Ring Attention with Blockwise Transformers for Near-Infinite Context

Hao Liu, Matei Zaharia, Pieter Abbeel
UC Berkeley
hao.liu@cs.berkeley.edu

Abstract

Transformers have emerged as the architecture of choice for many state-of-the-art AI models, showcasing exceptional performance across a wide range of AI applications. However, the memory demands imposed by Transformers limit their ability to handle long sequences, thereby creating challenges for tasks involving extended sequences or long-term dependencies. We present a distinct approach, Ring Attention, which leverages blockwise computation of self-attention to distribute long sequences across multiple devices while overlapping the communication of key-value blocks with the computation of blockwise attention. Ring Attention enables training and inference of sequences that are up to device count times longer than those of prior memory-efficient Transformers, effectively eliminating the memory constraints imposed by individual devices. Extensive experiments on language modeling tasks demonstrate the effectiveness of Ring Attention in allowing large sequence input size and improving performance.

1 Introduction

Transformers [39] have become the backbone of many state-of-the-art AI systems that have demonstrated impressive performance across a wide range of AI problems. Transformers achieve this success through their architecture design that uses self-attention and position-wise feedforward mechanisms. These components facilitate the efficient capture of long-range dependencies between input tokens, and enable scalability through highly parallel computations.

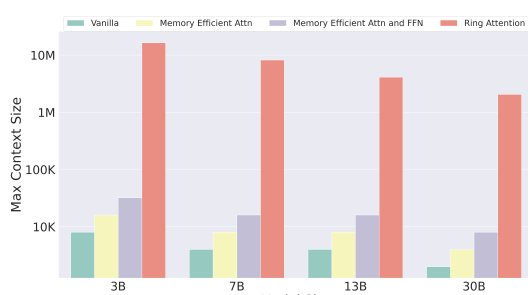


Figure 1: Maximum context length under end-to-end large-scale training on TPUv4-1024. Baselines are vanilla transformers [39], memory efficient transformers [31], and memory efficient attention and feedforward (blockwise parallel transformers) [24]. Our proposed approach Ring Attention allows training up to device count times longer sequence than baselines and enables the training of sequences that exceed millions in length without making approximations to attention.

However, scaling up the context length of Transformers is a challenge [30], since the inherited architecture design of Transformers, *i.e.* the self-attention has memory cost quadratic in the input sequence length, which makes it challenging to scale to longer input sequences. Large context Transformers are essential for tackling a diverse array of AI challenges, ranging from processing books and high-resolution images to analyzing long videos and complex codebases. They excel at extracting information from the interconnected web and hyperlinked content, and are crucial for handling complex scientific experiment data. There have been emerging use cases of language models with significantly expanded context than before: GPT-3.5 [33] with context length 16K, GPT-4 [30] with context length 32k, MosaicML's MPT [26] with context length 65k, and Anthropic's Claude [1] with context length 100k.

Driven by the significance, there has been surging research interests in reducing memory cost. One line of research leverages the observation that the softmax matrix in self-attention can be computed without materializing the full matrix [25] which has led to the development of blockwise computation of self-attention and feedforward [31, 9, 24] without making approximations. Despite the reduced memory, a significant challenge still arises from storing the output of each layer. This necessity arises from self-attention's inherent nature, involving interactions among all elements (n to n interactions). The subsequent layer's self-attention relies on accessing all of the prior layer's outputs. Failing to do so would increase computational costs cubically, as every output must be recomputed for each sequence element, rendering it impractical for longer sequences. To put the memory demand in perspective, even when dealing with a batch size of 1, processing 100 million tokens requires over 1000GB of memory for a modest model with a hidden size of 1024. This is much greater than the capacity of contemporary GPUs and TPUs, which typically have less than 100GB of high-bandwidth memory (HBM).

To tackle this challenge, we make a key observation: by performing self-attention and feedforward network computations in a blockwise fashion [24], we can distribute sequence dimensions across multiple devices, allowing concurrent computation and communication. This insight stems from the fact that when we compute the attention on a block-by-block basis, the results are invariant to the ordering of these blockwise computations. Our method distributes the outer loop of computing blockwise attention among hosts, with each device managing its respective input block. For the inner loop, every device computes blockwise attention and feedforward operations specific to its designated input block. Host devices form a conceptual ring, where during the inner loop, each device sends a copy of its key-value blocks being used for blockwise computation to the next device in the ring, while simultaneously receiving key-value blocks from the previous one. Because block computations take longer than block transfers, overlapping these processes results in no added overhead compared to standard transformers. By doing so, each device requires memory only proportional to the block size, which is independent of the original input sequence length. This effectively eliminates the memory constraints imposed by individual devices. Since our approach overlaps the communication of key-value blocks between hosts in a ring with blockwise computation, we name it Ring Attention.

We evaluate the effectiveness of our approach on language modeling benchmarks. Our experiments show that Ring Attention can reduce the memory requirements of Transformers, enabling us to train more than 500 times longer sequence than prior memory efficient state-of-the-arts and enables the training of sequences that exceed 100 million in length without making approximations to attention. Importantly, Ring Attention eliminates the memory constraints imposed by individual devices, empowering the training and inference of sequences with lengths that scale in proportion to the number of devices, essentially achieving near-infinite context size.

Our contributions are twofold: (a) proposing a memory efficient transformers architecture that allows the context length to scale linearly with the number of devices while maintaining performance, eliminating the memory bottleneck imposed by individual devices, and (b) demonstrating the effectiveness of our approach through extensive experiments.

