Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Leveraging Dense Text Representations for Efficient Language Modeling

Master's Thesis

Philippe Schläpfer

pschlaepfer@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Peter Belcák
Prof. Dr. Roger Wattenhofer

April 24, 2024

Acknowledgements

I want to express my gratitude to the people who have supported me in various ways, not only during the work on this thesis but also throughout my studies.

Firstly, I thank my supervisor Peter Belcák for his guidance and invaluable input and for opening my eyes to what's possible in the realm of efficient NLP models.

Next, I thank Prof. Roger Wattenhofer for giving me the opportunity to do the thesis in his lab and for the access to the lab's GPU resources.

Lastly, I thank my parents, siblings, and brother-in-law, for their unconditional support throughout my studies, for always having confidence in me, and for the numerous Sunday evening dinners. These gatherings have helped me recharge my batteries when I needed it the most.

Special thanks go out to Dano and Nic for their feedback and engaging in countless discussions, which helped shape and refine my ideas significantly.

Abstract

This thesis introduces a transformer-based text compression model that is able to efficiently operate on dense, compressed sentence representations. This is achieved by incorporating an artificial bottleneck into the BERT architecture. The model includes sentence-level transformer blocks that require only a 768 or 1024-dimensional embedding, representing an entire sentence, to reconstruct the masked-language-modeling objective. Consequently, the model serves as a general-purpose, encoder-only model.

With this approach, the model can achieve a text compression rate of up to 128x compared to the traditional BERT model. An additional benefit of the approach is that the model can do language modeling on a sentence level. This results in better perplexity scores than BERT.

Our approach achieves scores on the GLUE benchmark close to those of the BERT model while having roughly the same number of parameters. The most efficient configuration of the model achieves a speedup of up to 45% compared to BERT, potentially giving the model the capability of extending its context window to 8192 tokens or even more.

The model is implemented in the Crammed-BERT framework, making it trainable on a single A6000 GPU in approximately 5 days.

Contents

Chapter 1

Introduction

1.1 Language Modeling

With the advent of the transformer architecture [1], language models have exponentially increased in size. Starting with GPT-1 (110 million parameters) [2] in 2018 to GPT-3 [3] or Google's PaLM [4] (175 billion, 540 billion respectively) in 2022. This corresponds to a more than 1000-fold increase in the number of parameters in four years.
Recently it seems like the NLP community has focused on improving the LLMs' efficiency with approaches like the Mixture-of-Expert models [5] (like Mixtral [6]) or by improving the training signals (like Orca 2 [7]).

However, despite major efforts to reduce the footprint of the LLMs, all LLMs operate on (sub)words. Each of these subwords is represented with one embedding. In the case of GPT-3, one such embedding has a dimension of 12288. Even with newer, more efficient models, like Mixtral, the dimension of the embeddings is still 4096. According to Huggingface [1], the Mixtral model can directly be loaded in half-precision (i.e. float16). Therefore, each subword has a size of 8KB. If we use Mixtral with 4-bit quantization, each token would still be represented by an embedding with a size of 2KB. Thus it seems like we hit a natural limit by just trying to decrease the precision of the models.

In addition to the memory needed for each embedding, the processing of subwords during a forward pass also needs to be considered. In deep learning architectures, the embeddings are multiplied with matrices over and over again and since matrix multiplication is an expensive operation, the length of the sequence maintained during the forward pass has a direct impact on the efficiency of a language model. This is especially relevant for transformer-based models, whose attention mechanism scales quadratically in the length of the input. Consequently, if the length of the sequence could be reduced it would have a drastic impact on the efficiency of such architectures.

[1] https://huggingface.co/docs/transformers/en/model_doc/mixtral

Going beyond subwords. The goal of this thesis is to find a deep learning-based approach, which compresses enough meaning into sentence-level embeddings, such that these embeddings can be used to solve NLP tasks. If there is a way to do that, there are two major ways that it could influence the NLP landscape.
First and foremost, it would mean that we could do NLP on sentence level. This would imply that only one embedding would need to be processed to solve sentence-level tasks like GLUE [8], for instance. This would have a major impact on the efficiency of language models. A quick look at a subset of the Wikipedia dataset [9], shows that on average a sentence is made up of 26 subwords. If the sentences could be compressed into one embedding of the same size, this would be a compression of factor 26.
On the other hand, it could potentially make it possible to process longer sequences with no significant increase in model size. One major advantage of LLMs is their ability to process longer sequences. Mixtral for example has a context size of 32k tokens. Some commercial LLMs like GPT-4 [10] or Gemini [11] claim to have a context size up to 10 million tokens. The hypothesis here is that the sentence embeddings can be processed by a transformer network in the same manner as subword embeddings. Hence, training a transformer model with an input length of 128 would mean that whole paragraphs up to 128 sentences could be processed in one go. Assuming that an average sentence consists of 26 tokens, it would mean the resulting model has a context size of 3328 tokens with the parameter count of a normal BERT [12] model.
This thesis focuses on the masked language modeling objective and builds upon the BERT architecture.

1.2 Thesis overview

Chapter 2 discusses related work which influenced design decisions of this thesis and helps understand the whole field better. It also introduces related compression-based language models which are used as baselines to compare against.
Chapter 3 provides both a high-level overview and a detailed description of the approach. It also outlines the data preprocessing steps.
In Chapter 4, the performance on language modeling tasks is discussed and compared to different baselines described in Chapter 2.
Chapter 5 discusses the results of multiple efficiency and speed measurements of the developed models. These are also compared to baseline models.
To conclude the thesis, chapter 6 reviews the results, discusses the limitations of the approach and provides some ideas for future work.

Chapter 2

Related Work

2.1 Efficient Language Modeling

A lot of effort is being put into improving the efficiency of large language models. As described in Chapter 1.1, there's a trend towards larger and more memory- and compute-intensive language models. Hence, the NLP community has developed a lot of different approaches to make LLMs more accessible. The recently published survey paper "The Efficiency Spectrum of Large Language Models: An Algorithmic Survey" [13] has a comprehensive overview of all the different aspects of LLM efficiency. In this chapter, the different aspects are only explained briefly. For the interested reader, the above-mentioned survey paper is a good starting point.
The authors of that paper put the different aspects of efficiency into five different groups:

- Budget efficiency (mainly about scaling laws)
- Data efficiency (including Data filtering, active learning & curriculum learning)
- Architecture efficiency (including Efficient attention & sparse modeling)
- Training & Tuning efficiency (including Mixed precision training, Parameter-efficient fine-tuning, etc.)
- Inference efficiency (including pruning, distillation, quantization & low-rank decomposition)

Taking these groups as a basis, the work in this thesis can be put into both groups *architecture efficiency* as well as *inference efficiency*. It's important to note that this is just an attempt to somehow classify the different approaches.

2.2 Compression-based approaches

As can be seen from the previous section, there's not a lot of attention in the research community on sentence-based NLP. Nevertheless, there are many similar research directions.

2.2.1 Distillation for language models

A very successful approach to training efficient language models is distillation [14]. Distillation is the process of compressing the knowledge of a larger model, called the teacher model, into a smaller one, called the student model. To achieve this, the student model tries to predict either the logits or the softmax outputs of the teacher model. Note that the teacher model is a model that is already trained on a task and achieves a good performance on it.
In DistilBERT [15], the authors apply distillation to BERT by training a student model that has the same architecture as the original BERT model but with half as many layers. They can retain 97% of BERT's performance with 40% fewer parameters.
The authors of MobileBERT [16] build upon the distillation technique used in DistilBERT. They enhance this approach by integrating bottleneck layers into the architecture, with which the hidden dimension of the model gets decreased to achieve an even smaller model size. The resulting model is 4.3 times smaller than $\text{BERT}_{\text{BASE}}$ and 5.5 times faster while achieving competitive results on NLP benchmarks.
In the paper "Well-read students learn better" [17], the authors show that it is beneficial to first pre-train a model with a small architecture and then do knowledge-distillation on that model instead of doing knowledge-distillation only.

2.2.2 Sentence compression

Somewhat related to the topic of this thesis is the field of sentence embeddings, i.e. learning meaningful representations of sentences, which in turn can be used for downstream tasks. This can also be seen as some kind of compression since whole sentences are stored in one vector. The goal is to learn a fixed-size vector space in which sentences with similar meanings are close to each other. These approaches are usually evaluated with semantic textual similarity tasks or transfer learning tasks.
SentenceBERT [18] achieves meaningful sentence representations by modifying the original BERT architecture with siamese and triplet network structures. A major drawback of this and similar methods is the fact that they need labeled parallel data to be able to fine-tune their model on sentence pairs. TSDAE [19] solves this problem by using a denoising autoencoder. It works by adding noise to the input sentence (either deleting or swapping tokens in the input sequence)

and then learning to reconstruct the input. It does that by having a transformer encoder that encodes the input into a fixed-size vector that is fed into a decoder trying to reconstruct the input sequence.
The authors of the paper "Sentence Bottleneck Autoencoders from Transformer Language Models" [20] use a similar setup. They also use a denoising autoencoder with a transformer-based encoder that compresses the input into a fixed-size vector $\mathbf{z}$, which is then fed into a transformer decoder layer. Somewhat novel is their proposed gating mechanism (similar to the one in LSTMs) to solve the problem where only one single representation is given to the decoder. The model is evaluated on sentence classification, sentence similarity and sentence generation tasks. To make the training converge faster, the encoder model is initialized with the weights of RoBERTa [21].

2.2.3 Sentence-level NLP

Research in the field of sentence-level NLP tries to design models that can capture higher-level information of text. E.g. sentences and the relationships among them. The authors of SLM [22] propose a **S**entence-Level **L**anguage **M**odel that uses a hierarchical transformer to reconstruct the order of the input sentences, which have been shuffled.
In [23] the authors present a transformer-based model with the goal of generating more fluent stories. The proposed model does this by treating the story so far as a sequence of pre-trained sentence embeddings. The task of the model is to predict the next sentence embedding. This is treated as a standard classification task by presenting the model N candidate sentences of which only one is the actual next sentence in the story.

2.2.4 Efficient transformer architectures

A major disadvantage of the transformer architecture and self-attention in particular is the quadratic time- and space-complexity in the length of the input. While there has been groundbreaking research in this field (with a huge practical impact), like Flash Attention [24], the focus of this chapter lies on the improvement of the transformer architecture as a whole.

Funnel transformer

Funnel transformer [25] tries to solve the problem of "maintaining full-length token-level presentations" which BERT-like architectures do. The architecture gradually decreases the hidden size of the token representations during the forward pass. The saved compute can in turn be used to make the model deeper.

Funnel-Transformer outperforms the standard Transformer on many tasks while needing comparable or fewer FLOPs (floating point operations per second).
This is achieved by introducing pooling layers during the encoding step, which shorten the representations along the sequence dimension. The shortened representation is used as the query vector of the self-attention mechanism, while the unpooled representation functions as the key & value.
For the decoding step, an up-sampling operation is used, which repeats each hidden vector. Importantly, since all of these vectors contain the same information, a residual connection is used which adds the last representation of the full-sized vector of the encoder to the upsampled representation. This in turn makes the training easier and lets the model converge faster.
For downstream tasks where no token-level prediction is needed, the decoder is not used.

Hierarchical transformers

In [26], the authors use a very similar approach as in the Funnel transformer paper, but in contrast to that, their *Hourglass* architecture is autoregressive. They show that their architecture can efficiently handle longer contexts.

Lite Transformer

The Lite Transformer architecture [27] aims to create an efficient transformer architecture by combining CNNs and traditional Multihead-Attention layers, which the authors call Long-Short Range Attention (LSRA). CNNs allow the model to efficiently extract local information, whereas the Multihead-Attention layers maintain the Transformer architecture's strength in modeling complex, long-range dependencies.

2.2.5 Hierarchical representation of language

A related line of research tries to make use of the hierarchical forms of language by adding an inductive bias to the model. This is often done by first processing words or subwords, which are summarized in single sentence embeddings. These sentence embeddings are then further processed to get a representation of multiple sentences, often called documents in the literature.

Hierarchical Attention Networks for Document Classification

Early work on hierarchical representation learning is done by [28]. The authors use a Gated Recurrent Unit (GRU) [29] with an attention mechanism to encode the text on a word level. These hidden, contextualized representations of words are then used as input for the next layer, which processes multiple such sentence embeddings to classify documents.

Document-Level Neural Machine Translation with Hierarchical Attention Networks

The work of [30] is very similar to [28], but instead of a GRU, Multihead-Attention is used. Alongside this, a "Context Gating" mechanism is used to regulate the information at a sentence and document level.

HIBERT

HIBERT [31] uses sentence-level embeddings to encode whole documents and utilizes the document representations to do document summarization. They use a transformer-based sentence encoder to represent each sentence in a document by one vector followed by a document encoder (also transformer-based) to learn sentence representations given surrounding sentences as context. For the representation of a sentence, the last token of a sentence is taken (i.e. the EOS token). To force the model to use attention on the sentence level, 15% of the sentences are randomly masked (like in BERT, but on a sentence level). By "masking sentences“ the authors mean that all of the words in a sentence are masked. This approach is the one that is most related to the work in this thesis.

A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents

In [30] an encoder-decoder bi-LSTM [32] network is proposed in which the encoder is a hierarchical RNN encoding whole documents. Their "discourse-aware decoder" can attend to the words in the document and also to the relevant "discourse sections", which they call the encodings of the sentences in the source document. This approach is used for abstractive summarization of documents.

Hierarchical Learning for Generation with Long Source Sequences

The authors of [33] propose a Hierarchical Attention Transformer-based architecture (HAT) which extends the original transformer architecture with a "Hierarchical Learning" component on top of the encoder as well as in the decoder. They include beginning of sequence (BOS) tokens at the beginning of each sentence in the input. The hidden representations of these tokens are then fed into the hierarchical component of the encoder. The output of this component is fed into its counterpart component in the decoder part of the transformer.

2.3 Evaluation of Language models

Language models are usually evaluated on a variety of downstream tasks. These tasks can cover a lot of different aspects of language (text classification, machine translation, coding tasks, math problems, commonsense reasoning, etc.).

