequence. Although the LED and Reformer models can encode documents with a sequence length of up to 16,384 and 524,288 tokens respectively, limiting the sequence length allows a comparison to the results obtained from BigBird and Longformer without obfuscating the results by allowing more context to enter the training of the classification task.

[1] https://huggingface.co/
[2] https://www.tensorflow.org
[3] https://pytorch.org

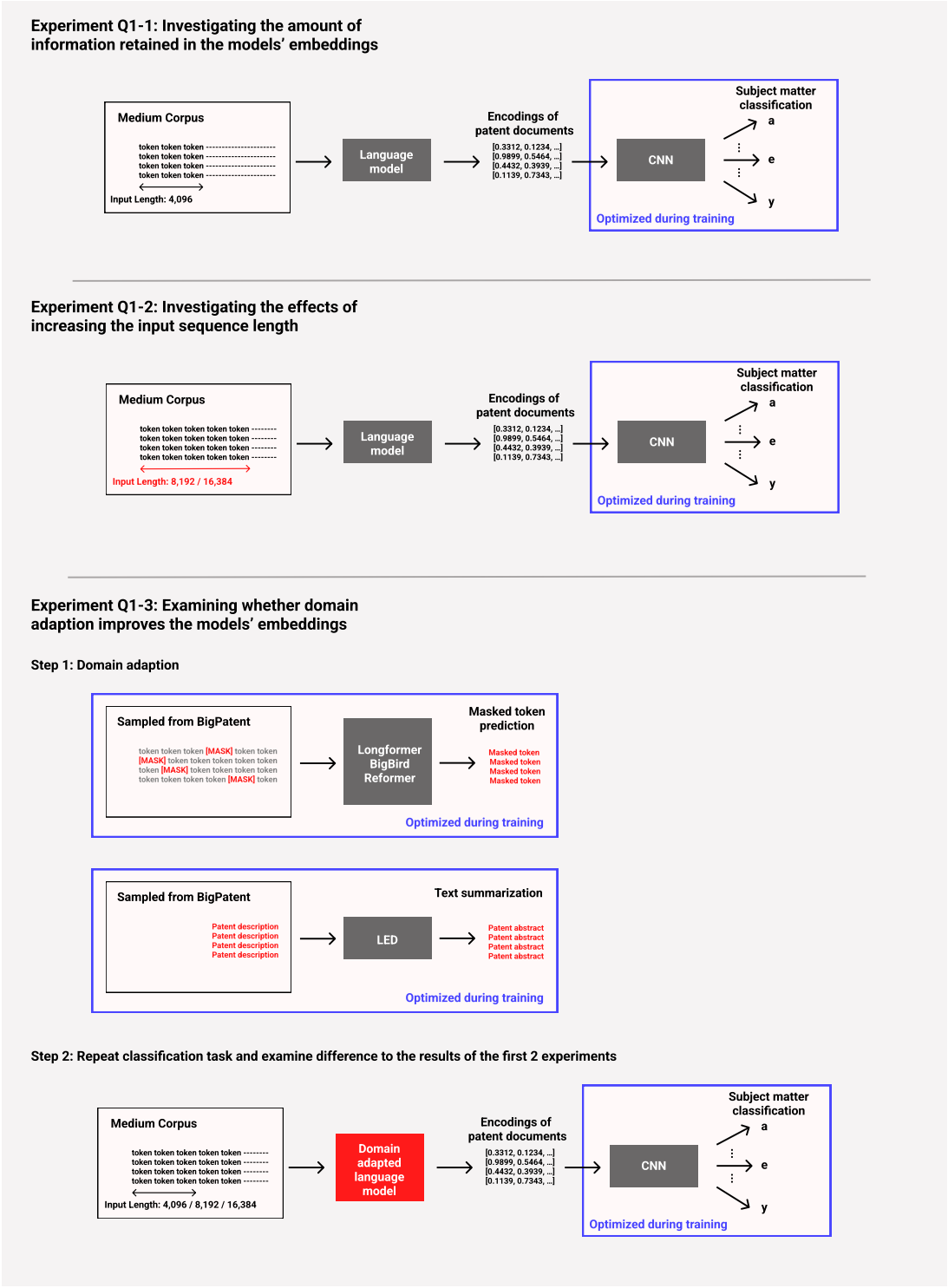


Figure 5.1.: Methodology of the experiments conducted to investigate question 1.

Experiment Q1-2: Investigating the effects of increasing the input sequence length

To examine the effects of increasing the length of the input sequence, models using the LED's and Reformer's encodings up to 8,192 and 16,384 tokens are trained.

Experiment Q1-3: Examining whether domain adaption improves the models' embeddings

The domain of the patent data used during this project can differ vastly from the data used to pretrain the considered models. This can lead a model to produce rather poor representations of domain specific words not present in the copora used during pretraining. All models are, therefore, domain adapted by further pretraining them on samples of the BigPatent corpus in order to evaluate to what degree the quality of their embeddings can be improved indepentently from a specific downstream task. The Longformer, BigBird, and Reformer models are further pretrained on masked token prediction with a masking probability of 15% on a training set of 60k documents. Due to resource limitations, mostly time constraints, the Longformer and BigBird cannot be trained using all 60k documents at once. Thus, 20k patent documents are repeatedly sampled for three training runs. Consequently, there is, however, a limited probability of a patent being sampled multiple times. These documents are then separated into chunks of 512 tokens in order to reduce memory requirements. The LED is domain adapted by sampling 30k documents from BigPatent's train split and training the model on text summarization using a document's description and abstract. To reduce memory requirements, the encoder's input and the decoder's output are limited to 8,192 and 512 tokens respectively. The further pretrained models' encodings are then used to train classification models in the manner as described above.

The checkpoint[4] available for the Reformer language model was pretrained using an English translation of Fyodor Dostoevsky's novel *Crime and Punishment*. While the resulting language model demonstrates the Reformer's ability to process up to 524,288 tokens at once, it has limited ability to encode patent documents. Since the model's vocabulary size is limited to 320, a large number of tokens are unknown and the corresponding tokens are uniformly marked as such. The model's tokenizer is, therefore, extended using the entire BigPatent's train split to increase the model's vocabulary size to 52k. Moreover, the model's architecture consisting of six hidden layers, two attention heads, and a dimensionality of 512 for the feed-forward networks of the Reformer blocks is rather elementary. The architecture is, therefore, adapted to twelve hidden layers, twelve attention heads, and a feed-forward size of 1,024. This,

[4] https://huggingface.co/google/reformer-crime-and-punishment

however, results in a large number of newly initialized parameters. In addition to the domain adaption process described above, the resulting Reformer model is, therefore, further pretrained on masked token prediction with a masking probability of 15% and 120k further examples of the BigPatent corpus sampled in two tranches, 50k examples of Wikipedia English[5] and 100k examples of the Arxiv[6] scientific papers corpus [10].

Hardware

The majority of models are trained on a 16GB GPU for the described experiments. The LED-base model receiving sequences of 16,384 tokens, however, is trained on a 40GB GPU.

5.2. Question 2: How does adapting a model's attention mechanism to accommodate longer sequences effect performance on downstream machine learning tasks?

In the second phase the models are evaluated according to their ability to incorporate information while being finetuned on a downstream task. Unlike the first research question, this section is rather concerned with the models' ability to absorb information about a patent document's subject matter during training and adapt the respective encodings accordingly. The models' weights are, therefore, not frozen during training. In order to facilitate comparability, all models are newly initialized without having received any pretraining for the first two experiments. The BigBird, Longformer, Performer, Linformer, and Reformer attention mechanisms are evaluated in the described manner. A CNN classification head is used for all experiments in phase two. See Figure 5.2. and Figure 5.3 for an overview of the methodology followed in the project's second phase.

Experiment Q2-1: Investigating the various attention mechanisms' ability to absorb information during training on a downstream task

The pytorch-performer[7] and linformer[8] python packages are used to create Performer and Linformer based language models respectively. Note, however, that these packages were not implemented by the authors of the original models themselves. See here[9]

[5] https://huggingface.co/datasets/wikipedia
[6] https://huggingface.co/datasets/arxiv_dataset
[7] https://pypi.org/project/performer-pytorch/
[8] https://pypi.org/project/linformer/
[9] https://github.com/lucidrains/performer-pytorch

