Sequence Accumulation and Beyond: Infinite Context Length on Single GPU and Large Clusters

Weigao Sun[1*], **Yongtuo Liu**[2], **Xiaqiang Tang**[3], **Xiaoyu Mo**[4*]
[1]Shanghai AI Laboratory
[2]University of Amsterdam
[3]The Hong Kong University of Science and Technology (Guangzhou)
[4]Nanyang Technological University
sunweigao@outlook.com, y.liu6@uva.nl, xtang771@connect.hkust-gz.edu.cn, xiaoyu006@e.ntu.edu.sg

Abstract

Linear sequence modeling methods, such as linear attention, state space modeling, and linear RNNs, have recently been recognized as potential alternatives to softmax attention thanks to their linear complexity and competitive performance. However, although their linear-memory advantage during training enables dealing with long sequences, it is still hard to handle extremely long sequences with very limited computational resources. In this paper, we propose Sequence Accumulation (SA) which leverages the common recurrence feature of linear sequence modeling methods to manage infinite context length even on a single GPU. Specifically, SA divides long input sequences into fixed-length sub-sequences and accumulates intermediate states sequentially, which reaches only constant-memory consumption. Additionally, we further propose Sequence Accumulation with Pipeline Parallelism (SAPP), to train large models with infinite context length, without incurring any additional synchronization costs in the sequence dimension. Extensive experiments with a wide range of context lengths are conducted to validate the effectiveness of SA and SAPP on both single and multiple GPUs. Results show that SA and SAPP enable the training of infinite context length on even very limited resources, and are well compatible with the out-of-the-box distributed training techniques.

Introduction

The attention mechanism (Vaswani et al. 2017; Qu et al. 2024) is recognized as a crucial component for effective sequence modeling. However, softmax attention has a complexity that grows quadratically with context length, making it inherently costly. Although recent advancements (Dao et al. 2022; Dao 2023; Shah et al. 2024) have enabled the scaling of softmax attention to longer sequences by optimizing intermediate computations for hardware, these approaches still necessitate the storage of key and value vectors. Managing this "KV cache" can become cumbersome when dealing with extremely long sequences.

Meanwhile, linear sequence modeling techniques, such as linear attention (Katharopoulos et al. 2020; Qin et al. 2022a; Yang et al. 2023; Qin et al. 2024c; Shen et al. 2024), state space modeling (SSM) (Gu and Dao 2023; Dao and Gu 2024), and linear RNNs (Peng et al. 2023, 2024; Qin et al. 2024d), have recently emerged as compelling alternatives to the conventional softmax-attention-based transformer architecture. These approaches are characterized by their linear complexity in both training computation and memory usage, which also eliminates the need for the KV cache, allowing for constant-memory inference. Although these methods originate from different technical backgrounds, recent works (Qin et al. 2024a; Yang et al. 2024) underscore the commonalities among these models, leading to the development of common techniques that boost the efficiency of linear sequence modeling methods.

Although the linear sequence modeling methods show the advantages of linear complexity and competitive performance, they still face challenges when modeling extremely long sequences with limited GPU resources. This is due to their linear-memory complexity with respect to the context length during training. To model extremely long sequences, previous methods rely on sequence or context parallelism techniques. However, sequence or context parallelism does not change the linear-memory complexity of linear sequence models, which also requires a lot more GPU resources and introduces heavy communication overheads.

In the paper, we propose Sequence Accumulation (SA) for linear sequence modeling methods to achieve constant-memory training. In this way, SA can enable infinite context length modeling even on a single GPU. Specifically, SA works by dividing a long input sequence into multiple sub-sequences (abbreviated as sub-seqs) and sequentially accumulating the intermediate states from each sub-seq into a single state. SA offers a distinct advantage when training with extremely long context lengths on limited computational resources. Additionally, we further propose Sequence Accumulation with Pipeline Parallelism (SAPP) to integrate SA with PP for training large models with infinite context length. SAPP accumulates intermediate states across model partitions on different devices without incurring any additional synchronization costs in the sequence dimension.

Through experiments conducted both on a single GPU and across multiple GPUs, we demonstrate the effectiveness of SA and SAPP on managing extremely long context lengths. Additionally, we evaluate the compatibility of SAPP with other parallel training strategies such as data parallelism (DP), tensor parallelism (TP), and context parallelism (CP), on a cluster with multiple nodes. The key contributions of this

