Longformer: The Long-Document Transformer

Iz Beltagy* **Matthew E. Peters*** **Arman Cohan***
Allen Institute for Artificial Intelligence, Seattle, WA, USA
{beltagy,matthewp,armanc}@allenai.org

Abstract

Transformer-based models are unable to process long sequences due to their self-attention operation, which scales quadratically with the sequence length. To address this limitation, we introduce the Longformer with an attention mechanism that scales linearly with sequence length, making it easy to process documents of thousands of tokens or longer. Longformer's attention mechanism is a drop-in replacement for the standard self-attention and combines a local windowed attention with a task motivated global attention. Following prior work on long-sequence transformers, we evaluate Longformer on character-level language modeling and achieve state-of-the-art results on `text8` and `enwik8`. In contrast to most prior work, we also pretrain Longformer and finetune it on a variety of downstream tasks. Our pretrained Longformer consistently outperforms RoBERTa on long document tasks and sets new state-of-the-art results on WikiHop and TriviaQA. We finally introduce the Longformer-Encoder-Decoder (LED), a Longformer variant for supporting long document generative sequence-to-sequence tasks, and demonstrate its effectiveness on the arXiv summarization dataset.[1]

1 Introduction

Transformers (Vaswani et al., 2017) have achieved state-of-the-art results in a wide range of natural language tasks including generative language modeling (Dai et al., 2019; Radford et al., 2019) and discriminative language understanding (Devlin et al., 2019). This success is partly due to the self-attention component which enables the network to capture contextual information from the entire sequence. While powerful, the memory and computational requirements of self-attention grow quadratically with sequence length, making it infeasible (or very expensive) to process long sequences.

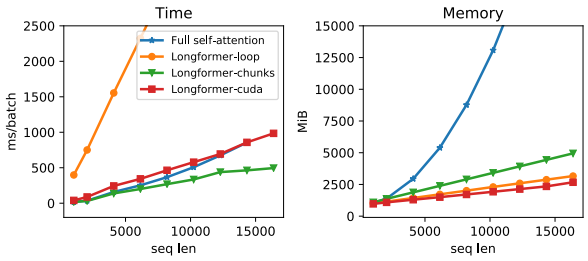


Figure 1: Runtime and memory of full self-attention and different implementations of Longformer's self-attention; `Longformer-loop` is non-vectorized, `Longformer-chunk` is vectorized, and `Longformer-cuda` is a custom cuda kernel implementations. Longformer's memory usage scales linearly with the sequence length, unlike the full self-attention mechanism that runs out of memory for long sequences on current GPUs. Different implementations vary in speed, with the vectorized `Longformer-chunk` being the fastest. More details are in section 3.2.

To address this limitation, we present Longformer, a modified Transformer architecture with a self-attention operation that scales linearly with the sequence length, making it versatile for processing long documents (Fig 1). This is an advantage for natural language tasks such as long document classification, question answering (QA), and coreference resolution, where existing approaches partition or shorten the long context into smaller sequences that fall within the typical 512 token limit of BERT-style pretrained models. Such partitioning could potentially result in loss of important cross-partition information, and to mitigate this problem, existing methods often rely on complex architectures to address such interactions. On the other hand, our proposed Longformer is able to build contextual representations of the entire context using multiple layers of attention, reducing the

* Equal contribution.
[1] https://github.com/allenai/longformer

need for task-specific architectures.

Recent work has addressed the computational inefficiency of Transformers on long sequences (see Tab. 1). However, they primarily focus on autoregressive language modeling (LM), while the application of long document transformers to document-level NLP tasks in the transfer learning setting (Dai and Le, 2015; Peters et al., 2018; Howard and Ruder, 2018; Devlin et al., 2019) has remained largely unexplored. We address this gap and show that Longformer's attention mechanism can act as a drop-in replacement for the self-attention mechanism in pretrained Transformers, and leads to gains across a suite of document NLP tasks.

Longformer's attention mechanism is a combination of a windowed local-context self-attention and an end task motivated global attention that encodes inductive bias about the task. Through ablations and controlled trials we show both attention types are essential – the local attention is primarily used to build contextual representations, while the global attention allows Longformer to build full sequence representations for prediction.

We first evaluate Longformer on autoregressive character-level language modeling using a combination of windowed and a new dilated attention pattern, allowing the model to process sequences of up to 32K characters on modern GPUs. We achieve state-of-the-art results on `text8` and `enwik8` benchmark datasets, demonstrating the effectiveness of Longformer in long document modeling.

Then, to evaluate Longformer's ability to replace the full self-attention operation of existing pretrained models, we pretrain it with the masked language modeling (MLM) objective, continuing from the RoBERTa (Liu et al., 2019) released checkpoint. After pretraining, we apply it to downstream language tasks through finetuning and demonstrate that Longformer consistently outperforms RoBERTa on a wide range of document-level natural language tasks including text classification, QA, and coreference resolution, achieving state-of-the-art results on two of these datasets.

We finally introduce a variant of Longformer which instead of an encoder-only Transformer architecture, it follows an encoder-decoder architecture similar to the original Transformer model (Vaswani et al., 2017), and it is intended for sequence-to-sequence (seq2seq) learning (Sutskever et al., 2014). We call this model Longformer-Encoder-Decoder (LED) that uses Longformer's efficient attention pattern on the encoder network, allowing it to address long document seq2seq tasks such as summarization. We demonstrate the effectiveness of LED on the arXiv summarization dataset (Cohan et al., 2018).

Model	attention matrix	char-LM	other tasks	pretrain
Transformer-XL (2019)	ltr	yes	no	no
Adaptive Span (2019)	ltr	yes	no	no
Compressive (2020)	ltr	yes	no	no
Reformer (2020)	sparse	yes	no	no
Sparse (2019)	sparse	yes	no	no
Routing (2020)	sparse	yes	no	no
BP-Transformer (2019)	sparse	yes	MT	no
Blockwise (2019)	sparse	no	QA	yes
Our Longformer	sparse	yes	multiple	yes

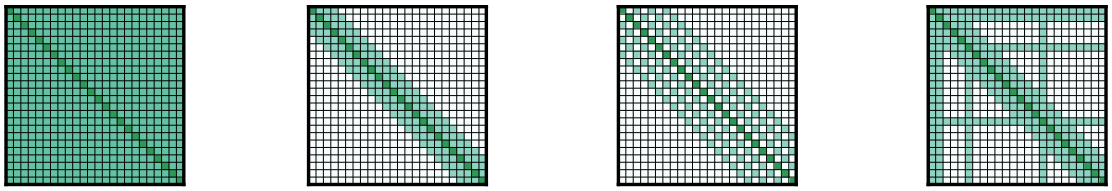
Table 1: Summary of prior work on adapting Transformers for long documents. ltr: left-to-right.

2 Related Work

Long-Document Transformers Tab. 1 summarizes recent prior work on long documents. Two types of self-attention approaches have been explored. The first is a left-to-right (ltr) approach that processes the document in chunks moving from left-to-right. While such models have been successful in autoregressive language modeling, they are unsuitable for transfer learning approaches with tasks that benefit from bidirectional context.