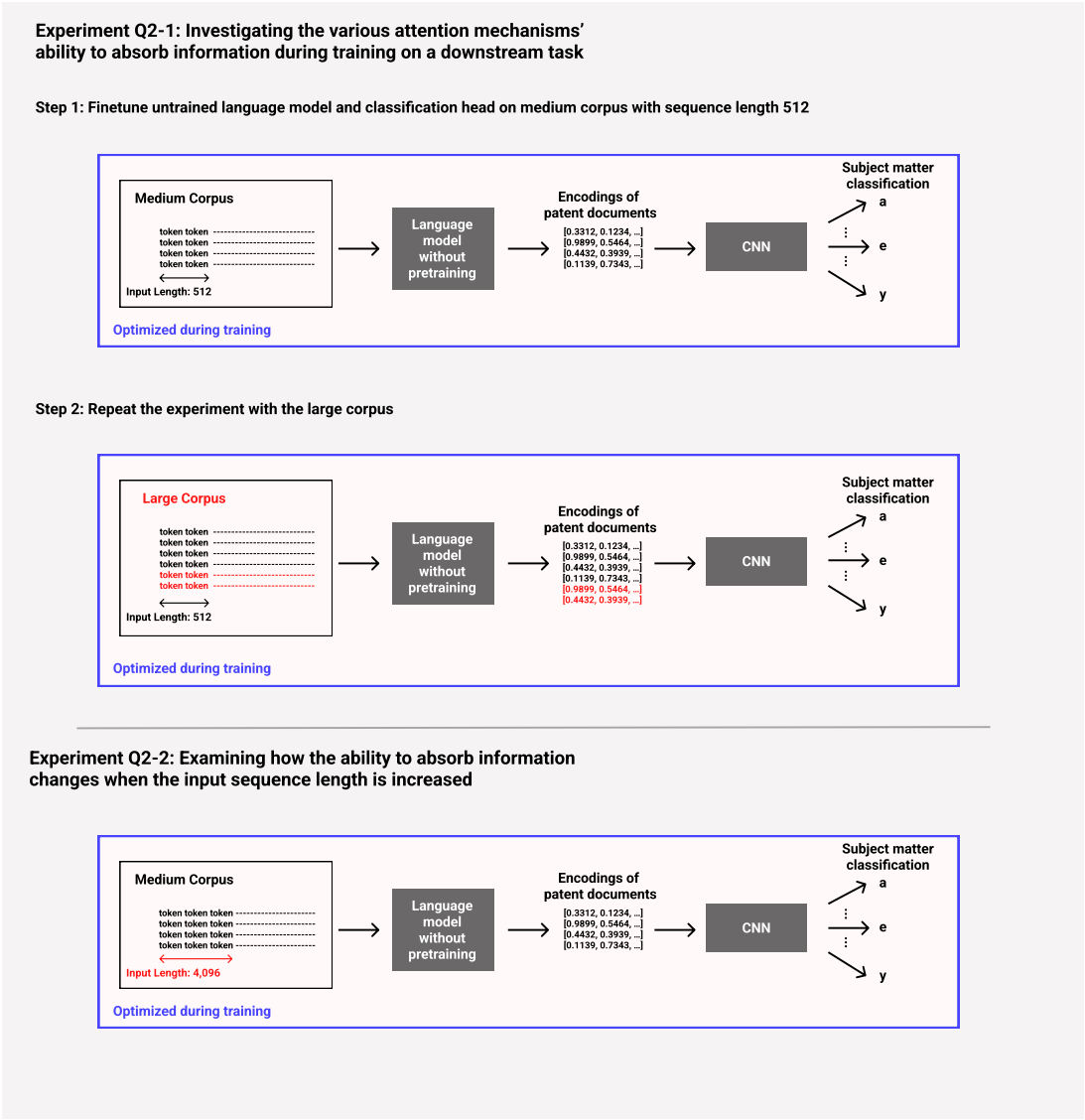


Figure 5.2.: Methodology of the experiments conducted to investigate question 2 (Q2-1 & Q2-2).

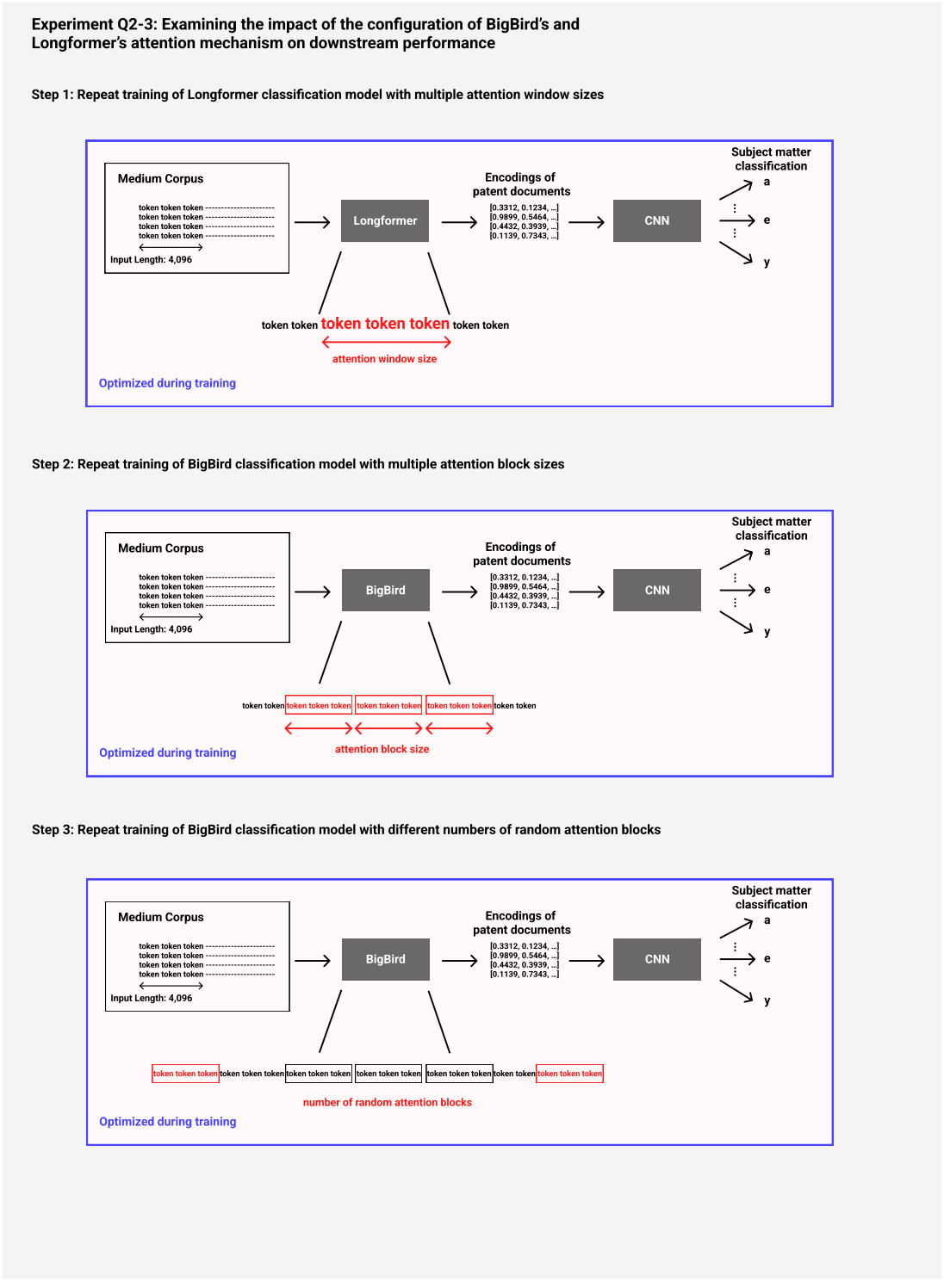


Figure 5.3.: Methodology of the experiments conducted to investigate question 2 (Q2-3).

for the code base of the pytorch-performer package and here[10] for the one of the linformer. To exclude the possibility of different model architectures influencing the results, all language models are initialized using the roBERTa-base architecture, i.e., twelve hidden layers, twelve attention heads, and a dimension of the attention heads of 64. The roBERTa-base model itself is used to create a baseline, in order to be able to compare the various efficient attention mechanisms to the original full self-attention. The maximum sequence length is set to 512 tokens. Consequently, the Longformer's attention window size and BigBird's attention block size are also reduced by a factor of eight to 64 and eight respectively to ensure comparability. The performer-pytorch and linformer packages used in this project do not allow to individually adjust the dimensions of the feed-forward layers used in the attention heads. Said dimensions are automatically set to the dimensions of the attention layer. The roBERTa, Longformer, and BigBird models are thus also trained with a feed-forward dimension of 768 instead of the default size of 3,072. The experiment is conducted both with the medium as well as the large sample corpus to examine how increasing the amount of training data influences the results.

Experiment Q2-2: Examining how the ability to absorb information changes when the input sequence length is increased

In the next step, the maximum sequence length is increased to 4,096 to compare the ability of the investigated efficient attention mechanisms to incorporate information when being finetuned with longer sequences.

Experiment Q2-3: Examining the impact of the configuration of BigBird's and Longformer's attention mechanism on downstream performance

In this experiment the degree to which various decisions made in the architecture of the Longformer's and BigBird's attention mechanism influence their ability to absorb information about a patent's subject matter during training is investigated. Firstly, a pretrained Longformer-base model is finetuned on the given classification task using varying window sizes in order to examine the relationship between the length of the full self-attention window and downstream performance. Moreover, a pretrained BigBird-base model is trained with varying attention block sizes and multiple numbers of random attention blocks.

[10] https://github.com/lucidrains/linformer

Hardware

All models are trained using 16GB GPUs.

5.3. Question 3: Which classification model is most appropriate for patent subject matter classification?

The project's last phase examines which combination of language model and classification head is most suited for classifying patent documents according to their subject matter. The goal is to obtain the best possible test accuracy on the large sample corpus. See Figure 5.4. for an overview of the methodology followed in the project's last phase.

Experiment Q3-1: Testing various classification heads

The base versions of the Longformer and BigBird are finetuned on the given classification task using both the CNN as well as an FNN head. The medium sized corpus is used for the described experiment. The results obtained are used to inform the decisions about which hyperparameters and classification head are used with the large model versions in the subsequent experiment.

Experiment Q3-2: Optimizing performance on the test set of the large corpus

In a first step, the large versions of BigBird, Longformer, and Longformer Encoder-Decoder are domain adapted on the BigPatent corpus following the same procedure described in phase one. For the former two models 24k patents are sampled in three tranches and the LED-large models is domain adapted on 30k patents. Lastly, the large domain-adapted models as well as a Reformer model are finetuned using the previously chosen classification heads and hyperparameters, as well as the large corpus to determine the most performant way to classify patent documents according to their subject matter.

Hardware

The large model versions are trained using 80GB GPUs, while the base versions are finetuned on 16GB GPUs.