2 Large Context Memory Constraint

Given input sequences $Q, K, V \in \mathbb{R}^{s \times d}$ where s is the sequence length and d is the head dimension. We compute the matrix of outputs as:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d}})V,$$

where $\mathrm{softmax}$ is applied row-wise. Each self-attention sub-layer is accompanied with a feedforward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

Blockwise Parallel Transformers. Prior state-of-the-arts have led to substantial reductions in memory utilization, achieved through innovative techniques that enable attention computation without full materialization by computing attention in a block by block manner [31, 9, 24]. These advancements lowered the memory overhead of attention to $2bsh$ bytes per layer, where b represents the batch size,

s denotes the sequence length, and h stands for the hidden size of the model. To further reduce memory usage, blockwise parallel transformer (BPT) [24] introduced a strategy where the feedforward network associated with each self-attention sub-layer is computed in a block-wise fashion. This approach effectively limits the maximum activation size of feedforward network from $8bsh$ to $2bsh$. For a more detailed analysis of memory efficiency, please refer to the discussion provided therein. In summary, the state-of-the-art transformer layer's memory cost of activation is $2bsh$.

Large Output of Each Layer. While BPT significantly reduces memory demand in Transformers, it still presents a major challenge for scaling up context length because it requires storing the output of each layer. This storage is crucial due to the inherent nature of self-attention, which involves interactions among all elements (n to n interactions). Without these stored outputs, the subsequent layer's self-attention becomes computationally impractical, necessitating recomputation for each sequence element. To put it simply, processing 100 million tokens with a batch size of 1 requires over 1000GB of memory even for a modest model with a hidden size of 1024. In contrast, modern GPUs and TPUs typically provide less than 100GB of high-bandwidth memory (HBM), and the prospects for significant HBM expansion are hindered by physical limitations and high manufacturing costs.

3 Ring Attention

Our primary objective is to eliminates the memory constraints imposed by individual devices by efficiently distribute long sequences across multiple hosts without adding overhead. To achieve this goal, we propose an enhancement to the blockwise parallel transformers (BPT) framework [24]. When distributing an input sequence across different hosts, each host is responsible for running one element of the outer loop of blockwise attention corresponding to its designated block, as well as the feedforward network specific to that block. These operations do not necessitate communication with other hosts. However, a challenge arises in the inner loop, which involves key-value block interactions that require fetching blocks from other hosts. Since each host possesses only one key-value block, the naive approach of fetching blocks from other hosts results in two significant issues. Firstly, it introduces a computation delay as the system waits to receive the necessary key-value blocks. Secondly, the accumulation of key-value blocks leads to increased memory usage, which defeats the purpose of reducing memory cost.

Ring-Based Blockwise Attention. To tackle the aforementioned challenges, we leverage the permutation invariance property of the inner loop's key-value block operations. This property stems from the fact that the self-attention between a query block and a group of key-value blocks can be computed in any order, as long as the statistics of each block are combined correctly for rescaling. We leverage this property by conceptualizing all hosts as forming a ring structure: host-1, host-2, ..., host-N. As we compute blockwise attention and feedforward, each host efficiently coordinates by concurrently sending key-value blocks being used for attention computation to the next host while receiving key-value blocks from the preceding host, effectively overlapping transferring of blocks with blockwise computation. Concretely, for any host-i, during the computation of attention between its query block and a key-value block, it concurrently sends key-value blocks to the next host-$(i+1)$ while receiving key-value blocks from the preceding host-$(i-1)$. If the computation time exceeds the time required for transferring key-value blocks, this results in no additional communication cost. This overlapping mechanism applies to both forward and backward passes of our approach since the same operations and techniques can be used.

Arithmetic Intensity Between Hosts. In order to determine the minimal required block size to overlap transferring with computation, assume that each host has F FLOPS and that the bandwidth between hosts is denoted as B. It's worth noting that our approach involves interactions only with the immediately previous and next hosts in a circular configuration, thus our analysis applies to both GPU all-to-all topology and TPU torus topology. Let's consider the variables: block size denoted as c and hidden size as d. When computing blockwise self-attention, we require $2dc^2$ FLOPs for calculating attention scores using queries and keys, and an additional $2dc^2$ FLOPs for multiplying these attention scores by values. In total, the computation demands amount to $4dc^2$ FLOPs. We exclude the projection of queries, keys, and values, as well as blockwise feedforward operations, since they only add compute complexity without any communication costs between hosts. This simplification leads to more stringent condition and does not compromise the validity of our approach. On the communication front, both key and value blocks require a total of $2cd$ bytes. Thus, the combined communication demand is $4cd$ bytes. To achieve an overlap between communication and

Figure 2: **Top (a):** We use the same model architecture as the original Transformer but reorganize the compute. In the diagram, we explain this by showing that in a ring of hosts, each host holds one query block, and key-value blocks traverse through a ring of hosts for attention and feedforward computations in a block-by-block fashion. As we compute attention, each host sends key-value blocks to the next host while receives key-value blocks from the preceding host. The communication is overlapped with the computation of blockwise attention and feedforward. **Bottom (b):** We compute the original Transformer block-by-block. Each host is responsible for one iteration of the query's outer loop, while the key-value blocks rotate among the hosts. As visualized, a device starts with the first query block on the left; then we iterate over the key-value blocks sequence positioned horizontally. The query block, combined with the key-value blocks, are used to compute self-attention (yellow box), whose output is pass to feedforward network (cyan box).

computation, the following condition must hold: $4dc^2/F \geq 4cd/B$. This implies that the block size, denoted as c, should be greater than or equal to F/B. Effectively, this means that the block size needs to be larger than the ratio of FLOPs over bandwidth.

Memory Requirement. A host needs to store multiple blocks, including one block size to store the current query block, two block sizes for the current key and value blocks, and two block sizes for receiving key and value blocks. Furthermore, storing the output of blockwise attention and feedforward necessitates one block size, as the output retains the shape of the query block. Therefore, a total of six blocks are required, which translates to $6bch$ bytes of memory. It's worth noting that the blockwise feedforward network has a maximum activation size of $2bch$ [24]. Consequently, the total maximum activation size remains at $6bch$ bytes. Table 1 provides a detailed comparison of the

Table 1: Comparison of maximum activation sizes among different Transformer architectures. Here, b is batch size, h is hidden dimension, n is number of head, s is sequence length, c is block size, the block size (c) is independent of the input sequence length (s). The comparison is between vanilla Transformer [39], memory efficient attention [31], memory efficient attention and feedforward [24], and our proposed approach Ring Attention. Numbers are shown in bytes per layer, assuming *bfloat16* precision.