Our work falls within the other general approach that defines some form of sparse attention pattern and avoids computing the full quadratic attention matrix multiplication. The model with the most similar attention pattern to ours is Sparse Transformer (Child et al., 2019), which uses a form of dilated sliding window of blocks of size 8x8 provided by BlockSparse (Gray et al., 2017). Our implementation (§3) also includes a custom CUDA kernel, but it is more flexible and maintainable than BlockSparse which is implemented in C++, and designed for a specific version of TensorFlow. We also introduce additional task motivated global attention patterns suitable for common NLP tasks (§3) and show they are essential for good performance in the transfer learning setting.

A few models tried tasks other than autoregressive language modeling, which is a step forward because arguably focusing on language modeling as the primary evaluation has led to the development of models with limited applicability. BP-Transformer (Ye et al., 2019) evaluated on machine



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

translation (MT), but didn't explore the pretrain-finetune setting. Blockwise attention (Qiu et al., 2019) pretrained their models and evaluated on question answering (QA). However, the evaluation is limited as it doesn't include language modeling, and the QA datasets are of relatively short documents,[2] therefore the effectiveness of this model on long document tasks remains unexplored.

Task-specific Models for Long Documents Many task-specific approaches have been developed to workaround the 512 limit of pretrained transformer models like BERT. The simplest approach just truncates the document, commonly used for classification (Xie et al., 2019). Another approach chunks the document into chunks of length 512 (could be overlapping), processes each chunk separately, then combines the activations with a task specific model (Joshi et al., 2019). A third approach popular for multihop and open domain QA tasks uses a two-stage model where the first stage retrieves relevant documents that are passed onto the second stage for answer extraction (Clark and Gardner, 2017; Chen et al., 2017). All of these approaches suffer from information loss due to truncation or cascading errors from the two stage approach. In contrast, Longformer can process long sequences without truncating or chunking, allowing us to adopt a much simpler approach that concatenates the available context and processes it in a single pass.

A few contemporaneous works[3] have explored similar ideas to Longformer using local + global attention in Transformers, and pre-training it for long document natural language tasks. In particular, ETC (Ainslie et al., 2020) uses a similar local + global attention instead of full self-attention to scale Transformers to long documents. Different from Longformer, ETC uses relative position embeddings (which we only used for the Autoregressive LM setting), introduces an additional training objective (CPC loss) for pre-training, and configures global attention in a slightly different way. It shows strong results on several tasks including reading comprehension and classification. GMAT (Gupta and Berant, 2020) uses a similar idea of few global locations in the input serving as global memory. BigBird (Zaheer et al., 2020) is an extension over ETC with evaluation on additional tasks, including summarization. Importantly, through theoretical analysis, BigBird shows that sparse Transformers are universal approximators of sequence functions and preserve these properties of the full self-attention.

3 Longformer

The original Transformer model has a self-attention component with $O(n^2)$ time and memory complexity where n is the input sequence length. To address this challenge, we sparsify the full self-attention matrix according to an "attention pattern" specifying pairs of input locations attending to one another. Unlike the full self-attention, our proposed attention pattern scales linearly with the input sequence, making it efficient for longer sequences. This section discusses the design and implementation of this attention pattern.

3.1 Attention Pattern

Sliding Window Given the importance of local context (Kovaleva et al., 2019), our attention pattern employs a fixed-size window attention surrounding each token. Using multiple stacked layers of such windowed attention results in a large receptive field, where top layers have access to all input locations and have the capacity to build representations that incorporate information across the entire input, similar to CNNs (Wu et al., 2019). Given a fixed window size w, each token attends to $\frac{1}{2}w$ tokens on each side (Fig. 2b). The computation complexity of this pattern is $O(n \times w)$,

[2] SQuAD contexts typically fit within the 512 limit, and MRQA is constructed by dropping long-document examples.

[3] All were published on arXiv after Longformer.

which scales linearly with input sequence length n. In a transformer with ℓ layers, the receptive field size at the top layer is $\ell \times w$ (assuming w is fixed for all layers). Depending on the application, it might be helpful to use different values of w for each layer to balance between efficiency and model representation capacity (§4.1).

Dilated Sliding Window To further increase the receptive field without increasing computation, the sliding window can be "dilated". This is analogous to dilated CNNs (van den Oord et al., 2016) where the window has gaps of size dilation d (Fig. 2c). Assuming a fixed d and w for all layers, the receptive field is $\ell \times d \times w$, which can reach tens of thousands of tokens even for small values of d.

In multi-headed attention, each attention head computes a different attention score. We found settings with different dilation configurations per head improves performance by allowing some heads without dilation to focus on local context, while others with dilation focus on longer context.

Global Attention In state-of-the-art BERT-style models for natural language tasks, the optimal input representation differs from language modeling and varies by task. For masked language modeling (MLM), the model uses local context to predict the masked word, while for classification, the model aggregates the representation of the whole sequence into a special token (`[CLS]` in case of BERT). For QA, the question and document are concatenated, allowing the model to compare the question with the document through self-attention.

In our case, the windowed and dilated attention are not flexible enough to learn task-specific representations. Accordingly, we add "global attention" on few pre-selected input locations. Importantly, we make this attention operation symmetric: that is, a token with a global attention attends to all tokens across the sequence, and all tokens in the sequence attend to it. Fig. 2d shows an example of a sliding window attention with global attention at a few tokens at custom locations. For example for classification, global attention is used for the `[CLS]` token while in QA global attention is provided on all question tokens. Since the number of such tokens is small relative to and independent of n the complexity of the combined local and global attention is still $O(n)$. While specifying global attention is task specific, it is a easy way to add inductive bias to the model's attention, and it is much simpler than existing task specific approaches that use complex architecture to combine information across smaller input chunks.

Linear Projections for Global Attention Recall that given the linear projections Q, K, V, the Transformer model (Vaswani et al., 2017) computes attention scores as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

We use two sets of projections, Q_s, K_s, V_s to compute attention scores of sliding window attention, and Q_g, K_g, V_g to compute attention scores for the global attention. The additional projections provide flexibility to model the different types of attention, which we show is critical for best performance on downstream tasks. Q_g, K_g, V_g are all initialized with values that match Q_s, K_s, V_s.

3.2 Implementation

In regular transformers, attention scores are computed as in Eqn. 1. The expensive operation is the matrix multiplication QK^T because both Q and K have n (sequence length) projections. For Longformer, the dilated sliding window attention computes only a fixed number of the diagonals of QK^T. As shown in Fig. 1, this results in a linear increase in memory usage compared to quadratic increase for full self-attention. However, implementing it requires a form of banded matrix multiplication that is not supported in existing deep learning libraries like PyTorch/Tensorflow. Fig. 1 compares the performance of three different ways of implementing it: `loop` is a memory efficient PyTorch implementation that supports dilation but is unusably slow and only used for testing; `chunks` only supports the non-dilated case and is used for the pretraining/finetuning setting; and `cuda` is our fully functioning highly optimized custom CUDA kernel implemented using TVM (Chen et al., 2018) and used for the language modeling experiments (see Appendix A for more details).

4 Autoregressive Language Modeling

Autoregressive or left-to-right language modeling is loosely defined as estimating the probability distribution of an existing token/character given its previous tokens/characters in an input sequence. This task is considered one of the fundamental tasks in natural language and recent prior work on modeling long sequences using transformers has relied

on this task as their primary evaluation (Dai et al., 2019; Rae et al., 2020; Sukhbaatar et al., 2019). Similarly, we develop and evaluate our model on autoregressive language modeling.