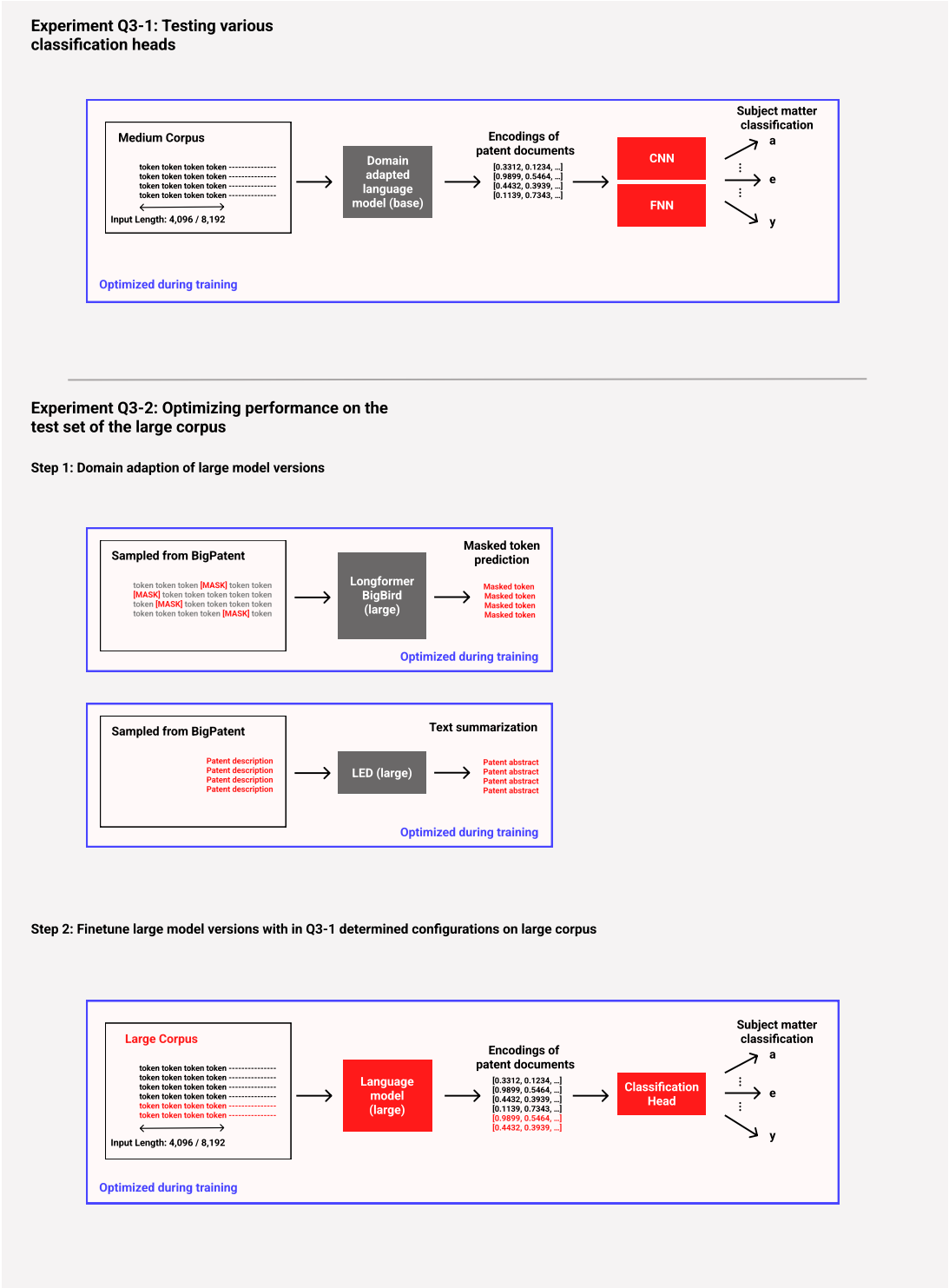


Figure 5.4.: Methodology of the experiments conducted to investigate question 3.

6. Experiments

6.1. Research Question 1: Which methods and models to encode long text sequences are most suited for downstream machine learning tasks?

The goal of this project's first phase is to evaluate the quality of the encodings produced by efficient Transformer models with regard to downstream tasks. The following experiments examine how much information about the original patent are retained by the encodings produced by various language models. A variety of Reformer-based language models were used throughout the project. Table 6.1. shows the various Reformer architectures as well as the data they were pretrained with.

Experiment Q1-1: Investigating the amount of information retained in the models' embeddings

The first experiment evaluates the degree to which a classification model can learn to infer a patent's subject matter from the encodings produced by the examined efficient Transformer language models. All models are trained for ten epochs with with batch size one, a learning rate of 2e-5 and a CNN classification head. See Figure 6.1. for an illustration of the used classification head. The batch size of one is used to conserve memory resources and allow for the experiments to be conducted with a 16GB GPU. The exact hyperparameters, such as the learning rate, are of lesser importance during the first experiments as long as the same configurations are used for all models, to ensure that the obtained results are comparable. Since the weights of the respective language model are frozen during training, the models are trained for ten epochs to allow them to fit the training data to a sufficient degree. See Table 6.2. for a summary of the experiment's results.

The roBERTa and BART classification models fit the training data less well than their efficient transformer counterparts (BigBird and Longformer for the former and LED for the latter). They do, however, yield competitive results or even outperform their counterparts. Of note is that chunking and averaging the roBERTa encodings produces

significantly worse results than merely truncating the input sequence does. The reason for this may lie in the length of the input sequences. Longer sequences result in a greater number of chunks. Averaging the respective encodings could potentially obscure a large amount of information.
The best results of the efficient Transformer models can be obtained with the BigBird and LED models with a performance of 53.58% and 53.7% respectively. Using the Longformer encodings yields a comparably low accuracy of 48.02%. Of note is, however, that the classification model which receives the Longformer encodings fits the training data nearly perfectly. The models receiving BigBird and LED encodings on the other hand merely yield a training accuracy of 81.41% and 75.47% respectively. There appears to be a clear trade-off relationship between the degree to which the models are able to fit to the training data and the results obtained on the test data. This is also evident in the performance of the baseline models. See Figure 6.2. for an illustration of the described trade-off.
While the Reformer-1 classification model is able to fit the training data to a high degree (95.8%), it performs worse on the test set (26.3%). The Reformer-1 model, however, incorporates a rather simple architecture and has received little pretraining compared to the residual models.

It appears that the encodings produced by efficient Transformer models do retain more information of the original patent documents than the ones of their full self-attention counterparts, as they are able to process a larger portion of the original input sequence at once. This allows the classification models to fit the training data better. That, however, seems to have diametrical effects on their performance on the test data in this particular case.

Experiment Q1-2: Investigating the effects of increasing the input sequence length

The second experiment is concerned with investigating the quality improvements of the embeddings produced by efficient Transformers when the context length is increased. Table 6.3. shows the corresponding results. Just as in the first experiment, all models are trained for ten epochs, with a batch size of one, a learning rate of 2e-5 and a CNN classification head.

Increasing the input sequence length from 4,096 to 8,192 tokens increases the LED classification model's performance by 1.61 percentage points to 55.31%. The corresponding accuracy on the training data, however, is 3.26 percentage points lower. The described performance gain might, therefore, not necessarily stem from the increase in the context length, but could also result from the model not overfitting to the training set as much.

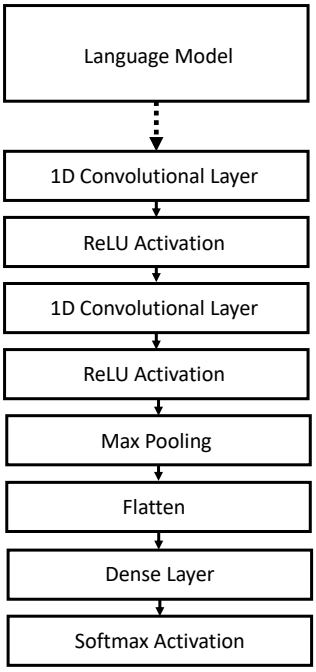


Figure 6.1.: Illustration of the CNN classification head used throughout the project.

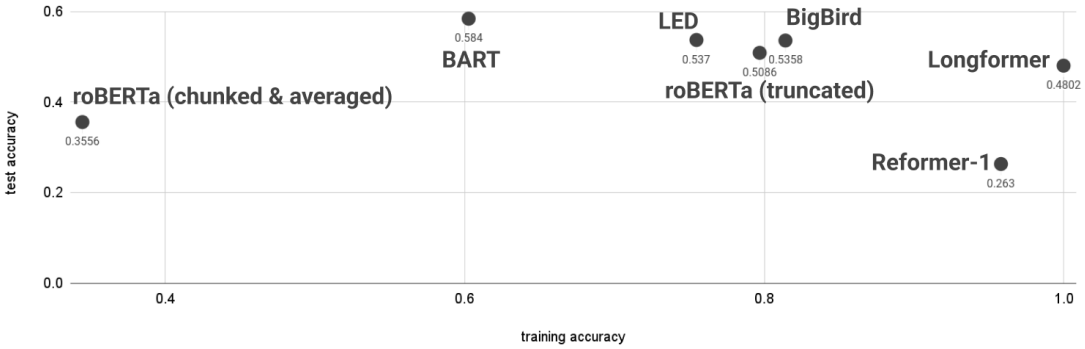


Figure 6.2.: Results of Experiment Q1-1: Illustration of the relationship between training and test accuracy in experiment Q1-1.

Reformer Model	Number of Hidden Layers	Number of Attention Heads	Feed-forward size	Vocab Size	Pretraining
Reformer-1	6	2	512	320	Crime and Punishment
Reformer-2	6	2	512	320	Crime and Punishment, 60k patent documents
Reformer-3	6	2	512	320	Crime and Punishment, 180k patent documents, 100k scientific papers, 50k Wikipedia English articles
Reformer-4	12	12	1,024	52,000	Crime and Punishment, 60k patent documents
Reformer-5	12	12	1,024	52,000	Crime and Punishment, 180k patent documents, 100k scientific papers, 50k Wikipedia English articles

Table 6.1.: Architecture and pretraining of the evaluated Reformer language models.

Language Model	Input Length	Training Accuracy	Test Accuracy
roBERTa-base	512 (truncated)	0.7969	0.5086
roBERTa-base	512 (chunked & averaged)	0.3447	0.3556
BART-base	1,024	0.6026	**0.584**
BigBird-base	4,096	0.8141	0.5358
Longformer-base	4,096	**0.9998**	0.4802
LED-base	4,096	0.7547	0.537
Reformer-1	4,096	0.958	0.263

Table 6.2.: Results of Experiment Q1-1: Comparing the degree to which a machine learning model can learn to infer a patent's subject matter from the encodings produced by efficient transformer language models (all models are trained for 10 epochs with batch size 1, learning rate 2e-5 and a CNN classification head).

Language Model	Input Length	Training Accuracy	Test Accuracy
LED-base	8,192	0.7215	**0.5531**
LED-base	16,384	0.7517	0.5432
Reformer-1	8,192	**0.975**	0.2765
Reformer-1	16,384	0.9616	0.2802

Table 6.3.: Results of Experiment Q1-2: Investigating the quality improvement of the embeddings produced by efficient transformers when the context length is increased (all models are trained for 10 epochs with batch size 1, learning rate 2e-5 and a CNN classification head).

