arXiv:2208.04347v1 [cs.CL] 8 Aug 2022

Investigating Efficiently Extending Transformers for Long Input Summarization

Jason Phang [1]* **Yao Zhao** [2] **Peter J. Liu** [2]
[1]New York University, [2]Google Research, Brain Team
jasonphang@nyu.edu
{yaozhaoyz, peterjliu}@google.com

Abstract

While large pretrained Transformer models have proven highly capable at tackling natural language tasks, handling long sequence inputs continues to be a significant challenge. One such task is long input summarization, where inputs are longer than the maximum input context of most pretrained models. Through an extensive set of experiments, we investigate what model architectural changes and pretraining paradigms can most efficiently adapt a pretrained Transformer for long input summarization. We find that a staggered, block-local Transformer with global encoder tokens strikes a good balance of performance and efficiency, and that an additional pretraining phase on long sequences meaningfully improves downstream summarization performance. Based on our findings, we introduce PEGASUS-X, an extension of the PEGASUS model with additional long input pretraining to handle inputs of up to 16K tokens. PEGASUS-X achieves strong performance on long input summarization tasks comparable with much larger models while adding few additional parameters and not requiring model parallelism to train.

1 Introduction

Large pretrained Transformer models have proven to be extremely capable at tackling natural language tasks (Devlin et al., 2018; Brown et al., 2020). However, handling long textual sequences continues to be a significant challenge for these models. Training models to handle long sequences is expensive in both computation and memory, and moreover requires training and evaluating on long sequence data, which can be rarer and more costly to collect. Given the broad success of Transformer models on short-sequence language tasks, our goal is to investigate the best way to extend these models to handle longer sequences.

* Work done while at Google.

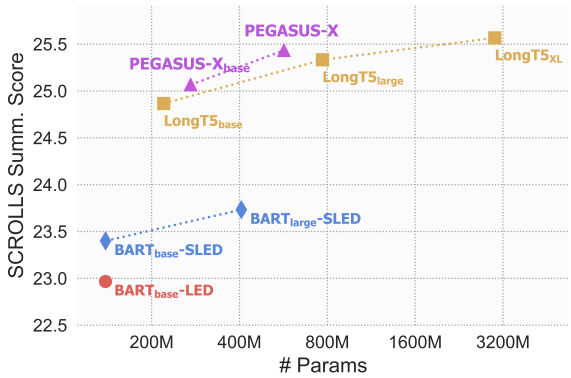

Figure 1: Model scores on SCROLLS (Shaham et al., 2022) summarization tasks. All models evaluated on up to 16K input tokens. PEGASUS-X outperforms other models at comparable model sizes. Scores are computed by taking the average of the geometric mean of ROUGE-1/2/L.

In this work, we focus on the task of long input summarization: summarizing long input documents into shorter textual sequences. The input documents of such tasks are often significantly longer than the maximum context lengths of most standard Transformer models, and hence warrant both specialized model architecture modifications as well as new training regimes to handle. For instance, to avoid the quadratic growth in memory consumption of the attention computation in Transformers, many memory-efficient Transformer variants have been proposed (Tay et al., 2020, 2021). However, the manner in which these changes are incorporated into models has been inconsistent and ad-hoc, and there are few established best-practices. For instance, some works add an additional long input pretraining stage to adapt the model weights to the new architecture (Beltagy et al., 2020), while others directly fine-tune on the long-input summarization data without any pre-adaptation (Zaheer et al., 2020; Pang et al., 2022). Because of the high cost of training these models, there has yet to be a systematic study of how best to adapt models for

long input sequences. Hence, it has been difficult to establish which model and training changes are necessary or complementary.

To answer these questions, we conduct an extensive empirical investigation into the architectural changes, model configurations and pretraining schemes to identify the better approaches to training Transformer models to tackle long input summarization. We evaluate a set of efficient Transformer variants, and propose a simpler block-wise local Transformer architecture with staggered blocks and global tokens that strikes a good balance of performance and memory efficiency. We also show that given a fixed token budget, pretraining on short sequences and then pre-adapting the model to an efficient Transformer architecture on long sequence for additional training steps leads to superior performance compared to only long input pretraining or no adaptation at all. We also investigate several other model design choices such as position encoding schemes, encoder-decoder layer distributions, and the impact of discrepancies between pretraining and fine-tuning architecture hyperparameters.

Based on the findings from our empirical investigation, we adapt the pretrained $\text{PEGASUS}_{\text{Large}}$ model (Zhang et al., 2020) to tackle long input summarization on up to 16K input tokens. The resulting model, which we call PEGASUS-X attains top scores on long summarization tasks, outperforming much larger models like LongT5 (Guo et al., 2021) in some cases, and sets the state of the art of two tasks: GovReport and PubMed. Morever, impact on short input summarization performance is minimal. A smaller version which we call $\text{PEGASUS-X}_{\text{Base}}$ attains similar scores with much fewer parameters. The code and weights for both models will be released at `https://github.com/google-research/pegasus` and as well as in Hugging Face Transformers (Wolf et al., 2020). Beyond long input summarization, we believe that many of our findings will be useful to the community for efficiently adapting Transformer models to handle ever longer input sequences for other tasks.

In summary, our contributions are:

1. We evaluate a series of proposed efficient Transformer architectures as well as a host of other model tweaks, and report their efficacy as well as trade-offs on computational resources when applied to long input summarization tasks.
2. Based on our findings, we propose a recipe for adapting a short-context, pretrained Transformer encoder-decoder to longer inputs, and apply it to PEGASUS to greatly improve its long-document summarization performance, with comparable short-input performance.
3. We release model checkpoints for the resulting 568M-parameter model, which we call PEGASUS-X, and a smaller 272M-parameter model with most of the performance, $\text{PEGASUS-X}_{\text{Base}}$.

2 Challenges of Long Input Summarization

2.1 Computational Challenges

While summarization is fundamentally about extracting and compressing information from longer to shorter sequences, most commonly studied summarization tasks have had inputs on average shorter than the input sequence lengths of Transformer language models–typically 512 to 2048 tokens. As the ability for models to handle language has improved, the field has pushed for more challenging summarization tasks with longer input lengths. The quadratic scaling of the memory requirements and computation for the attention mechanism in Transformers poses a challenge to tackling these longer summarization tasks. Many memory- and compute-efficient variants of Transformers (Beltagy et al., 2020; Zaheer et al., 2020; Choromanski et al., 2021; Wang et al., 2020; Kitaev et al., 2020) have been proposed to address this constraint. However, even when incorporating efficient Transformer architectures that achieve approximately linear memory scaling with input sequences, it is still common for models to be pretrained on short sequence inputs and only be adapted to handle long sequences when fine-tuning on a downstream task, which may be suboptimal.

While using decoder-only autoregressive language models for summarization has received some recent attention (Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2022), encoder-decoder models still generally perform better and remain the architecture of choice for the task (Wang et al., 2022b). The asymmetry between the input length and summary lengths requires new considerations for resource limitations of models. Consider a summarization model with 12 encoder and 12 decoder layers, pretrained on an input length of 512 and fine-tuned on a task with input sequence length 16384,

using output length of 512 in both cases. Since pretraining is typically done with shorter sequences while fine-tuning uses long inputs for summaries, fine-tuning can now be more resource intensive and slower than pretraining, which is contrary to the conventional paradigm. Since the encoder inputs have increased 32×, the quadratic scaling in the memory consumption of the self-attention operation means that we expect the encoder self-attention to consume 1024× the amount of memory in fine-tuning relative to pretraining. Even if we use an efficient Transformer variant that achieves linear scaling in memory consumption and computation, both the encoder self-attention and decoder cross-attention operations still consume 32× the memory compared to pretraining. Besides attention, expensive operations such as the FFN that scale linearly with the input length also greatly increase the computation required both at training and inference.

On the other hand, the unique characteristics of long-document summarization may also prompt new solutions to these issues. For instance, if encoder computations over long sequences pose a compute bottleneck, we may consider using fewer encoder layers and more decoder layers, exchanging decoding speed at inference for faster training. The higher relative cost of fine-tuning can also justify greater efforts to adapt the pretrained model to fine-tune more quickly, via mixing short- and long input training curricula, adapting the model to efficient Transformer architectures via additional pretraining, and so on.