4.1 Attention Pattern

For autoregressive language modeling we use our dilated sliding window attention. Following Sukhbaatar et al. (2019) we use differing window sizes across the layers. In particular, we use small window sizes for the lower layers and increase window sizes as we move to higher layers. This allows the top layers to learn higher-level representation of the entire sequence while having the lower layers capture local information. In addition, it provides balance between efficiency (smaller window sizes are less computationally expensive due to fewer nonzero values) and performance (larger window sizes have richer representation power and often result in performance improvements).

We do not use dilated sliding windows for lower layers to maximize their capacity to learn and utilize the immediate local context. For the higher layers, we use a small amount of increasing dilation only on 2 heads. This gives the model the ability to directly attend to distant tokens without sacrificing local context.

4.2 Experiment Setup

To compare to prior work we focus on character-level LM (`text8` and `enwik8`; Mahoney, 2009).

Training Ideally, we would like to train our model on the largest window size and sequence length we can fit in a modern GPU memory. However, we found that the model needs a large number of gradient updates to learn the local context first, before learning to utilize longer context. To accommodate this, we adopt a staged training procedure where we increase the attention window size and sequence length across multiple training phases. In particular, in the first phase we start with a short sequence length and window size, then on each subsequent phase, we double the window size and the sequence length, and halve the learning rate. This makes training fast, while keeping the slow part (longest sequences and window sizes) to the end. We train the model over 5 total phases with starting sequence length of 2,048 and ending sequence length of 23,040 on the last phase (see Appendix B for detailed configurations of each phase, and for all other hyperparameters).

Model	#Param	Dev	Test
Dataset `text8`			
T12 (Al-Rfou et al., 2018)	44M	-	1.18
Adaptive (Sukhbaatar et al., 2019)	38M	1.05	1.11
BP-Transformer (Ye et al., 2019)	39M	-	1.11
Our Longformer	41M	1.04	**1.10**
Dataset `enwik8`			
T12 (Al-Rfou et al., 2018)	44M	-	1.11
Transformer-XL (Dai et al., 2019)	41M	-	1.06
Reformer (Kitaev et al., 2020)	-	-	1.05
Adaptive (Sukhbaatar et al., 2019)	39M	1.04	1.02
BP-Transformer (Ye et al., 2019)	38M	-	1.02
Our Longformer	41M	1.02	**1.00**

Table 2: *Small* model BPC on `text8` & `enwik8`

Model	#Param	Test BPC
Transformer-XL (18 layers)	88M	1.03
Sparse (Child et al., 2019)	≈100M	0.99
Transformer-XL (24 layers)	277M	0.99
Adaptive (Sukhbaatar et al., 2019)	209M	0.98
Compressive (Rae et al., 2020)	277M	0.97
Routing (Roy et al., 2020)	≈223M	0.99
Our Longformer	102M	0.99

Table 3: Performance of *large* models on `enwik8`

Evaluation We evaluate with sequences of length 32,256. Following Dai et al. (2019), we split the dataset into overlapping sequences of size 32,256 with a step of size 512, and report the performance on the last 512 tokens on the sequence.

4.2.1 Results

Tab. 2 and 3 summarize evaluation results on `text8` and `enwik8` datasets. We achieve a new state-of-the-art on both `text8` and `enwik8` using the small models with BPC of **1.10** and **1.00** on `text8` and `enwik8` respectively, demonstrating the effectiveness of our model.

For large models, given how expensive these experiments are, and following recent work (Kitaev et al., 2020; Rae et al., 2020), we are only evaluating on `enwik8`. Tab. 3 shows that Longformer outperforms the comparable Transformer-XL model, matches the performance of the comparable Sparse Transformer (Child et al., 2019), and matches or slightly underperforms recent models that have more than twice the number of parameters. It is worth noting that Adaptive Span (Sukhbaatar et al., 2019) and Compressive Transformer (Rae et al., 2020) are not good fit for the pretraining-finetuning paradigm as discussed in §2.

Model	Dev BPC
Decreasing w (from 512 to 32)	1.24
Fixed w (= 230)	1.23
Increasing w (from 32 to 512)	**1.21**
No Dilation	1.21
Dilation on 2 heads	**1.20**

Table 4: Top: changing window size across layers. Bottom: with/without dilation (@ 150K steps on phase1)

Model	base	large
RoBERTa (seqlen: 512)	1.846	1.496
Longformer (seqlen: 4,096)	10.299	8.738
+ copy position embeddings	1.957	1.597
+ 2K gradient updates	1.753	1.414
+ 65K gradient updates	1.705	1.358
Longformer (train extra pos. embed. only)	1.850	1.504

Table 5: MLM BPC for RoBERTa and various pretrained Longformer configurations.

4.2.2 Ablation Study

To show the importance of the design choices of our attention patterns, we tried different variants and report their controlled experiment results. To make the ablation study more manageable, we train each configuration for 150K steps[4] with phase 1 configuration on a small model on `text8`, then report the BPC performance on the dev set.

The top of Tab. 4 demonstrates the impact of different ways of configuring the window sizes per layer. We observe that increasing the window size from the bottom to the top layer leads to the best performance, arranging them in the reverse way leads to worse performance, and using a fixed window size (the average of window sizes of the other configuration) leads to a performance that it is in between. The bottom of Tab. 4 shows the impact of adding dilation. Adding some dilation to two heads leads to some improvement compared with no dilation at all.

5 Pretraining and Finetuning

Current state-of-the-art systems for many NLP tasks finetune a pretrained model with task supervision (e.g. BERT). One of our main motivations is to develop such a model suitable for long document tasks. To do so, we pretrained Longformer on a document corpus and finetune it for six tasks, including classification, QA and coreference resolution. The resulting model can process sequences up to 4,096 tokens long (8 times longer than BERT)[5].

We pretrain Longformer with masked language modeling (MLM), where the goal is to recover randomly masked tokens in a sequence. Since MLM pretraining is expensive, we continue pretraining from the RoBERTa (Liu et al., 2019) released checkpoint, while only making the minimal changes necessary to support Longformer's attention mechanism. Note that our attention pattern can be plugged into any pretrained transformer model without the need to change the model architecture.

Attention Pattern We use sliding window attention with window size of 512, therefore using the same amount of computation as RoBERTa.[6]

Position Embeddings RoBERTa uses learned absolute position embeddings with the maximum position being 512. To support longer documents, we add extra position embeddings to support up to position 4,096. To leverage RoBERTa's pretrained weights, instead of randomly initializing the new position embeddings, we initialize them by copying the 512 position embeddings from RoBERTa multiple times as analysis of BERT's attention heads shows a strong learned bias to attending to local context, including the previous or next token (Clark et al., 2019). Using the copy initialization preserves this local structure everywhere except at the partition boundaries. Despite its simplicity, we found this to be a very effective (see Tab. 5), allowing Longformer pretraining to rapidly converge with a small number of gradient updates.

Continued MLM Pretraining We pretrain Longformer using fairseq (Ott et al., 2019) on a corpus of long documents that we compiled (see Appendix C for corpus details). We train two model sizes, a base model and a large model. Both models are trained for 65K gradient updates with sequences length 4,096, batch size 64 (2^{18} tokens), maximum learning rate of 3e-5, linear warmup of 500 steps, followed by a power 3 polynomial decay. The rest of the hyperparameters are the same as RoBERTa.