2.3.1 GLUE Benchmark

In this thesis the GLUE benchmark [8] is used to evaluate the performance of the trained models. It is one of the most widely used benchmarks and consists of multiple different tasks, all related to language understanding. Hence, it is well-suited to compare the performance of our model to other pre-trained language models. The different tasks are the following:

- CoLA: English acceptability judgments from books and journals on linguistic theory. The goal is to predict if a given sentence is linguistically acceptable or not. Thus, it's a binary classification problem but evaluated with the Matthews correlation coefficient.
- SST-2: Sentences from movie reviews with a binary label (positive/negative)
- MRPC: Given two sentences, the goal is to predict whether these are semantically equivalent.
- QQP: A collection of question pairs from Quora. The goal is to predict if a pair of questions are semantically equivalent.
- STS-B: A collection of sentence pairs, where the task is to predict a similarity score between 1 and 5.
- MNLI: Given a premise sentence and a hypothesis sentence, the goal is to predict whether the premise entails the hypothesis. There are three labels for this task: entailment, contradiction and neutral.
- QNLI: For each sample, there's a question and a sentence. The task is to determine whether the sentence entails the question or not.
- RTE: Given two sentences, the task consists of predicting whether the first sentence entails the second.
- WNLI: Same task as RTE.

2.3.2 Evaluation of computational resources

Since the goal of this work is to come up with an efficient (sentence-level) architecture, we also compare the model to BERT based on FLOPS (Floating point operations per second) [1] & MACs (multiply–accumulate operations) [2]. Moreover, the actual time spent processing is measured to get a better picture of the efficiency of this approach.

2.3.3 Perplexity for masked language models

In addition to the two above-mentioned evaluation schemes, the models are also evaluated using the perplexity [34], defined as follows:

$$\mathrm{PPL}(X) = exp\{-\frac{1}{t}\sum_{i}^{|X|}\log p_\theta(x_i|x_{<i})\} \tag{2.1}$$

Where $X = (x_0, x_1, ..., x_t)$ is a sequence of words, $p_\theta(x_i|x_{<i})$ denotes the probability for a word x_i given all previous words in the sequence and θ are trainable parameters. A caveat of this definition is that it is only applicable to autoregressive models, which do not have access to tokens x_m, where $m > i$. Since the language model in this work is a masked language model, it has access to the whole word sequence at once. Therefore we use the definition of [35], which the authors call *pseudo-log-likelihood scores* (PLL):

$$\mathrm{PLL(X)} = \sum_{t=1}^{|X|}\log p_\theta(x_t|X_{\backslash t}) \tag{2.2}$$

$p_\theta(x_t|X_{\backslash t})$ is the (masked language model) probability of token x_t given all other tokens in the sequence X.

The perplexity is low if the probability of a given token is high. Hence, in general, we strive to train language models which achieve a low perplexity.

2.4 Training Data & Implementation

For all of the pre-trainings in this work the BookCorpus [36] and English Wikipedia are used, which are also used for the original BERT pre-training.
The whole approach is implemented within the Crammed-BERT [37] framework [3]. This is done because the framework provides a way to train a BERT model

[1]https://en.wikipedia.org/wiki/FLOPS
[2]https://en.wikipedia.org/wiki/Multiply%E2%80%93accumulate_operation
[3]code forked from https://github.com/JonasGeiping/cramming

within one day on consumer-grade hardware, hence it facilitates fast iteration and it doesn't need as much resources. Furthermore, it provides all of the necessary building blocks like transformer encoders, embedding layers, etc. Therefore it makes sense that the data preprocessing for baseline models is directly used from that framework. For our approach, we use the same data, but since our approach operates on a document level, we have to implement our own data preprocessing. We keep the maximum length of input sentences at 128 tokens since the whole Crammed-BERT setup is designed to work best that way. More info regarding data preprocessing can be found in section 3.2.

Chapter 3

Approach

The architecture of the model is very similar to BERT, but with the difference of having an artificial bottleneck to force the model to learn dense sentence representations. It also has elements of an autoencoder, since the model is trained on a masked language modeling objective (like BERT), but it has to predict all masked tokens in a sentence from just one hidden vector. This task is harder than the BERT task, since in the BERT architecture, a hidden vector for each input token is maintained throughout the whole forward pass.
For the remainder of this thesis, the terms *documents* and *paragraphs* are used interchangeably. Both refer to a collection of related sentences.

3.1 Architecture

As discussed in section 1.1, this thesis focuses on sentence-level language modeling with the potential of processing whole paragraphs or documents at once. Therefore, the highest level of abstraction we have to maintain is at the document or paragraph level. In contrast, traditional methods simply involve storing sequences of tokens. This can be viewed as augmenting the data with an extra layer, creating a new, additional dimension. Instead of sequences $X^{(i)} = (x_0, x_1, x_2, ..., x_n)$ containing n tokens, this model processes documents D, which themselves contain sequences of tokens $X^{(i)}$.

$$\begin{aligned} D_i &= (X^{(0)}, X^{(1)}, ..., X^{(m)}) \\ &= ((x_0^{(0)}, x_1^{(0)}, ..., x_{n_0}^{(0)}), (x_0^{(1)}, x_1^{(1)}, ..., x_{n_1}^{(1)}), ..., (x_0^{(m)}, x_1^{(m)}, ..., x_{n_2}^{(m)})) \end{aligned} \tag{3.1}$$

As can be seen in figure 3.1, we start with documents as the basis. These documents are preprocessed such that each document is processed as a whole unit. This is necessary since during the forward pass, the model needs to keep track of which sentence belongs to which document. Otherwise, attention between sentences (within a document) wouldn't be possible.

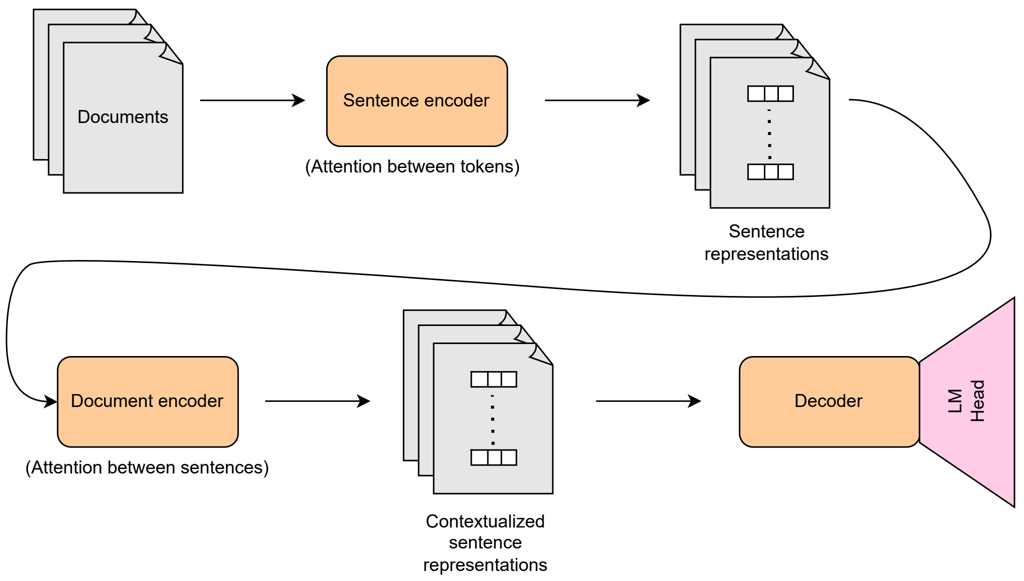

Documents
Sentence encoder
(Attention between tokens)
Sentence representations
Document encoder
(Attention between sentences)
Contextualized sentence representations
Decoder
LM Head

Figure 3.1: Simplified overview of the architecture.

1. The individual sentences are processed by the *sentence-level encoder*, which outputs representations for each sentence. This step applies the attention mechanism on a token level to capture relationships between tokens within a single sentence. The output of this step is a list of sentence embeddings for each document.

2. The sentence embeddings of the previous steps are fed into the *document-level encoder*. The job of this encoder is to capture any relationships between the sentences. The output can be thought of as contextualized sentence representations. This is also the component from which speed gains come. Since whole sentences are processed as one single unit (i.e. one vector), this operation is very efficient.

3. The contextualized sentence representations are then fed into the decoder part of the model, which is responsible for reconstructing the individual tokens from the input such that the masked language modeling loss can be calculated.

All of the orange boxes are regular transformer *encoder* blocks. The decoder is also a transformer encoder (although it's called a decoder) since it has a similar job as the decoder in an autoencoder. It's worth mentioning that a hierarchical encoding is done with this architecture by first encoding the sentences and then using the sentence embeddings to encode whole documents, similar to approaches described in section 2.2.5.

3.1.1 Fine-tuning & Inference

The decoder is not used for inference as well as fine-tuning on downstream tasks. Usually, the output of the document encoder is passed through a feed-forward layer with the number of output neurons needed to solve a specific task.

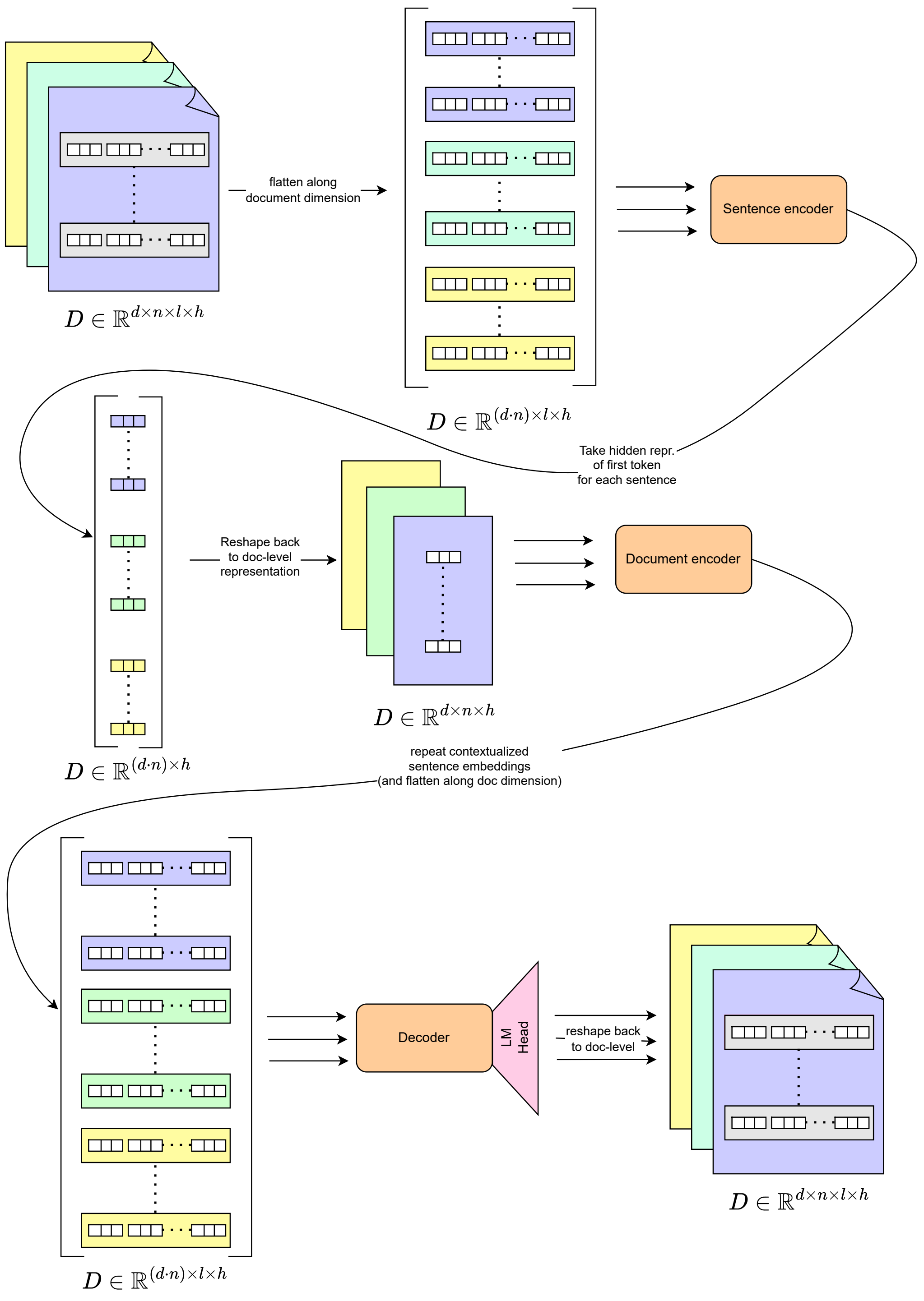
3.1.2 How can the standard transformer blocks handle the 3D data?

They can't. However, since the architecture consists of a sentence-level encoder, the input documents can be reshaped such that each sentence is processed individually by the sentence encoder. 3.2 shows a detailed version of a forward pass through the architecture. Here d is the number of documents in the batch, n is the number of sentences in each document, l is the maximum length of the sentences (which is capped at 128 tokens) and h is the hidden dimension (the size of the embeddings).

The following steps describe an exemplary forward pass depicted in figure 3.2 with three documents as input.

1. Starting with a batch $D \in \mathbb{R}^{d \times n \times l \times h}$ consisting of three documents, the first step is to flatten the data along the first dimension to get a list of sentences $\in \mathbb{R}^{(d \cdot n) \times l \times h}$.
2. Now the data is ready to be applied to attention on a sentence level. Each sentence is processed separately. There is no explicit information to which document each sentence belongs.
3. The next step is to take the first b embeddings of each processed sentence as a representation for each sentence. Most of the time (and also for this example), the representation of the first token is taken (the representation of the BOS token). This implies that the bottleneck size directly depends on the size of the hidden dimension. The result is a two-dimensional tensor of shape $(d \cdot n) \times h$. This 2D array is reshaped back into document-level. But since each representation of a sentence consists of only one element, it's now possible to pass it through a transformer. Comparing it to the initial input, we basically got rid of the length dimension by compressing each sentence of length represented by a 2D tensor $l \times h$ into just one vector of dimension h.
This is the step where the efficiency comes into play. Since we can operate on just one embedding representing the whole sentence instead of 128 embeddings, this block is a lot more efficient than the sentence-level encoders.