To address these questions and challenges, we conduct a series of ablation experiments investigating which approaches can lead to improvements in downstream summarization results, as well as the computational trade-offs therein.

2.2 Task/Dataset Challenges

A challenge in building long-document summarization models is the relative scarcity of long-input summarization datasets with sufficient data to train and evaluate models on. Recent work introducing new long-document summarization datasets has alleviated this problem somewhat (Chen et al., 2022; Shaham et al., 2022; Kryściński et al., 2021), although the relative scarcity of good datasets continue to make this a challenging problem to make progress on. The main issues in current datasets are: relative simplicity of summarization, lack of diverse inputs, potential leakage of data due to the data collection procedure, and low quantity of examples for training. We refer the reader to Wang et al. (2022a) for more discussion on the challenges of creating large, high-quality long-document summarization datasets.

3 Experimental Setup

Similar to Zhang et al. (2020), we perform the majority of our experiments with a $\text{PEGASUS}_{\text{Base}}$-sized model, before applying our findings to $\text{PEGASUS}_{\text{Large}}$-sized model.

3.1 Pretraining

We generally follow the recipe from PEGASUS (Zhang et al., 2020) for pretraining $\text{PEGASUS}_{\text{Base}}$-sized models. All experiments in our ablation study performed pretraining with C4 (Raffel et al., 2020) for 500k steps with 512 input tokens and 256 output tokens and a masking ratio of 45%, unless otherwise stated. For long input pretraining we extend the input length to 4096 tokens, adjust the masking ratio from 45% to 5.625%, reducing the ratio by a factor of 8 to account for the 8x increase in input sequence length. We also filter for only documents longer than 10000 characters.

3.2 Fine-tuning

We evaluate our pretrained models by fine-tuning on the arXiv (Cohan et al., 2018) and GovReport (Huang et al., 2021) long-context summarization tasks. Where relevant, we also fine-tune on the shorter-context XSUM and CNN/DailyMail tasks. For each experiment, we report the best validation set scores based on the geometric average (RG) of ROUGE-1, ROUGE-2 and ROUGE-L scores (Lin, 2004) based on the `rouge-score` package.[1] For arXiv, we fine-tune with an input length of up to 16384 tokens and 256 output tokens, while for GovReport we use an input length of 10240 input tokens and 1024 output tokens given the longer summaries for the task. For XSUM and CNN/Daily Mail, with use an input length of 512, and output lengths of 64 and 128 respectively, following PEGASUS hyperparameters. The full set of hyperparameters for fine-tuning can be found in Appendix 7. Unless otherwise stated, we directly switch over to the efficient Transformer architectures between pretraining (on shorter context) and

[1] https://github.com/google-research/google-research/tree/master/rouge

Encoder	XSUM		CNN/DM		arXiv		GovReport			
	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	Steps/s	Mem
Transformer	**40.0 / 16.9 / 32.0**	**27.9**	**39.5 / 19.0 / 28.6**	**27.8**	- / - / -	-	- / - / -	-	-	-
BigBird	39.6 / 16.7 / 31.7	27.6	39.3 / 18.2 / 28.1	27.2	46.8 / 19.6 / 28.0	29.5	60.5 / 28.5 / 30.1	37.3	0.31	1.88
Performer	36.5 / 14.0 / 28.7	24.5	37.4 / 17.4 / 26.9	26.0	39.0 / 13.2 / 23.8	23.1	55.8 / 20.2 / 24.7	30.3	0.96	1.12
Local	38.5 / 15.7 / 30.6	26.4	39.0 / 18.4 / 28.1	27.2	46.5 / 19.7 / 27.9	29.5	60.2 / 28.3 / 30.0	37.1	**1.00**	**1.00**
Global-Local	38.7 / 16.2 / 31.2	26.9	39.0 / 18.6 / 28.2	27.3	**47.6 / 20.2 / 28.5**	**30.1**	**61.4 / 29.3 / 30.6**	**38.0**	0.87	1.08

Table 1: Comparison of different encoder architectures on short (XSUM, CNN/DM) and long (arXiv, GovReport) summarization tasks. Training steps per second and memory are computed based on arXiv, and normalized to Local Transformer performance.

fine-tuning (on longer contexts), with no adaptation phase in between.

4 Experiments

4.1 Encoder architectures

We begin by investigating the efficacy of swapping the encoder for an efficient Transformer encoder to allow our models to incorporate longer input sequences while consuming reasonable amounts of device memory. We first consider two efficient encoder architectures that exemplify two different approaches to memory-efficient attention. Big Bird (Zaheer et al., 2020) takes the approach of using sparse attention computation, combining sliding-window attention, random attention and a set of global-attention tokens. Conversely, Performer (Choromanski et al., 2021) takes the approach of factorizing attention matrices via orthogonal random features. Both Big Bird and Performer have the benefit of requiring no new parameters to be introduced, and hence the weights from a pretrained Transformer can be ported directly to these architectures. Both model also performed well on the Long Range Arena tasks (Tay et al., 2021). However, for this experiment, we perform both pretraining and fine-tuning with the same encoder architecture to avoid the issue of mismatch between pretraining and fine-tuning architectures.

In addition, we also introduce two simple variants of local attention Transformer encoders. First, we use a simple block-local Transformer (Local), where encoder input tokens are divided into non-overlapping blocks, tokens can only attend to other tokens within the block. Second, we extend this local Transformer by adding a set of global tokens with learnable embeddings, that can attend to and be attended from every encoder token (Global-Local). These components are similar in principle to the sliding window attention and global token attention of Big Bird, as well as similar constructs in other efficient Transformers such as ETC (Ainslie et al., 2020) and Longformer (Beltagy et al., 2020). However, we opt for the simpler block-local attention rather than sliding window attention, and compensate for the lack of overlapping blocks by staggering the local attention blocks, which we elaborate on in Section 4.2. As we show below, the performance is highly competitive despite its simplicity.

BigBird, Local and Global-Local all use a block size of 64, and 32 global tokens where relevant. Performer uses 256 random features.

Results on short and long summarization tasks are shown in Table 1, with the relative training steps per second and memory consumed per device for fine-tuning on arXiv shown in the right-most columns. Among the short tasks, the full-attention Transformer performs best, followed by BigBird. On the long tasks, Big Bird and Global-Local models perform best, but Big Bird consumes significantly more memory and trains much more slowly than the other architectures. Conversely, we find that although the Performer has relatively low memory consumption and trains efficiently, it performs the worst out of the architectures we tested by a noticeable margin.

On the other hand, we find that the Local and Global-Local encoders strike a good balance of both performance and efficiency. The simple local attention encoder, which uses a block-local attention mechanism, attains performance surprisingly close to that of Big Bird while being much faster and using much less memory. The Global-Local encoder trades off a small amount of speed and memory for better performance, outperforming Big Bird. While both Local and Global-Local models underperform Big Bird and Transformer for short-tasks, it appears that the model architectures make the right trade-offs for performance on long summarization tasks.

Encoder	Stagger Local Blocks	Use Global In Decoder	arXiv		GovReport	
			R1 / R2 / RL	RG	R1 / R2 / RL	RG
Global-Local	✓	✓	**48.1 / 20.3 / 28.5**	**30.3**	60.5 / 28.8 / 30.5	37.6
Global-Local		✓	47.0 / 19.5 / 27.9	29.5	60.9 / 28.9 / 30.2	37.6
Global-Local	✓		47.7 / 20.4 / 28.6	30.3	**61.3 / 29.4 / 30.8**	**38.1**
Global-Local			46.7 / 19.5 / 27.9	29.4	59.5 / 27.8 / 29.4	36.5
Local	✓	-	46.8 / 19.7 / 28.0	29.6	59.2 / 27.9 / 30.0	36.7
Local		-	46.5 / 19.2 / 27.5	29.1	58.8 / 27.5 / 28.9	36.0

Table 2: Comparison of architectural tweaks to Local and GlobalLocal encoder. Staggering local blocks uses different blocks boundaries for different layers in block-local attention. Global information is incorporated in the decoder via an additional cross-attention before cross-attention over the encoded input.