Tab. 5 shows the BPC on the development set of our training corpus. The first row shows a 1.846

[4] One caveat is that the ordering of end performance will not agree with that at step 150K. However, this approximation saves the huge cost of running every experiment to completion.

[5] Sequences up to 16K are possible on current GPUs.

[6] Adding dilation on a few heads as in §4.1 hurt performance, likely because it is not compatible with the pretrained RoBERTa weights. Retraining such model from scratch might be needed to improve performance.

Wordpieces	WH	TQA	HQA	ON	IMDB	HY
avg.	1,535	6,589	1,316	506	300	705
95th pctl.	3,627	17,126	1,889	1,147	705	1,975

Table 6: Average and 95th percentile of context length of datasets in wordpieces. WH: WikiHop, TQA: TriviaQA, HQA: HotpotQA, ON: OntoNotes, HY: Hyperpartisan news

BPC using RoBERTa-base, which is comparable to the 1.880 BPC reported on the RoBERTa paper on their corpus. This indicates our training corpus is from a distribution close to that used to train RoBERTa. The following two rows show the performance of Longformer before pretraining with randomly initialized position embeddings and with copied position embeddings. The significant difference indicates the importance of the copy initialization, and the relative small difference between the RoBERTa BPC and the initialized BPC indicates that our sliding window attention is working well with the RoBERTa weights. The following two rows show the impact of continuing pretraining. Traininig for 2K steps improves BPC from 1.957 to 1.753, which further decreases to 1.705 after 65K steps, demonstrating the model is learning to better utilize the sliding window attention and longer context. Similar patterns are observed with RoBERTa-large and Longformer-large.

Frozen RoBERTa Weights We also pretrained Longformer while freezing all RoBERTa weights, and only training the new position embeddings. The motivation for this configuration is to perfectly preserve the RoBERTa performance on short documents. This configuration has a BPC of 1.850 (down from 1.957 at initialization), but higher than 1.705 where all the weights are trainable.

6 Tasks

We apply Longformer to multiple long document tasks, including QA, coreference resolution and classification. Tab. 6 shows the evaluation datasets have contexts significantly longer than 512 wordpieces. Our primary goal is to evaluate whether our attention mechanism can act as a replacement for the standard self-attention mechanism in BERT style models, and to perform controlled trials against a strong baseline. We are also interested in evaluating whether we can replace complicated task specific models necessitated by BERT's limited context with simpler models that just concatenate all available context into a single sequence.

Our baseline is a RoBERTa based model that breaks the context into the longest possible segment, passes each individually through RoBERTa, and concatenates the activations for further processing. For QA tasks, we also concatenate the question to each segment so that RoBERTa can condition it's contextual representations of the context on the question. The Longformer variant replaces the RoBERTa self-attention mechanism with our windowed attention used during pretraining, plus a task motivated global attention. The global attention uses additional linear projections (§3.1).

6.1 Question answering

We used three datasets: WikiHop (Welbl et al., 2018), TriviaQA (Joshi et al., 2017, Wikipedia setting), and HotpotQA, (Yang et al., 2018, distractor setting).[7]

For WikiHop and TriviaQA we follow the simple QA model of BERT (Devlin et al., 2019), and concatenate question and documents into one long sequence, run it through Longformer, then have a dataset-specific prediction layer. WikiHop uses a classification layer for the candidate while TriviaQA uses the loss function of Clark and Gardner (2017) to predict answer span. We include global attention to question tokens and answer candidates for WikiHop and to question tokens for TriviaQA.

HotpotQA is a multihop QA dataset that involves extracting answer spans and evidence sentences from 10 Wikipedia paragraphs, 2 of which are relevant and the rest are distractors. We use a two-stage model that first selects the most relevant paragraphs then passes them to a second stage for answer extraction. Both stages concatenate question and context into one sequence, run it through Longformer, then use task-specific prediction layers. We train the models in a multi-task way to predict relevant paragraphs, evidence sentences, answer spans and question types (yes/no/span) jointly. Note that this model is simpler than recent SOTA models that include complex task-specific architectures (e.g., (Tu et al., 2019; Chen et al., 2019; Tu et al., 2020; Groeneveld et al., 2020)). See Appendix D for further details about the models and hyperparameters.

6.2 Coreference Resolution

We use OntoNotes (Pradhan et al., 2012), and the model from Joshi et al. (2019), a modification of

[7] We use the full version of TriviaQA and HotpotQA, not the simplified versions in MRQA (Fisch et al., 2019).

	QA			Coref.	Classification	
Model	WikiHop	TriviaQA	HotpotQA	OntoNotes	IMDB	Hyperpartisan
RoBERTa-base	72.4	74.3	63.5	78.4	95.3	87.4
Longformer-base	**75.0**	**75.2**	**64.4**	**78.6**	**95.7**	**94.8**

Table 7: Summary of finetuning results on QA, coreference resolution, and document classification. Results are on the development sets comparing our Longformer-base with RoBERTa-base. TriviaQA, Hyperpartisan metrics are F1, WikiHop and IMDB use accuracy, HotpotQA is joint F1, OntoNotes is average F1.

the system from Lee et al. (2018) to replace ELMo with BERT. The Longformer system is a straightforward adaption of the baseline model by replacing RoBERTa with Longformer and extending the sequence length. We didn't use global attention for this task.

6.3 Document Classification

We evaluate on IMDB (Maas et al., 2011) and Hyperpartisan news detection (Kiesel et al., 2019) datasets.[8] IMDB is a standard sentiment classification datasets consisting of movie reviews. While most documents in this dataset are short, about 13.6% of them are larger than 512 wordpieces (Tab. 6). Documents in Hyperpartisan are relatively long, and it is small with only 645 documents making it a good test for Longformer's ability to adapt to limited data. We use global attention on the `[CLS]` token.

6.4 Results

Main Result Tab. 7 summarizes the results of all our finetuning experiments. We observe that Longformer consistently outperforms the RoBERTa baseline. Its performance gain is especially obvious for tasks that require long context such as WikiHop and Hyperpartisan. For TriviaQA, the improvement is more modest as the local context is often sufficient to answer the question. In the case of HotpotQA, the supporting fact auxiliary supervision allows models to easily find relevant contexts and then focus on local context, leading to smaller gains. This is contrasted with WikiHop that only includes distant supervision of intermediate reasoning chains, where our approach excels by reasoning over the entire context. On the IMDB and OntoNotes datasets the performance gains are smaller. For IMDB, the majority of the dataset consists of short documents and thus it is expected to see smaller improvements. For OntoNotes, we found that the distance between any two mentions is typically quite small so that a baseline that processes smaller chunks separately is able to stitch together mentions into coreference chains without considering cross chunk interactions.

[8]For Hyperpartisan we split the training data into 80/10/10 train/dev/test sets, and report mean F1 across five seeds.

Model	WikiHop	TriviaQA	HotpotQA
Current* SOTA	78.3	73.3	**74.2**
Longformer-large	**81.9**	**77.3**	73.2

Table 8: Leaderboard results of Longformer-large at time of submission (May 2020). All numbers are F1 scores.