4. Passing the documents through the document encoder results in sentence embeddings that have context from the other sentences in the same documents. It's reshaped back to sentence level and the output for each sentence is repeated l times. Although this doesn't add any information, experiments showed that it works slightly better compared to just padding with zero values. This representation is passed through the decoder with a language modeling head on top to predict the masked tokens.
   Note that, unlike the funnel transformer architecture, this approach does not need a skip connection from the full-sized hidden representations to make the decoding easier. We want the architecture to be able to reconstruct the MLM objective only from one embedding.



$D \in \mathbb{R}^{d \times n \times l \times h}$
flatten along document dimension
$D \in \mathbb{R}^{(d \cdot n) \times l \times h}$
Sentence encoder
Take hidden repr. of first token for each sentence
$D \in \mathbb{R}^{(d \cdot n) \times h}$
Reshape back to doc-level representation
$D \in \mathbb{R}^{d \times n \times h}$
Document encoder
repeat contextualized sentence embeddings (and flatten along doc dimension)
$D \in \mathbb{R}^{(d \cdot n) \times l \times h}$
Decoder
LM Head
reshape back to doc-level
$D \in \mathbb{R}^{d \times n \times l \times h}$

Figure 3.2: Detailed overview of the architecture.

3.2 Data preprocessing

Given a list of documents, the first step is to split up the given documents into sentences (sometimes also called *sentencizing*). For the sentencize step the `tokenize.sent_tokenize()` method of the nltk library [38] is used. After that, the individual sentences are tokenized into subwords. The documents are then treated as single training samples, the same way as text sequences in traditional pre-training are treated.
Since the Wikipedia and Bookcorpus datasets both can have very large documents, the number of sentences for each document is capped at 64 to make the training a bit easier. This means that the theoretically maximal context size of this model is 8192 tokens (128 tokens per sentence times 64 sentences per document). The whole process is schematically visualized in figure 3.3.
There are some tricks needed during the preprocessing which make the reshaping operations shown in figure 3.2 work efficiently:

- Each batch consists of documents with the same number of sentences. This is necessary such that the reshaping operations discussed in the previous section can be handled efficiently. Also, if that wouldn't be the case, there could be batches in which one document has only one sentence, and another one has 64. Hence, the document with only one sentence would have to be padded with 63 empty sentences, which in turn would make the forward pass make a lot of unnecessary calculations.

- To better control the size of the batches, we defined a constraint that helps fill up batches with documents up to a pre-defined maximum number of sentences. We defined n_{max} as the total maximum number of sentences in a batch (over multiple documents). For a group of documents, each consisting of n sentences, the maximum number of documents d_{max} in a batch is constrained by:
$$d_{max} \leq \frac{n_{max}}{n} \tag{3.2}$$
During preprocessing the documents are grouped into lists of documents such that equation 3.2 is satisfied.

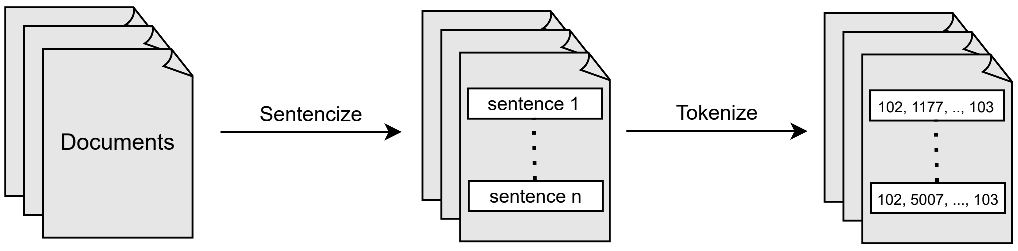


Figure 3.3: Preprocessing steps

3.3 Pre-training details

All of the models were trained with a one-cycle learning rate schedule [39] for $2'000'000$ steps with a peak learning rate of 0.001. As an optimizer, AdamW [40] was used with weight decay 0.01, $\epsilon = 1^{-12}$, $\beta = (0.9, 0.98)$. For the rest of the hyperparameters, the default values of the Crammed-BERT framework were used.

For the THF model (short for Thoughtformer), we've chosen the format (s_e-d_e-d), to describe the number of sentence encoder layers (s_e), the number of document level encoder layers (d_e) and the number of decoder layers (d). Since the decoder layers are not used for downstream tasks, it makes sense to directly compare a BERT model with n layers to a THF model with $n = s_e + d_e$. But of course, it also makes sense to compare architectures with different numbers of encoders.

Training for the largest model (i.e. the (12-4-4) configuration) took 5 days on a single A100 GPU and the (6-2-4) configuration was finished in roughly 3.5 days, also on an A100 GPU.

Chapter 4

Experiments on Language Modeling

4.1 Attention over sentence embeddings

In section 1.1 it's stated that with an encoder that can operate on a sentence level, it would be possible to apply attention on a document level. Unfortunately, without a loss function that explicitly forces the document-level encoder to train the sentence-level attention mechanism, fine-tuning does not work if the input is split up into individual sentences. This can be seen in table 4.1. Surprisingly, it works if the whole input is packed into one input sentence, even though the model is trained on individual sentences. Note that the MRPC task is a binary classification task with a class balance of 31% for one class and 69% for the other class. Therefore, having an accuracy of 69% is the same as random guessing.

We've tried two different approaches to force the document-level encoder to learn relations between sentences. Both are discussed below.

4.1.1 Masking sentences

Inspired by the HIBERT paper [31], we've tried to mask whole sentences for a fraction of the training samples. The intuition behind this idea is that (some) of the words in the masked sentence can be predicted based on the surrounding sentences.
Since the fraction of sentences to mask is a hyperparameter, different values were tested (15%, 20%, 50%). The rest of the training samples were kept to the MLM objective. Although the loss went down, it didn't go as far down as the MLM-only objective. The downstream performance was on the level of random guessing.
We hypothesize that the task is just too hard for a language model (probably also for a human).

4.1.2 Unshuffling of sentences

The SLM paper [22] has a different approach to training a sentence-level language model. The authors shuffle the sentences of the input sequence in 50% of the training samples. To solve the shuffling task, a separate decoder network is introduced, which is responsible for solving the unshuffling task. The model is then trained jointly on both tasks such that the final loss function looks like this:

$$\mathcal{L}_{total} = \mathcal{L}_{MLM} + \mathcal{L}_{shuffle} \tag{4.1}$$

We adapt this approach by shuffling the hidden representations of the input sentences before they are passed through the doc-level encoder. The output of the doc-level encoder is then passed through the sentence-unshuffling network as well as through the decoder.
This approach unfortunately didn't work either and due to time constraints was not further pursued.

Model	**type**	**dim**	**MRPC (acc./f1)**	**STS-B**
THF-12-4-4	vanilla THF	1024	69.2/79.2	40.1
THF-12-4-4	masking 20% of the sentences	1024	67.9/77.9	42.2
THF-12-4-4	shuffle all sentences	1024	66.9/77.2	38.7
THF-12-4-4	shuffle 50% of the sentences	1024	69/78.5	66.6
THF-12-4-4	shuffle 15% of the sentences	1024	67.9/78.3	52.4
Bi-LSTM	-	1500	74.3/81.8	67.8

Table 4.1: Results on the MRPC and STS-B tasks from GLUE. As a baseline a vanilla bidirectional LSTM [32] was used. Note that both approaches get outperformed by the baseline. This hints at the fact that the fine-tuning procedure of multiple sentence embeddings has to be improved for it to work.

4.2 GLUE

Since the attention over sentence embeddings did not work for GLUE, we've resorted to an approach which we call *packed sequencing*. Meaning the whole input is packed into one sequence instead of splitting it up into individual sentences. The sentences are separated by special tokens ([SEP] tokens in this case). This is how multiple input sentences are handled for the traditional BERT models. Although this is not the initially intended way to measure the performance, it's the only way we've achieved high scores on GLUE. Also, it has the side benefit of achieving the highest compression rate. Since the bottleneck consists of only one hidden vector for the whole input sequence (which for most glue tasks consists of multiple sentences). This means that a compression of up to 128x can be achieved this way.

The performance of the model is evaluated on the GLUE dev set. The results are compared to the original BERT model, DistilBERT, a 6-layer well-read-student model and multiple Crammed-BERT versions with different architecture hyperparameters like hidden layers, hidden dimensions, etc. to have a direct comparison of the THF model to the BERT models. The results are displayed in table 4.2.

For fine-tuning, the same hyperparameters as in the Crammed-BERT paper [37] are used, but with the following modifications:

- The models are fine-tuned for 3 epochs instead of 5.
- learning rate: 5e-5 instead of 4e-5

We report the median GLUE scores over five runs for each task. The same hyperparameters are used for all the GLUE tasks.

Model	**dim**	**MNLI**	**SST-2**	**STSB**	**RTE**	**QNLI**	**QQP**	**MRPC**	**Avg**
THF-12-4-4	1024	79.9	90.6	83.5	54.9	86.7	86.7	87.6	81.39
THF-8-2-2	1024	78.7	89.9	82.5	54.2	85.5	85.8	87	80.51
THF-10-2-4	1024	79.3	89.3	83.9	56.7	86.4	86.2	86.6	81.2
THF-6-2-4	1024	78.5	89.4	82.9	57.8	85.8	85.6	86.2	80.89
THF-8-2-4	1024	79.2	89.4	83.1	54.2	85.4	86	86	80.44
THF-8-4-4	1024	79.1	90.3	82.7	55.6	85.5	86.1	85.6	80.7
THF-6-2-4	768	77.6	89.3	82.2	56.7	84.6	84.8	83.9	79.87
Baselines									
BERT$_{\text{Base}}$-Orig	768	**86.7**	**92.7**	**89**	**69.3**	**91.8**	**89.6**	88.6	**86.81**
BERT$_{\text{Base}}$-Cram	768	83.4	91.9	86.7	59.2	90.6	87.7	89.3	84.11
Cramming-12	768	84.1	92.2	84.6	53.8	89.5	87.3	87.5	82.71
DistilBERT	768	82.2	91.3	86.9	59.9	89.2	88.5	87.5	83.64
BERT-6	768	82.5	**92.7**	-	66.7	89.4	87.7	**89.4**	84.73$^{(*)}$
Cramming-8	1024	35.4	90.8	85.2	52.7	49.5	63.2	84.8	65.94
Cramming-8	768	81	90.4	85	55.2	88.7	85.9	85.7	81.7

Table 4.2: Results on the GLUE dev set. BERT-6 is the 6-layer BERT model of the "Well-read students" paper. BERT$_{\text{Base}}$-Orig is the model from the original paper and BERT$_{\text{Base}}$-Cram denotes the results of the Crammed-BERT paper. Cramming-x denotes a BERT model with x layers trained using the Crammed-BERT framework. **Bold** are the best overall scores and violet are the best scores amongst all THF models.
$^{(*)}$: Scores for STSB and CoLA are not reported in the paper. Hence, the average is not directly comparable to the averages of the other models.

4.2.1 Findings

The first thing that stands out is that the differences in the overall GLUE scores between the individual THF setups are not that large, ranging from 79.87 overall to 81.39. This fact implies that there's an upper limit to the information that the bottleneck can carry, which makes a lot of sense intuitively. Nevertheless, the best (and largest) model comes close to the 12-layer Crammed-BERT model trained by us, which has a score of 82.71.

Hidden dimension size. The size of the hidden dimension is the hyperparameter responsible for the largest difference in GLUE scores. The difference is clearly visible between the (6-2-4) setups where one setup has a hidden dimension of 1024 and the other has a setup of 768. The difference is biggest for the CoLA and MRPC tasks.

Bottleneck width. To get an idea of the influence of the bottleneck width, two almost identical models were trained with the only difference being the width of the bottleneck. The models have a (6-2-4) configuration with a hidden dimension of 768. This means that the smaller bottleneck has a size of 768, whereas the larger bottleneck's dimension is 1536. Table 4.3 shows the results on the GLUE dev-set. Unsurprisingly, the model with the larger bottleneck performs better, but the difference is marginal. Nevertheless, it's still surprising that the difference is so small since the model with the larger bottleneck should have twice the capacity to store necessary information. A possible explanation for this outcome is that the supplementary embedding does not significantly enrich the information already captured by the [CLS] token representation, which is commonly used for fine-tuning in the traditional BERT setting. Except for the CoLA task, the differences in scores between the models are within a range of ±1.2.

width	MNLI	SST-2	STSB	RTE	QNLI	QQP	MRPC	CoLA	Avg
1	77.6	89.3	82.2	56.7	84.6	84.8	83.9	50.5	76.2
2	77.6	88.1	82.1	56.7	84.6	85.1	84.3	51.9	76.3

Table 4.3: Glue dev scores of two THF models with the configuration (6-2-4) and hidden dimension 768. The only difference is the bottleneck width. A width of two means that the hidden representations of the first two tokens were taken and concatenated.

Number of decoder layers. When comparing the (8-2-2) configuration with the (8-2-4), we can see that their scores are almost equal (80.51 vs 80.44). This makes sense intuitively, since for the decoding step, the hidden representation is just repeated n times. Nevertheless, when comparing the CoLA scores, we see that the configuration with more decoders performs better. Also, since the decoder is not used for fine-tuning and inference, most of the configurations are trained with 4 decoders.

Number of sentence encoder layers. There's clear evidence that more sentence encoder layers lead to higher scores. Comparing the Glue scores of (x-2-4) models with a hidden dimension of 1024 and with either 6, 8, 10 or 12 sentence encoders, there's a clear pattern (although the differences are small and there's a drop-off going from 6 encoder layers to 8 layers):

- 6 layers: 80.89
- 8 layers: 80.44
- 10 layers: 81.2
- 12 layers: 81.39

Number of document encoder layers. Comparing two setups which only differ in the number of document level encoders, we see that the setup with more document encoder layers performs slightly better (80.44 vs. 80.7). This can be seen when comparing the two setups (8-2-4) and (8-4-4). But the difference is small and the model with more document encoders only outperforms the smaller model in 5 of the 8 tasks.