*Corresponding Authors

Modeling Method	Formulation	Choice of λ
Linear Attention (Katharopoulos et al. 2020)	$KV_t = KV_{t-1} + K_t^T V_t$	$\setminus$
Lightning Attention (Qin et al. 2023b)	$KV_t = \lambda KV_{t-1} + K_t^T V_t$	$\lambda \in \mathcal{R}$
RetNet (Sun et al. 2023)	$KV_t = \lambda KV_{t-1} + K_t^T V_t$	$\lambda \in \mathcal{R}$
GLA (Yang et al. 2023)	$KV_t = diag\{\lambda_t\} KV_{t-1} + K_t^T V_t$	$\lambda_t \in \mathcal{R}^d$
DeltaNet (Yang et al. 2024)	$KV_t = (I - \lambda_t K_t^\top K_t) KV_{t-1} + \lambda_t K_t^T V_t$	$\lambda_t \in \mathcal{R}$
Mamba2 (Dao and Gu 2024)	$KV_t = \lambda_t KV_{t-1} + K_t^T V_t$	$\lambda_t \in \mathcal{R}$
RWKV-6 (Peng et al. 2024)	$KV_t = diag\{\lambda_t\} KV_{t-1} + K^T V_t$	$\lambda_t \in \mathcal{R}^d$
HGRN2 (Qin et al. 2024d)	$KV_t = diag\{\lambda_t\} KV_{t-1} + (1 - \lambda_t)^T V_t$	$\lambda_t \in \mathcal{R}^d$

Table 1: **Instances of Unified Linear Sequence Modeling.** All instances listed follow the unified formulation in Eq. 3. Here, λ represents a fixed constant, $\lambda_t \in \mathcal{R}$ refers to a time-dependent scalar, and $\lambda_t \in \mathcal{R}^d$ indicates a time-dependent vector. It is important to note that λ may denotes different constants or variables in each specific method.

paper are summarized as follows:

- *Sequence Accumulation.* We introduce the SA method, which accumulates intermediate states from each sub-seq, enabling constant-memory training with infinite context length even on a single GPU.
- *Sequence Accumulation Pipeline Parallelism.* SA is integrated with PP, allowing the accumulation of intermediate states across model partitions on different devices, thus facilitating the training of large models with infinite context lengths on large clusters.
- *Compatibility with Hybrid Parallelism.* We implement and evaluate the compatibility of SAPP with existing parallel training techniques, including DP, TP, and CP.

Preliminary

Notation Throughout this paper, we maintain consistent notations for arithmetic expressions. For clarity, matrices are denoted by uppercase letters, while vectors are represented by lowercase letters, both in non-boldface. Scalar or matrix multiplication is indicated by the symbol "$\cdot$" or is implied when the symbol is omitted, while Hadamard (element-wise) multiplication is represented by "$\odot$". The scalars N and d refer to the context length and hidden dimension, respectively. To simplify the notation, we omit the dimensions related to batch size and the number of heads in the tensor shapes.

Linear Attention The standard softmax attention (Vaswani et al. 2017), commonly used in transformer models, can typically be expressed as:

$$O = \text{softmax}(Q \cdot K^\top / \sqrt{d}) \cdot V. \tag{1}$$

Here, the matrices $Q, K, V, O \in \mathcal{R}^{N \times d}$ correspond to the query, key, value, and output matrices, respectively. The matrices Q, K, and V are linear projections of the input matrix $X \in \mathcal{R}^{N \times d}$, defined as $Q = XW_Q$, $K = XW_K$, and $V = XW_V$, where $W_Q, W_K, W_V \in \mathcal{R}^{d \times d}$ are learnable weight matrices.

Linear attention (Katharopoulos et al. 2020) introduces two key modifications to the standard softmax attention: 1) it eliminates the softmax$(\cdot)$ operation, thereby removing the need for the scaling factor $1/\sqrt{d}$; and 2) it alters the order of matrix multiplications by first computing $K^\top V$, followed by $Q(K^\top V)$. These adjustments reduce both the computational and memory complexity of attention from $O(N^2 d)$ to $O(Nd^2)$. This technique is often referred to as the right-product kernel trick because it prioritizes the multiplication on the right side first.

Another important feature of linear attention is that its computation of KV state can be processed in a recurrent form, similar to RNN, as follows:

$$KV_t = KV_{t-1} + K_t^\top \cdot V_t. \tag{2}$$

In this formulation, the intermediate state KV is updated at each time step t by adding the product of $K^\top \cdot V$ from the t-th input to the previous computed state KV_{t-1}. Note that the term KV here represents a single state or matrix, which is distinct from the key K and value V matrices.