Further increasing the length of the input sequences to 16,384 tokens yields a lower test accuracy of 54.32% with a higher training accuracy of 75.17%.
Increasing the sequence length from 4,096 to 8,192 and 16,384 tokens improves the Reformer-1 model's performance from 26.3% to 27.65% and 28.02% respectively.

The average sequence length of the medium sample's test split is 3,698.7. There are notable exceptions. For instance, the average length in class *c* is 5,304.61 and the dataset's longest document consists of 35,959 tokens. Increasing the number of tokens to be considered in the classification is, however, expected to have marginally diminishing returns in terms of performance improvements.

Experiment Q1-3: Examining whether domain adaption improves the models' embeddings

The considered language models are domain adapted by further pretraining them on samples of the BigPatent corpus, in order to evaluate to what degree the quality of their embeddings can be improved. Table 6.4. shows the corresponding results yielded by these models on the given classification task.
Domain adapting BigBird increases the accuracy obtained on the training set, but has no substantial impact on the test accuracy.
While the model receiving the encodings produced by the Longformer still learns to classify the training data nearly perfectly, domain adaption appears to improve the model's ability to generalize these learnings to previously unseen data. The corresponding test accuracy improves from 48.02% to 54.57%.
The domain adapted LED classification model receiving sequences with a length of up to 8,192 tokens achieves a higher accuracy on both the training as well as the test data. The former increases from 72.15% to 76.07% and the latter from 55.31% to 56.42%.
Domain adaption causes the model trained on input sequences up to 16,384 tokens long to learn the training data far better. The training accuracy improves from 75.17% to 93.18%. This, however, fails to have any significant effect on the model's performance on the test set.
Domain adapting the Reformer-2 model increases the test accuracy for sequences with a length of up to 4,096 tokens by 1.48 percentage points. Further pretraining the model using other data sources only has marginal effects and actually leads the test accuracy to decrease slightly. The Reformer-4 and Reformer-5 fail to learn the training data (11.64% and 11.47% accuracy respectively) and thus also produce poor results on the test set. Since the vocabulary size and the architecture of these models have been significantly increased, the models would require a substantial amount of further pretraining to produce results competitive to those of the other models examined.

In general, domain adaption appears to allow the models to better generalize the relationship between a patent's encodings and its subject matter learned during training. It can, thus, be used to mitigate the adverse effects of the models overfitting to the training data.

Language Model	Input Length	Training Accuracy	Test Accuracy
BigBird-base (da)	4,096	0.8488	0.532
Longformer-base (da)	4,096	**0.9997**	0.5457
LED-base (da)	8,192	0.7607	**0.5642**
LED-base (da)	16,384	0.9318	0.5418
Reformer-2	4,096	0.9728	0.2778
Reformer-3	4,096	0.9781	0.2741
Reformer-4	4,096	0.1163	0.1160
Reformer-5	4,096	0.1147	0.1049

Table 6.4.: Results of Experiment Q1-3: Investigating the effects of domain adaption on the embeddings produced by efficient transformers (all models are trained for 10 epochs with batch size 1, learning rate 2e-5 and a CNN classification head), da = domain adapted.

6.2. Question 2: How does adapting a model's attention mechanism to accommodate longer sequences effect performance on downstream machine learning tasks?

The project's second phase evaluates how well the different attention mechanisms adapt the embeddings they produce while being finetuned on a downstream task. Moreover, the implications of changing the configuration of BigBird's and Longformer's attention mechanism are examined.

Experiment Q2-1: Investigating the various attention mechanisms' ability to absorb information during training on a downstream task

The first experiment is designed to compare the ability of different attention mechanisms to incorporate information about a patent's subject matter during training. In order to isolate their ability to adapt their encodings during training, all models are initialized without pretraining. The models are finetuned on the given classification task for ten epochs, a batch size of one, a learning rate of 6.25e-07 and a CNN classification head. The batch size of one is again chosen to preserve memory resources. The lower learning rate of 6.25e-07 is used, since the gradient is passed back to the respective language model during training. The maximum sequence length is limited to 512 tokens to be able to compare the results to the ones of the full self-attention mechanism of the roBERTa model. Table 6.5. shows the experiment's results. The dimensions of

the feed-forward networks is changed from 3,072 to 768 in the roBERTa, BigBird, and Longformer models. Table 6.6. thus shows the results of the same experiment with the original feed-forward size for reference purposes.

The untrained roBERTa model (feed-forward size of 768) which serves as a baseline here yields a training accuracy of 33.84% and a test accuracy of 26.42% on the medium sized corpus. The Longformer model with a feed-forward size of 768 merely achieves an accuracy of 15.1% and 16.17% on the training and test set respectively. The ones obtained with a feed-forward size of 3,072 are 39.42% and 27.41%. At least for such a small window size (64 tokens), the feed-forward size appears to be of significant consequence for the Longformer's dilated window attention mechanism.
Despite its rather small window size, the BigBird classification model outperforms roBERTa's full self-attention mechanism both in terms of training (40.28%) as well as test accuracy (37.53%).
In addition to that, the Performer's training results (35.07%) are comparable to the ones of roBERTa. However, it yields a significantly higher accuracy on the test set (32.84%).
The Linformer and Reformer models incorporate information about a patent's subject matter far quicker during training than the roBERTa model does. Both models also yield far better results on the test data. Their training and test accuracy is 78.69% and 57.33% as well as 36.05% and 47.41% respectively.

The performance gap between the untrained roBERTa and the residual models is less severe when the large corpus is used. The margin between the performance of the Reformer and the roBERTa model shrinks from 20.99 to 7.71 percentage points. The Longformer with a feed-forward size of 768 also yields the worst performance in this case. See Table 6.7. for an overview of the results obtained by using the large corpus.

Apart from the one of the Longformer model, all alternative attention mechanisms examined are able to adapt their encodings during training far quicker than the original full self-attention mechanism of the roBERTa model. The difference in performance, however, is less severe when a larger training corpus is used. Moreover, the Performer, Linformer, and Reformer models which perform an approximated attention mechanism for all tokens, rather than limiting the number of full attention operations that need to be computed, also outperform the Longformer and BigBird models in both cases. These models may, thus, be better suited for downstream tasks with comparably small amounts of data. They might, however, be more prone to overfitting than a model using full self-attention.

Language Model	Training Accuracy	Test Accuracy
roBERTa-base	0.3384	0.2642
Longformer-base	0.151	0.1617
BigBird-base	0.4038	0.3753
Performer	0.3507	0.3284
Linformer	**0.7869**	0.3605
Reformer	0.5733	**0.4741**

Table 6.5.: Results of Experiment Q2-1 with medium sized corpus: To investigate their ability to adapt their encodings during a classification task, the models are initialized without pretraining and finetuned with a maximum sequence length of 512 tokens (all models are trained for 10 epochs with batch size 1, learning rate 6.25e-07 and a CNN classification head).

Language Model	Training Accuracy	Test Accuracy
roBERTa-base	0.3814	0.2765
Longformer-base	0.3942	0.2741
BigBird-base	**0.4072**	**0.3889**

Table 6.6.: Results of Experiment Q2-1 with medium sized corpus: The models are initialized without pretraining (with a feed-forward size of 3,072 for reference) and finetuned during the classification task with a maximum sequence length of 512 tokens (all models are trained for 10 epochs with batch size 1, learning rate 6.25e-07 and a CNN classification head).

Language Model	Training Accuracy	Test Accuracy
roBERTa-base	0.468	0.4296
Longformer-base	0.3355	0.3111
BigBird-base	0.4593	0.4459
Performer	0.5441	0.4785
Linformer	**0.6693**	0.5052
Reformer	0.5848	**0.5067**

Table 6.7.: Results of Experiment Q2-1 with large corpus: To investigate their ability to adapt their encodings during a classification task, the models are initialized without pretraining (with a feed-forward size of 768) and finetuned with a maximum sequence length of 512 tokens (all models are trained for 10 epochs with batch size 1, learning rate 6.25e-07 and a CNN classification head).

Language Model	Training Accuracy	Test Accuracy
roBERTa-base	**0.5006**	**0.4596**
Longformer-base	0.3871	0.3596
BigBird-base	0.4051	0.4033

Table 6.8.: Results of Experiment Q2-1 with large corpus: The models are initialized without pretraining (with a feed-forward size of 3,072 for reference) and finetuned during the classification task with a maximum sequence length of 512 tokens (all models are trained for 10 epochs with batch size 1, learning rate 6.25e-07 and a CNN classification head).

Language Model	Training Accuracy	Test Accuracy
Longformer-base	0.3935	0.3037
BigBird-base	0.4072	0.3889
Performer	0.6398	0.4272
Linformer	**0.7928**	0.3704
Reformer	0.6013	**0.479**

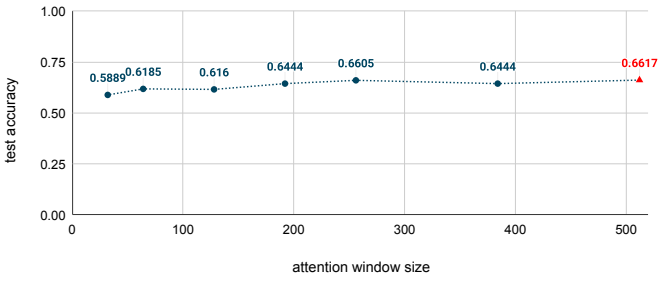
Table 6.9.: Results of Experiment Q2-2: To investigate their ability to adapt their encodings during a classification task, the models are initialized without pretraining (with a feed-forward size of 768) and fine-tuned during the classification task with a maximum sequence length of 4,096 tokens (all models are trained for 10 epochs with batch size 1, learning rate 6.25e-07 and a CNN classification head).

Experiment Q2-2: Examining how the ability to absorb information changes when the input sequence length is increased

In order to examine the ability of different attention mechanisms to adapt their encodings during a classification task with longer sequences, the length of the inputs is increased to 4,096 tokens. Table 6.9. shows the corresponding results obtained with the medium sized corpus.

The Longformer model underperforms the residual models and achieves a test accuracy that is 8.52 and 17.53 points lower than the ones of the BigBird and Reformer respectively. The former incorporates an attention mechanism that follows a similar approach to the Longformer's and the latter model is the best performing one. While the Linformer model achieves the highest training accuracy (79.28%), its performance on the test set is 1.85 points lower than the one of the BigBird model which achieves an accuracy of mereley 40.72% on the training data.