(a) Block-local attention

(b) Block-local attention with staggered blocks

Figure 2: In block-local attention (a), the same block boundaries are used across all layers, preventing information from being shared across blocks. Staggering the block boundaries (b) be shifting the boundaries every other layer allows for cross-block interactions with minimal additional computational cost or complexity.

Takeaways: Local attention is a surprisingly strong baseline, while adding global tokens significantly improves performance, and both models are resource-efficient.

4.2 Local and Global-Local configurations

Given the good performance of both Local and Global-Local encoder variants, we next consider further architectural tweaks to these models.

First, we introduce ***staggering*** of local attention blocks. Unlike in sliding window attention, in block-local attention tokens can only attend to other tokens within the same block. If the input tokens are divided up into the same blocks in every layer, this means that no information is exchanged across blocks through the entire encoder. To address this pitfall, we introduce a small architectural change wherein we stagger the block allocation across alternating layers. We show an example of this in Figure 2. Concretely, we stagger attention blocks by shifting the block boundaries by half a block every other layer: in practice, we implement this by padding the hidden representations on either side by half a block and masking accordingly.

Secondly, in the Global-Local model, the decoder only attends to the encoded token representations, and not the global token representations. We consider a variant where we supply the global token representations to the decoder, and in particular introduce a second encoder-decoder cross-attention that attends only to the global tokens, before performing cross-attention over the encoded tokens. Our goal is to allow the decoder to incorporate global information before performing cross-attention over the encoded sequence.

We show the results of both of these changes in Table 2. We find that staggering local blocks improves performance in both Local and Global-Local models by a noticeable amount. We highlight that this improves performance even with the Global-Local models, which already has a channel for cross-block interactions via global tokens, indicating that both of these model improvements are complementary. Conversely, we did not find that incorporating global token information in the decoder led to much of a performance improvement, particular once staggered local blocks were used.

Takeaways: Staggering local attention blocks significantly improves performance, and is complementary to global tokens.

4.3 Global-Local: Block Size and Number of Global Tokens

Next, we vary the block size and number of global tokens for the Global-Local encoder, with results shown in Table 3.[2]

[2] A number of experiments with very small block sizes or number global tokens ran into memory issues, owing to the

Block Size	Global Tokens	arXiv R1 / R2 / RL	arXiv RG	GovReport R1 / R2 / RL	GovReport RG	Steps/s	Mem
4	8	46.2 / 19.1 / 27.5	29.0	60.1 / 28.0 / 29.7	36.8	0.77	1.27
	32	46.1 / 18.8 / 27.2	28.7	60.1 / 27.6 / 28.9	36.3	0.65	1.70
	64	- / - / -	-	60.1 / 27.7 / 29.0	36.4	-	-
	128	- / - / -	-	- / - / -	-	-	-
16	8	46.9 / 19.6 / 27.9	29.5	60.1 / 28.2 / 29.7	36.9	0.98	1.03
	32	47.1 / 20.0 / 28.3	29.9	59.7 / 27.8 / 29.2	36.5	0.92	1.15
	64	46.8 / 19.7 / 28.0	29.6	60.8 / 28.6 / 30.0	37.4	0.75	1.54
	128	47.7 / 20.0 / 28.2	30.0	60.7 / 28.8 / 30.2	37.5	0.58	1.70
64	8	46.8 / 19.8 / 28.0	29.6	61.2 / 28.8 / 30.2	37.6	0.98	1.06
	32	47.7 / 20.3 / 28.5	30.2	61.0 / 29.3 / 30.8	38.0	0.47	1.07
	64	47.4 / 20.2 / 28.5	30.1	60.9 / 29.1 / 30.7	37.9	0.94	1.10
	128	**47.8 / 20.4 / 28.6**	**30.3**	60.9 / 29.0 / 30.3	37.7	0.85	1.26
128	8	- / - / -	-	- / - / -	-	-	-
	32	46.9 / 19.7 / 28.0	29.6	60.9 / 28.7 / 30.1	37.5	**1.00**	**1.00**
	64	47.4 / 20.2 / 28.4	30.1	60.9 / 28.9 / 30.8	37.8	0.96	1.05
	128	47.1 / 20.0 / 28.3	29.9	61.0 / 28.9 / 30.6	37.8	0.90	1.15
256	8	46.8 / 20.0 / 28.2	29.8	60.7 / 29.3 / 30.9	38.0	0.96	1.07
	32	47.3 / 20.2 / 28.3	30.0	61.6 / 29.4 / 30.7	38.2	0.92	1.11
	64	47.2 / 20.2 / 28.4	30.0	59.2 / 28.6 / 30.5	37.2	0.88	1.16
	128	48.1 / 20.5 / 28.6	30.4	**61.7 / 29.3 / 30.8**	**38.2**	0.83	1.26
512	8	- / - / -	-	- / - / -	-	-	-
	32	46.7 / 19.7 / 28.1	29.6	59.8 / 28.2 / 29.8	36.9	0.77	1.35
	64	47.2 / 20.1 / 28.2	29.9	61.1 / 29.3 / 30.7	38.0	0.75	1.40
	128	47.2 / 20.0 / 28.2	29.9	61.0 / 29.3 / 30.7	38.0	0.71	1.51

Table 3: Varying the block size and the number of global tokens of a GlobalLocal encoder. Training steps per second and memory are computed based on arXiv, and normalized to the run with Block Size=128 and Global Tokens=32.

Broadly, we find that increasing either block size or global tokens leads to improved performance, with a corresponding increase in memory consumption and computation time. However, the effect size from going to larger block sizes is not large, and appears to saturate as we get to larger block sizes or number of global tokens. As such, increasing either of these hyperparameters is preferable if resources allow, but may not be a high priority compared to other potential model improvements. For the remainder of the ablation experiments, we stick to a block size of 64 and 32 global tokens for consistency.

Takeaways: Larger block sizes and/or number of global tokens leads to improved performance, although the effect saturates.

4.4 Position Encoding Schemes

New position encoding schemes encoding schemes such as RoPE (Su et al., 2021) and ALiBi (Press et al., 2022) have garnered recent attention, showing improved performance on downstream evaluations. As input sequence lengths have gotten much longer, and in particular longer than the dimensions of hidden representations, previous choices of position encoding may no longer be optimal. Moreover, relative position encodings such as RoPE, T5 and ALiBi may be better suited for adapting models to different input lengths between pretraining and fine-tuning. Hence, this is a good opportunity to revisit the choice of positioning encoding schemes in encoder models.

Because of the more complex interaction between local attention blocks and relative position encoding implementations, we conduct a preliminary investigation with a full-attention Transformer. We pretrain with an input length of 512, and fine-tune with an input length of 2048 for the long sequence tasks – this experiment also tests the propensity for position encodings to be adapted to longer sequences downstream. In addition to the sinusoidal position encoding used in PEGASUS and Vaswani et al. (2017), we also consider the bucket-based relative position encoding scheme of T5, RoPE, absolute position embeddings, and

way in which TPUs pad small dimensions of arrays to certain minimum lengths, leading to larger than expected memory consumption.

Position Encoding	XSUM		CNN/DM		arXiv		GovReport		Step/s
	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	
None	34.3 / 12.5 / 26.8	22.6	25.6 / 7.8 / 17.7	15.2	36.1 / 9.8 / 22.0	19.8	38.3 / 13.2 / 18.7	21.1	0.96
Sinusoidal	39.8 / 16.9 / 31.8	27.8	**40.0 / 18.6 / 28.4**	**27.6**	44.5 / 17.6 / 26.7	27.6	40.0 / 18.8 / 22.3	25.6	0.96
T5	**40.1 / 17.1 / 32.0**	**28.0**	39.8 / 18.8 / 28.6	27.8	**44.9 / 17.9 / 26.8**	**27.8**	**40.2 / 19.5 / 22.9**	**26.2**	0.53
RoPE	39.8 / 16.9 / 31.8	27.8	39.2 / 18.7 / 28.5	27.5	43.5 / 17.2 / 26.5	27.1	40.0 / 19.1 / 22.6	25.8	0.85
Absolute	39.1 / 16.4 / 31.3	27.2	39.7 / 18.7 / 28.5	27.7	44.3 / 17.5 / 26.5	27.4	38.6 / 17.5 / 21.1	24.2	**1.00**

Table 4: Comparison of position encodings schemes for a Transformer encoder-decoder. Training steps per second are computed based on arXiv summarization. Absolute position embeddings are replicated to longer input sequences, following Beltagy et al. (2020). Training steps per second is computed based on arXiv, and normalized to the run with absolute position embeddings.