THF excels on the CoLA task. One thing that stands out is that all of the THF architectures seem to excel on the CoLA task. Table 4.4 shows that the (6-2-4) configuration with a hidden size of 768 for example performs almost on par with DistilBERT and the same configuration with a wider bottleneck outperforms it. The configurations with larger hidden sizes even outperform the original BERT architecture, which is not the case for any other task. This is remarkable since [37] reports that approaches that distill BERT into smaller architectures, often perform worse on this task. On the other hand, performance on CoLA can be strongly influenced by hyperparameters [41].

Model	**dim**	**CoLA**	**Average**
THF-12-4-4	1024	**58**	78.46
THF-8-2-2	1024	53.2	77.1
THF-10-2-4	1024	**58**	78.3
THF-6-2-4	1024	53.2	77.43
THF-8-2-4	1024	56.2	77.41
THF-8-4-4	1024	55.6	77.56
THF-6-2-4	768	50.5	76.2
THF-6-2-4 (bottleneck width=2)	768	51.9	76.3
Baselines			
$\text{BERT}_{\text{Base}}$-Orig	768	56.3	83
$\text{BERT}_{\text{Base}}$-Cram	768	56.5	80.66
Cramming-12	768	44.5	77.94
DistilBERT	768	51.3	79.6
Cramming-8	1024	42.7	63.04
Cramming-8	768	38.7	76.33

Table 4.4: Results on the GLUE dev set for the CoLA task. The average column is the overall score for all GLUE tasks. BERT-6 is not in this table because the score on CoLA is not reported in the paper.

Pre-train and distill. Inspired by the "well-read students" paper, we've applied the same training regime to a THF model to see if the scores can be boosted. After pre-training a (8-2-4) configuration with a hidden dimension of 1024, we've used $\text{BERT}_{\text{LARGE}}$ as a teacher to continue pre-training with the distillation step. The results can be found in table 4.5.
Apart from CoLA, for which this model is almost on par with the largest THF configurations, this approach does not seem to improve the scores by much. Although the distillation step does improve the scores on all tasks except MRPC, the improvements are at most 0.5.

MNLI	**SST-2**	**STSB**	**RTE**	**QNLI**	**QQP**	**MRPC**	**CoLA**	**Avg. (w/o CoLA)**	**Overall Avg.**
79.3	89.9	83.5	54.5	85.7	86	84.9	57	80.54	77.6

Table 4.5: Results of the pre-train & distillation approach for a (8-2-4) THF model.

Comparison to BERT and DistilBERT. Comparing the performance of THF to the original BERT model, we see that the differences for MNLI, STSB and RTE are still quite large. On the other hand, for SST-2, MRPC and CoLA

the THF models are either almost on par with BERT or even outperform it.
The comparison with DistilBERT shows a similar picture, although the differences are smaller in general.

4.3 Masked Perplexity

In addition to the evaluation on the GLUE benchmark, the models are also evaluated and compared with the masked perplexity, as described in 2.3.3. The comparison is done by splitting up texts into chunks of different lengths (multiples of 16 in this case) and making sure that all of these buckets have the same number of samples. The lengths of texts are measured by the number of tokens. The models are evaluated on unseen text data sampled from the Openwebtext corpus [42]. The evaluations in this section are done on the *unpacked* version (i.e. individual sentences, hence sentence attention is being done) unless stated otherwise.
Since a probability distribution over the whole vocabulary is needed to calculate the perplexity, the whole THF model including the decoder is used. Also, the tokenizers for all the compared models are identical. Otherwise, it would be difficult to compare the perplexities of the models.

Comparing the perplexity scores of $\text{BERT}_{\text{BASE}}$ to the THF model with a (10-2-4) configuration and a hidden dimension of 1024 (figure 4.1), one can see that for longer text sequences (starting from lengths 256), the THF model has a lower perplexity. It's also worth noting that the THF model has a much lower variance than BERT. This is surprising since BERT has a much higher score on the GLUE task. Figure A.1 shows the same finding for $\text{BERT}_{\text{LARGE}}$ and table 4.6 shows some more statistics for the models discussed in this section.
The same holds for the comparison of DistilBERT against a (6-2-4) configuration of THF with a hidden dimension of 768, which can be seen in figure 4.2. Except the difference in perplexity scores between DistilBERT and THF is even larger for longer sequences.
Interestingly enough though, the perplexity scores of the 6-layer "well-read students" model are lower than the 6-layer THF model and therefore also lower than the scores of the original BERT models. Even the variances of the scores are lower than the THF model. This can be seen in figure A.2.

Why do THF models have lower perplexity but worse GLUE scores?

There are two possible explanations for this phenomenon:

1. The design of the architecture, as described in section 3.1, operates on shorter sequences, which could make it easier for the model to handle longer sequences. This stems from the fact that each sentence in the input sequence is processed individually and hence there's no cross-contamination

[43] happening between sequences. In other words, since the input sequence is split up into sentences, the attention mechanisms of the *sentence-level encoders* can operate on individual sentences, without having to deal with cross-contamination. Although in theory, the length of the sequence shouldn't make any difference, since BERT was explicitly trained on sequences of up to 512 tokens.

2. Another possibility is that the document-level layers actually learned some relations between sentence embeddings, but it's not reflected in the GLUE scores. A reason for that could be that the fine-tuning approach needs some tweaks to make it work for sentence-level inputs. Nevertheless, the plots serve as yet another proof that the compression works, and therefore the sentence embeddings contain all of the information needed to do language modeling with it.

Of course, it could also be a combination of both above-mentioned factors.

Model	**Median**	**Mean**	**Std.**
$\text{BERT}_{\text{BASE}}$	9.16	12.75	**11.56**
THF-10-2-4	**8.47**	**10.96**	12.15
DistilBERT	15.15	18.4	15.84
THF-6-2-4	9.56	12.54	14.83

Table 4.6: Statistics for the perplexities of the models shown in the plots in figure 4.1 and 4.2. The (10-2-4) configuration has a hidden dimension of 1024, while the (6-2-4) configuration has a hidden dimension of size 768.

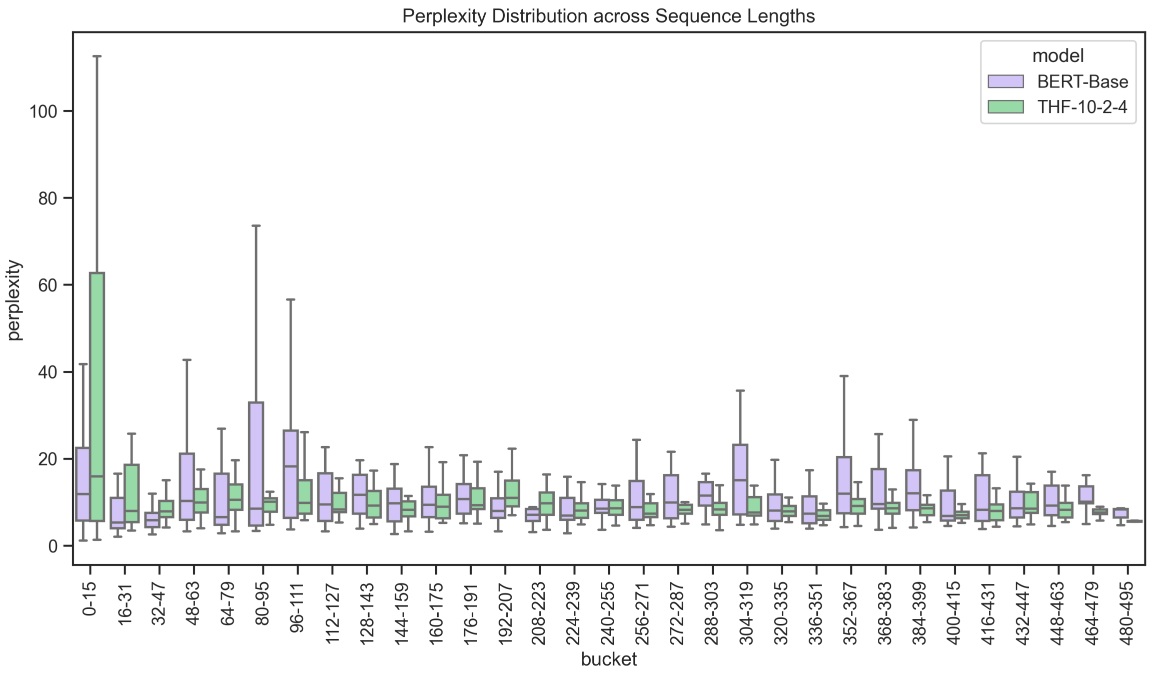


Figure 4.1: Masked perplexity for BERT$_{\text{BASE}}$ and THF-10-2-4 with a hidden dimension of 1024. All of the buckets have the same number of text samples.

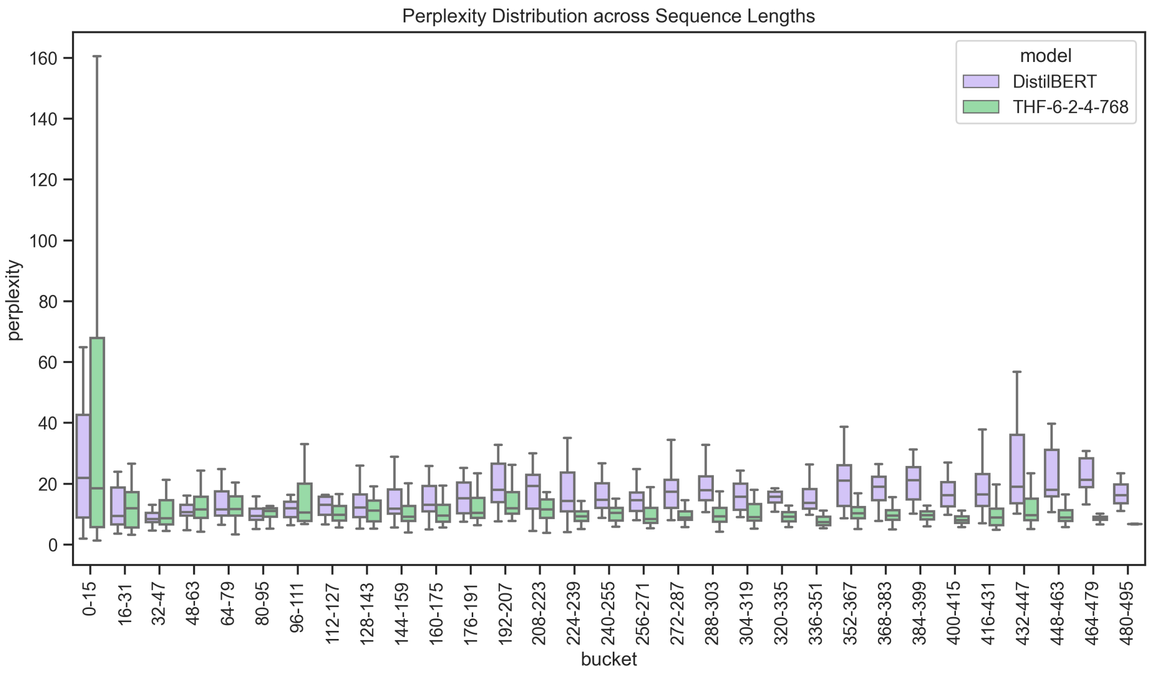


Figure 4.2: Masked perplexity for DistilBERT and THF-6-2-4 with a hidden dimension of 768. All of the buckets have the same number of text samples.

4.3.1 Packed vs. Non-packed

To get an idea of how a non-packed version of THF could perform if it would work on downstream tasks, we plot the perplexity scores of the THF-6-2-4 configuration with the only difference that for one model we supply a packed sequence whereas for the other we use individual sentences as input. 4.3 shows the differences very clearly. While the scores for the first bucket are identical (since these sequences all consist of only one sentence), we notice that the longer the sequences get, the larger the differences are. That applies to both the median as well as the variance. Note that the same model is used. It's only the shape of the data that's different.

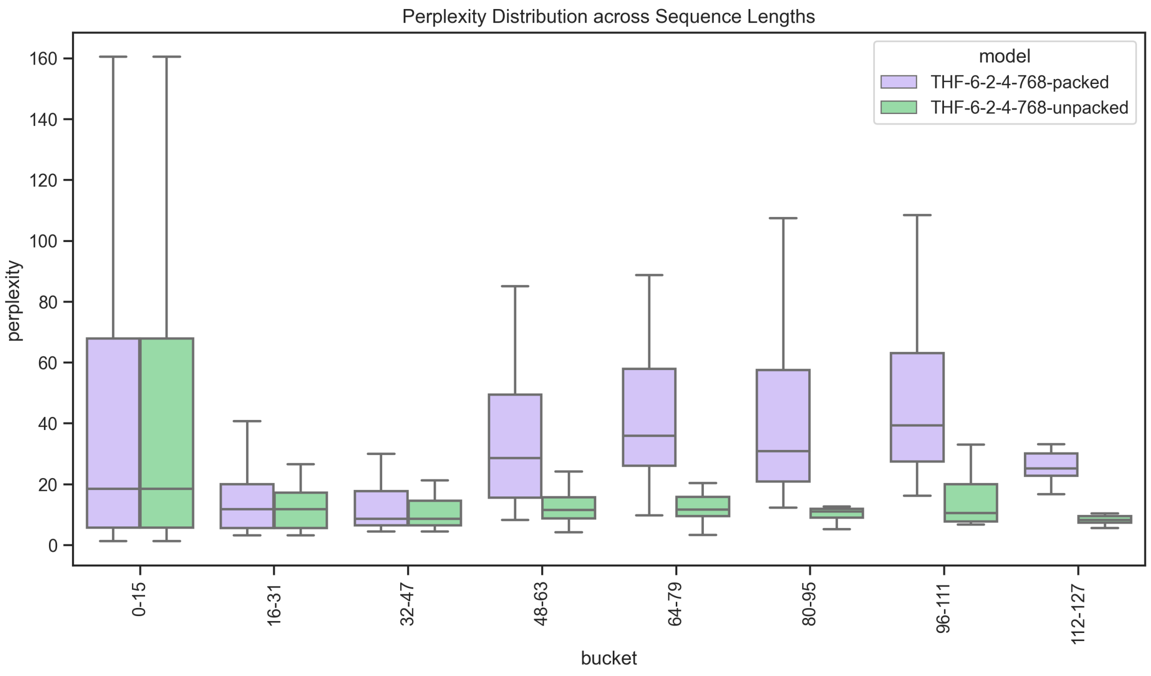


Figure 4.3: Perplexity scores for a THF-6-2-4 model. Purple are the scores for the packed input, green are the scores for the unpacked input.