The Performer, Linformer, and Reformer models appear to also incorporate information about a patent's subject matter much faster during training than the Longformer and BigBird when the sequence length is increased to 4,096. The models' ability to learn the relationship between a patent document and its subject does not necessary translate into a higher performance on previously unseen data. However, the Reformer classification model produces the best results both for a sequence length of 512 as well as 4,096 tokens.



Window Size	Accuracy
512 (default)	**0.6617**
384	0.6444
256	0.6605
192	0.6444
128	0.6160
64	0.6185
32	0.5889

Figure 6.3.: Results of Experiment Q2-3 (Longformer): The Longformer-base (no pretraining) classification model is finetuned on the medium sized corpus with varying sizes of the attention window (all models are trained for 3 epochs with batch size 1, learning rate 1.25e-06 and a CNN classification head).

Experiment Q2-3: Examining the impact of the configuration of BigBird's and Longformer's attention mechanism on downstream performance

Figure 6.3. shows the test accuracy obtained by finetuning a pretrained Longformer classification model on the medium corpus with varying sizes of the sliding window used in its attention mechanism. The models are trained for three epochs using a learning rate of 1.25e-06. As the goal is not to optimize performance on the test set, but rather compare the models' performance, training for three epochs is sufficient. The default window size of 512 tokens yields a test accuracy of 66.17%. Of note is that the model's performance does not monotonically decrease as the window size is lowered. The accuracy obtained by a model using a window size of 256 tokens is 1.61 percentage points higher than the accuracy of one using 384 tokens and only slightly lower than the one of the default model.

Table 6.10. shows the test accuracy obtained by finetuning a pretrained BigBird classification model on the medium corpus with varying attention block sizes and different numbers of random attention blocks. All models are trained for three epochs using a learning rate of 1.25e-06. Increasing the size of the attention blocks to 128 tokens or decreasing it to 32 tokens appears to have only marginal effects on the model's performance on the given task. The former yields an increase in accuracy of 1.01 percentage points and the latter merely decreases the obtained performance by 0.12 points. Increasing the number of random attention blocks to five leads to an increase in performance of 2.72 percentage points, while lowering it to one has no effect at all.

Attention Block Size	Accuracy
128	**0.6691**
64 (default)	0.6580
32	0.6568

Number of Random Attention Blocks	Accuracy
5	**0.6852**
3 (default)	0.6580
1	0.6580

Table 6.10.: Results of Experiment Q2-2 (BigBird): The BigBird-base (no pretraining) classification model is finetuned on the medium sized corpus with varying block sizes and number of random blocks used in its attention mechanism (all models are trained for 3 epochs with batch size 1, learning rate 1.25e-06 and a CNN classification head).

The experiment's results imply that one can potentially significantly lower the Longformer's and BigBird's memory and compute requirements by adjusting the particular attention window size, attention block size, and number of random attention blocks without substantially impairing their performance on the given classification task.

6.3. Question 3: Which classification model is most appropriate for patent subject matter classification?

The project's last phase examines which combination of language model, classification head, and hyperparameters achieves the highest performance on the given classification task.

Experiment Q3-1: Testing various classification heads

Table 6.11. shows the results obtained by finetuning the domain adapted base versions of the Longformer and BigBird models with the medium sized corpus and various classification heads. Figure 6.4. illustrates the feed-forward network head considered during this project.

The combination of the Longformer and the CNN classification head produces results that are 2.96 percentage points higher than the ones obtained by using the FNN head. For the BigBird model the CNN head also achieves marginally better results.

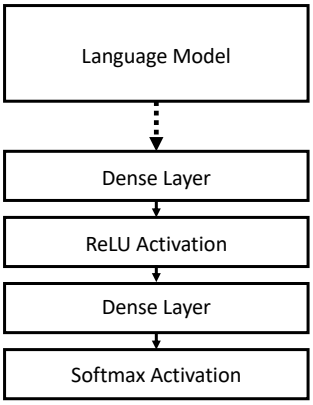


Figure 6.4.: Illustration of the FNN classification head used throughout the project.

Language Model	Classification Head	Test Accuracy
Longformer-base (da)	FNN	0.6519
Longformer-base (da)	CNN	**0.6815**
BigBird-base (da)	FNN	0.6901
BigBird-base (da)	CNN	**0.7**

Table 6.11.: Finetinuning various combinations of classification heads and language models on the medium sample corpus to optimize performance on the test set (all models are trained for 10 epochs with batch size 1, learning rate 1.25e-06).

Language Model	Input Length	Batch Size	Learning Rate	Number of Epochs	Test Accuracy
Longformer-large (da)	4,096	8	1e-05	5	0.7041
LED-base (da)	8,192	4	5e-06	5	0.7007
BigBird-large (da)	4,096	8	1e-05	5	**0.7102**
Reformer-5	16,384	4	5e-06	10	0.4

Table 6.12.: Finetuning the best performing combinations of pretrained language models and classification heads on the large sample corpus to optimize results on the test set.

Experiment Q3-2: Optimizing performance on the test set of the large corpus

Table 6.11. shows the results obtained by finetuning pretrained and domain adapted efficient transformer language models in combination with a CNN head on the large sample corpus.
Even the 80GB GPUs used in this experiment do not supply enough memory to train a classification model that incorporates the large LED model version. Attempting to train such a model leads to an exception both when limiting the input sequence length to 8,192 as well as 4,096 tokens. Consequently, the LED-base model is used in the remainder of the experiment. The sequence length is nonetheless limited to 8,192 tokens, as the corresponding models in previous experiments outperformed those using a sequence length of up to 16,384 tokens.
The Longformer-large, LED-base, and BigBird-large models yield comparable performances on the test data. The BigBird-large model obtains the highest accuracy of 71.02%. Using the large model checkpoints, therefore, only marginally improves the observed results. Classifying patents according to their subject matter appears to be a rather difficult task. One reason for this could be that the CPC scheme classifies patents in fairly broad categories. The category *F: Mechanical Engineering; Lightning; Heating; Weapons; Blasting* , for instance, includes a multitude of different topics and *Y: General tagging of new or cross-sectional technology* contains aspects of multiple categories by design.
Of interest is that the Reformer-5 model achieves a mere 40% test accuracy. The untrained Reformer model used in experiment Q2-1, thus, outperforms the Reformer-5 model by 10 percentage points. One reason for this might be that the former model consists of fewer parameters than the latter and thus requires less training to complete the classification task.

7. Conclusion and Future Work

Making a general assessment of the investigated models' ability to encode sequences is particularly difficult. Which models are most suited for downstream tasks cannot be conclusively answered. However, this project's results indicate that embeddings produced by long sequence Transformer models do retain more information of the original patent documents than the ones of their full self-attention counterparts. The corresponding classification models do fit the training data better. Yet, this appears to have a negative effect on their performance on previously unseen data. Consequently, they seem to be more prone to overfitting.
Moreover, increasing the input sequences' length does not automatically result in a higher accuracy on the test data.
Domain adapting the considered language models can possibly mitigate some of the adverse effects of overfitting to the training data. Further pretraining the individual language models on domain specific data can, therefore, improve the performance on downstream machine learning tasks.

The majority of the considered efficient Transformer models seem to be able to incorporate information about a data point's classification far quicker than the original full self-attention mechanism of the roBERTa model. The performance gap, however, is less severe as the size of the training corpus increases. Moreover, the Performer, Linformer, and Reformer models which perform an approximated attention mechanism for all tokens, rather than limiting the number of full attention operations, also incorporate such information faster than the Longformer and BigBird models. These models are, therefore, potentially better suited for downstream tasks with comparably small amounts of data. However, they might also be more prone to overfitting than a model using full self-attention. The project's results also imply that the Longformer's attention window size as well as BigBird's attention block size and number of random attention blocks are of lesser importance for their performance on downstream tasks. One can, therefore, potentially significantly lower the Longformer's and BigBird's memory and compute requirements by adjusting their configuration.

Using a CNN classification head rather than an FNN one yields a superior performance for all models considered. Classification models that incorporate the Longformer-large,

LED-base, and BigBird-large language models deliver comparable performances on the test split of the large sample corpus. No model achieves an accuracy that significantly exceeds 70%. Finetuning the respective Reformer model, however, yields rather poor results.

The project's findings only allow conclusions about the classification of patent documents according to their subject matter. The data within individual classes of other tasks maybe more homogeneous, resulting in a less severe trade-off between overfitting to the training data and accurately classifying unseen documents. The information distribution of documents of other corpora may also differ from the one of patents. Increasing the sequence length in these cases could potentially lead to a higher increase in performance than was observed here. Future work will, therefore, likely include testing the examined models on tasks involving data of other domains.
Moreover, the classification task set up for this project is focused on encoder architectures. Benchmark tasks evaluating decoder or encoder-decoder architectures for language modeling or sequence-to-sequence tasks will thus also be part of future experiments.
The project does show that newly introduced Transformer architectures can deliver competitive results to the original full-self attention mechanism. A comparison of their performance on downstream tasks will likely be easier, as more pretrained language models incorporating their approaches become available. Of particular interest, for instance, is the backwards compatibility of the Performer. This could allow pretraining with the benefits of full self-attention, while using the less resource intensive attention of the Performer on downstream tasks.
Yet, it is apparent that most of the examined models, nonetheless, require vast amounts of resources when processing longer sequences. All approaches followed with these models either limit the number of attention operations that need to be performed or approximate the results of full self-attention. None of them, therefore, question the merits of full self-attention itself. Of interest is consequently also whether model architectures for sequential data will emerge that follow a completely different approach.

A. Appendix

A.1. Medium Sample (extended Reformer tokenizer)

Category	Mean	Median	Mininum	Maximum
a	3,493.06	2,757	305	35,282
b	2,908.16	2,467	412	25,724
c	5,475.75	4,198.5	370	130,193
d	2,872.62	2,575.5	421	23,098
e	2,767.67	2,463	470	28,093
f	2,817.77	2,413	299	18,572
g	3,970.62	3,302.5	528	28,182
h	3,368.35	2,886.5	544	29,193
y	3,302.34	2,816.5	438	29,650
overall	3,441.82	2,589	299	130,193

Table A.1.: Distribution of the number of tokens (extended Reformer tokenizer) of the medium sized sample corpus (train split).