Architecture	Enc	Dec	XSUM		CNN/DM		arXiv		GovReport	
			R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG
Local	18	6	37.4 / 15.0 / 29.7	25.5	39.0 / 18.2 / 27.9	27.0	46.0 / 19.4 / 27.6	29.1	58.9 / 27.4 / 29.1	36.1
	12	12	37.5 / 14.9 / 29.7	25.5	38.5 / 18.0 / 27.6	26.7	45.4 / 18.9 / 27.3	28.6	59.2 / 27.6 / 29.3	36.3
	6	18	37.7 / 15.1 / 29.9	25.7	38.5 / 18.1 / 27.7	26.9	46.3 / 19.3 / 27.6	29.1	59.4 / 27.8 / 29.5	36.5
Global-Local	18	6	**38.6 / 15.9 / 30.9**	**26.7**	39.2 / 18.5 / 28.2	27.3	47.3 / 20.1 / 28.3	30.0	60.2 / 28.7 / 30.6	37.5
	12	12	38.6 / 15.9 / 30.7	26.6	**40.0 / 18.6 / 28.3**	**27.6**	47.5 / 20.1 / 28.3	30.0	**61.1 / 29.3 / 30.7**	**38.1**
	6	18	37.7 / 15.1 / 29.9	25.7	38.5 / 18.1 / 27.7	26.9	46.4 / 19.5 / 27.9	29.3	60.3 / 28.6 / 30.0	37.2
Global-Local	18	12	38.5 / 15.7 / 30.6	26.4	38.7 / 18.4 / 28.1	27.1	47.3 / 20.0 / 28.3	29.9	60.2 / 29.2 / 31.0	37.9
	12	18	38.6 / 15.8 / 30.5	26.5	38.6 / 18.3 / 28.0	27.0	**47.5 / 20.3 / 28.5**	**30.2**	60.9 / 29.0 / 30.4	37.7

Table 5: Varying the distribution of encoder/decoder layers)

no position encoding as a baseline. For absolute position embeddings, we follow the recipe of Beltagy et al. (2020) and duplicate the learned position embeddings to handle longer sequences before fine-tuning. The chosen position encoding scheme is applied to all parts of the model, including both the encoder and the decoder. We do not experiment with ALiBi, as we found no natural way to adapt ALiBi to cross-attention.

Our results are shown in Table 4. We find that although T5 performs the best, it is also almost twice as slow as the other position encoding schemes, which is consistent with the findings of Press et al. (2022). Sinusoidal position encodings and RoPE perform only slightly worse than T5 with much better efficiency, making them more desirable choices. Given the much simpler implementation of sinusoidal position encodings, we opt to stick with them for the remainder of the experiments.

Takeaways: Sinusoidal position encodings still remain a good choice for long input Transformers.

4.5 Scaling Encoder and Decoder Layers

Scaling laws (Kaplan et al., 2020; Ghorbani et al., 2021; Zhang et al., 2022) that describe the empirical relationship between model sizes and performance have proven surprisingly consistent and gotten significant attention in recent years. We present in this section a small set of scaling experiments, exploring the distribution of layers between encoder and decoder.

Our results are shown in Table 5. In the top half, we fix the total number of layers to 24, and consider both encoder-heavy and decoder-heavy distributions, for both Local and Global-Local models. We observe that impact of distribution of encoder and decoder layers on performance is relatively small. For Local models, we see a slight boost from decoder-heavy models. For Global-Local models, we observe that a balanced encoder-decoder outperforms encoder- and decoder-heavy models, both of which perform about comparably.

We also consider cases where we further increase the size of either the encoder or decoder to 18 layers, shown in the second half of Table 5. We observe no improvement in performance over the 12/12-layer encoder-decoder, and suspect that other hyperparameters (e.g. hidden size) might be the bottleneck rather than the number of layers.

We highlight here that because of the asymmetry of the input and output lengths, there are different computational trade-offs to different balances of encoder and decoder layers. Encoder-heavy models require more memory because of the long input sequences, whereas decoder-heavy models are relative slower at inference because of the autoregressive nature of decoding. Given the relatively small

Pretraining → Fine-tuning	Block Size	arXiv		GovReport	
		R1 / R2 / RL	RG	R1 / R2 / RL	RG
Transformer → Local	4	46.2 / 19.6 / 27.9	29.3	60.0 / 28.3 / 29.8	37.0
	16	46.4 / 19.6 / 27.9	29.4	59.6 / 28.2 / 29.9	36.9
	64	46.5 / 19.5 / 27.8	29.3	59.5 / 28.0 / 29.6	36.7
	256	46.8 / 19.7 / 28.0	29.6	59.8 / 28.0 / 29.8	36.8
Local → Local	4	45.0 / 18.2 / 26.6	27.9	59.1 / 27.1 / 28.8	35.9
	16	45.9 / 19.1 / 27.5	28.9	59.0 / 27.5 / 29.3	36.2
	64	46.5 / 19.5 / 27.8	29.3	59.7 / 28.1 / 29.8	36.8
	256	47.1 / 19.9 / 28.1	29.8	59.7 / 28.5 / 30.3	37.2
Transformer → Global-Local	4	44.6 / 18.0 / 26.6	27.7	59.5 / 27.0 / 28.6	35.8
	16	46.0 / 19.2 / 27.5	29.0	60.3 / 28.2 / 29.8	37.0
	64	47.0 / 20.0 / 28.2	29.8	60.8 / 28.7 / 30.1	37.4
	256	47.6 / 20.3 / 28.4	30.2	60.8 / 28.7 / 30.0	37.4
Global-Local → Global-Local	4	46.1 / 18.8 / 27.2	28.7	60.1 / 27.6 / 28.9	36.3
	16	47.1 / 20.0 / 28.3	29.9	59.7 / 27.8 / 29.2	36.5
	64	**47.7 / 20.3 / 28.5**	**30.2**	61.0 / 29.3 / 30.8	38.0
	256	47.3 / 20.2 / 28.3	30.0	**61.6 / 29.4 / 30.7**	**38.2**

Table 6: Comparison of adapting models architectures between pretraining and fine-tuning.

difference in the margin of performance, memory or computational constraints may outweigh the performance differences in practical scenarios.

Takeaways: A balanced Global-Local model outperforms other variants, but the difference in performance may be outweighed by other resource considerations.

4.6 Pretraining vs Fine-tuning Architectures

Previous works using efficient Transformer encoders have generally taken the model weights of a full-attention Transformer pretrained on a shorter sequence, and adapted them to the efficient architecture either directly during fine-tuning (Zaheer et al., 2020), or with an intermediate stage of additional pretraining (Beltagy et al., 2020). In this section, we investigate if such an approach is optimal, or if the model would benefit from being pretrained with the efficient encoder from the beginning. Note that we are still performing pretraining on a short sequence (512 tokens), even with an efficient encoder.

We consider both pretraining with a Transformer and pretraining with the efficient architecture for both Local and Global-Local models. We also vary the block size, as the main difference between a Transformer and Local Transformer is the block size (aside from staggering, a Local model with block size 512 is equivalent to a dense Transformer), and hence the difference in block size also corresponds to the extent to which the model needs to adapt between architectures. When adapting from a pretrained Transformer encoder to a Global-Local architecture, because the Global-Local model relies on newly introduced global token embeddings, we initialize them by randomly sampling tokens from the vocabulary embeddings.