Longformer-large for QA We also evaluate the performance of Longformer-large on long context QA tasks. Tab. 8 shows that our Longformer-large achieves new state-of-the-art results[9] on WikiHop and TriviaQA by large margins (3.6 and 4 points respectively), and for HotpotQA, it underperforms the current state-of-the-art (Fang et al., 2020) by a point. Tab. 9 shows the detailed results of HotpotQA compared with published and unpublished concurrent models. Longformer places second on the published leaderboard, outperforming all other published results except for HGN (Fang et al., 2020). All published top performing models in this task (Tu et al., 2019; Fang et al., 2020; Shao et al., 2020) use GNNs (Kipf and Welling, 2017) or graph network of entities, which seem to encode an important inductive bias for the task and can potentially improve our results further. Nevertheless, Longformer performs strongly outperforming all other methods including the recent non-GNN methods (Glaß et al., 2019; Shao et al., 2020; Groeneveld et al., 2020).

Model	ans.	supp.	joint
TAP 2 (ensemble) (Glaß et al., 2019)	79.8	86.7	70.7
SAE (Tu et al., 2019)	79.6	86.7	71.4
Quark (dev) (Groeneveld et al., 2020)	81.2	87.0	72.3
C2F Reader (Shao et al., 2020)	81.2	87.6	72.8
Longformer-large	81.3	88.3	73.2
ETC-large† (Ainslie et al., 2020)	81.2	89.1	73.6
GSAN-large†	81.6	88.7	73.9
HGN-large (Fang et al., 2020)	82.2	88.5	74.2

Table 9: HotpotQA results in distractor setting test set. Quark's test results are not available. All numbers are F1 scores. † shows contemporaneous leaderboard submissions.

Model	Accuracy / Δ
Longformer (seqlen: 4,096)	73.8
RoBERTa-base (seqlen: 512)	72.4 / -1.4
Longformer (seqlen: 4,096, 15 epochs)	75.0 / +1.2
Longformer (seqlen: 512, attention: n^2)	71.7 / -2.1
Longformer (seqlen: 2,048)	73.1 / -0.7
Longformer (no MLM pretraining)	73.2 / -0.6
Longformer (no linear proj.)	72.2 / -1.6
Longformer (no linear proj. no global atten.)	65.5 / -8.3
Longformer (pretrain extra position embed. only)	73.5 / -0.3

Table 10: WikiHop development set ablations

6.5 Ablations on WikiHop

Tab. 10 presents an ablation study for WikiHop on the development set. All results use Longformer-base, fine-tuned for five epochs with identical hyperparameters except where noted. Longformer benefits from longer sequences, global attention, separate projection matrices for global attention, MLM pretraining, and longer training. In addition, when configured as in RoBERTa-base (seqlen: 512, and n^2 attention) Longformer performs slightly worse then RoBERTa-base, confirming that performance gains are not due to additional pretraining. Performance drops slightly when using the RoBERTa model pretrained when only unfreezing the additional position embeddings, showing that Longformer can learn to use long range context in task specific fine-tuning with large training datasets such as WikiHop.

[9] At submission time, May 2020. Later, BigBird (Zaheer et al., 2020) improved leaderboard results on these datasets. There are confounding factors such as using 16X more compute in BigBird's pretraining compared with Longformer, potentially affecting the performance.

7 Longformer-Encoder-Decoder (LED)

The original Transformer (Vaswani et al., 2017) consisted of an encoder-decoder architecture, intended for sequence-to-sequence tasks (Sutskever et al., 2014), such as summarization and translation. While encoder-only Transformers are effective on a variety of NLP tasks, pre-trained encoder-decoder Transformer models (e.g. BART (Lewis et al., 2020) and T5 (Raffel et al., 2020)) have achieved strong results on tasks like summarization. Yet, such models can't efficiently scale to seq2seq tasks with longer inputs.

To facilitate modeling long sequences for seq2seq learning, we propose a Longformer variant that has both the encoder and decoder Transformer stacks but instead of the full self-attention in the encoder, it uses the efficient local+global attention pattern of the Longformer. The decoder uses the full self-attention to the entire encoded tokens and to previously decoded locations. We call this model Longformer-Encoder-Decoder (LED) which scales linearly with the input. Since pre-training LED is expensive, we initialize LED parameters from the BART, and follow BART's exact architecture in terms of number of layers and hidden sizes. The only difference is that to process longer inputs, we extend position embedding to 16K tokens (up from BART's 1K tokens) and we initialize the new position embedding matrix by repeatedly copying BART's 1K position embeddings 16 times as in Section 5 for RoBERTa. Following BART, we release two model sizes, LED-base and LED-large, which respectively have 6 and 12 layers in both encoder and decoder stacks.

We evaluate LED on the summarization task using the arXiv summarization dataset (Cohan et al., 2018) which focuses on long document summarization in the scientific domain. The 90th percentile of document lengths is 14.5K tokens, making it an appropriate testbed for evaluating LED. LED's encoder reads the document and its decoder generates the output summary. The encoder uses local attention with window size 1,024 tokens and global attention on the first <s> token. The decoder uses full attention to the entire encoder and previously decoded locations. As standard in seq2seq models, LED is trained using teacher forcing on gold training summaries and uses beam search at inference.

Tab. 11 demonstrates the results of LED-large 16K on the arXiv summarization task. This model is merely initialized from BART, with no additional

	R-1	R-2	R-L
Discourse-aware (2018)	35.80	11.05	31.80
Extr-Abst-TLM (2020)	41.62	14.69	38.03
Dancer (2020)	42.70	16.54	38.44
Pegasus (2020)	44.21	16.95	38.83
LED-large (seqlen: 4,096) (ours)	44.40	17.94	39.76
BigBird (seqlen: 4,096) (2020)	**46.63**	19.02	41.77
LED-large (seqlen: 16,384) (ours)	**46.63**	**19.62**	**41.83**

Table 11: Summarization results of Longformer-Encoder-Decoder (LED) on the arXiv dataset. Metrics from left to right are ROUGE-1, ROUGE-2 and ROUGE-L.

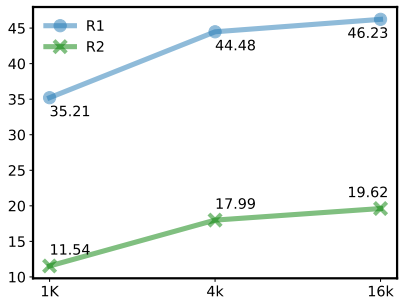


Figure 3: ROUGE-1 and ROUGE-2 of LED when varying the input size (arXiv validation set).

pre-training. We observe that LED achieves state-of-the-art results on arXiv, slightly outperforming BigBird (Zaheer et al., 2020). Note that the BigBird summarization model supports sequence length of 4K tokens but starts from and continues pre-training Pegasus (Zhang et al., 2020), a model specifically designed and pre-trained for summarization. With no pre-training or task-specific initialization, but with ability to process longer inputs, LED can slightly outperform BigBird. Further improvements should be possible through pre-training of LED. Fig. 3 further illustrates the importance of sequence length showing the ablility to process longer input significantly improves the results.

8 Conclusion and Future Work

We present Longformer, a transformer-based model that is scalable for processing long documents and that makes it easy to perform a wide range of document-level NLP tasks without chunking/shortening the long input and without complex architecture to combine information across these chunks. Longformer employs an attention pattern that combines local and global information while also scaling linearly with the sequence length. Longformer achieves state-of-the-art results on the character-level language modeling tasks of `text8` and `enwik8`. When pretrained, Longformer consistently outperforms RoBERTa on long document tasks and sets new state-of-the-art results on WikiHop and TriviaQA. We further present LED, an encoder-decoder variant of Longformer for modeling sequence-to-sequence tasks, and achieve state-of-the-art results on the arXiv long document summarization task. For future work, we would like to study other pretraining objectives, especially for LED, increase the sequence length, and explore other tasks that might benefit from our model.