Layer Type	Self-Attention	FeedForward	Total
Vanilla	$2bns^2$	$8bsh$	$2bhs^2$
Memory efficient attention	$2bsh + 4bch$	$8bsh$	$8bsh$
Memory efficient attention and feedforward	$2bsh$	$2bsh$	$2bsh$
Ring Attention	$6bch$	$2bch$	$6bch$

Table 2: Minimal sequence length needed on each device. Interconnect Bandwidth is the unidirectional bandwidth between hosts, *i.e.*, NVLink / InfiniBand bandwidth between GPUs and ICI bandwidth between TPUs. The minimal block size required $c = \text{FLOPS}/\text{Bandwidth}$, and minimal sequence length $s = 6c$.

Spec Per Host	FLOPS	HBM	Interconnect Bandwidth	Minimal Blocksize	Minimal Sequence Len
	(TF)	(GB)	(GB/s)	($\times$1e3)	($\times$1e3)
A100 NVLink	312	80	300	1.0	6.2
A100 InfiniBand	312	80	12.5	24.5	149.5
TPU v3	123	16	112	1.1	6.6
TPU v4	275	32	268	1.0	6.2
TPU v5e	196	16	186	1.1	6.3

memory costs between our method and other approaches. Notably, our method exhibits the advantage of linear memory scaling with respect to the block size c, and is independent of the input sequence length s.

Our analysis shows that the model needs to have a sequence length of $s = 6c$, which is six times the minimal block size. Requirements for popular computing servers are shown in Table 2. The required minimal sequence length (rightmost column) for each host varies between 6K and 10K, and the minimal block size (second-to-rightmost column) for each host is around 1K for TPUs and GPUs with high bandwidth interconnect. For GPUs connected via InfiniBand, which offers lower bandwidth, the requirements are more strict. In this case, we can consider extending Ring Attention to harness the GPU all-to-all topology, which we leave for future work. These requirements are easy to meet with parallelism and memory efficient blockwise attention and feedforward [31, 9, 24], which we will show in experiment Section 5.

Algorithm and Implementation. Algorithm 1 provides the pseudocode of the algorithm. Ring Attention is compatible with existing code for memory efficient transformers: Ring Attention just needs to call whatever available memory efficient computation locally on each host, and overlap the communication of key-value blocks between hosts with blockwise computation. We use collective operation `jax.lax.ppermute` to send and receive key value blocks between nearby hosts. A Jax implementation is provided in Appendix A.

4 Setting

We evaluate the impact of using Ring Attention in improving Transformer models by benchmarking maximum sequence length and model flops utilization.

Model Configuration. Our study is built upon the LLaMA architecture, we consider 3B, 7B, 13B, and 30B model sizes in our experiments.

Baselines. We evaluate our method by comparing it with vanilla transformers [39] which computes self-attention by materializing the attention matrix and computes the feedforward network normally,

Algorithm 1 Reducing Transformers Memory Cost with Ring Attention.

```
Required: Input sequence x. Number of hosts N_h.
Initialize
Split input sequence into N_h blocks that each host has one input block.
Compute query, key, and value for its input block on each host.
for Each transformer layer do
  for count = 1 to N_h − 1 do
    for For each host concurrently. do
      Compute memory efficient attention incrementally using local query, key, value blocks.
      Send key and value blocks to next host and receive key and value blocks from previous
      host.
    end for
  end for
  for For each host concurrently. do
    Compute memory efficient feedforward using local attention output.
  end for
end for
```

transformers with memory efficient attention [31] and its efficient CUDA implementation [9], and transformers with both memory efficient attention and feedforward [24].

Training Configuration. For all methods, we apply full gradient checkpointing [5] to both attention and feedforward, following prior works [31, 24]. The experiments are on both GPUs and TPUs. For GPUs, we consider both single DGX A100 server with 8 GPUs and distributed 32 A100 GPUs. We also experiment with TPUs, from older generations TPUv3 to newer generations of TPUv4 and TPUv5e. We note that all of our results are obtained using full precision instead of mixed precision.

5 Results

In our experiments, our primary objective is to comprehensively evaluate the performance of Ring Attention across multiple key metrics, including maximum supported sequence length within accelerator memory, model flops utilization, and throughput. We compare Ring Attention's performance with several baseline models , including the vanilla transformers [39], transformers with memory efficient attention [31], and transformers with both memory efficient attention and feedforward [24], across different model sizes and accelerator configurations.

5.1 Evaluating Max Context Size

We evaluate maximum supported context length using fully sharded tensor parallelsim (FSDP) [11] which is widely used in prior end-to-end training [38, 12]. We note that no tensor parallelism is considered in our evaluations since our approach is independent of tensor parallelism. Practitioners can combine our method with tensor parallelism, which we will show in Section 5.2. Using FSDP allows us to set the same batch size in tokens for baselines and our approach, ensuring a fair comparison. Concretely, on n devices, FSDP is used to shard the model for baselines, which gives a sequence length of l. The total batch size in tokens is nl. We utilize FSDP along with Ring Attention to extend the sequence length to $\frac{nl}{m}$ and m sequences. This means that the total batch size in tokens remains the same, but Ring Attention enables a significantly larger context size. Table 3 summarizes the results of our experiments.