Category	Mean	Median	Mininum	Maximum
a	3,839.23	2,600	642	35,959
b	2,691.26	2,334.5	554	8,329
c	4,479.83	2,842	645	26,092
d	2,825.61	2,502.5	428	12,512
e	2,482.91	2,096.5	876	7,080
f	2,647.07	2,229	899	15,513
g	4,156.71	3,390.5	884	17,607
h	3,633.90	2,733	800	16,185
y	3,647.35	2,959	642	13,087
overall	3378.21	2552	428	29422

Table A.2.: Distribution of the number of tokens (extended Reformer tokenizer) of the medium sized sample corpus (test split).

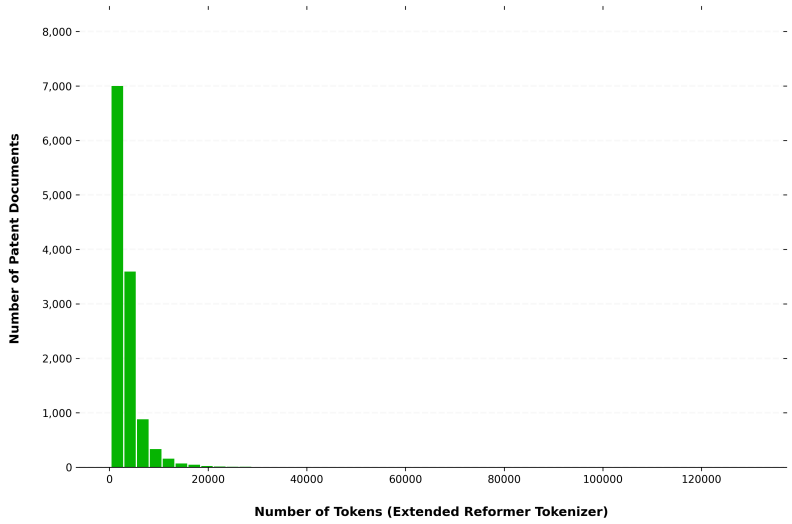


Figure A.1.: Distribution of the number of tokens (extended Reformer tokenizer) of the medium sized sample corpus (train split).

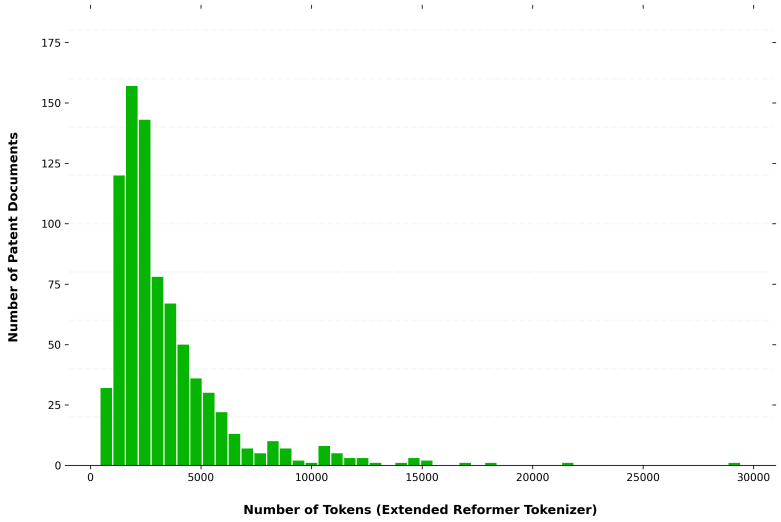


Figure A.2.: Distribution of the number of tokens (extended Reformer tokenizer) of the medium sized sample corpus (test split).

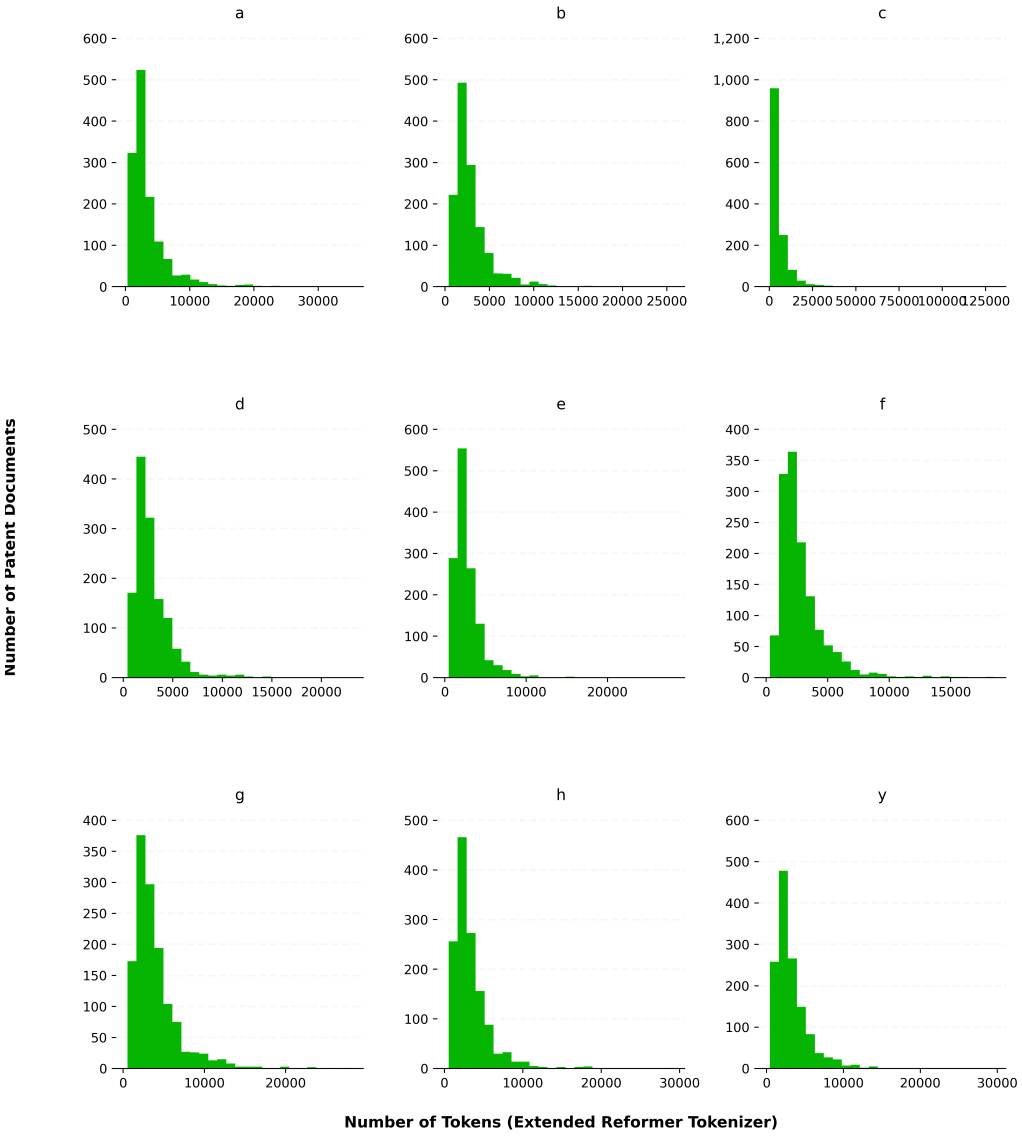


Figure A.3.: Distribution of the number of tokens (extended Reformer tokenizer) of the medium sized sample corpus (train split) grouped by category.

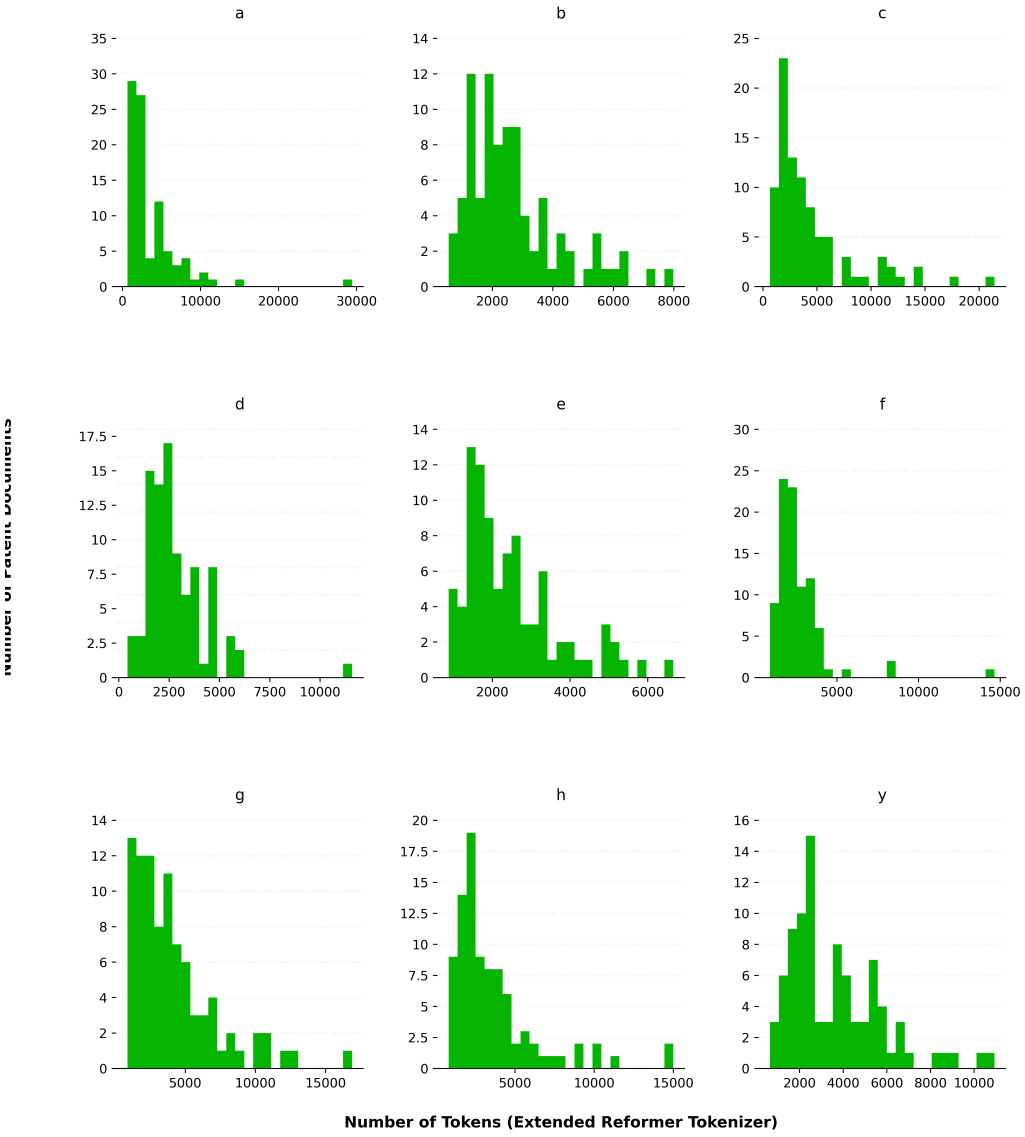


Figure A.4.: Distribution of the number of tokens (extended Reformer tokenizer) of the medium sized sample corpus (test split) grouped by category.