Unified Linear Sequence Modeling Linear attention simplifies the standard softmax attention using the right-product kernel trick, which is a constructive step towards linear sequence modeling. Follow-up studies on state space modeling and linear RNNs, also demonstrate the linear sequence modeling is comparable with transformer architecture from the perspectives of control theory and routine RNN. Recent studies (Qin et al. 2024a; Yang et al. 2024) suggest that the linear attention, state space, and linear RNN sequence modeling methods can be expressed within a unified recurrence framework as:

$$\begin{aligned} \widehat{M}_t &= f(K_t^\top, V_t), \\ M_t &= \Theta_t \diamond M_{t-1} + \widehat{M}_t. \end{aligned} \tag{3}$$

In this formulation, $\widehat{M}_t \in \mathcal{R}^{d \times d}$ represents the memory state corresponding to the t-th input, which is a function of $K_t^\top$ and V_t. And Θ_t denotes a coefficient matrix that may be time-varying or constant (and also can be a vector or scalar). The operator "$\diamond$" can denote either standard matrix multiplication or a Hadamard product. We collect some recent linear sequence modeling methods which follow the unified formulation in Eq. 3 and list them in Table 1.

Method

In this section, we begin by introducing the SA method, followed by an explanation of its integration with PP. The visual

representations of SA and SAPP are provided in Figure 1 and Figure 2, respectively.

Sequence Accumulation

SA is a co-design of algorithm and system techniques aimed at enabling the training of models with long context length using minimal GPU resources, which introduces a new concept that differs from sequence parallelism. In existing sequence or context parallelism approaches, the input sequence is divided into multiple shards, which are then distributed across a group of devices to compute the individual Q, K, V components concurrently. These methods require complex communication operations to gather these shards, allowing attention computation to be performed with complete information across the sequence dimension.

Similar to sequence or context parallelism, SA also divides the entire input sequence X into T sub-sequences, denoted as $\{X_1, X_2, \cdots, X_T\}$. Differently, SA sequentially computes the query, key, and value states for each sub-sequence on the same device:

$$\begin{aligned} Q_t &= X_t W_Q, \quad K_t = X_t W_K, \\ V_t &= X_t W_V, \quad t \in \{1, 2, \cdots, T\}. \end{aligned} \tag{4}$$

Leveraging the recurrent form of linear sequence modeling as described in Eq. 3, the memory state variation $\widehat{M}_t$ for each sub-seq t can be computed and accumulated into M_t sequentially. The finally updated memory state M_t is then utilized to produce the output O_t. The accumulation process in SA closely mirrors the recurrent computation form of linear sequence modeling and can be expressed as:

$$\begin{aligned} \widehat{M}_t &= f(K_t^\top, V_t), \\ M_t &= \Theta \diamond M_{t-1} + \widehat{M}_t, \\ O_t &= Q_t M_t, \\ t &= 1, 2, \cdots, T, \end{aligned} \tag{5}$$

with the initial memory state

$$M_0 = 0 \in \mathcal{R}^{d \times d}. \tag{6}$$

The specific selections of f, Θ, and $\diamond$ in Eq. 5 determine how the SA computation process is instantiated. For instance, in the basic linear transformer model used in our experiments, $f(K_t^\top, V_t)$ is defined as $K_t^\top V_t$, Θ is set as a constant $\lambda \in \mathcal{R}$, and $\diamond$ corresponds to the matrix multiplication.

The forward computation for the t-th sub-sequence depends on the memory state from the preceding $(t-1)$ sub-sequences, denoted as M_{t-1}, which is precomputed and can be reused on the same device. During the backward pass, gradient computation for the t-th sub-sequence requires M_t as an intermediate activation. To prevent redundant computation of M_t and to expedite subsequent forward passes, we store the updated memory state M_t in the High-Bandwidth Memory (HBM) of the GPU. This allows M_t to be efficiently accessed during both the backward pass and the next forward pass. The forward procedure for SA is detailed in Algorithm 1, and the backward pass is provided in Algorithm 2. The backward pass is not performing any sort of truncation as is typical in truncated backpropagation through time in RNNs.

Algorithm 1 Sequence Accumulation (Forward)

Input: input sequence in embedding space $X \in \mathcal{R}^{N \times d}$.
Split X into T sub-seqs $\{X_1, X_2, \cdots, X_T\}$, obtain sub-seq length $S = N/T$.
Initialize memory state $M_0 = 0 \in \mathcal{R}^{d \times d}$.
Instantiate f, Θ and $\diamond$.
for sub-seq $t \in \{1, 2, \cdots, T\}$ on the same device **do**
    Compute $Q_t = X_t W_Q$, $K_t = X_t W_K$, $V_t = X_t W_V$.
    Compute $\widehat{M}_t = f(K_t^\top, V_t)$.
    Accumulate $M_t = \Theta_t \diamond M_{t-1} + \widehat{M}_t$.
    Compute $O_t = Q_t M_t$.
end for
Return: $O = [O_t]$, with $t \in \{1, 2, \cdots, T\}$.