Acknowledgment

We would like to thank Noah Smith, Dan Weld, Dirk Groeneveld, Kyle Lo, Daniel King and Doug Downey for helpful discussions and feedback, and the AI2 infrastructure team for technical support.

References

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, Online. Association for Computational Linguistics.

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2018. Character-level language modeling with deeper self-attention. In *AAAI*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *ACL*.

Jifan Chen, Shih-Ting Lin, and Greg Durrett. 2019. Multi-hop question answering via reasoning chains. *arXiv preprint*, abs/1910.02610.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *OSDI*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint*, abs/1604.06174.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint*, abs/1904.10509.

Christopher Clark and Matt Gardner. 2017. Simple and effective multi-paragraph reading comprehension. In *ACL*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does bert look at? an analysis of bert's attention. *arXiv preprint*, abs/1906.04341.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *NAACL-HLT 2018*.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *NeurIPS*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. 2020. Hierarchical graph network for multi-hop question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8823–8838, Online. Association for Computational Linguistics.

Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. 2019. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *MRQA workshop at EMNLP*.

Alexios Gidiotis and Grigorios Tsoumakas. 2020. A divide-and-conquer approach to the summarization of academic articles. *ArXiv*, abs/2004.06190.

Michael Glaß, Alfio Massimiliano Gliozzo, Rishav Chakravarti, Anthony Ferritto, Lin Pan, Gaudani Bhargav, Dinesh Garg, and Avirup Sil. 2019. Span selection pre-training for question answering. *arXiv preprint*, abs/1909.04120.

Scott Gray, Alec Radford, and Diederik P. Kingma. 2017. Gpu kernels for block-sparse weights.

Dirk Groeneveld, Tushar Khot, Mausam, and Ashish Sabhwaral. 2020. A simple yet strong pipeline for HotpotQA. *arXiv preprint*, abs/2004.06753.

Ankit Gupta and Jonathan Berant. 2020. Gmat: Global memory augmentation for transformers. *ArXiv*, abs/2006.03274.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *ACL*.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*.

Mandar Joshi, Omer Levy, Luke Zettlemoyer, and Daniel Weld. 2019. BERT for coreference resolution: Baselines and analysis. In *EMNLP-IJCNLP*.

Johannes Kiesel, Maria Mestre, Rishabh Shukla, Emmanuel Vincent, Payam Adineh, David Corney, Benno Stein, and Martin Potthast. 2019. SemEval-2019 task 4: Hyperpartisan news detection. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 829–839, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *ICLR*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *ICLR*.

Olga V. Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of bert. In *EMNLP/IJCNLP*.

Kenton Lee, Luheng He, and Luke Zettlemoyer. 2018. Higher-order coreference resolution with coarse-to-fine inference. In *NAACL*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint*, abs/1907.11692.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Matt Mahoney. 2009. Large text compression benchmark.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. In *SSW*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible

toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 1–40, Jeju Island, Korea. Association for Computational Linguistics.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Yih, Sinong Wang, and Jie Tang. 2019. Blockwise self-attention for long document understanding. *arXiv preprint*, abs/1911.02972.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *ICLR*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, W. Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient content-based sparse attention with routing transformers. *arXiv preprint*, abs/2003.05997.

Nan Shao, Yiming Cui, Ting Liu, Shijin Wang, and Guoping Hu. 2020. Is graph structure necessary for multi-hop reasoning? *arXiv preprint*, abs/2004.03096.

Sandeep Subramanian, Raymond Li, Jonathan Pilault, and C. Pal. 2020. On extractive and abstractive neural document summarization with transformer language models. In *EMNLP*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. In *ACL*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Trieu H. Trinh and Quoc V. Le. 2018. A simple method for commonsense reasoning. *arXiv preprint*, abs/1806.02847.

Ming Tu, Jinke Huang, Xiaodong He, and Bowen Zhou. 2020. Graph sequential network for reasoning over sequences. In *NeurIPS Graph Representation Learning workshop*.

Ming Tu, Kevin Huang, Guangtao Wang, Jing Huang, Xiaodong He, and Bufang Zhou. 2019. Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. *arXiv preprint*, abs/1911.00484.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. 2018. Constructing datasets for multi-hop reading comprehension across documents. *TACL*, 6:287–302.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint*, abs/1901.10430.

Qizhe Xie, Zihang Dai, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. 2019. Unsupervised data augmentation for consistency training. *arXiv preprint*, abs/1904.12848.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shu xin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Li-Wei Wang, and Tie-Yan Liu. 2020. On layer normalization in the transformer architecture. *arXiv preprint*, abs/2002.04745.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.

Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. 2019. BP-Transformer: Modelling long-range context via binary partitioning. *arXiv preprint*, abs/1911.04070.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, C. Alberti, S. Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, L. Yang, and A. Ahmed. 2020. Big bird: Transformers for longer sequences. *ArXiv*, abs/2007.14062.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. In *NeurIPS*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. *ICML*.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *ICCV*, pages 19–27.

A Implementation Details

Implementing Longformer's dilated sliding window attention requires a form of banded matrix multiplication (matrix multiplication where the output is all zero except certain diagonals) that is not directly supported in existing deep learning libraries like PyTorch/Tensorflow. Fig. 1 compares the runtime and memory of three different ways of implementing it.

`Longformer-loop` is a naive implementation that computes each diagonal separately in a loop. It is memory efficient because it only computes the non-zero values, but it is unusably slow. We only use it for testing because it is easy to implement but don't use it to run experiments.

`Longformer-chunks` only supports the non-dilated case. It chunks Q and K into overlapping blocks of size w and overlap of size $\frac{1}{2}w$, multiplies the blocks, then mask out the diagonals. This is very compute efficient because it uses a single matrix multiplication operation from PyTorch, but it consumes 2x the amount of memory a perfectly optimized implementation should consume because it computes some of the zero values. Because of the compute efficiency, this implementation is most suitable for the pretrain/finetune case. We didn't find the increase in memory to be a problem for this setting.

`Longformer-cuda` is a custom CUDA kernel that we implement using TVM (Chen et al., 2018). It is a fully functioning implementation of our attention (not limited as `Longformer-chunks`), it is the most memory efficient, and it is as fast as the highly optimized full self-attention.[10] We mainly use this implementation for the autoregressive language modeling experiments because of the memory efficiency (allows the longest sequences) and the support of dilation (needed for character-LM experiments).

Tensor Virtual Machine (TVM) We build our custom CUDA kernel using TVM (Chen et al., 2018), a deep learning compiler stack that compiles high level description of a function into optimized device-specific code. Using TVM, we describe our banded matrix multiplication in high-level python constructs, then TVM generates the corresponding CUDA code and compiles it for GPUs.