4.4 The impact of more document-level layers on perplexity

Do more document-level encoder layers have an impact on language modeling performance? In this section, we compare two THF models with the same configuration with the only difference that one model has only 2 document-level encoders, while the other has 4 document-level encoders. The hypothesis is that

with more document-level encoders, the model can better capture relationships between sentences. If this is the case, it would provide further evidence that the document-level encoder learned something about relationships between sentences.
Figure 4.4 shows the perplexities for two models with the same data setup as described in 4.3. It shows that for every bucket, the model with more document-level encoders has a lower median as well as a lower standard deviation.
Table 4.7 shows the statistics over all buckets. The detailed statistics can be found in appendix B.

Model	**Median**	**Mean**	**Std.**
THF-8-2-4	8.96	11.32	10.53
THF-8-4-4	7.88	10.24	10.92

Table 4.7: Overall statistics of the two THF models.

Comparing the values in table 4.7 with the values in table 4.6, we see an interesting fact. Namely, the THF configuration with 10 sentence-level encoders but only 2 document-level encoders is in between the (8-2-4) and (8-4-4) configurations when looking at the perplexity scores. As can be seen in table 4.8, the (8-4-4) configuration has the same number of parameters as the (10-2-4) configuration. This supports the hypothesis that more document-level layers help with language modeling.

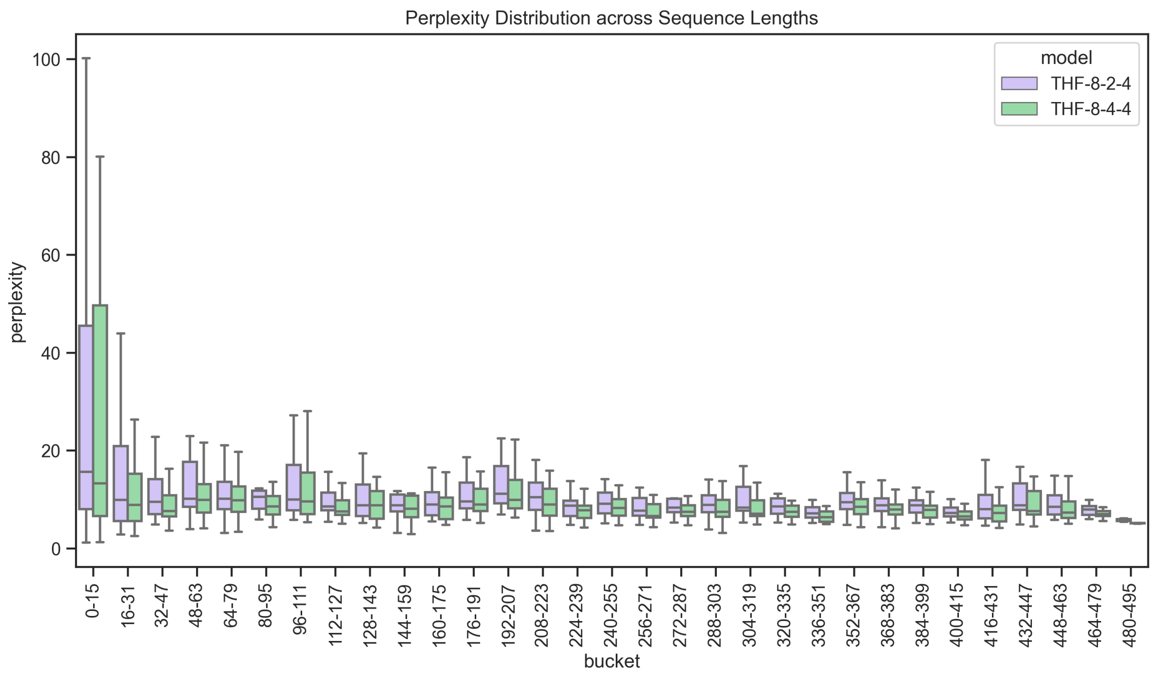


Figure 4.4: Masked perplexity for THF-8-2-4 and THF-8-4-4 with a hidden dimension of 1024. All of the buckets have the same number of text samples.

4.5 Model sizes

For the sake of completeness, we also report the size of the models in the number of parameters. We make a distinction between the number of parameters used for pre-training and inference.

Model	**dim**	**# Parameters for pre-training (in millions)**	**# Parameters for inference (in millions)**
BERT$_{\text{BASE}}$-Orig	768	110	110
DistilBERT	768	66	66
Cramming-8	768	71	71
BERT-6	768	67	67
THF-12-4-2	1024	192	174
THF-8-2-2	1024	138	121
THF-10-2-4	1024	174	139
THF-6-2-4	1024	138	104
THF-8-2-4	1024	156	121
THF-dist-8-2-4	1024	156	121
THF-6-2-4	1024	138	104
THF-8-4-4	1024	174	139
THF-6-2-4	768	138	104

Table 4.8: Model sizes for different BERT and THF configurations.

Chapter 5

Speed experiments

In this section, we report and analyze the speed of the different sizes of THF models compared to the BERT architecture. We focus on inference, hence for THF the decoder blocks are not taken into account.
We split the results of the speed measurements into two sections. One section is for the packed version, which was used for the GLUE scores. The other section describes the results of the non-packed version.

5.1 Packed sequence

5.1.1 speed comparison of the whole architectures

Theoretical performance

To measure the theoretical performance of the models, the calflops library [44] is used, which supports calculations of parameters, FLOPS & MACs for PyTorch-based models. The results can be found in table 5.1.

Model	**dim**	**batch-size**	**# tokens**	**GFLOPS**	**GMACs**
THF-12-4-4	1024	1	128	30.71	15.34
THF-12-4-4	1024	128	16384	3930	1960
THF-8-2-2	1024	1	128	20.46	10.22
THF-8-2-2	1024	128	16384	2620	1310
THF-10-2-4	1024	1	128	25.56	12.77
THF-10-2-4	1024	128	16384	3270	1630
THF-6-2-4	1024	1	128	15.35	7.67
THF-6-2-4	1024	128	16384	1970	981.94
THF-6-2-4	768	1	128	10.0	5.0
THF-6-2-4	768	128	16384	**1280**	**639.56**
Baselines					
$\text{BERT}_{\text{Base}}$	768	1	128	22.36	11.17
$\text{BERT}_{\text{Base}}$	768	128	16384	2860	1430
BERT-6	768	1	128	11.18	5.59
BERT-6	768	128	16384	1430	715.19
Cramming-8	768	1	128	13.3	6.64
Cramming-8	768	128	16384	1700	850.5

Table 5.1: Theoretical number of Giga-FLOPS and Giga-MACs for different model configurations and batch sizes. BERT-6 is the 6-layer BERT model of the "Well-read students" paper. **Bold** denotes the lowest number of operations for batch size 128, violet denotes the lowest number of operations for batch size 1.

Findings:

- Comparing the smallest THF model (6-2-4) to the 8-layer BERT, we see that the whole model uses roughly 25% less operations for GFLOPS & GMACs than the 8-layer BERT for batch-sizes 1 as well as 128.
- Even with a larger hidden dimension (1024), the (6-2-4) model is still 20% more efficient than the 8-layer BERT with a hidden dimension of 768.
- Comparing the (6-2-4)-models with dimensions 768 and 1024, we see that the smaller hidden dimensions make a difference of about 35%.

Notice that the number of FLOPS is roughly two times the number of MACs for all measurements. Hence, if there's nothing noteworthy between the two, we only report FLOPS from here on out. This stems from the fact that for MACs, Multiply-Accumulate operations are counted as one, whereas for FLOPS that's two operations.

Empirical inference speed

The results of the speed measurements can be found in table 5.2.

- Overall, the BERT models are faster when processing only 1 sample, even though the GFLOPS and GMACs are lower. For a batch size of 128, the THF models are faster. Hence, the fastest model for batch size 1 is the 8-layer BERT, which is faster than the (6-2-4) configuration of THF.
- On the other hand, the fastest model for batch size 128 is the (6-2-4) configuration of THF with embedding size 768.
- Figure 5.1 shows that the 8-layer BERT model is only faster for batch sizes smaller than 8. While the inference speed for both models grows linearly, as the batch size gets larger the difference between the THF model and the BERT model gets bigger. One hypothesis for this behavior is that the THF model does have an overhead caused by the reshaping and padding operations. So the efficiency gains only kick in when the batch size grows.

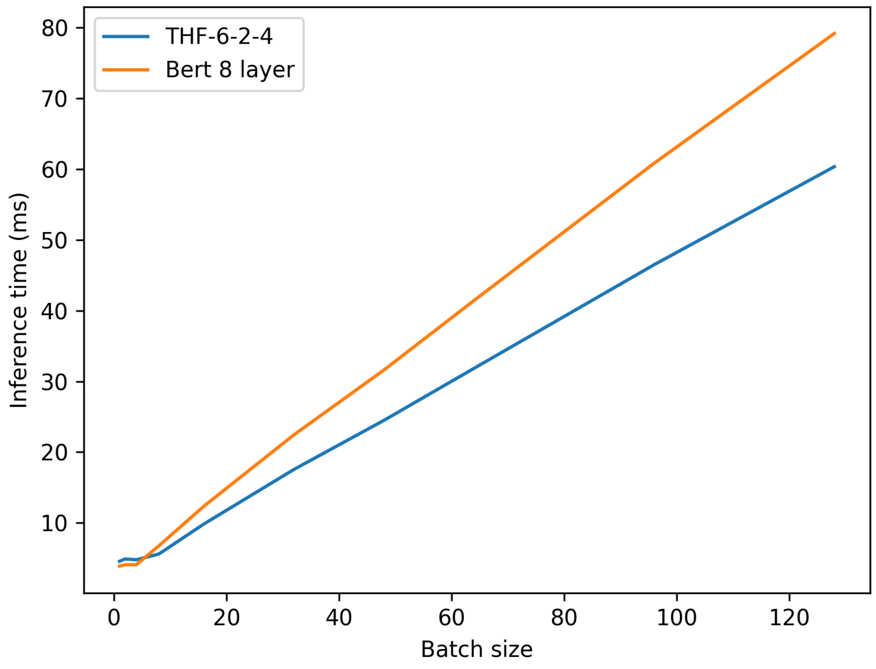


Figure 5.1: Inference speed for difference batch sizes.

Model	**dim**	**batch size**	**tokens**	**median inference time per batch (ms)**	**avg. inference time per batch (ms)**	**std. dev.**	**avg. tokens per second**
THF-12-4-4	1024	1	128	8.23	8.12	0.84	15552
THF-12-4-4	1024	128	16384	162.3	162.26	0.48	100976
THF-8-2-2	1024	1	128	5.28	5.38	0.73	23813
THF-8-2-2	1024	128	16384	109.76	109.69	0.35	149368
THF-10-2-4	1024	1	128	6.26	6.39	0.72	20045
THF-10-2-4	1024	128	16384	136.37	136.35	0.43	120162
THF-6-2-4	1024	1	128	4.28	4.36	0.55	29331
THF-6-2-4	1024	128	16384	81.14	81.09	0.54	201924
THF-6-2-4	768	1	128	4.25	4.38	0.71	29224
THF-6-2-4	768	128	16384	**60.46**	**60.45**	0.07	270984
Baselines							
$\text{BERT}_{\text{Base}}$	768	1	128	6.95	7.1	0.58	17428
$\text{BERT}_{\text{Base}}$	768	128	16384	137.77	137.76	0.59	118930
BERT-6	768	1	128	4.07	4.03	0.22	31415
BERT-6	768	128	16384	81.64	81.31	2.14	200691
Cramming-8	768	1	128	3.81	3.74	0.39	33596
Cramming-8	768	128	16384	79.28	79.25	0.09	206666
Cramming-10	1024	1	128	4.52	4.5	0.08	28296
Cramming-10	1024	128	16384	134.52	134.49	0.43	121792
Cramming-12	1024	1	128	5.36	5.31	0.24	23900
Cramming-12	1024	128	16384	161.48	161.57	0.59	101463

Table 5.2: Measured inference times for different model configurations and batch sizes. For each configuration, the average speed of 100 samples was taken. The inference was run on a single GeForce RTX 3090 GPU.
Cramming-x denotes a BERT model with x layers trained with the Crammed-BERT framework. **Bold** denotes the lowest inference time for batch size 128, violet denotes the lowest inference time for batch size 1.

5.1.2 Document level block vs. BERT encoder block

While the above sections discuss the performance of the models as a whole, in this section we want to compare the transformer blocks in isolation. To that end, we compare the speed of just one BERT encoder to the speed of one document-level encoder.

Model	**dim**	**MACs** (%) **of one layer**	**FLOPS** (%) **of one layer**	**MFLOPS total**
BERT$_{\text{Base}}$	768	8.33	8.32	
BERT-8-Cram	768	12.5	12.48	1660
THF-12-4-4	1024	0.06493	0.06488	19.93
THF-8-4-4	1024	0.09727	0.0972	19.93
THF-6-2-4	768	0.12985	0.12976	12.99

Table 5.3: Total and relative amount of Mega-MACs (MMACs) and Mega-FLOPS (MFLOPS) of a BERT layer compared to different configurations of a single THF layer.

Theoretical performance

Table 5.3 shows the theoretical performance of THF document layers and BERT encoder layers. We see that the total number of MACs and FLOPS is only influenced by the hidden dimension of the model.
While one encoder layer of BERT makes up roughly $\frac{1}{n}$ (n = number of layers) operations for a BERT model, the document encoder layers are in the range of one-tenth of a percent. The large difference in performance can also be seen in the total number of operations. That's not surprising since in the THF model, only one embedding is used for a sequence compared to 128 embeddings in BERT. The results for all configurations can be found in table C.3.