Our results are shown in Table 6. For Local models, we find that pretraining with local attention using small block sizes tends to hurt performance, but at moderate block sizes (e.g. 64) there is little difference between the two approaches. In contrast, we find that for Global-Local, pretraining with the efficient architecture tends to perform better. We hypothesize that this difference arises because of the presence of the learned global embedding tokens, which are randomly initialized when adapting from a pretrained Transformer and hence may benefit from pretraining and being jointly trained with the local attention.

Takeaways: For moderate block sizes, either pretraining or adapting to a Local encoder performs about equally well, but pretraining with a Global-Local encoder performs slightly better.

4.7 Pretraining Schemes

Up to this point, we have only considered pretraining with short sequences. We might expect that pretraining with longer sequences ought to improve performance of our model on downstream long input summarization. However, pretraining only on long sequences is computationally expensive and requires a large collection of long input documents, which are relatively rarer. Moreover, long documents may contain different information from short documents, hence limiting training to only

Pretraining Scheme	Encoder	XSUM		CNN/DM		arXiv		GovReport	
		R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG
Short (50%)	Local	38.4 / 15.8 / 30.6	26.5	39.2 / 18.1 / 27.9	27.1	46.8 / 19.7 / 28.0	29.6	60.1 / 28.3 / 29.8	37.0
	Global-Local	39.4 / 16.5 / 31.5	27.4	39.1 / 18.6 / 28.3	27.4	47.7 / 20.4 / 28.6	30.3	61.9 / 29.6 / 30.8	38.4
Short (100%)	Local	39.2 / 16.3 / 31.3	27.1	39.2 / 18.6 / 28.3	27.4	46.9 / 19.7 / 28.0	29.6	60.1 / 28.3 / 29.8	37.0
	Global-Local	**39.9 / 17.0 / 31.9**	**27.9**	39.8 / 18.6 / 28.3	27.6	48.1 / 20.5 / 28.7	30.5	61.9 / 29.6 / 30.8	38.4
Short (75%) → Long (25%)	Local	38.8 / 15.9 / 30.7	26.7	39.1 / 18.2 / 28.0	27.1	47.5 / 20.1 / 28.2	30.0	60.6 / 28.9 / 30.6	37.7
	Global-Local	39.6 / 16.8 / 31.7	27.6	**39.8 / 18.8 / 28.5**	**27.7**	48.4 / 20.7 / 28.8	30.7	61.8 / 29.8 / 31.1	38.5
Short (50%) → Long (50%)	Local	38.4 / 15.7 / 30.5	26.4	39.4 / 18.1 / 27.9	27.1	47.7 / 20.2 / 28.3	30.1	60.9 / 29.1 / 30.7	37.9
	Global-Local	39.3 / 16.4 / 31.4	27.3	39.4 / 18.3 / 28.1	27.3	**48.4 / 20.9 / 29.1**	**30.9**	**61.7 / 30.0 / 31.2**	**38.7**
Long (100%)	Local	36.0 / 14.0 / 28.6	24.3	38.4 / 17.7 / 27.4	26.5	46.7 / 19.5 / 27.7	29.3	59.8 / 28.0 / 29.5	36.7
	Global-Local	36.4 / 14.3 / 28.9	24.7	38.5 / 17.8 / 27.5	26.6	47.3 / 19.9 / 28.1	29.8	61.1 / 29.1 / 30.7	37.9

Table 7: Comparison of different pretraining formats, given a input token budget of 131B tokens, which corresponds to 1M steps with 512 input tokens. Short pretraining uses 512 input tokens, whereas long pretraining uses 4096 input tokens.

long inputs maybe reduce the diversity of training data. Different long context Transformers have taken different approaches to pretraining on long inputs. For instance, Longformer (Beltagy et al., 2020) performed several additional stages of increasingly longer-sequence pretraining to adapt the initial RoBERTa to long sequence inputs. On the other hand, LongT5 (Guo et al., 2021) is pretrained exclusively with long input sequences. Others (Zaheer et al., 2020; Ivgi et al., 2022) perform no long input pretraining at all. In this section, we investigate how the balance of short and long pretraining impact downstream performance, and try to find the best trade-off between pretraining cost and downstream performance.

We consider two setups for pretraining: *short-input pretraining*, with 512 input tokens and 256 output tokens, and *long-input pretraining*, with 4096 input tokens and 256 output tokens. We describe the corresponding differences in data preprocessing in Section 3.1. We choose to fix the number of input tokens seen during training as the constraint, and vary configurations subject to this constraint. This constraint roughly proxies for the amount of compute consumed as well as corresponds to the number of input tokens seen during pretraining, in contrast to fixing the number of steps, where long-input pretraining would consume far more compute for the same number of steps.

In contrast to the above experiments where we generally performed short pretraining for 500k steps, we set our total input token budget at 131 billion tokens, which correponds to 1 million steps with 512 input tokens. This larger budget ensures that when we do only long-input pretraining, the model is still pretrained for a reasonable number of steps. Given this budget, we consider four configurations:

- Short-input pretraining for 100% of tokens (1M steps)
- Short-input for 75% of tokens (98.3B, 750k steps), then long-input for 25% of tokens (32.8B, 31.25k steps)
- Short-input for 50% of tokens (62.5B, 500k steps), then long-input for 50% of tokens (62.5B, 62.5k steps)
- Long-input pretraining for 100% of tokens (125k steps)

We compare the performance of the different pretraining scehemes in Table 7. We also include the short-input pretraining for 500k steps for comparison. First, comparing short-input pretraining for 500k and 1M steps, we find that more pretraining still improves performance, indicating that our base models may still be undertrained at 500k steps. Secondly, we observe that long-input pretraining performs consistently worse than the other variants, which we attribute to the fewer number of training steps taken, again highlighting the issue of potential under-training. Focusing our analysis on the middle three configurations, on the long tasks, we find that all three non-long-only variants atttain similar scores, with more long-input pretraining having slightly better performance, particularly on the ROUGE-2 and ROUGE-L scores. While the small absolute differences in scores make it hard to draw strong conclusions, we lean towards the conclusion that adding a short phase of long input pretraining can be beneficial can improve performance on long input summarization tasks.[3]

[3]One major difference from Longformer is that Longformer uses absolute position embeddings, hence it is potentially more important the model to have some pretraining with longer sequences to adapt the replicated position embed-

Cross-Attention	XSUM		CNN/DM		arXiv		GovReport		Step/s	Mem
	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG		
Full	**38.8 / 16.0 / 31.0**	**26.8**	39.5 / 18.6 / 28.4	27.5	47.7 / 20.4 / 28.6	30.3	**61.3 / 29.4 / 30.8**	**38.1**	1.00	1.00
Cross[0,2,4,6,8,10]	38.3 / 15.6 / 30.5	26.3	**39.8 / 18.8 / 28.5**	**27.7**	**48.1 / 20.4 / 28.6**	**30.4**	61.0 / 29.0 / 30.7	37.9	1.10	0.90
Cross[0,3,6,9,11]	38.0 / 15.3 / 30.2	26.0	38.8 / 18.4 / 28.1	27.2	46.9 / 19.9 / 28.2	29.7	60.1 / 28.6 / 30.2	37.3	1.15	0.88
Cross[0,4,8,11]	37.8 / 15.3 / 30.1	25.9	38.5 / 18.1 / 27.9	26.9	47.6 / 20.2 / 28.4	30.1	60.9 / 28.9 / 30.3	37.6	1.15	0.86
Cross[0,6,11]	37.4 / 14.8 / 29.7	25.4	38.8 / 18.1 / 27.9	27.0	46.9 / 19.7 / 28.1	29.6	60.3 / 28.5 / 30.2	37.3	1.18	0.87
Cross[0,6]	37.5 / 14.9 / 29.7	25.5	38.3 / 18.0 / 27.8	26.8	47.1 / 19.8 / 28.1	29.7	60.4 / 28.1 / 29.7	36.9	**1.21**	**0.85**

Table 8: Comparison of models with cross-attention only in a subset of the 12 decoder layers. Training steps per second and memory are computed based on arXiv, and normalized to the Cross[0,6] run.