A.2. Large Sample (extended Reformer tokenizer)

Category	Mean	Median	Mininum	Maximum
a	3,698.88	2,660.5	339	52,986
b	2,927.02	2,390.5	348	59,307
c	5,347.46	3,508.5	372	59,111
d	2,948.7	2,384.5	267	32,375
e	2,792.62	2,309.5	286	23,000
f	2,726.6	2,231	433	25,140
g	3,995.81	3,076.5	583	47,569
h	3,562.51	2,844	320	43,187
y	3,361.16	2,598.5	380	40,685
overall	3,484.53	2,615	267	59,307

Table A.3.: Distribution of the number of tokens (extended Reformer tokenizer) of the large sample corpus (train split).

Category	Mean	Median	Mininum	Maximum
a	3650.49	2,664	591	28,341
b	2857.63	2,340.5	618	11,317
c	5953.13	3,704.5	575	45,947
d	2796.14	2,331.5	404	13,846
e	2979.69	2,316.5	605	19,466
f	2704.9	2,240	576	14,023
g	4095.54	3,052	357	54,645
h	3372.49	2,777	749	12,504
y	3179.4	2,627.5	508	14,119
overall	3,509.94	2,608.5	357	54,645

Table A.4.: Distribution of the number of tokens (extended Reformer tokenizer) of the large sample corpus (test split).

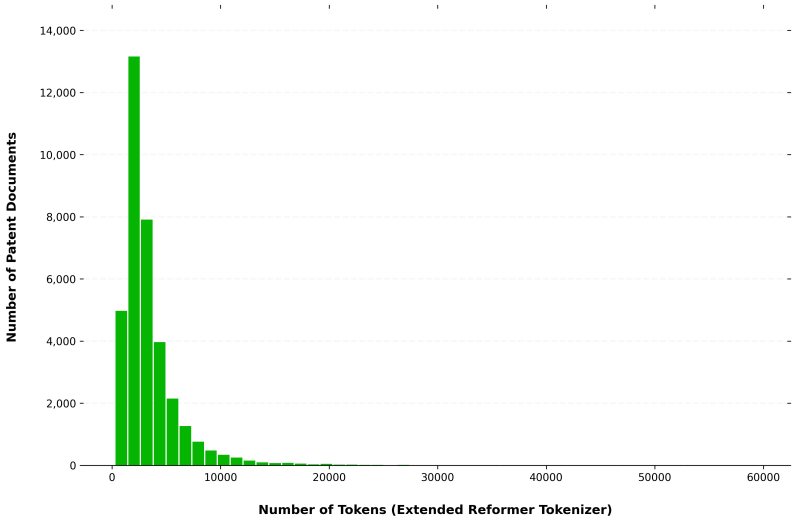


Figure A.5.: Distribution of the number of tokens (extended Reformer tokenizer) of the large sample corpus (train split).

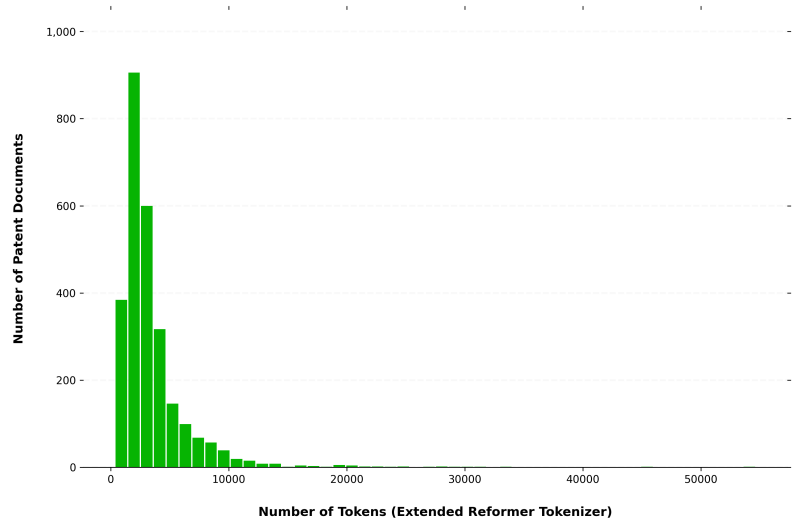


Figure A.6.: Distribution of the number of tokens (extended Reformer tokenizer) of the large sample corpus (test split).

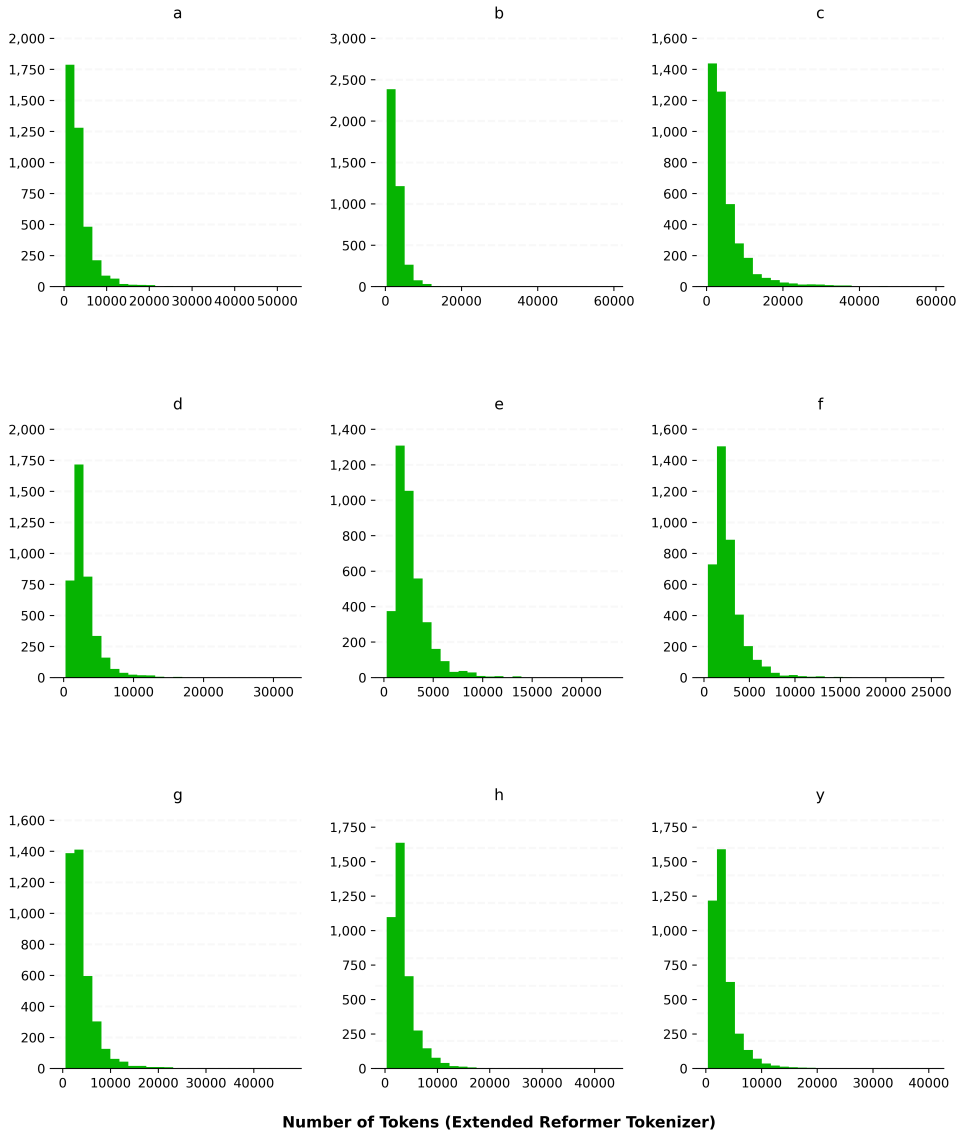


Figure A.7.: Distribution of the number of tokens (extended Reformer tokenizer) of the large sample corpus (train split) grouped by category.