Empirical performance

As can already be seen in table 5.2, the efficiency gains of the THF model appear for larger batch sizes. It becomes even more apparent when comparing just one document encoder layer to one BERT layer, as depicted in figure 5.2 (denoted as *THF 1-128*). While the runtime for BERT layers grows linearly in the number of samples, the runtime of the document layer is (almost) constant.
This makes sense when we have a look at the shape of the data that is processed by both layers. For an input shape *(batch size, sequence length, hidden dimension)*, the BERT layer has to process batches of size *(batch size, 128, 768)*, while the body layer only has to handle batches of size *(batch size, 1, 768)*. Here, the second dimension corresponds to the sequence length. Since the attention mechanism is the part that needs the most compute and is also dependent on the length of the input, this explains the difference in run times.
Let's have a closer look at the reason why the document-level encoders seem to be able to process the inputs in almost constant time: Let n be the length of an arbitrary input sequence. As we know, the complexity of the attention mechanism is $\mathcal{O}(n^2)$. Now, if the input length can somehow be compressed by a

factor of m, the attention mechanism operating on the compressed representation would lead to $\mathcal{O}((\frac{n}{m})^2) = \mathcal{O}(\frac{n^2}{m^2})$. In our case, m is given by the average size of all sentences in the sequence. For a given sequence m is constant, hence it does not change the complexity class. But the denominator still grows quadratically.

A simplified example which illustrates this point: Let's assume an input length of 128, which consists of four sentences with varying lengths. A standard BERT layer has to apply attention to 128 tokens, which would result in 16384 operations. In contrast, a doc-level encoder only has to apply attention to a sequence of input length four, which results in 16 operations. This emphasizes the fact that the efficiency of these layers is directly dependent on the length of the sentences. Also, since in our case n does not grow very large, it can make the impression that the document-level layers can handle the input lengths in constant time, even though it's still in the quadratic complexity class.

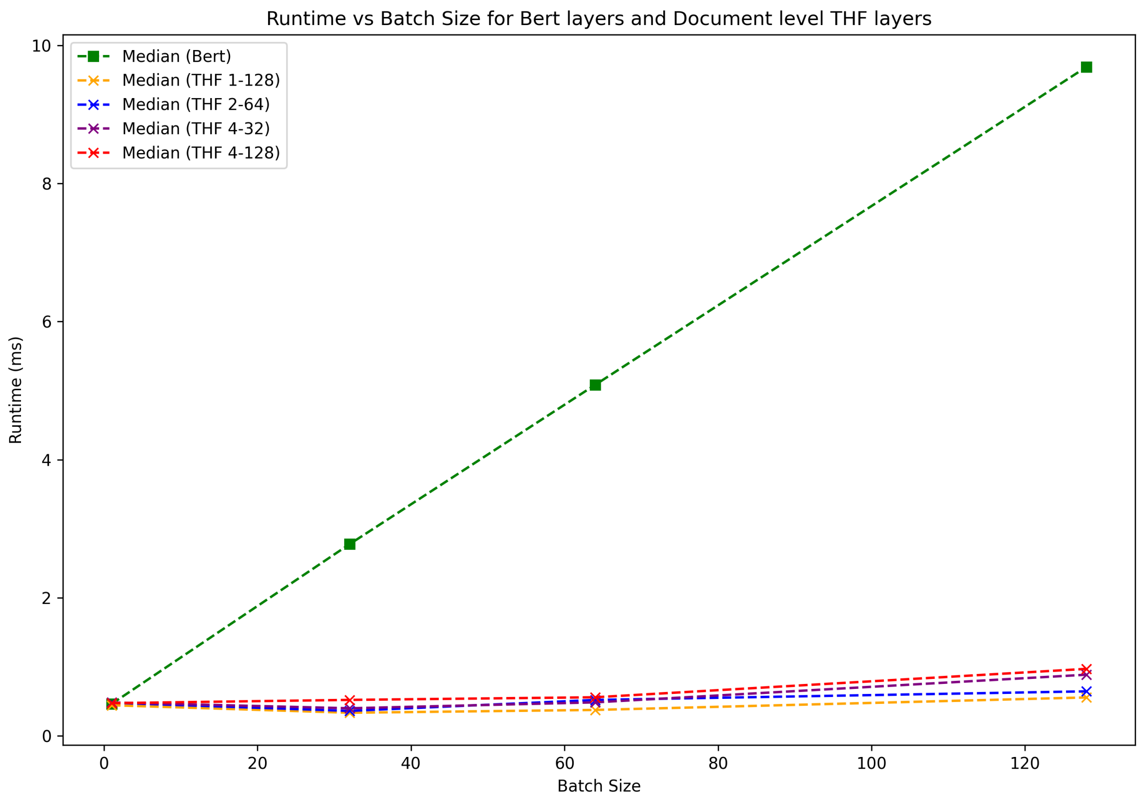


Figure 5.2: Runtime of one THF document layer vs. one BERT layer for batch sizes 1, 32, 64 & 128. To keep it comparable, the measurements of both models were done within the Crammed-BERT framework. (1-128) stands for the version with one input sentence with a length of 128 tokens, (2-64) stands for 2 input sentences with a length of 64 each, etc.

5.2 Non-packed sequences

Along with the results of the packed version of THF, we also report the speed measurements of THF models which operate on individual sentences and compare them to their BERT counterparts. Table 5.2 shows the theoretical as well as the empirical speed comparisons for selected THF configurations and their corresponding baselines for batch size 1. By corresponding baseline, we mean that the number of sentence-level encoders and doc-level encoders added is either equal or almost equal to the total number of BERT layers. Table 5.2 shows the

same but for batch size 128.
Since the goal of this evaluation is to compare sequence lengths of 512, we compare it to traditional BERT models, as the Crammed-BERT version is only trained on sequence lengths up to 128 tokens. This also highlights a strength of the THF approach. The individual sentences are capped at 128 tokens. But this is not a disadvantage, because sentences are rarely longer than 128 tokens, and processing longer sequences is done by splitting up the input into sentences, which cannot be done with traditional BERT approaches.

The results show that it does not matter how exactly the inputs are split up. It's very robust to differences in input shapes. Whether the input consists of many shorter sequences or a few longer sequences, the differences are marginal. It again becomes apparent that the advantages of the THF architecture come to light with larger batch sizes where efficiency gains of up to 45% were measured for the THF-6-2-4 vs. the 6-layer BERT (even though the THF needs 8 layers for a forward pass in this case). The full results with more THF configurations can be found in appendix C.

Figure 5.2 shows how the unpacked version compares to the packed version as well as to the BERT layers. Since the comparison is done within the Crammed-BERT framework, the focus is on sequence lengths of up to 128 tokens. Hence, we've compared input shapes of one sentence of length 128, two sentences of each length 64, and four sentences of length 32. Consequently, all versions process sequences of length 128. Additionally, to show how efficient these layers are, we've added one forward pass which processes 4 sentences, each of length 128. This might seem unfair since in that case 512 tokens are being processed. But we see that it is only slightly slower than the other THF options and a lot faster than the BERT layer (which in this case is also only processing 128 tokens at a time).

Model	**Sequence length**	**number of sentences**	**GFLOPS**	**Median inference time per Batch(ms)**	**Std.**	**Tokens per second**
THF-10-2-4	128	4	102.23	6.81	1.09	72262
THF-10-2-4	32	16	102.71	6.8	1.03	73037
THF-6-2-4 (768 dim.)	128	4	40	4.64	0.81	107000
THF-6-2-4 (768 dim.)	32	16	40.31	4.9	0.8	101743
THF-6-2-4	128	4	61.4	4.68	0.76	106604
THF-6-2-4	32	16	61.88	4.61	0.82	107576
Baselines						
BERT$_{\text{BASE}}$	512	-	96.72	7.52	0.27	66774
BERT-6	512	-	48.36	**4.29**	1.21	108442

Table 5.4: Speed measurements for selected THF configurations and the baselines for batch-size 1. All of the THF models have a hidden dimension of 1024 unless stated otherwise. The baseline's hidden dimension is 768.

Model	**Sequence Length**	**number of sentences**	**TFLOPS**	**Median inference per Batch (ms)**	**std.**	**Tokens per second**
THF-10-2-4	128	4	6.54	265.20	1.25	123447
THF-10-2-4	32	16	6.57	258.46	1.29	126851
THF-6-2-4 (768 dim.)	128	4	2.56	112.31	0.47	291676
THF-6-2-4 (768 dim.)	32	16	2.58	**109.13**	0.27	300244
THF-6-2-4	128	4	3.93	157.58	0.37	207906
THF-6-2-4	32	16	3.96	154.53	0.47	212003
Baselines						
BERT$_{\text{BASE}}$	512	-	6.19	348.12	1.94	94052
BERT-6	512	-	3.10	198.79	0.82	164840

Table 5.5: Speed measurements for selected THF configurations and the baselines for batch-size 128.

Chapter 6

Conclusion and Outlook

This thesis introduces a transformer-based language model - trained on the masked token prediction task - that can operate on dense sentence representations. We were able to show that it's possible to compress the meaning of whole sentences into dense embeddings of the size 768 or 1024 and not only reconstruct it but to do language modeling with it. The approach is implemented in the Crammed-BERT framework, making it efficient to pre-train.

The model is evaluated on multiple different tasks. First and foremost on the GLUE benchmark, on which it achieves scores close to those of BERT, despite operating on embeddings that are up to 128 times smaller than the ones BERT is operating on.
We also did an extensive evaluation of the efficiency of the model regarding FLOPS and MACs. Along with the evaluation of the efficiency of the model as a whole, we've also evaluated and compared the speed of the document-level layers to the speed of traditional BERT layers, finding speed gains in the order of magnitudes. We were also able to show that the architecture excels on larger batch sizes, achieving a speedup of up to 45% compared to traditional BERT architectures. All of the efficiency evaluations were done theoretically as well as empirically.

Although the attention on a sentence level did not work for the downstream GLUE tasks, we were able to show that for longer sequences (consisting of multiple sentences), the perplexity scores were lower than the ones from BERT or DistilBERT. Subsequently, we were also able to show that more document-level encoders help with language modeling (i.e. lower perplexity). This hints at the fact that the attention mechanism on the document level did learn something about the relationships of sentences, opening doors for future work which could lead to large improvements in the language model's abilities to process longer sequences.

6.1 Limitations and future work

Sentence-level attention. The biggest impact on the NLP community could be had if the attention mechanism over sentence embeddings worked. The possibility of having a BERT-sized model that could attend over 64, 128, or even 256 sentences, each with a maximum length of 128, would mean that the context size of such models could get as large as 32768 tokens. This would be on par with current state-of-the-art large language models like Mixtral, for example. Under the assumption that the sentence embeddings do capture all of the needed information, a transformer block should be able to handle such input lengths without any problem (this would be identical to an input length of 256 tokens, which transformer models can handle with ease). These models could then be evaluated on NLP tasks that contain longer sequences. Like the SCROLLS [45] benchmark for example.

Autoregressive models. Current LLMs are all decoder-only language models, which are very well-suited for text generation. This is where the architecture described in this thesis has a disadvantage since it is an encoder-only architecture, like BERT. Using the compression-based approach for text generation tasks would most certainly need some work to adapt the architecture. Mostly because it's not straightforward to generate text from only a sentence embedding. The decoder layers would most likely need to be involved in some way during inference and fine-tuning.

Evaluate sentence embeddings. As described in section 2.2.2, there exists a whole field of research with the goal of getting good sentence representations. Since it was not the main focus of this thesis, the compressed sentence representations were not evaluated on any tasks. Nevertheless, it would be interesting to see how the sentence embeddings perform on such tasks. This could potentially have an impact on fields like text indexing or retrieval-based applications.

Push the boundaries. Although we've trained and evaluated a lot of different configurations of the THF model, there's still more to be explored. One could push the boundaries and try to make models as small as possible to see how small the models can be made while still having a good performance on downstream tasks. Furthermore, scaling up the models even more would show where the upper limit of compressed sentence embeddings would be.

Improve the bottleneck. Most of the models in this thesis were trained by using the hidden representation of a sequence as the bottleneck. For longer bottlenecks, the first n hidden representations are concatenated. That's a rather naive approach, which probably could be improved by concatenating every n-th embedding for example.

Improve the data loading for pre-training. The data preprocessing described in section 3.2 has some disadvantages, which make the data handling somewhat inflexible. The fact that for each batch only documents with the same number of sentences are processed means that either the whole corpus has to be stored that way (already sentencized) or parts of the corpus have to be loaded into memory (more than just a batch) to construct the next batch, which adds an overhead to the data preprocessing (or data loading, depending on whether it's done on-the-fly or as a preprocessing step before training). The defined constraint to make better use of the VRAM (3.2) suffers from the same drawback. For larger datasets, the handling of the data could become a challenge.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[2] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," in *Improving Language Understanding by Generative Pre-Training*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," 2022.

[5] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” 2017.

[6] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mixtral of experts,” 2024.

[7] A. Mitra, L. D. Corro, S. Mahajan, A. Codas, C. Simoes, S. Agarwal, X. Chen, A. Razdaibiedina, E. Jones, K. Aggarwal, H. Palangi, G. Zheng, C. Rosset, H. Khanpour, and A. Awadallah, “Orca 2: Teaching small language models how to reason,” 2023.

[8] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, “GLUE: A multi-task benchmark and analysis platform for natural language understanding,” in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, T. Linzen, G. Chrupała, and A. Alishahi, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 353–355. [Online]. Available: https://aclanthology.org/W18-5446

[9] W. Foundation. Wikimedia downloads. [Online]. Available: https://dumps.wikimedia.org

[10] OpenAI, :, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger,

V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "Gpt-4 technical report," 2023.