Our Ring Attention model consistently surpasses baselines, delivering superior scalability across diverse hardware setups. For example, with 32 A100 GPUs, we achieve over 1 million tokens in context size for 7B model, a 32 times improvement over previous best. Furthermore, when utilizing larger accelerators like TPUv4-512, Ring Attention enables a 256 times increase in context size, allows training sequences of over 30 million tokens. Furthermore, our Ring Attention model scales linearly with the number of devices, as demonstrated by the 8x improvement over previous best on 8 A100 and the 256x improvement on TPUv3-512. If a model can be trained with context size s on n GPUs using the blockwise attention and feedforward, with our Ring Attention approach, it becomes possible to train a model with a context size of ns.

Table 3: The maximum context length supported in end-to-end training using fully sharded data parallelism and various transformers architectures. We show different model sizes and accelerators. Baselines are vanilla transformer [39], transformer with memory efficient attention [31], and transformer with memory efficient attention and feedforward [24]. The context size is reported in tokens (1e3). Our Ring Attention substantially outperforms baselines and enables training sequences that are up to device count times longer than prior state-of-the-arts.

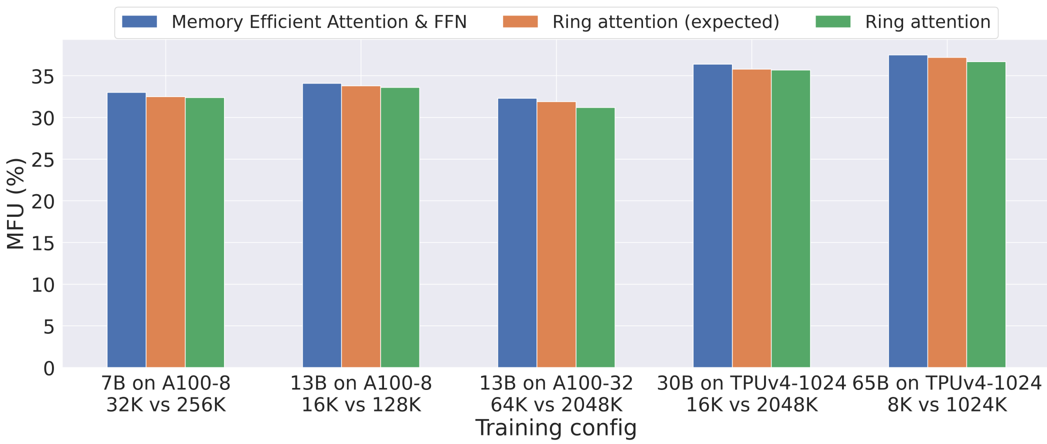
	Max context size supported ($\times 1e3$)				
	Vanilla	Memory Efficient Attn	Memory Efficient Attn and FFN	Ring Attention (Ours)	Ours vs SOTA
8x A100 NVLink					
3B	4	32	64	**512**	8x
7B	2	16	32	**256**	8x
13B	2	4	16	**128**	8x
32x A100 InfiniBand					
7B	4	64	128	**4096**	32x
13B	4	32	64	**2048**	32x
TPUv3-512					
7B	1	4	8	**2048**	256x
13B	1	2	8	**1024**	128x
TPUv4-1024					
3B	8	16	32	**16384**	512x
7B	4	8	16	**8192**	512x
13B	4	8	16	**4096**	256x
30B	2	4	8	**2048**	256x
TPUv5e-256					
3B	4	8	32	**4096**	128x
7B	2	8	16	**2048**	128x

5.2 Evaluating Model Flops Utilization

We evaluate the model flops utilization (MFU) of Ring Attention in standard training settings using fully sharded data parallelism(FSDP) [11] and tensor parallelism following LLaMA and OpenLLaMA [38, 12] with Jax SPMD. The batch size in tokens are 2M on 8/32x A100 and 4M on TPUv4-256. Our goal is investigating the impact of model size and context length on MFU, a critical performance metrics while highlighting the benefits of our approach. Table 5.1 presents the results of our experiments on MFU for different model sizes and context lengths. We present the achieved MFU using state-of-the-art memory efficient transformers BPT [24], compare it to our anticipated MFU based on these results, and demonstrate the actual MFU obtained with our approach (Ring Attention). For fair comparison, both BPT and our approach are based on the same BPT implementation on both GPUs and TPUs. It's worth noting that on GPUs our approach Ring Attention can be also integrated with the more compute efficient Triton code [18] or CUDA code [9] of memory efficient attention [31], similarly on TPUs it is also compatible with Pallas [37]. Combing these low level kernels implementations with our approach can maximize MFU, we leave that to future work.

Ring Attention trains much longer context sizes for self-attention, resulting in higher self-attention FLOPs compared to baseline models. Since self-attention has a lower MFU than feedforward, Ring Attention is expected to have a lower MFU than the baseline models. Our method offers a clear advantage in terms of maintaining MFU while enabling training with significantly longer context lengths. As shown in Table 5.1, when comparing our approach to prior state-of-the-arts, it is evident that we can train very large context models without compromising MFU or throughput.

Table 4: Model flops utilization (MFU) with different training configurations: model sizes, compute, and context lengths. Ring Attention enables training large models (7B-65B) on large input context sizes (over 4M) with negligible overheads.



	Model size	7B	13B	13B	30B	65B
	Compute	8x A100	8x A100	32x A100	TPUv4-1024	TPUv4-1024
Memory efficient attention & FFN	Context size ($\times$1e3)	32	16	64	16	8
Ring Attention	Context size ($\times$1e3)	256	128	2048	2048	1024

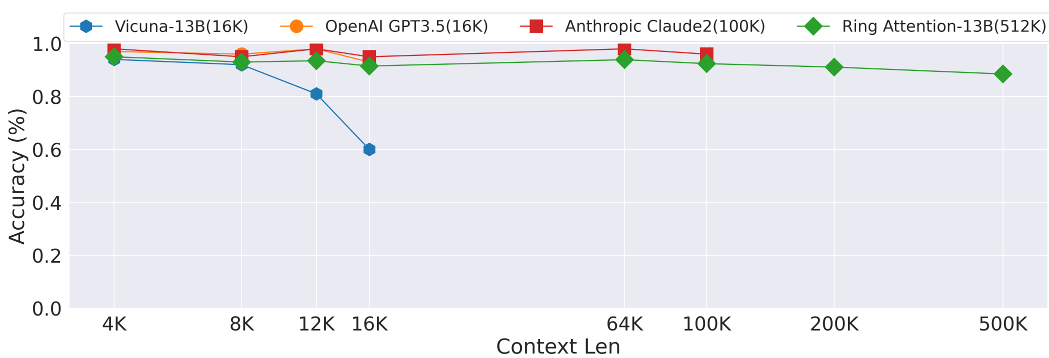


Figure 3: Comparison of different models on the long-range line retrieval task.