Cross-Attention	Model	arXiv		GovReport	
		R1 / R2 / RL	RG	R1 / R2 / RL	RG
Pretrained	Full	47.7 / 20.4 / 28.6	30.3	**61.3 / 29.4 / 30.8**	**38.1**
	Cross[0,2,4,6,8,10]	**48.1 / 20.4 / 28.6**	**30.4**	61.0 / 29.0 / 30.7	37.9
	Cross[0,6]	47.1 / 19.8 / 28.1	29.7	60.4 / 28.1 / 29.7	36.9
Converted	Cross[0,2,4,6,8,10]	46.4 / 19.7 / 28.1	29.5	60.2 / 28.8 / 30.3	37.4
	Cross[0,6]	46.2 / 19.7 / 28.1	29.5	60.2 / 28.1 / 29.8	36.9

Table 9: Comparison of models pretrained with cross-attention for a subset of layers, and adapting a pretrained model by dropping cross-attention layers only during fine-tuning

Takeaways: Given a fixed compute budget, allocating some portion of training to long-input training can improve performance, although the precise optimal allocation is difficult to determine. Exclusively long pretraining results in worse performance.

4.8 Partial Cross Attention

Given the use of an efficient attention architecture, which has memory consumption scale linearly rather than quadratically in input sequence length, another major memory bottleneck is the encoder-decoder cross-attention. Because each decoder layer attends separately to the long encoder representations, and the attention is dense, this is a large contiguous chunk of memory that we could seek to reduce.

Perceiver AR (Hawthorne et al., 2022) demonstrated strong performance by using only a single cross-attention at the bottom layer of an autoregressive language model. Based on these results, we investigate the impact of only having cross-attention on a subset of decoder layers. In Table 8, we show the results of pretraining and fine-tuning Global-Local models with cross-attention only on specific layers on a variety of configurations. We find that reducing the number of cross-attention layers leads to a drop in performance, but the impact on performance is smaller than expected. For instance, with only cross-attention on the first and sixth layer, the Global-Local model still outperforms a Local model. The reduction of cross-attention layers also leads to a corresponding improvement in training step and reduction in memory consumption.

Given the small drop in performance from using fewer decoder layers with cross-attention, we consider the viability of dropping cross-attention layers after pretraining. In other words, we take a Global-Local model pretrained with full cross-attention, drop the cross-attention for a subset of layers, and fine-tune directly. Our results are shown in Table 9. We find that dropping the cross-attention after pretraining again only leads to a small (additional) dip in performance. This indicates that dropping cross-attention may be a viable strategy for further reducing memory requirements for an existing pretrained model with a small performance trade-off, and pretraining a separate model from scratch is not necessary.

Takeaways: Dropping cross-attention for a fraction of decoder layers can reduce memory consumption at the cost of slight performance regression. Cross-attention can be dropped after pretraining, with an associated performance trade-off.

5 PEGASUS-X

Based on our findings above, we settle on the following recipe for adapting the PEGASUS models

dings to capture different position information. In contrast, because our models use sinusoidal position encodings which can naturally extrapolate to longer input lengths, we find that fine-tuning has been sufficient to adapt the model to reasonable performance.

(Zhang et al., 2020) to long sequence summarization.

- We use a Global-Local architecture with block staggering, a large number of global tokens, and large block sizes during pretraining.
- We conduct an additional stage of long input pretraining on 4096 token inputs for 300,000 steps.
- We extend input sequences up to 16384 input tokens in fine-tuning, depending on the task.

We experiment with two model sizes **PEGASUS-X** (PEGASUS e**X**tended), based on $\text{PEGASUS}_{\text{Large}}$; and $\text{PEGASUS-X}_{\text{Base}}$, based on a newly trained $\text{PEGASUS}_{\text{Base}}$ model which we call $\text{PEGASUS}_{\text{Base+}}$. In a similar finding as Hoffmann et al. (2022), we found that $\text{PEGASUS}_{\text{Base}}$ benefits from training on significantly more tokens, which we set to the same as $\text{PEGASUS}_{\text{Large}}$.

We initialize the weights of PEGASUS-X and $\text{PEGASUS-X}_{\text{Base}}$ on the pretrained weights of $\text{PEGASUS}_{\text{Large}}$ and $\text{PEGASUS}_{\text{Base+}}$ respectively. Only two new sets of parameters introduced: the global token embeddings, and a separate LayerNorm for the global input representations in each Transformer layer. This is approximately 1M more parameters for $\text{PEGASUS-X}_{\text{Base}}$ and 2M more for PEGASUS-X. We initialize the global token embeddings by randomly sampling tokens from the input token embedding, and we initialize the LayerNorm weights with the regular input LayerNorm weights.

The task- and model-specific hyperparameters for fine-tuning can be found in Appendix 15. For this section, we report ROUGE-Lsum[4] rather than ROUGE-L for consistency with the metrics reported in other papers and leaderboards.

	$\text{PEGASUS-X}_{\text{Base}}$	PEGASUS-X
# Parameters	272M	568M
# Global Tokens	128	128
Block Size	512	512
Batch Size	512	1024
Additional Pretraining	300K steps	300K steps

Table 10: Hyperparameters of Pegasus-X Models

5.1 Results on Summarization tasks

Long summarization tasks In Table 11, we compare the performance of PEGASUS models to those of PEGASUS-X on three long-input summarization tasks: arXiv, Big Patent and PubMed. In all three tasks, we see significant improvements in performance of $\text{PEGASUS-X}_{\text{Base}}$ over $\text{PEGASUS}_{\text{Base+}}$, and PEGASUS-X over $\text{PEGASUS}_{\text{Large}}$. To isolate the impact of additional long input pretraining compared to only switching the architecture to accomodate long input sequences, we also include evaluation on the PEGASUS models using the Global-Local architecture with no further pretraining, which we list in the table as $\text{PEGASUS}_{\text{Base+}}$ + Global-Local.

We also compare to reported results of $\text{PEGASUS}_{\text{Large}}$ using the Big Bird architecture (Zaheer et al., 2020), Longformer encoder-ecoder (LED; Beltagy et al., 2020), the Top-Down Transformer (Pang et al., 2022) in both Average-Pool (AvgP) and Adaptive-Pool (AdaP) variants, the Large and XL sizes of LongT5, and the SLED (Ivgi et al., 2022). LED, Top-Down and SLED are all initialized with $\text{BART}_{\text{Large}}$ weights with no additional pretraining on long input sequences, although AdaP has a multi-step fine-tuning setup (see below).

We note that the Big Bird-PEGASUS uses only 3072 tokens context, which is likely due to the larger memory consumption of Big Bird. We find that PEGASUS-X outperforms Big Bird-PEGASUS on all tasks, and Top-Down-AvgP on both compared tasks. Top-Down-AdaP still outperforms PEGASUS-X, we highlight that Top-Down-AdaP uses a much more complex, multi-step fine-tuning setup, involving using an importance tagger on reference summaries to construct weights for pooling tokens within segments. In contrast, PEGASUS-X is fine-tuned with the standard fine-tuning pipeline. Even so, PEGASUS-X still outperforms Top-Down with adaptive pooling on PubMed. PEGASUS-X also outperforms LongT5 on both arXiv and PubMed summarization, despite both compared LongT5 models having more parameters. However, we find that LongT5 performs much better on BigPatent, which is a largely extractive summarization task. We hypothesize that a much larger much may be better at extraction over

[4] https://github.com/google-research/google-research/blob/master/rouge/README.md#two-flavors-of-rouge-l