[11] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. Lillicrap, J. baptiste Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser, I. Antonoglou, R. Anil, S. Borgeaud, A. Dai, K. Millican, E. Dyer, M. Glaese, T. Sottiaux, B. Lee, F. Viola, M. Reynolds, Y. Xu, J. Molloy, J. Chen, M. Isard, P. Barham, T. Hennigan, R. McIlroy, M. Johnson, J. Schalkwyk, E. Collins, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, C. Meyer, G. Thornton, Z. Yang, H. Michalewski, Z. Abbas, N. Schucher, A. Anand, R. Ives, J. Keeling, K. Lenc, S. Haykal, S. Shakeri, P. Shyam, A. Chowdhery, R. Ring, S. Spencer, E. Sezener, L. Vilnis, O. Chang, N. Morioka, G. Tucker, C. Zheng, O. Woodman, N. Attaluri, T. Kocisky, E. Eltyshev, X. Chen, T. Chung, V. Selo, S. Brahma, P. Georgiev, A. Slone, Z. Zhu, J. Lottes, S. Qiao, B. Caine, S. Riedel, A. Tomala, M. Chadwick, J. Love, P. Choy, S. Mittal, N. Houlsby, Y. Tang, M. Lamm, L. Bai, Q. Zhang, L. He, Y. Cheng, P. Humphreys, Y. Li, S. Brin, A. Cassirer, Y. Miao, L. Zilka, T. Tobin, K. Xu, L. Proleev, D. Sohn, A. Magni, L. A. Hendricks, I. Gao, S. Ontañón, O. Bunyan, N. Byrd, A. Sharma, B. Zhang, M. Pinto, R. Sinha, H. Mehta, D. Jia, S. Caelles, A. Webson, A. Morris, B. Roelofs, Y. Ding, R. Strudel, X. Xiong, M. Ritter, M. Dehghani, R. Chaabouni, A. Karmarkar, G. Lai, F. Mentzer, B. Xu, Y. Li, Y. Zhang, T. L. Paine,

A. Goldin, B. Neyshabur, K. Baumli, A. Levskaya, M. Laskin, W. Jia, J. W. Rae, K. Xiao, A. He, S. Giordano, L. Yagati, J.-B. Lespiau, P. Natsev, S. Ganapathy, F. Liu, D. Martins, N. Chen, Y. Xu, M. Barnes, R. May, A. Vezer, J. Oh, K. Franko, S. Bridgers, R. Zhao, B. Wu, B. Mustafa, S. Sechrist, E. Parisotto, T. S. Pillai, C. Larkin, C. Gu, C. Sorokin, M. Krikun, A. Guseynov, J. Landon, R. Datta, A. Pritzel, P. Thacker, F. Yang, K. Hui, A. Hauth, C.-K. Yeh, D. Barker, J. Mao-Jones, S. Austin, H. Sheahan, P. Schuh, J. Svensson, R. Jain, V. Ramasesh, A. Briukhov, D.-W. Chung, T. von Glehn, C. Butterfield, P. Jhakra, M. Wiethoff, J. Frye, J. Grimstad, B. Changpinyo, C. L. Lan, A. Bortsova, Y. Wu, P. Voigtlaender, T. Sainath, C. Smith, W. Hawkins, K. Cao, J. Besley, S. Srinivasan, M. Omernick, C. Gaffney, G. Surita, R. Burnell, B. Damoc, J. Ahn, A. Brock, M. Pajarskas, A. Petrushkina, S. Noury, L. Blanco, K. Swersky, A. Ahuja, T. Avrahami, V. Misra, R. de Liedekerke, M. Iinuma, A. Polozov, S. York, G. van den Driessche, P. Michel, J. Chiu, R. Blevins, Z. Gleicher, A. Recasens, A. Rrustemi, E. Gribovskaya, A. Roy, W. Gworek, S. Arnold, L. Lee, J. Lee-Thorp, M. Maggioni, E. Piqueras, K. Badola, S. Vikram, L. Gonzalez, A. Baddepudi, E. Senter, J. Devlin, J. Qin, M. Azzam, M. Trebacz, M. Polacek, K. Krishnakumar, S. yiin Chang, M. Tung, I. Penchev, R. Joshi, K. Olszewska, C. Muir, M. Wirth, A. J. Hartman, J. Newlan, S. Kashem, V. Bolina, E. Dabir, J. van Amersfoort, Z. Ahmed, J. Cobon-Kerr, A. Kamath, A. M. Hrafnkelsson, L. Hou, I. Mackinnon, A. Frechette, E. Noland, X. Si, E. Taropa, D. Li, P. Crone, A. Gulati, S. Cevey, J. Adler, A. Ma, D. Silver, S. Tokumine, R. Powell, S. Lee, M. Chang, S. Hassan, D. Mincu, A. Yang, N. Levine, J. Brennan, M. Wang, S. Hodkinson, J. Zhao, J. Lipschultz, A. Pope, M. B. Chang, C. Li, L. E. Shafey, M. Paganini, S. Douglas, B. Bohnet, F. Pardo, S. Odoom, M. Rosca, C. N. dos Santos, K. Soparkar, A. Guez, T. Hudson, S. Hansen, C. Asawaroengchai, R. Addanki, T. Yu, W. Stokowiec, M. Khan, J. Gilmer, J. Lee, C. G. Bostock, K. Rong, J. Caton, P. Pejman, F. Pavetic, G. Brown, V. Sharma, M. Lučić, R. Samuel, J. Djolonga, A. Mandhane, L. L. Sjösund, E. Buchatskaya, E. White, N. Clay, J. Jiang, H. Lim, R. Hemsley, J. Labanowski, N. D. Cao, D. Steiner, S. H. Hashemi, J. Austin, A. Gergely, T. Blyth, J. Stanton, K. Shivakumar, A. Siddhant, A. Andreassen, C. Araya, N. Sethi, R. Shivanna, S. Hand, A. Bapna, A. Khodaei, A. Miech, G. Tanzer, A. Swing, S. Thakoor, Z. Pan, Z. Nado, S. Winkler, D. Yu, M. Saleh, L. Maggiore, I. Barr, M. Giang, T. Kagohara, I. Danihelka, A. Marathe, V. Feinberg, M. Elhawaty, N. Ghelani, D. Horgan, H. Miller, L. Walker, R. Tanburn, M. Tariq, D. Shrivastava, F. Xia, C.-C. Chiu, Z. Ashwood, K. Baatarsukh, S. Samangooei, F. Alcober, A. Stjerngren, P. Komarek, K. Tsihlas, A. Boral, R. Comanescu, J. Chen, R. Liu, D. Bloxwich, C. Chen, Y. Sun, F. Feng, M. Mauger, X. Dotiwalla, V. Hellendoorn, M. Sharman, I. Zheng, K. Haridasan, G. Barth-Maron, C. Swanson, D. Rogozińska, A. Andreev, P. K. Rubenstein, R. Sang, D. Hurt, G. Elsayed, R. Wang, D. Lacey, A. Ilić,

Y. Zhao, L. Aroyo, C. Iwuanyanwu, V. Nikolaev, B. Lakshminarayanan, S. Jazayeri, R. L. Kaufman, M. Varadarajan, C. Tekur, D. Fritz, M. Khalman, D. Reitter, K. Dasgupta, S. Sarcar, T. Ornduff, J. Snaider, F. Huot, J. Jia, R. Kemp, N. Trdin, A. Vijayakumar, L. Kim, C. Angermueller, L. Lao, T. Liu, H. Zhang, D. Engel, S. Greene, A. White, J. Austin, L. Taylor, S. Ashraf, D. Liu, M. Georgaki, I. Cai, Y. Kulizhskaya, S. Goenka, B. Saeta, K. Vodrahalli, C. Frank, D. de Cesare, B. Robenek, H. Richardson, M. Alnahlawi, C. Yew, P. Ponnapalli, M. Tagliasacchi, A. Korchemniy, Y. Kim, D. Li, B. Rosgen, Z. Ashwood, K. Levin, J. Wiesner, P. Banzal, P. Srinivasan, H. Yu, Çağlar Ünlü, D. Reid, Z. Tung, D. Finchelstein, R. Kumar, A. Elisseeff, J. Huang, M. Zhang, R. Zhu, R. Aguilar, M. Giménez, J. Xia, O. Dousse, W. Gierke, S. H. Yeganeh, D. Yates, K. Jalan, L. Li, E. Latorre-Chimoto, D. D. Nguyen, K. Durden, P. Kallakuri, Y. Liu, M. Johnson, T. Tsai, A. Talbert, J. Liu, A. Neitz, C. Elkind, M. Selvi, M. Jasarevic, L. B. Soares, A. Cui, P. Wang, A. W. Wang, X. Ye, K. Kallarackal, L. Loher, H. Lam, J. Broder, D. Holtmann-Rice, N. Martin, B. Ramadhana, D. Toyama, M. Shukla, S. Basu, A. Mohan, N. Fernando, N. Fiedel, K. Paterson, H. Li, A. Garg, J. Park, D. Choi, D. Wu, S. Singh, Z. Zhang, A. Globerson, L. Yu, J. Carpenter, F. de Chaumont Quitry, C. Radebaugh, C.-C. Lin, A. Tudor, P. Shroff, D. Garmon, D. Du, N. Vats, H. Lu, S. Iqbal, A. Yakubovich, N. Tripuraneni, J. Manyika, H. Qureshi, N. Hua, C. Ngani, M. A. Raad, H. Forbes, A. Bulanova, J. Stanway, M. Sundararajan, V. Ungureanu, C. Bishop, Y. Li, B. Venkatraman, B. Li, C. Thornton, S. Scellato, N. Gupta, Y. Wang, I. Tenney, X. Wu, A. Shenoy, G. Carvajal, D. G. Wright, B. Bariach, Z. Xiao, P. Hawkins, S. Dalmia, C. Farabet, P. Valenzuela, Q. Yuan, C. Welty, A. Agarwal, M. Chen, W. Kim, B. Hulse, N. Dukkipati, A. Paszke, A. Bolt, E. Davoodi, K. Choo, J. Beattie, J. Prendki, H. Vashisht, R. Santamaria-Fernandez, L. C. Cobo, J. Wilkiewicz, D. Madras, A. Elqursh, G. Uy, K. Ramirez, M. Harvey, T. Liechty, H. Zen, J. Seibert, C. H. Hu, M. Elhawaty, A. Khorlin, M. Le, A. Aharoni, M. Li, L. Wang, S. Kumar, A. Lince, N. Casagrande, J. Hoover, D. E. Badawy, D. Soergel, D. Vnukov, M. Miecnikowski, J. Simsa, A. Koop, P. Kumar, T. Sellam, D. Vlasic, S. Daruki, N. Shabat, J. Zhang, G. Su, J. Zhang, J. Liu, Y. Sun, E. Palmer, A. Ghaffarkhah, X. Xiong, V. Cotruta, M. Fink, L. Dixon, A. Sreevatsa, A. Goedeckemeyer, A. Dimitriev, M. Jafari, R. Crocker, N. FitzGerald, A. Kumar, S. Ghemawat, I. Philips, F. Liu, Y. Liang, R. Sterneck, A. Repina, M. Wu, L. Knight, M. Georgiev, H. Lee, H. Askham, A. Chakladar, A. Louis, C. Crous, H. Cate, D. Petrova, M. Quinn, D. Owusu-Afriyie, A. Singhal, N. Wei, S. Kim, D. Vincent, M. Nasr, C. A. Choquette-Choo, R. Tojo, S. Lu, D. de Las Casas, Y. Cheng, T. Bolukbasi, K. Lee, S. Fatehi, R. Ananthanarayanan, M. Patel, C. Kaed, J. Li, J. Sygnowski, S. R. Belle, Z. Chen, J. Konzelmann, S. Põder, R. Garg, V. Koverkathu, A. Brown, C. Dyer, R. Liu, A. Nova, J. Xu, S. Petrov, D. Hassabis, K. Kavukcuoglu, J. Dean, and O. Vinyals, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," 2024.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[13] T. Ding, T. Chen, H. Zhu, J. Jiang, Y. Zhong, J. Zhou, G. Wang, Z. Zhu, I. Zharkov, and L. Liang, "The efficiency spectrum of large language models: An algorithmic survey," 2023.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[15] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[16] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," 2020.

[17] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," 2019.

[18] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. [Online]. Available: https://aclanthology.org/D19-1410

[19] K. Wang, N. Reimers, and I. Gurevych, "TSDAE: Using transformer-based sequential denoising auto-encoderfor unsupervised sentence embedding learning," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 671–688. [Online]. Available: https://aclanthology.org/2021.findings-emnlp.59

[20] I. Montero, N. Pappas, and N. A. Smith, "Sentence bottleneck autoencoders from transformer language models," 2021.

[21] 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 pre-training approach," 2019.

[22] H. Lee, D. A. Hudson, K. Lee, and C. D. Manning, "SLM: Learning a discourse language representation with sentence unshuffling," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 1551–1562. [Online]. Available: https://aclanthology.org/2020.emnlp-main.120

[23] D. Ippolito, D. Grangier, D. Eck, and C. Callison-Burch, "Toward better storylines with sentence-level language models," 2020.

[24] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022.

[25] Z. Dai, G. Lai, Y. Yang, and Q. Le, "Funnel-transformer: Filtering out sequential redundancy for efficient language processing," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4271–4282. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/2cd2915e69546904e4e5d4a2ac9e1652-Paper.pdf

[26] P. Nawrot, S. Tworkowski, M. Tyrolski, L. Kaiser, Y. Wu, C. Szegedy, and H. Michalewski, "Hierarchical transformers are more efficient language models," in *Findings of the Association for Computational Linguistics: NAACL 2022*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds. Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 1559–1571. [Online]. Available: https://aclanthology.org/2022.findings-naacl.117

[27] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, "Lite transformer with long-short range attention," 2020.

[28] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Knight, A. Nenkova, and O. Rambow, Eds. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 1480–1489. [Online]. Available: https://aclanthology.org/N16-1174

[29] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://aclanthology.org/D14-1179

[30] L. Miculicich, D. Ram, N. Pappas, and J. Henderson, “Document-level neural machine translation with hierarchical attention networks,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2947–2954. [Online]. Available: https://aclanthology.org/D18-1325

[31] X. Zhang, F. Wei, and M. Zhou, “HIBERT: Document level pre-training of hierarchical bidirectional transformers for document summarization,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 5059–5069. [Online]. Available: https://aclanthology.org/P19-1499

[32] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[33] T. Rohde, X. Wu, and Y. Liu, “Hierarchical learning for generation with long source sequences,” 2021.

[34] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, “Perplexity—a measure of the difficulty of speech recognition tasks,” *The Journal of the Acoustical Society of America*, vol. 62, no. S1, pp. S63–S63, 1977.

[35] J. Salazar, D. Liang, T. Q. Nguyen, and K. Kirchhoff, “Masked language model scoring,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 2699–2712. [Online]. Available: https://aclanthology.org/2020.acl-main.240

[36] 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,” 2015.