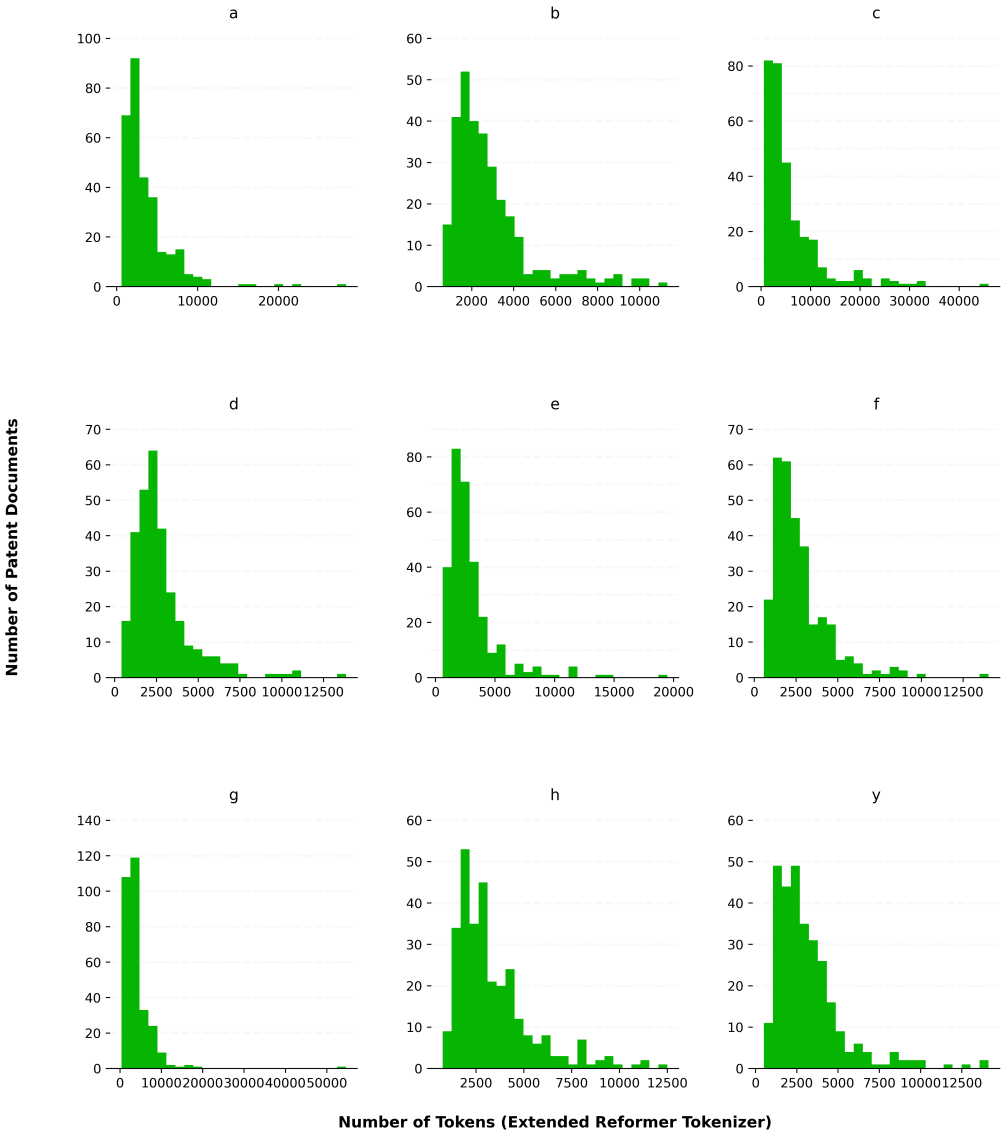


Figure A.8.: Distribution of the number of tokens (extended Reformer tokenizer) of the large sample corpus (test split) grouped by category.

Bibliography

[1] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. "Practical and optimal LSH for angular distance." In: *Advances in neural information processing systems* 28 (2015).

[2] I. Beltagy, M. E. Peters, and A. Cohan. "Longformer: The long-document transformer." In: *arXiv preprint arXiv:2004.05150* (2020).

[3] C. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[4] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, et al. "Findings of the 2014 workshop on statistical machine translation." In: *Proceedings of the ninth workshop on statistical machine translation*. 2014, pp. 12–58.

[5] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, et al. "{TVM}: An automated {End-to-End} optimizing compiler for deep learning." In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 578–594.

[6] Z. Chen, H. Zhang, X. Zhang, and L. Zhao. *Quora question pairs*. 2018.

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

[8] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, D. Belanger, L. Colwell, et al. "Masked language modeling for proteins via linearly scalable long-context transformers." In: *arXiv preprint arXiv:2006.03555* (2020).

[9] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. "Rethinking attention with performers." In: *arXiv preprint arXiv:2009.14794* (2020).

[10] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian. "A discourse-aware attention model for abstractive summarization of long documents." In: *arXiv preprint arXiv:1804.05685* (2018).

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[12] Q. Fournier, G. M. Caron, and D. Aloise. "A practical survey on faster and lighter transformers." In: *arXiv preprint arXiv:2103.14636* (2021).

[13] A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. "The reversible residual network: Backpropagation without storing activations." In: *Advances in neural information processing systems* 30 (2017).

[14] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang. "Retrieval augmented language model pre-training." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3929–3938.

[15] J. He, L. Wang, L. Liu, J. Feng, and H. Wu. "Long document classification from local word glimpses via recurrent attention learning." In: *IEEE Access* 7 (2019), pp. 40707–40718.

[16] D. Hendrycks and K. Gimpel. "Gaussian error linear units (gelus)." In: *arXiv preprint arXiv:1606.08415* (2016).

[17] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. "Teaching machines to read and comprehend." In: *Advances in neural information processing systems* 28 (2015).

[18] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension." In: *arXiv preprint arXiv:1705.03551* (2017).

[19] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. "Transformers are rnns: Fast autoregressive transformers with linear attention." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5156–5165.

[20] J. Kiesel, M. Mestre, R. Shukla, E. Vincent, P. Adineh, D. Corney, B. Stein, and M. Potthast. "Semeval-2019 task 4: Hyperpartisan news detection." In: *Proceedings of the 13th International Workshop on Semantic Evaluation*. 2019, pp. 829–839.

[21] N. Kitaev, Ł. Kaiser, and A. Levskaya. "Reformer: The efficient transformer." In: *arXiv preprint arXiv:2001.04451* (2020).

[22] S. B. Kotsiantis. "Supervised Machine Learning: A Review of Classification Techniques." In: *Informatica* 31.4 (2007), pp. 249–268.

[23] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown. "Text Classification Algorithms: A Survey." In: *Information* 10 (2019), p. 150.

[24] A. Krizhevsky, G. Hinton, et al. "Learning multiple layers of features from tiny images." In: (2009).

[25] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, et al. "Natural questions: a benchmark for question answering research." In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 453–466.

[26] J.-S. Lee and J. Hsiang. "Patent classification by fine-tuning BERT language model." In: *World Patent Information* 61 (2020), p. 101965.

[27] J. Lever, M. Krzywinski, and N. Altman. "Classification evaluation." In: *Nature Methods* 13 (2016), pp. 603–604.

[28] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." In: *arXiv preprint arXiv:1910.13461* (2019).

[29] T. Lin, Y. Wang, X. Liu, and X. Qiu. "A survey of transformers." In: *arXiv preprint arXiv:2106.04554* (2021).

[30] D. Linsley, J. Kim, V. Veerabadran, C. Windolf, and T. Serre. "Learning long-range spatial dependencies with horizontal gated recurrent units." In: *Advances in neural information processing systems* 31 (2018).

[31] 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." In: *arXiv preprint arXiv:1907.11692* (2019).

[32] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. "Learning word vectors for sentiment analysis." In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 2011, pp. 142–150.

[33] S. Merity, C. Xiong, J. Bradbury, and R. Socher. "Pointer sentinel mixture models." In: *arXiv preprint arXiv:1609.07843* (2016).

[34] N. Nangia and S. R. Bowman. "Listops: A diagnostic dataset for latent tree learning." In: *arXiv preprint arXiv:1804.06028* (2018).

[35] S. Narayan, S. B. Cohen, and M. Lapata. "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization." In: *arXiv preprint arXiv:1808.08745* (2018).

[36] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "Wavenet: A generative model for raw audio." In: *arXiv preprint arXiv:1609.03499* (2016).

[37] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. "fairseq: A fast, extensible toolkit for sequence modeling." In: *arXiv preprint arXiv:1904.01038* (2019).

[38] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel. "Language models as knowledge bases?" In: *arXiv preprint arXiv:1909.01066* (2019).

[39] S. Pradhan, A. Moschitti, N. Xue, O. Uryupina, and Y. Zhang. "CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes." In: *Joint Conference on EMNLP and CoNLL-Shared Task*. 2012, pp. 1–40.

[40] J. Qiu, H. Ma, O. Levy, S. W.-t. Yih, S. Wang, and J. Tang. "Blockwise self-attention for long document understanding." In: *arXiv preprint arXiv:1911.02972* (2019).

[41] D. R. Radev, P. Muthukrishnan, V. Qazvinian, and A. Abu-Jbara. "The ACL anthology network corpus." In: *Language Resources and Evaluation* 47.4 (2013), pp. 919–944.

[42] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. "Squad: 100,000+ questions for machine comprehension of text." In: *arXiv preprint arXiv:1606.05250* (2016).

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[44] E. Sharma, C. Li, and L. Wang. "Bigpatent: A large-scale dataset for abstractive and coherent summarization." In: *arXiv preprint arXiv:1906.03741* (2019).

[45] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. "Recursive deep models for semantic compositionality over a sentiment treebank." In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[46] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng. "Synthesizer: Rethinking self-attention for transformer models." In: *International conference on machine learning*. PMLR. 2021, pp. 10183–10192.

[47] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan. "Sparse sinkhorn attention." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9438–9447.

[48] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. "Long range arena: A benchmark for efficient transformers." In: *arXiv preprint arXiv:2011.04006* (2020).

[49] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. "Efficient transformers: A survey." In: *ACM Computing Surveys (CSUR)* (2020).

[50] T. H. Trinh and Q. V. Le. "A simple method for commonsense reasoning." In: *arXiv preprint arXiv:1806.02847* (2018).

[51] 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* 30 (2017).

[52] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. "GLUE: A multi-task benchmark and analysis platform for natural language understanding." In: *arXiv preprint arXiv:1804.07461* (2018).

[53] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. "Linformer: Self-attention with linear complexity." In: *arXiv preprint arXiv:2006.04768* (2020).

[54] J. Welbl, P. Stenetorp, and S. Riedel. "Constructing datasets for multi-hop reading comprehension across documents." In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 287–302.

[55] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. "HotpotQA: A dataset for diverse, explainable multi-hop question answering." In: *arXiv preprint arXiv:1809.09600* (2018).

[56] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. "Big bird: Transformers for longer sequences." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17283–17297.

[57] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi. "Defending against neural fake news." In: *Advances in neural information processing systems* 32 (2019).

[58] X. Zhang, J. Zhao, and Y. LeCun. "Character-level convolutional networks for text classification." In: *Advances in neural information processing systems* 28 (2015).

[59] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 19–27.

List of Figures

List of Tables