B Character LM Hyperparameters

We evaluate on `text8` and `enwik8`, both contain 100M characters from Wikipedia split into 90M, 5M, 5M for train, dev, test. Our model only specifies how the self-attention component works, and it is agnostic to the other design choices for the transformer model. Our implementation is based on the Transformer-XL (Dai et al., 2019) code[11] with the memory mechanism disabled. We use relative position embeddings with sinusoidal weights as in Dai et al. (2019). We use two different model sizes, a small (12 layers, 512 hidden size) model as in Dai et al. (2019), and a large (30 layers, 512 hidden size) model as in Child et al. (2019). We employed mixed precision training (floating points 16 and 32) using apex[12] to reduce memory consumption and speed-up training. However, we kept the attention computation in fp32 to avoid numerical instability issues.[13] We used gradient checkpointing (Chen et al., 2016) to reduce memory usage, and ran our experiments on 48GB RTX8000 GPUs. All hyperparameters and stage configurations are listed in Tab. 12. Our CUDA kernel supports the autoregressive mode where each token attends to a window of previous tokens only. Our implementation also includes a version of the relative position embedding that is compatible with our dilated sliding window attention.

We ran the small model experiments on 4 RTX8000 GPUs for 16 days. For the large model, we ran experiments on 8 RTX8000 GPUs for 13 days. Most of our hyperparameter search is similar to the ablation in Tab. 4 where we run the configuration for 150K steps on `text8`. We experimented with absolute position embeddings and learned position embeddings, dropout values of [0.1, 0.2] (small model) and [0.1, 0.4] (large model), prelayernorm and post-layernorm (Xiong et al., 2020), learning rate (LR) of phase1 of values [2.5e-5, 5e-4, 1e-4] constant and cosine LR schedules, and different configurations for dilation (on all heads, on 2 heads, no dilation). Number of gradient updates/phase reported in Tab. 12 is determined by running each phase until the validation BPC stops

[10] It is worth noting that theoretically, a perfectly optimized `Longformer-cuda` should be faster than the n^2 computation. However, achieving this level of performance requires special knowledge of low-level GPU programming, similar to implementing a highly optimized matrix multiplication. Our current implementation is sufficiently fast and practical to use.

[11] https://github.com/kimiyoung/transformer-xl

[12] https://github.com/NVIDIA/apex

[13] We found that using fp16 in attention operation results in floating point overflow and NaNs in later stages of training.

getting better.

C Pretraining Data

In order to allow the model to learn long dependencies in pretraining, we compiled a corpus of long documents. Some of these data sources were also included in the original RoBERTa pretraining including the Books corpus (Zhu et al., 2015) plus English Wikipedia. We additionally included one third of a subset of the Realnews dataset (Zellers et al., 2019) with documents longer than 1,200 tokens as well as one third of the Stories (Trinh and Le, 2018) corpus. Our goal was to include a mix of long and short documents to both allow the model to learn longer dependencies while not to forget information from the original RoBERTa pretraining. The statistics of the pretraining data is shown in Tab. 13.

D Task specific model details

All the QA and classification models are implemented using PyTorch-Lightning[14]. We use the official train/dev/test splits of all datasets except for the Hyperpartisan news which we randomely split into 80/10/10 for train/dev/test.

WikiHop Instances in WikiHop consist of: a question, answer candidates (ranging from two candidates to 79 candidates), supporting contexts (ranging from three paragraphs to 63 paragraphs), and the correct answer. The dataset does not provide any intermediate annotation for the multihop reasoning chains, requiring models to instead infer them from the indirect answer supervision.

To prepare the data for input to Longformer and RoBERTa, we first tokenize the question, answer candidates, and support contexts using RoBERTa's wordpiece tokenizer. Then we concatenate the question and answer candidates with special tokens as `[q] question [/q] [ent] candidate1 [/ent] ... [ent] candidateN [/ent]`. The contexts are also concatenated using RoBERTa's document delimiter tokens as separators: `</s> context1 </s> ... </s> contextM </s>`. The special tokens `[q]`, `[/q]`, `[ent]`, `[/ent]` were added to the RoBERTa vocabulary and randomly initialized before task finetuning.

[14] https://github.com/PyTorchLightning/pytorch-lightning

After preparing the input data, we compute activations from the top layer of each model as follows. We take the question and answer candidates and concatenate them to as much context as possible up to the model sequence length (512 for RoBERTa, 4,096 for Longformer), run the sequence through the model, collect the output activations, and repeat until all of the context is exhausted (for all models except Longformer-large, where we just include the first 4,096 length sequence due to memory requirements). Then all activations for all chunks are concatenated into one long sequence. In the case of Longformer, we use global attention to the entire question and answer candidate sequence.

For prediction, we attach a linear layer to each `[ent]` that outputs a single logit, average over all logits for each candidate across the chunks, apply a softmax and use the cross entropy loss with the correct answer candidate.

Training used the Adam optimizer with linear warmup over 200 gradient updates to a maximum LR, and linear decay over the remainder of training. We used gradient accumulation to effective batch size of 32 instances, checking the development accuracy every 250 gradient updates and reported the maximum development accuracy. Other hyperparameters (dropout, weight decay) were identical to RoBERTa pretraining.

In general, we ran minimal hyperparameter trials, but for fair comparison between Longformer and RoBERTa ran an identical hyperparameter search with Longformer-base and RoBERTa-base. This consisted of a grid search of LR in [2e-5, 3e-5, 5e-5] and number epochs in [5, 10, 15]. The best Longformer-base configuration used lr=3e-5, 15 epochs. We ran two hyperparameter trials for Longformer-large, lr=3e-5 and number epochs in [5, 15] (the 5 epoch model had higher dev accuracy of 77.6, and was the single model submitted to the public leaderboard for test set evaluation). All models were trained on a single RTX8000 GPU, with Longformer-base taking about a day for 5 epochs.

TriviaQA TriviaQA has more than 100K question, answer, document triplets for training. Documents are Wikipedia articles, and answers are named entities mentioned in the article. The span that answers the question is not annotated, but it is found using simple text matching.

Similar to WikiHop, we tokenize the question and the document using RoBERTa's tokenizer, then form the input as `[s] question [/s]`

Param	Value
Position Embeddings	Relative and Sinusoidal as in Dai et al. (2019)
Small model config	12 layers, 8 heads, 512 hidden size as in Dai et al. (2019)
Large model config	30 layers, 8 heads, 512 hidden size as in Child et al. (2019)
Optimizer	AdamW
Dropout	0.2 (small model), 0.4 (large model)
Gradient clipping	0.25
Weight Decay	0.01
Layernorm Location	pre-layernorm (Xiong et al., 2020)
Activation	GeLU
Number of phases	5
Phase 1 window sizes	32 (bottom layer) - 8,192 (top layer)
Phase 5 window sizes	512 (bottom layer) - (top layer)
Phase 1 sequence length	2,048
Phase 5 sequence length	23,040 (gpu memory limit)
Phase 1 LR	0.00025
Phase 5 LR	000015625
Batch size per phase	32, 32, 16, 16, 16
#Steps per phase (small)	430K, 50k, 50k, 35k, 5k
#Steps per phase (large)	350K, 25k, 10k, 5k, 5k
Warmup	10% of the phase steps with maximum 10K steps
LR scheduler	constant throughout each phase
Dilation (small model)	0 (layers 0-5), 1 (layers 6-7), 2 (layers 8-9), 3 (layers 10-11)
Dilation (large model)	0 (layers 0-14), 1 (layers 15-19), 2 (layers 20-24), 3 (layers 25-29)
Dilation heads	2 heads only

Table 12: Hyperparameters for the best performing model for character-level language modeling

Source	Tokens	Avg doc len
Books (Zhu et al., 2015)	0.5B	95.9K
English Wikipedia	2.1B	506
Realnews (Zellers et al., 2019)	1.8B	1.7K
Stories (Trinh and Le, 2018)	2.1B	7.8K

Table 13: Pretraining data

`document [/s]`. We truncate the document at 4,096 wordpiece to avoid it being very slow. Afterwards, we get the activations from RoBERTa and Longformer similar to WikiHop (discussed above). We use global attention on all question tokens.