Model	#Params	arXiv		Big Patent		PubMed	
		R1 / R2 / RLs	RG	R1 / R2 / RLs	RG	R1 / R2 / RLs	RG
$\text{PEGASUS}_{\text{Base}}$	271M	34.8 / 10.2 / 22.5*	20.0*	43.5 / 20.4 / 31.8*	30.5*	40.0 / 15.2 / 25.2*	24.8*
$\text{PEGASUS}_{\text{Base+}}$	271M	42.2 / 15.8 / 37.3	29.2	51.2 / 32.6 / 41.0	40.9	44.1 / 18.3 / 40.1	31.9
$\text{PEGASUS}_{\text{Base+}}$ + Global-Local	272M	47.6 / 20.2 / 42.4	34.4	58.1 / 39.5 / 47.2	47.7	47.3 / 21.4 / 43.0	35.2
$\text{PEGASUS-X}_{\text{Base}}$	272M	49.4 / 21.6 / 44.0	36.1	61.3 / 42.6 / 50.1	50.8	49.6 / 23.6 / 45.2	37.5
$\text{PEGASUS}_{\text{Large}}$	567M	44.7 / 17.2 / 25.7*	27.0*	53.4 / 32.9 / 42.1*	42.0*	45.1 / 19.6 / 27.4*	28.9*
PEGASUS-X	568M	50.0 / 21.8 / 44.6	36.5	64.8 / 47.5 / 54.3	55.1	**51.0 / 24.7 / 46.6**	**38.9**
Longformer Encoder-Decoder	464M	46.6 / 19.6 / 41.8	33.7	–.- / –.- / –.-	–.-	–.- / –.- / –.-	–.-
Top-Down (AvgP)	464M	48.7 / 20.7 / 43.9	35.4	–.- / –.- / –.-	–.-	48.3 / 21.4 / 44.2	35.7
Top-Down (AdaP)	464M	**51.0 / 21.9 / 45.6**	**37.1**	–.- / –.- / –.-	–.-	51.1 / 23.3 / 46.5	38.1
Big Bird-Pegasus	567M	46.6 / 19.0 / 41.8	33.3	60.6 / 42.5 / 50.1	50.5	46.3 / 20.7 / 42.3	34.4
$\text{LongT5}_{\text{Large}}$	770M	48.3 / 21.6 / 44.1	35.8	70.4 / 56.8 / 62.7	63.1	50.0 / 24.7 / 46.5	38.6
$\text{LongT5}_{\text{XL}}$	3B	48.4 / 21.9 / 44.3	36.1	**76.9 / 66.1 / 70.8**	**71.1**	50.2 / 24.8 / 46.7	38.7

Table 11: Comparison on long summarization tasks (Test sets). Results for other models are taken from their respective papers. *: PEGASUS (Zhang et al., 2020) only reports ROUGE-L and not ROUGE-LSum.

Model	CNN/DailyMail		XSum	
	R1 / R2 / RLs	RG	R1 / R2 / RLs	RG
$\text{PEGASUS}_{\text{Base}}$	41.8 / 18.8 / 38.9	38.9	39.8 / 16.6 / 31.7	27.6
$\text{PEGASUS}_{\text{Base+}}$	42.5 / 20.1 / 39.6	32.4	43.8 / 21.2 / 36.0	32.2
$\text{PEGASUS-X}_{\text{Base}}$	42.5 / 20.1 / 39.6	32.4	42.9 / 20.1 / 35.0	31.2
$\text{PEGASUS}_{\text{Large}}$	**44.2 / 21.5 / 41.1**	**33.9**	**47.2 / 24.6 / 39.2**	**35.7**
PEGASUS-X	43.4 / 21.2 / 40.6	33.5	45.8 / 22.8 / 37.6	34.0

Table 12: Comparison on short summarization tasks (Test sets)

very long encoded sequences.

Short summarization tasks We show in Table 12 the performance of PEGASUS and PEGASUS-X models on shorter summarization tasks. We observe that there is a slight regression in performance of both PEGASUS-X models compared to their PEGASUS equivalents. We hypothesize that the long input pretraining might negatively impact the performance on shorter input tasks because of the difference data filtering for long documents, resulting in a potentially less diverse training data distribution.

5.2 Results on SCROLLS Summarization Tasks

We report the performance of the PEGASUS-X models on the summarization tasks in the recently introduced SCROLLS benchmark in Table 13. This includes GovReport (Huang et al., 2021), the ForeverDreaming subset of SummScreen (Chen et al., 2022), and QMSum (Zhong et al., 2021).

We observe that PEGASUS-X outperforms all other models on GovReport, setting the state of the art on the dataset. PEGASUS-X performs comparably to both $\text{LongT5}_{\text{Large}}$ and Top-Down-AvgP on SummScreen/FD, although it underperforms both LongT5 models on QMSum. Moreover, we find that $\text{PEGASUS-X}_{\text{Base}}$ also performs competitively, outperforming both LongT5 models on GovReport, and only a small margin behind PEGASUS-X on all three tasks. $\text{PEGASUS-X}_{\text{Base}}$ also outperforms $\text{BART}_{\text{Large}}$-SLED, a larger model with a similar 16K token context length. A major difference between PEGASUS-X and $\text{BART}_{\text{Large}}$-SLED, besides being based on PEGASUS and BART respectively, is that $\text{BART}_{\text{Large}}$-SLED does not have additional pretraining on long documents. We also note that UL2 only uses a context length of 2K tokens.

6 Related Work

Long Document Summarization Several new long input summarization datasets and benchmarks have been recently introduced, providing better measures of long input summarization capability as well as prompting new interest in this research direction. The BookSum dataset (Kryściński et al., 2021) consists of paragraph, chapter, and full summaries of books on Project Gutenberg based on web-scraped educational website. (Chen et al., 2022) consists of television show transcripts and episode summaries based on web-scraped fan-

Model	#Params	GovReport		SummScreen/FD		QMSum	
		R1 / R2 / RL	RG	R1 / R2 / RL	RG	R1 / R2 / RL	RG
PEGASUS-X$_{\text{Base}}$	272M	59.3 / 29.3 / 30.9	37.7	35.0 / 8.9 / 20.4	18.5	32.9 / 9.8 / 21.4	19.0
PEGASUS-X	568M	**60.3 / 30.0 / 31.5**	**38.5**	35.7 / 9.1 / 20.6	18.8	33.2 / 9.6 / 21.6	19.0
BART$_{\text{Large}}$-SLED	406M	58.0 / 26.9 / 27.6	35.1	33.8 / 8.0 / 18.5	17.1	32.1 / 10.2 / 21.0	19.0
Top-Down-AvgP	464M	–.- / –.- / –.-	–.-	35.8 / 8.9 /30.6*	21.4*	–.- / –.- / –.-	–.-
Top-Down-AdaP	464M	–.- / –.- / –.-	–.-	**36.8 / 9.2 /31.1***	**21.9***	–.- / –.- / –.-	–.-
LongT5$_{\text{Large}}$	770M	54.2 / 27.8 / 29.8	35.5	35.6 / 9.2 / 21.2	19.1	**35.1 / 12.0 / 23.3**	**21.4**
LongT5$_{\text{XL}}$	3B	54.7 / 28.2 / 30.2	36.0	35.8 / 9.6 / 21.1	19.4	34.9 / 11.8 / 23.5	21.3
UL2	20B	53.6 / 26.1 / 28.8	34.3	32.9 / 7.8 / 19.4	17.1	31.1 / 8.5 / 20.4	17.5

Table 13: Comparison on SCROLLS benchmark (Summarization tasks, Test sets). Results for SLED, LongT5 and UL2 models are taken from the SCROLLS benchmark leaderboard. *: Top-Down (Pang et al., 2022) reports much higher scores for ROUGE-L on SummScreen/FD than any other model, and may have been computed with a variant of ROUGE-L that involves splitting on sentences rather than newlines.

written summaries. The SCROLLS benchmark (Shaham et al., 2022) and the MuLD benchmark (Hudson and Al Moubayed, 2022) consist of multiple natural language tasks with long inputs, including long input summarization. The SQuALITY dataset (Wang et al., 2022a) consists of question-focused summaries of Project Gutenberg stories, where annotators write summaries based on different questions that cover different aspects of the same story.