5.3 Impact on LLM Performance

We evaluate Ring Attention by applying our method to finetune LLaMA model to longer context. In this experiment, while our approach enables training with millions of context tokens, we conducted finetuning on the LLaMA-13B model, limiting the context length to 512K tokens due to constraints on our cloud compute budget. This finetuning was carried out on 32 A100 GPUs, using the ShareGPT dataset, following methodologies as outlined in prior works [6, 13]. We then evaluated our finetuned model on the line retrieval test [21]. In this test, the model needs to precisely retrieve a number from a long document, the task can effectively capture the abilities of text generation, retrieval, and information association at long context, reflected by the retrieving accuracy. Figure 3 presents the accuracy results for different models across varying context lengths (measured in tokens). Notably, our model, Ring Attention-13B-512K, stands out as it maintains high accuracy levels even with long contexts. GPT3.5-turbo-16K, Vicuna-16B-16K, and Claude-2-100K demonstrate competitive accuracy within short context lengths. However, they cannot handle extended context lengths.

6 Related Work

Transformers have garnered significant attention in the field of AI and have become the backbone for numerous state-of-the-art models. Several works have explored memory-efficient techniques to address the memory limitations of Transformers and enable their application to a wider range of problems. Computing exact self-attention in a blockwise manner using the tiling technique [25] has led to the development of memory efficient attention mechanisms [31] and its efficient CUDA implementation [9], and blockwise parallel transformer [24] that proposes computing both feedforward and self-attention block-by-block, resulting in a significant reduction in memory requirements. In line with these advancements, our work falls into the category of memory efficient computation for Transformers. Other works have investigated the approximation of attention mechanisms, yet these efforts have often yielded sub-optimal results or encountered challenges during scaling up. For an in-depth review of these techniques, we recommend referring to the surveys [27, 36]. Another avenue of research explores various parallelism methods, including data parallelism [10], tensor parallelism [35], pipeline parallelism [28, 15, 29], sequence parallelism [22, 19, 17], and FSDP [11, 32]. The activations of self-attention take a substantial amount of memory for large context models. Tensor parallelism can only reduce parts of activations memory and sequence parallelism introduces a significant communication overhead that cannot be fully overlapped with computation. Our work leverages on blockwise parallel transformers to distribute blockwise attention and feedforward across devices and concurrently overlaps the communication of key-value blocks in a circular of hosts with the computation of query-key-value blocks and feedforward, allowing number of devices times longer sequences with negligible overheads. Overlapping communication with computation has been studied in high performance computing literature [7, 40, 8, *inter alia*]. While ring communication has found applications in other parallel computing scenarios [2, 16, 14, 34], our work stands out as the first work to show that it can be applied to self-attention as used in Transformers and to make it fit efficiently into Transformer training and inference without adding significant overhead by overlapping blockwise computation and communication.

7 Conclusion

In conclusion, we propose a memory efficient approach to reduce the memory requirements of Transformers, the backbone of state-of-the-art AI models. Our approach allows the context length to scale linearly with the number of devices while maintaining performance, eliminating the memory bottleneck imposed by individual devices. Through extensive experiments on language modeling and reinforcement learning, we demonstrate its effectiveness, enabling training sequences that are up to device count times longer than those of prior memory-efficient Transformers, exceeding a context length of 100 million without making approximations to attention.

Limitations and Future Work. Although our method achieves state-of-the-art context length for Transformer models, it does have some limitations that need to be addressed:

- *Scaled up training*: due to compute budget constraint, our experiments focus on evaluation the effectiveness of the proposed approach without large scale training models.
- *Optimal compute performance*: While Ring Attention scales context length with device count while maintaining performance, optimizing low-level operations is required for achieving optimal compute performance. We suggest considering porting our method to CUDA, OpenAI Triton, or Jax Pallas in the future, for both maximum sequence length and maximum compute performance.

In terms of future prospects, the possibility of near-infinite context introduces a vast array of exciting opportunities, such as large video-audio-language models, learning from extended feedback and trial-and-errors, understanding and generating codebase, and adapting AI models to understand scientific data such as gene sequences.

References

[1] Anthropic. Introducing claude, 2023. URL `https://www.anthropic.com/index/introducing-claude`.

[2] Christian Bischof. *Parallel computing: Architectures, algorithms, and applications*, volume 15. IOS Press, 2008.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[6] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna.lmsys.org*, 2023.

[7] Anthony Danalis, Ki-Yong Kim, Lori Pollock, and Martin Swany. Transformations to parallel codes for communication-computation overlap. In *SC'05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, pages 58–58. IEEE, 2005.

[8] Anthony Danalis, Lori Pollock, Martin Swany, and John Cavazos. Mpi-aware compiler optimizations for improving communication-computation overlap. In *Proceedings of the 23rd international conference on Supercomputing*, pages 316–325, 2009.

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[10] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[11] Facebook. Fully Sharded Data Parallel: faster AI training with fewer GPUs — engineering.fb.com. `https://engineering.fb.com/2021/07/15/open-source/fsdp/`, 2023.

[12] Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, may 2023. *URL https://github. com/openlm-research/open_llama*, 2023.