For prediction, we add one layer that predicts the beginning and end of the answer span. Because of the distant supervision nature of the training data (no gold answer spans), we use the loss function of Clark and Gardner (2017) which works like an OR that the model only needs to get one answer span right, not all of them.

Hyperparameters of the best configuration are listed in Tab. 14. All other hyperparameters are similar to RoBERTa's. For hyperparameter search, we only tuned LR for the RoBERTa baseline and tried rates [3e-5, 5e-5, 1e-4], then used the best, which is 3e-5, for all subsequent experiments with no further tuning. We trained the Longformer-large with the best configuration once and submitted its output to the leaderboard. We ran our experiments on 32GB V100 GPUs. Small model takes 1 day to train on 4 GPUs, while large model takes 1 day on 8 GPUs.

HotpotQA HotpotQA dataset involves answering questions from a set of 10 paragraphs from 10 different Wikipedia articles where 2 paragraphs are relevant to the question and the rest are distractors. It includes 2 tasks of answer span extraction and evidence sentence identification. Our model for HotpotQA combines both answer span extraction and evidence extraction in one joint model. We found a higher performance using a two-stage Longformer model with similar setup that first identifies relevant paragraphs and then does find the final answer span and evidence.[15] This is largely because removing the distracting paragraphs first reduces the noise for the final evidence and span detection as also found to be important by recent state-of-the-art methods in this dataset (Fang et al., 2020). Similar to Wikihop and TriviaQA, to prepare the data for input to Longformer, we concatenate question and then all the 10 paragraphs in one long context. We particularly use the following input format with special tokens: "`[CLS] [q] question [/q]` ⟨t⟩ $\texttt{title}_1$ ⟨/t⟩ $\texttt{sent}_{1,1}$ `[s]` $\texttt{sent}_{1,2}$ `[s]` ...

[15] The final dev performance of the two stage model improves over a single stage model by about 4.2 points on joint-F1 metric

⟨t⟩ title$_2$ ⟨/t⟩ sent$_{2,1}$ [s] sent$_{2,2}$ [s] ..." where [q], [/q], ⟨t⟩, ⟨/t⟩, [s], [p] are special tokens representing, question start and end, paragraph title start and end, and sentence, respectively. The special tokens were added to the Longformer vocabulary and randomly initialized before task finetuning. For Longformer, we use global attention to question tokens, paragraph title start tokens as well as sentence tokens. The model includes additional feedforward layers on top of paragraph title start tokens for prediction of relevant paragraphs, as well as sentence tokens for predicting evidence sentences. After training the first stage model, we predict relevant paragraph scores for both training and development set. We then keep up to 5 paragraphs whose raw score is higher than a pre-specified threshold (-3.0), and remove the other paragraphs from the context. We then train the second stage model on the resulting shortened context. For answer span extraction we use BERT's QA model (Devlin et al., 2019) with addition of a question type (yes/no/span) classification head over the first special token ([CLS]). For evidence extraction we apply 2 layer feedforward networks on top of the representations corresponding to sentence and paragraph tokens to get the corresponding evidence prediction scores and use binary cross entropy loss to train the model. At inference time for evidence extraction, we use a constrained decoding strategy similar to Groeneveld et al. (2020) that ensures that the evidence sentences come from exactly two paragraphs which is the setup of this dataset. We combine span, question classification, sentence, and paragraphs losses and train the model in a multitask way using linear combination of losses. Our experiments are done on RTX8000 GPUs and training each epoch takes approximately half a day on 4 GPUs. We trained the model using Adam optimizer with linear warmup (1000 steps) and linear decay. We used minimal hyperparameter tuning using LRs of 3e-5 and 5e-5 and epochs of 3 to 7 and found the model with LR of 3e-5 and 5 epochs to work best. We conduct the same hyperparameter search for the RoBERTa baseline as well. The rest of hyperparameters are reported in Tab 14.

Param	WikiHop	TriviaQA	HotpotQA
Epochs	15	5	5
LR	3e-5	3e-5	5e-5
Warmup steps	200	1000	1000
Batch size	32	32	32
Optimizer	Adam	Adam	Adam

Table 14: Hyperparameters of the QA models. All models use a similar scheduler with linear warmup and decay.

Coreference model details The coreference model is a straightforward adaptation of the coarse-to-fine BERT based model from Joshi et al. (2019). After preprocessing each document with the RoBERTa wordpiece tokenizer, it splits each document into non-overlapping segments up to the maximum sequence length, then concatenates the activations for the coarse-to-fine clustering stage that forms coreference clusters. The maximum sequence length was 384 for RoBERTa-base, chosen after three trials from [256, 384, 512] using the default hyperparameters in the original implementation.[16] For Longformer-base the sequence length was 4,096. Similar to the original implementation, different learning rates were used for the pretrained RoBERTa parameters and the randomly initialized task parameters. Using a larger learning rate in the task parameters allows the optimizer to adjust them farther from their randomly initialized values without destroying the information in the pretrained RoBERTa parameters.

Hyperparameter searches were minimal and consisted of grid searches of RoBERTa LR in [1e-5, 2e-5, 3e-5] and task LR in [1e-4, 2e-4, 3e-4] for both RoBERTa and Longformer for a fair comparison. The best configuration for Longformer-base was RoBERTa lr=1e-5, task lr=1e-4. All other hyperparameters were the same as in the original implementation. Training takes about 10 hours on a single GPU.

Our implementation is a superhack that involves PyTorch and Tensorflow sharing a single process and GPU. To avoid re-implementing the complicated coarse-to-fine logic from Tensorflow in PyTorch (that involves a highly optimized custom GPU kernel originally released by Lee et al. (2018)), we devised a system where the lower transformer portion of the model passes activations and gradients back and forth between PyTorch and Tensorflow. The input tensors are first run through the transformer in PyTorch, the activations are collected from the top layer, transferred from GPU to CPU then from CPU to Tensorflow and back to GPU to run the coarse-to-fine clustering and compute the loss. Then gradients are back propogated

[16] https://github.com/mandarjoshi90/coref

in Tensorflow to the top of the transformer and the process reversed to transfer them to PyTorch for back propogation through the remainder of the model. Separate optimizers are maintained with identical LR schedules for parameter updates. The overhead in this approach is minimal compared to the overall cost of running the model.

Text classification For classification, following BERT, we used a simple binary cross entropy loss on top of a first `[CLS]` token with addition of global attention to `[CLS]`. We used Adam optimizer with batch sizes of 32 and linear warmup and decay with warmup steps equal to 0.1 of the total training steps. For both IMDB and Hyperpartisan news we did grid search of LRs [3e-5, 5e-5] and epochs [10, 15, 20] and found the model with [3e-5] and epochs 15 to work best. Experiments were done on a single RTX8000 GPU.