[37] J. Geiping and T. Goldstein, “Cramming: Training a language model on a single gpu in one day,” 2022.

[38] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O’Reilly Media, Inc.", 2009.

[39] L. N. Smith and N. Topin, “Super-convergence: Very fast training of neural networks using large learning rates,” 2018.

[40] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” 2019.

[41] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," 2020.

[42] A. Gokaslan and V. Cohen, "Openwebtext corpus," http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[43] M. M. Krell, M. Kosec, S. P. Perez, and A. Fitzgibbon, "Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance," 2022.

[44] xiaoju ye. (2023) calflops: a flops and params calculate tool for neural networks in pytorch framework. [Online]. Available: https://github.com/MrYxJ/calculate-flops.pytorch

[45] U. Shaham, E. Segal, M. Ivgi, A. Efrat, O. Yoran, A. Haviv, A. Gupta, W. Xiong, M. Geva, J. Berant, and O. Levy, "Scrolls: Standardized comparison over long language sequences," 2022.

Appendix A

Perplexity plots

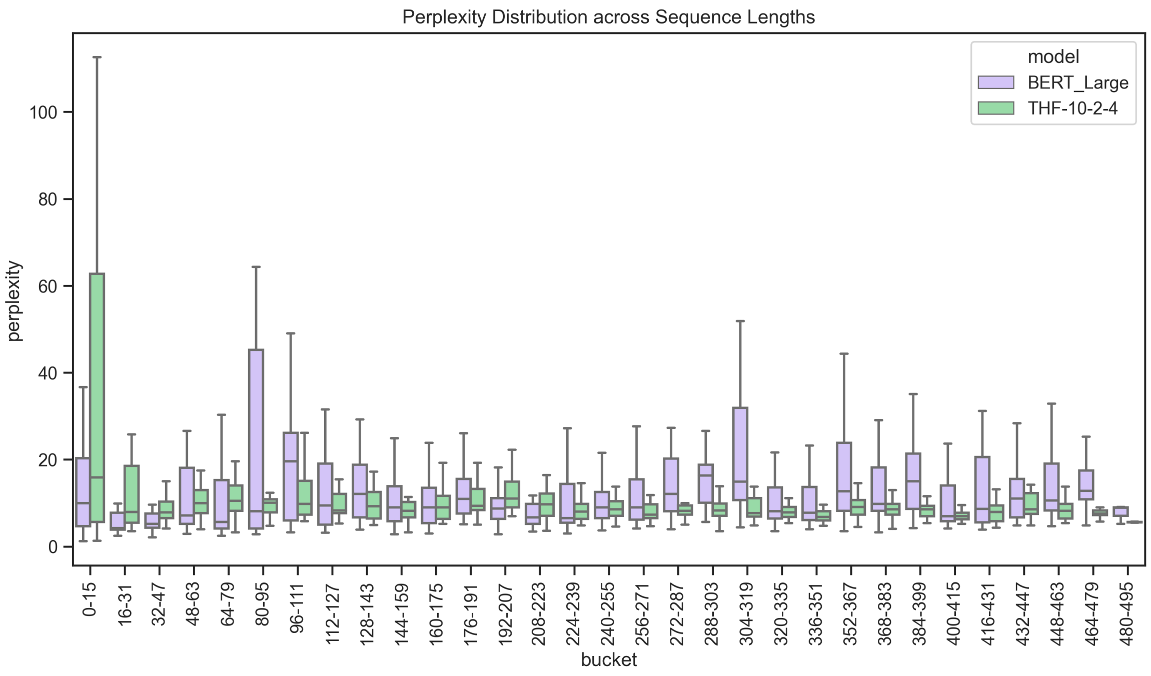


Figure A.1: Masked perplexity for $\text{BERT}_{\text{LARGE}}$ and THF-10-2-4 with a hidden dimension of 1024.

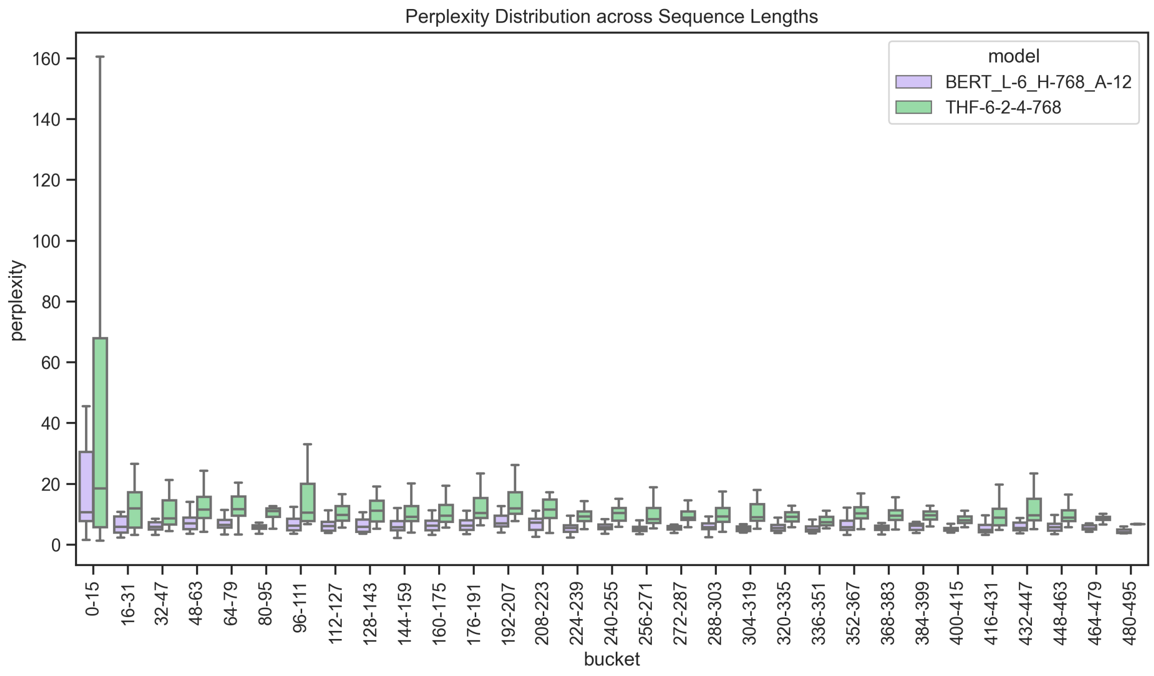


Figure A.2: Masked perplexity for the 6-layer BERT variant of the "well-read sutdents" paper and THF-10-2-4 with a hidden dimension of 1024.

Appendix B

Detailed perplexity results

B.1 Perplexities per bucket for THF-8-2-4

bucket	mean	std	min	50%	max
0-15	31.90	36.35	1.16	15.59	123.35
16-31	14.54	12.19	2.84	9.91	43.91
32-47	11.26	5.85	4.88	9.51	22.74
48-63	12.51	5.62	3.90	10.09	22.92
64-79	11.16	4.48	3.17	10.14	21.05
80-95	10.28	3.35	5.88	10.55	20.32
96-111	13.33	8.31	5.82	9.94	39.64
112-127	10.19	4.31	5.39	8.58	24.24
128-143	14.43	16.07	5.18	8.80	79.31
144-159	11.08	6.49	3.13	8.76	28.12
160-175	9.92	4.40	5.45	8.91	21.71
176-191	11.12	4.69	5.84	9.55	23.43
192-207	13.37	6.43	6.92	11.12	33.62
208-223	10.68	3.74	3.64	10.47	18.04
224-239	8.80	2.62	4.75	8.70	14.66
240-255	9.19	2.55	5.07	9.08	14.10
256-271	8.63	2.76	4.76	7.68	17.10
272-287	9.46	3.53	5.25	8.32	17.38
288-303	9.56	3.81	3.83	8.83	21.60
304-319	9.54	3.11	5.24	8.33	16.83
320-335	12.67	19.72	5.26	8.52	102.56
336-351	8.38	4.15	5.18	7.17	23.73

Table continued from previous page.

bucket	**mean**	**std**	**min**	**50%**	**max**
336-351	8.38	4.15	5.18	7.17	23.73
352-367	9.96	3.55	4.78	9.42	21.05
368-383	8.78	2.45	4.34	8.78	13.91
384-399	8.90	2.33	5.19	8.79	15.12
400-415	7.56	1.59	5.24	7.24	11.23
416-431	10.06	7.79	4.62	7.96	44.63
432-447	10.92	5.71	4.86	8.75	25.71
448-463	9.10	2.76	5.79	8.49	16.81
464-479	8.05	1.59	5.98	7.91	11.78
480-495	5.76	0.47	5.43	5.76	6.09

Table B.1: Statistics for the perplexity scores for each bucket of the 8-2-4 configuration of THF.

B.2 Perplexities per bucket for THF-8-4-4

bucket	**mean**	**std**	**min**	**50%**	**max**
0-15	32.59	41.49	1.23	13.28	184.52
16-31	15.53	18.70	2.50	8.85	73.56
32-47	9.74	5.75	3.57	7.61	26.33
48-63	10.75	4.91	4.10	9.92	22.27
64-79	10.49	4.21	3.37	9.81	19.71
80-95	9.18	3.08	4.35	8.59	16.85
96-111	12.05	7.23	5.33	9.61	33.25
112-127	9.23	4.30	5.00	7.50	22.51
128-143	12.26	12.75	4.20	8.75	63.10
144-159	9.93	6.06	2.89	8.08	25.29
160-175	9.12	3.99	4.79	8.52	19.53
176-191	10.10	3.86	5.18	8.92	19.43
192-207	12.14	6.12	6.30	9.93	32.70
208-223	9.55	3.45	3.54	8.99	15.88
224-239	7.89	2.43	4.26	7.79	14.49
240-255	8.26	2.26	4.74	8.24	12.86
256-271	7.60	2.41	4.30	6.61	15.51
272-287	8.33	3.14	4.74	7.42	15.97
288-303	8.36	3.30	3.11	7.42	18.94
304-319	8.09	2.32	4.85	7.05	13.42
320-335	10.10	12.60	4.82	7.42	67.24
336-351	7.32	3.62	4.95	6.33	21.03
352-367	8.73	2.83	4.28	8.44	16.67
368-383	7.69	2.06	4.05	7.91	11.98
384-399	7.91	2.05	4.91	7.82	12.95
400-415	6.70	1.23	4.74	6.50	9.12
416-431	8.82	6.78	4.16	7.20	38.97
432-447	9.74	4.94	4.50	7.62	21.41
448-463	8.07	2.50	5.04	7.30	14.74
464-479	7.14	1.16	5.56	7.00	9.86
480-495	5.11	0.12	5.03	5.11	5.20

Table B.2: Statistics for the perplexity scores for each bucket of the 8-4-4 configuration of THF.

Appendix C

Full speed measurements

Model	**Sequence length**	**number of sentences**	**GFLOPS**	**Median inference time per Batch(ms)**	**Std.**	**Tokens per second**
THF-12-4-4	4	128	122.803	8.66	0.4	58392
THF-12-4-4	32	16	123.76	8.79	1.07	58451
THF-10-2-4	4	128	102.23	6.81	1.09	72262
THF-10-2-4	32	16	102.71	6.8	1.03	73037
THF-6-2-4 (768 dim.)	4	128	40	4.64	0.81	107000
THF-6-2-4 (768 dim.)	32	16	40.31	4.9	0.8	101743
THF-6-2-4 (1024 dim.)	4	128	61.4	4.68	0.76	106604
THF-6-2-4 (1024 dim.)	32	16	61.88	4.61	0.82	107576
THF-8-2-4	4	128	81.82	5.73	0.99	86367
THF-8-2-4	32	16	82.3	5.75	0.79	87088
Baselines						
Bert-Base	512	-	96.72	7.52	0.27	66774
Bert-6	512	-	48.36	4.29	1.21	108442

Table C.1: Speed measurements for all THF configurations and the baselines for batch-size 1. All of the THF models have a hidden dimension of 1024, unless stated otherwise. The baselines' hidden dimension is 768.

Model	**Sequence Length**	**number of sentences**	**TFLOPS**	**Median inference per Batch (ms)**	**std.**	**Tokens per second**
THF-12-4-4	128	4	7.86	316.32	1.67	103519
THF-12-4-4	32	16	7.92	312.46	1.64	104878
THF-10-2-4	128	4	6.54	265.20	1.25	123447
THF-10-2-4	32	16	6.57	258.46	1.29	126851
THF-6-2-4 (768 dim.)	128	4	2.56	112.31	0.47	291676
THF-6-2-4 (768 dim.)	32	16	2.58	109.13	0.27	300244
THF-6-2-4	128	4	3.93	157.58	0.37	207906
THF-6-2-4	32	16	3.96	154.53	0.47	212003
THF-8-2-4	128	4	5.24	209.17	0.92	156693
THF-8-2-4	32	16	5.27	203.77	0.81	160808
Baselines						
Bert-Base	512	-	6.19	348.12	1.94	94052
Bert-6	512	-	3.10	198.79	0.82	164840

Table C.2: Speed measurements for all THF configurations and the baselines for batch-size 128. All of the THF models have a hidden dimension of 1024, unless stated otherwise. The baselines' hidden dimension is 768.

Model	**dim**	**MACs** (%) **of one layer**	**FLOPS** (%) **of one layer**	**MFLOPS total**
$\text{BERT}_{\text{Base}}$	768	8.33	8.32	
BERT-8-Cram	768	12.5	12.48	1660
THF-12-4-4	1024	0.06493	0.06488	19.93
THF-8-2-2	1024	0.09746	0.09738	19.93
THF-10-2-4	1024	0.078	0.07794	19.93
THF-6-2-4	1024	0.12985	0.12976	19.93
THF-8-2-4	1024	0.09746	0.09738	19.93
THF-6-2-4	1024	0.12985	0.12976	19.93
THF-8-4-4	1024	0.09727	0.0972	19.93
THF-6-2-4	768	0.12985	0.12976	12.99

Table C.3: Total and relative amount of Mega-MACs (MMACs) and Mega-FLOPS (MFLOPS) of a BERT layers compared to different configurations of single THF layers.