[13] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. Koala: A dialogue model for academic research. *Blog post, April*, 1, 2023.

[14] Andrew Gibiansky. Bringing hpc techniques to deep learning. *Baidu Research, Tech. Rep.*, 2017.

[15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[16] Joshua Hursey and Richard L Graham. Building a fault tolerant mpi application: A ring communication example. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1549–1556. IEEE, 2011.

[17] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

[18] OpenAI kernel team. Openai triton fused attention, 2023. URL `https://github.com/openai/triton/blob/main/python/tutorials/06-fused-attention.py`.

[19] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *arXiv preprint arXiv:2205.05198*, 2022.

[20] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark. *arXiv preprint arXiv:2110.15191*, 2021.

[21] Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can open-source llms truly promise on context length?, June 2023. URL `https://lmsys.org/blog/2023-06-29-longchat`.

[22] Shenggui Li, Fuzhao Xue, Yongbin Li, and Yang You. Sequence parallelism: Making 4d parallelism possible. *arXiv preprint arXiv:2105.13120*, 2021.

[23] Hao Liu and Pieter Abbeel. Emergent agentic transformer from chain of hindsight experience. *International Conference on Machine Learning*, 2023.

[24] Hao Liu and Pieter Abbeel. Blockwise parallel transformer for large context models. *Advances in neural information processing systems*, 2023.

[25] Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

[26] MosaicML. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL `https://www.mosaicml.com/blog/mpt-7b`.

[27] Sharan Narang, Hyung Won Chung, Yi Tay, William Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

[28] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[29] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, pages 7937–7947. PMLR, 2021.

[30] OpenAI. Gpt-4 technical report, 2023.

[31] Markus N Rabe and Charles Staats. Self-attention does not need o(n2) memory. *arXiv preprint arXiv:2112.05682*, 2021.

[32] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[33] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J. F. C. Uribe, L. Fedus, L. Metz, M. Pokorny, R. G. Lopes, S. Zhao, A. Vijayvergiya, E. Sigler, A. Perelman, C. Voss, M. Heaton, J. Parish, D. Cummings, R. Nayak, V. Balcom, D. Schnurr, T. Kaftan, C. Hallacy, N. Turley, N. Deutsch, and V. Goel. Chatgpt: Optimizing language models for dialogue. *OpenAI Blog*, 2022. URL `https://openai.com/blog/chatgpt`.

[34] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[35] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[36] Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q Tran, Dani Yogatama, and Donald Metzler. Scaling laws vs model architectures: How does inductive bias influence scaling? *arXiv preprint arXiv:2207.10551*, 2022.

[37] Jax team. Jax pallas fused attention, 2023. URL `https://github.com/google/jax/blob/main/jax/experimental/pallas/ops/tpu/flash_attention.py`.

[38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, et al. Overlap communication with dependent computation via decomposition in large deep learning models. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 93–106, 2022.

[41] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

A Code

The implementation of Ring Attention in Jax is provided in Figure 4. We use `defvjp` function to define both the forward and backward passes, and use collective operation `jax.lax.ppermute` to facilitate the exchange of key-value blocks among a ring of hosts. The provided code snippet highlights essential components of Ring Attention. We provide the complete code of our Ring Attention at `https://github.com/lhao499/llm_large_context`. It is worth mentioning that, for maximum MFU, Ring Attention can be integrated with exiting kernel-level fused-attention implementations, such as Ring Attention can be integrated with Triton / CUDA / Pallas based code [18, 9, 37].

Practitioner Guide. For large scale end-to-end training on TPU or on GPU cluster with high bandwidth inter connection, we recommend using FSDP to shard large models and using Ring Attention to achieve large context. If total batch size is too large, add tensor parallelism to reduce the global batch size. The degree of parallelism can be adjusted using the `mesh_dim` parameter within the codebase. To illustrate, consider a setup with 512 devices, such as 512x A100. If the model size is 30B, you can shard it across 8 devices and allocate the remaining 32 devices for Ring Attention. This setup allows the context size to be expanded 32 times more than if you didn't use Ring Attention. Conversely, for models sized 7B or 3B, there is no need for FSDP. This means you can utilize all 512 devices exclusively to expand the context using Ring Attention by 512 times. Building upon the result that our approach allows for a 256K context size when using 8x A100 GPUs, it suggests that by employing 512 A100 GPUs, the potential context size can be expanded to 16 million.

B Experiment Details

B.1 Evaluation of context length

In the experimental results presented in Section 5.1, we used fully sharded tensor parallelism (FSDP) to partition the model across GPUs or TPU devices. Our evaluation focused on determining the maximum achievable sequence length in commonly used FSDP training scenarios. For TPUs, we utilized its default training configuration, which involved performing matmul operations in `bfloat16` format with weight accumulation in `float32`. On the other hand, for GPUs, we adopted the default setup, where all operations were performed in `float32`.

B.2 Evaluation of MFU

In the evaluation presented in Section 5.2. The batch size in tokens is 2 million per batch on GPU and 4 million per batch on TPU. The training was conducted using FSDP [11] with Jax SPMD. For gradient checkpointing [5], we used `nothing_saveable` as checkpointing policies for attention and feedforward network (FFN). For more details, please refer to Jax documentation.

B.3 Evaluation on line retrieval

In the evaluation presented in Section 5.3, we finetuned the LLaMA-13B model [38], limiting context length to 512K tokens due to constraints on our cloud compute budget, the training was conducted on 32x A100 80GB Cloud GPUs. We use user-shared conversations gathered from ShareGPT.com with its public APIs for finetuning, following methodologies as outlined in prior works [6, 13]. ShareGPT is a website where users can share their ChatGPT conversations. To ensure data quality, we convert the HTML back to markdown and filter out some inappropriate or low-quality samples, which results in 125K conversations after data cleaning.