Efficient Transformers Many efficient Transformer variants have been introduced in recent years (Tay et al., 2020), and we discuss here the works more relevant to this manuscript. (Beltagy et al., 2020) use global tokens as well as a sliding window local attention, implemented using custom CUDA kernels. The ETC model (Ainslie et al., 2020) uses both global tokens and block-wise sliding window local attention, although the global attention is incorporated based on the first few tokens of a sequence, rather than separately learned global tokens. Zaheer et al. (2020) extend ETC by adding random attention blocks, but we found that this significantly increases code complexity and computational cost. Guo et al. (2021) similarly extend ETC's block-wise sliding window attention, but computes transient "global token" representations by pooling over blocks of tokens. Pang et al. (2022) propose to augment the Longformer encoder-decoder with additional pooling layers to improve long-sequence summarization performance. Ivgi et al. (2022) propose an alternative approach to sparse attention via encoding overlapping chunks and fusing information across chunks int he decoder. We highlight that while the final Global-Local model architecture that we settle on shares similarity with several other proposed efficient Transformer architectures, our key contribution lies in our extensive ablation study that identifies architectural tweaks that improve and, just as importantly, do not improve downstream performance.

Among the listed model architectures for long input summarization, LongT5 (Guo et al., 2021) is the most similar to PEGASUS-X, sharing a similar encoder-decoder architecture, a similar training objective in generating masked sentences, and a mix of local attention and global information sharing for the encoder. We briefly highlight the key differences between the two models. Firstly, LongT5 trains from scratch on long sequences, whereas we initialize our model weights with PEGASUS weights (which is trained on short sequences) before doing additional pretraining on long input sequences. This significantly reduces the overall pretraining cost, as short sequence pretraining and be performed much more economically. LongT5 also uses the T5 relative position biases whereas PEGASUS-X uses sinusoidal position embeddings–as shown in Section 4.4, T5 relative position biases perform slightly better but are significantly slower. The efficient encoder architecture between the two models is also different: LongT5 uses a transient global representations based on pooling chunks of tokens, whereas PEGASUS-X uses learned global token embeddings. LongT5 also uses a sliding window local attention based on ETC (Ainslie et al., 2020), whereas we use a simpler block-local attention with staggered blocks. Lastly, the largest LongT5 model is 3B parameters, more than 5$\times$ the size of PEGASUS-X.

More broadly, Tay et al. (2021) compare a variety of efficient Transformer architectures on a set of

tasks designed to probe long-sequence processing capability, evaluating the different models on both performance as well as computation requirements. Tay et al. (2022) further evaluate the scaling properties of novel Transformer architectures, finding that deviating from full attention tends to hurt downstream performance. Xiong et al. (2022) showed that simple local attention variants can be highly competitive with more complex sparse attention schemes, consistent with our findings.

7 Conclusion

In this work, we investigate a range of proposed improvements to allow Transformer models to effectively and economically handle long inputs in text summarization tasks. Through extensive ablation experiments, we find a simple but effective recipe for extending short input Transformers to tackle long-input summarization. Based on our findings, we introduce PEGASUS-X, an extended version of PEGASUS with a modified architecture and additional long-sequence pretraining. We show that PEGASUS-X sets the state of the art on two long input summarization tasks (GovReport and PubMed) and performs competitively on many others, even despite being much smaller than some compared models. Our findings can also be applied to extending models to handle long input sequences in other domains beyond summarization, both for pretraining long input models from scratch as well as extending already pretrained short sequence models.

References

Joshua Ainslie, Santiago Ontañón, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. Etc: Encoding long and structured data in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Mingda Chen, Zewei Chu, Sam Wiseman, and Kevin Gimpel. 2022. SummScreen: A dataset for abstractive screenplay summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8602–8615, Dublin, Ireland. Association for Computational Linguistics.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *International Conference on Learning Representations, ICLR 2021*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

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 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Behrooz Ghorbani, Orhan Firat, Markus Freitag, Ankur Bapna, Maxim Krikun, Xavier Garcia, Ciprian Chelba, and Colin Cherry. 2021. Scaling laws for neural machine translation. *arXiv preprint arXiv:2109.07740*.

Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2021. Longt5: Efficient text-to-text transformer for long sequences.

Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, Hannah Sheahan, Neil Zeghidour, Jean-Baptiste Alayrac, João Carreira, and Jesse Engel. 2022. General-purpose, long-context autoregressive modeling with perceiver ar.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. 2020. Flax: A neural network library and ecosystem for JAX.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *CoRR*, abs/2203.15556.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online. Association for Computational Linguistics.

George Hudson and Noura Al Moubayed. 2022. Muld: The multitask long document benchmark. In *Proceedings of the Language Resources and Evaluation Conference*, pages 3675–3685, Marseille, France. European Language Resources Association.

Maor Ivgi, Uri Shaham, and Jonathan Berant. 2022. Efficient long-text understanding with short-text models.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *CoRR*, abs/2001.04451.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. 2021. Booksum: A collection of datasets for long-form narrative summarization.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Bo Pang, Erik Nijkamp, Wojciech Kryściński, Silvio Savarese, Yingbo Zhou, and Caiming Xiong. 2022. Long document summarization with top-down and bottom-up inference.

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, and Omer Levy. 2022. Scrolls: Standardized comparison over long language sequences.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q. Tran, Dani Yogatama, and Donald Metzler. 2022. Scaling laws vs model architectures: How does inductive bias influence scaling?

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *CoRR*, abs/2009.06732.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Richard Yuanzhe Pang, Angelica Chen, Jason Phang, and Samuel R. Bowman. 2022a. SQuALITY: Building a long-document summarization dataset the hard way. *arXiv preprint 2205.11465*.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768.

Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. 2022b. What language model architecture and pretraining objective work best for zero-shot generalization?

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Wenhan Xiong, Barlas Oguz, Anchit Gupta, Xilun Chen, Diana Liskovich, Omer Levy, Scott Yih, and Yashar Mehdad. 2022. Simple local attentions remain competitive for long-context tasks. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1975–1986, Seattle, United States. Association for Computational Linguistics.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33.

Biao Zhang, Behrooz Ghorbani, Ankur Bapna, Yong Cheng, Xavier Garcia, Jonathan Shen, and Orhan Firat. 2022. Examining scaling and transfer of language model architectures for machine translation.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11328–11339. PMLR.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921, Online. Association for Computational Linguistics.

A Fine-tuning Hyperparameters

The hyperparameters for fine-tuning models are shown in Table 15.

B Engineering Details

The original PEGASUS model was trained using a codebase based on TensorFlow. The experiments in this paper were run using a new codebase written with JAX (Bradbury et al., 2018) and Flax (Heek et al., 2020). PEGASUS-$X_{Base}$and PEGASUS-Xwere trained by converting the weights from the TensorFlow checkpoint to a Flax checkpoint format, and then continuing with long input training.

	arXiv		GovReport	
Position Encoding	R1 / R2 / RL	RG	R1 / R2 / RL	RG
Factor=10000	48.1 / 20.4 / 28.6	30.4	60.9 / 29.3 / 30.8	38.0
Factor=50000	48.1 / 20.4 / 28.6	30.4	61.4 / 29.5 / 30.9	38.3

Table 14: Comparison of different scaling constants in sinusoidal position encodings.

Dataset	Batch Size	Learning Rate	Num Steps	Max Input Tokens	Max Output Tokens	Beam Size	Beam Alpha
PEGASUS-X_{Base}							
XSum	64	8e-4	97.5K	1024	128	4	0.8
CNN/DailyMail	64	8e-4	410K	1024	128	4	0.8
arXiv	64	8e-4	92.5K	16384	256	1	1
Big Patent	64	8e-4	272.5K	16384	256	1	1
PubMed	64	8e-4	85K	8096	256	1	1
GovReport	64	8e-4	40K	12288	1024	2	1
SummScreen	64	8e-4	90K	16384	256	1	1
QMSum	64	8e-4	7.5K	16384	256	1	1
PEGASUS-X							
XSum	64	8e-4	5k	1024	128	4	0.8
CNN/DailyMail	64	8e-4	7.5k	1024	128	4	0.8
arXiv	64	8e-4	85k	16384	256	1	1
Big Patent	64	8e-4	390k	12192	256	1	1
PubMed	64	8e-4	47.5k	12192	256	1	1
GovReport	64	8e-4	75K	12288	1024	1	1
SummScreen	64	8e-4	40K	12192	256	1	1
QMSum	64	8e-4	35K	12192	256	1	1

Table 15: Hyperparameters for fine-tuning models