C Training FLOPs Scaling of Context Size

Given that our proposed approach unlocks the possibility of training with a context size exceeding 100 million tokens and allows for linear scaling of the context size based on the number of devices, it is essential to understand how the training FLOPs per dataset scale with the context size. While a larger context size results in a higher number of FLOPs, the increased ratio does not scale quadratically because the number of tokens remains fixed. We present these results in Figure 5, which showcases various model sizes and context lengths, representing different computational budgets. The figure shows the ratio of FLOPs for larger context lengths compared to the same model with a shorter

```python
def _ring_attention_fwd(q, k, v, attn_bias, axis_name, float32_logits, blockwise_kwargs):
    if float32_logits:
        q, k = q.astype(jnp.float32), k.astype(jnp.float32)
    batch, q_len, num_heads, dim_per_head = q.shape
    batch, kv_len, num_heads, dim_per_head = k.shape
    numerator = jnp.zeros((batch, q_len, num_heads, dim_per_head)).astype(q.dtype)
    denominator = jnp.zeros((batch, num_heads, q_len)).astype(q.dtype)
    axis_size = lax.psum(1, axis_name)
    block_size = q_len # assumes this function is pre-sharded inside shard_map
    query_chunk_size = blockwise_kwargs["query_chunk_size"]
    key_chunk_size = blockwise_kwargs["key_chunk_size"]
    def scan_kv_block(carry, idx):
        prev_max_score, numerator, denominator, k, v = carry
        attn_bias_slice = lax.dynamic_slice_in_dim(attn_bias,
            (lax.axis_index(axis_name) - idx) % axis_size * kv_len, kv_len, axis=-1)
        q_block_idx = lax.axis_index(axis_name)
        k_block_idx = (lax.axis_index(axis_name) - idx) % axis_size
        q_chunk_idx_start = q_block_idx * (block_size // query_chunk_size)
        k_chunk_idx_start = k_block_idx * (block_size // key_chunk_size)
        numerator, denominator, max_score = _blockwise_attention_fwd(q, k, v,
            (numerator, denominator, prev_max_score), q_chunk_idx_start, k_chunk_idx_start,
            bias=attn_bias_slice, **blockwise_kwargs)
        k, v = map(lambda x: lax.ppermute(x, axis_name, perm=[(i, (i + 1) % axis_size)
            for i in range(axis_size)]), (k, v))
        return (max_score, numerator, denominator, k, v), None
    prev_max_score = jnp.full((batch, num_heads, q_len), -jnp.inf).astype(q.dtype)
    (max_score, numerator, denominator, _, _), _ = lax.scan(scan_kv_block,
        init=(prev_max_score, numerator, denominator, k, v), xs=jnp.arange(0, axis_size))
    output = numerator / rearrange(denominator, 'b h q -> b q h')[..., None]
    return output.astype(v.dtype), (output, q, k, v, attn_bias, denominator, max_score)

def _ring_attention_bwd(axis_name, float32_logits, blockwise_kwargs, res, g):
    output, q, k, v, attn_bias, denominator, max_score = res
    batch, kv_len, num_heads, dim_per_head = k.shape
    axis_size = lax.psum(1, axis_name)
    dq = jnp.zeros_like(q, dtype=jnp.float32)
    dk = jnp.zeros_like(k, dtype=jnp.float32)
    dv = jnp.zeros_like(v, dtype=jnp.float32)
    query_chunk_size = blockwise_kwargs["query_chunk_size"]
    key_chunk_size = blockwise_kwargs["key_chunk_size"]
    block_size = q.shape[1] # assumes this function is pre-sharded inside shard_map
    def scan_kv_block(carry, idx):
        dq, dk, dv, k, v = carry
        attn_bias_slice = lax.dynamic_slice_in_dim(attn_bias,
            (lax.axis_index(axis_name) - idx) % axis_size * kv_len, kv_len, axis=-1)
        q_block_idx = lax.axis_index(axis_name)
        k_block_idx = (lax.axis_index(axis_name) - idx) % axis_size
        q_chunk_idx_start = q_block_idx * (block_size // query_chunk_size)
        k_chunk_idx_start = k_block_idx * (block_size // key_chunk_size)
        dq, dk, dv = _blockwise_attention_bwd(q, k, v, g, (dq, dk, dv, output, denominator, max_score),
            q_chunk_idx_start, k_chunk_idx_start, bias=attn_bias_slice, **blockwise_kwargs)
        k, v, dk, dv = map(lambda x: lax.ppermute(x, axis_name, perm=[(i,
            (i + 1) % axis_size) for i in range(axis_size)]), (k, v, dk, dv))
        return (dq, dk, dv, k, v), None
    (dq, dk, dv, k, v), _ = lax.scan(scan_kv_block, init=(dq, dk, dv, k, v), xs=jnp.arange(0, axis_size))
    dq, dk, dv = dq.astype(q.dtype), dk.astype(k.dtype), dv.astype(v.dtype)
    return dq, dk, dv, None

@partial(jax.custom_vjp, nondiff_argnums=[4, 5, 6])
def ring_attention(q, k, v, attn_bias, axis_name, float32_logits, blockwise_kwargs):
    y, _ = _ring_attention_fwd(q, k, v, attn_bias, axis_name, float32_logits, blockwise_kwargs)
    return y

ring_attention.defvjp(_ring_attention_fwd, _ring_attention_bwd)
```

Figure 4: Key parts of the implementation of Ring Attention in Jax. We use collective operation `lax.ppermute` to send and receive key value blocks between previous and next hosts.

4K context size. We calculated the per sequence FLOPs using $(24bsh^2 + 4bs^2h)n$ where h is

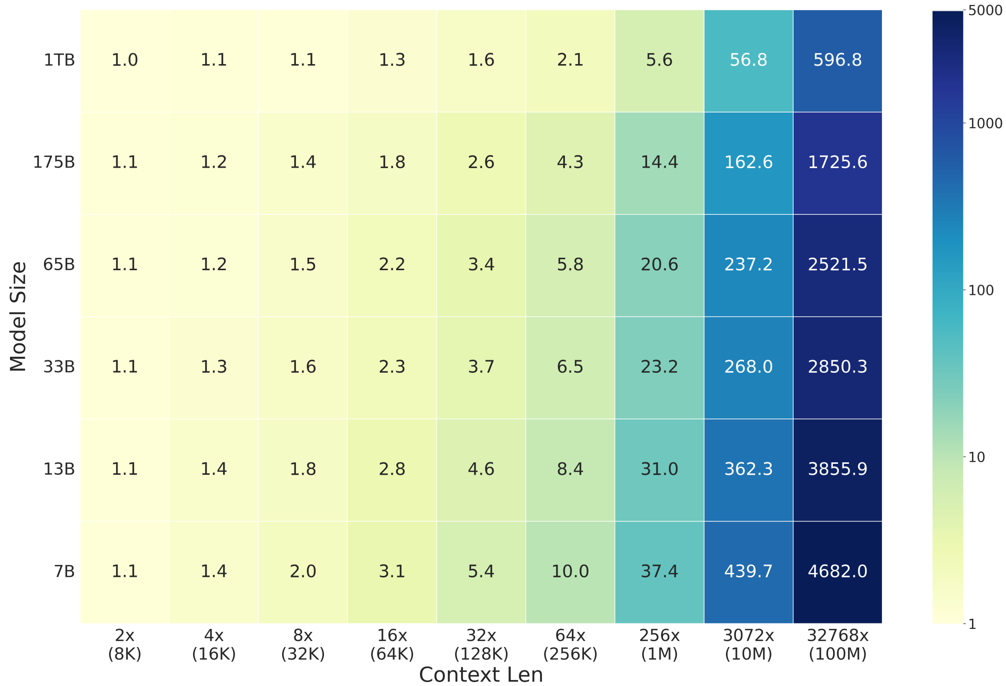


Figure 5: The per dataset trainig FLOPs cost ratio relative to a 4k context size, considering different model dimensions. On the x-axis, you'll find the context length, where, for example, 32x(128k) denotes a context length of 128k, 32x the size of the same model's 4k context length.

model hidden dimension, b is batch size, s is total sequence length, and n is number of layers. The per dataset FLOPs ratio is then given by $((24bs_2h^2 + 4bs_2{}^2h)/(24bs_1h^2 + 4bs_1{}^2h))/(s_2/s_1) = (6h + s_2)/(6h + s_1)$, where s_2 and s_1 are new and old context lengths. Model sizes and their hidden dimensions are as follows: LLaMA-7B (4096), LLaMA-13B (5140), LLaMA-33B (7168), LLaMA-65B (8192), GPT3-175B (12288), and 1TB (36864). These model configurations are from LLaMA [38] and GPT-3 [3] papers, except the 1TB model size and dimension were defined by us.

As depicted in Figure 5, scaling up small models to a 1M context size results in approximately 20-40 times more FLOPs, and even more for 10M and 100M token context sizes. However, as the model sizes increase, the cost ratio decreases. For instance, scaling up the 170B model from 4K to 10M incurs 162.6x higher per dataset FLOPs, despite the context size being 3072 times longer.

D Impact on In Context RL Performance

In addition to show the application of Ring Attention to finetune LLM in Section 5.3, we present additional results of applying Ring Attention for learning trial-and-error RL experience using Transformers. We report our results in Table 5, where we evaluate our proposed model on the ExoRL benchmark across six different tasks. On ExoRL, we report the cumulative return, as per ExoRL [41]. We compare BC, DT [4], AT [23], and AT with memory efficient attention [31] (AT+ME), AT with blockwise parallel transformers [24] (AT+BPT), and AT with our Ring Attention (AT+Ring Attention). The numbers of BC, DT, AT are from the ExoRL and AT paper. AT + Ring Attention numbers are run by ourselves. Since the ExoRL data is highly diverse, having been collected using unsupervised RL [20], it has been found that TD learning performs best, while behavior cloning struggles [41]. AT [23] shows that conditioning Transformer on multiple trajectories with relabeled target return can achieve competitive results with TD learning. For more details, please refer to their papers. We are interested in applying Ring Attention to improve the performance of AT by conditioning on a larger number of trajectories rather than 32 trajectories in prior works. It is worth noting that each trajectory has 1000×4 length where 1000 is sequence length while 4 is return-state-action-reward, making training 128 trajectories with modest 350M size model infeasible for prior state-of-the-art blockwise parallel transformers. Results in Table 5 show that, by scaling up the sequence length (number of

Table 5: Application of Ring Attention on improving Transformer in RL. BC and DT use vanilla attention. AT + ME denotes using memory efficient attention, AT + BPT denotes using blockwise parallel transformer. AT + RA denotes using Ring Attention.

ExoRL	**BC-10%**	**DT**	**AT + ME**	**AT + BPT**	**AT + BPT**	**AT + RA**
Task			N Trajs = 32	N Trajs = 32	N Trajs = 128	N Trajs = 128
Walker Stand	52.91	34.54	oom	95.45	oom	98.23
Walker Run	34.81	49.82	oom	105.88	oom	110.45
Walker Walk	13.53	34.94	oom	78.56	oom	78.95
Cheetah Run	34.66	67.53	oom	178.75	oom	181.34
Jaco Reach	23.95	18.64	oom	87.56	oom	89.51
Cartpole Swingup	56.82	67.56	oom	120.56	oom	123.45
Total Average	36.11	45.51	oom	111.13	oom	113.66

trajectories), AT + Ring Attention consistently outperforms oringal AT with BPT across all six tasks, achieving a total average return of 113.66 compared to the AT with BPT model's total average return of 111.13. The results show that the advantage of Ring Attention for training and inference with long sequences.