Algorithm 2 Sequence Accumulation (Backward)

Input: Q_t, K_t, V_t, O_t, dO_t for $t = T$.
Initialize $dM_{T+1} = 0 \in \mathcal{R}^{d \times d}$.
for $t \in \{T, \cdots, 2, 1\}$ on the same device **do**
    Compute $dQ_t = dO_t M_t^\top$.
    Compute $\widehat{dM}_t = Q_t^\top dO_t$.
    Accumulate $dM_t = \Theta \diamond dM_{t+1} + \widehat{dM}_t$.
    Compute $dK_t = V_t dM_t^\top$.
    Compute $dV_t = K_t dM_t$.
end for
Return: $dQ = [dQ_t]$, $dK = [dK_t]$, $dV = [dV_t]$, with $t \in \{1, 2, \cdots, T\}$.

Note that the accumulated memory state M_t has a shape of $d \times d$, so storing M_t incurs only a small memory cost, which is independent of the context length N or sub-seq length S. Additionally, no communication operations are required in SA, as the entire process is executed on the same device.

Otherwise, SA can be used in conjunction with DP, TP or even CP. In these scenarios, the SA process runs concurrently on each device, with synchronization occurring between devices at each iteration within the data, tensor or context parallelism communication groups.

Sequence Accumulation Pipeline Parallelism

To enable the application of SA to large models with long sequences, we discovered that SA can be seamlessly integrated with PP. This integration allows for efficient training of long context lengths on large models on limited GPU resources. For the sake of simplifying the analysis of SAPP, we use GPipe (Huang et al. 2019; Kim et al. 2020) as a representative example of PP. A detailed exploration of more complex PP methods is outside the scope of this paper.

Consider a deep neural network with a sequence of L layers, each layer L_i is composed of a forward computation function f_i. Considering the consecutive layers between layers i and j, its forward function would be $F_{ij} = f_j \circ \cdots \circ f_{i+1} \circ f_i$. The corresponding back-propagation function B_{ij} can be computed from F_{ij} using automatic symbolic differentiation. PP partitions the network into P cells and places the p-th cell on the p-th accelerator. Communication primitives are

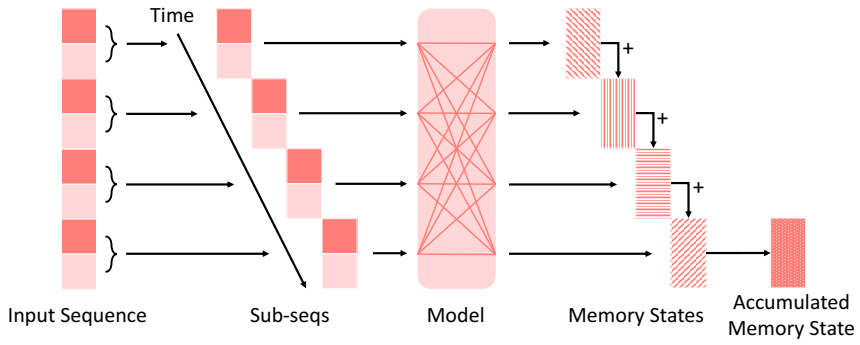

Figure 1: **Sequence Accumulation (SA).** The entire input sequence is divided into multiple sub-sequences, which are processed sequentially on the same device to compute their respective memory states. These memory states are progressively accumulated into a single state, which is then used to generate the output.

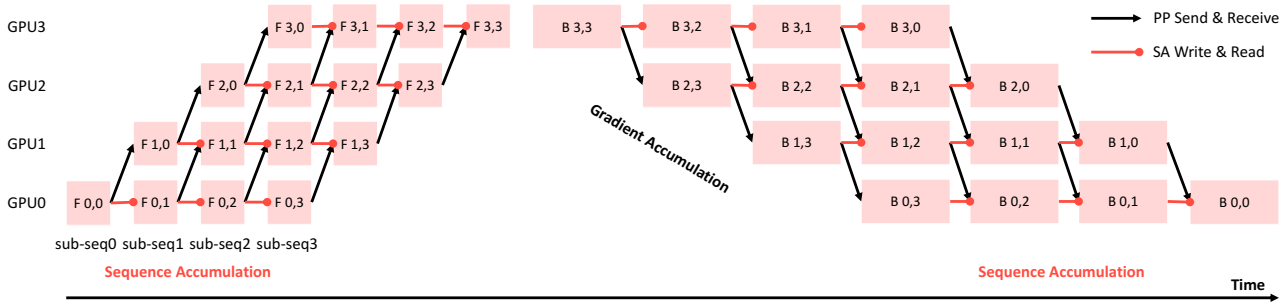

Figure 2: **Sequence Accumulation Pipeline Parallelism (SAPP).** We illustrate the operation of SAPP using a setup with 4 GPUs, employing code-style indexing to maintain consistency with other PP illustrations. The model is divided into 4 shards, each allocated to one of the 4 GPUs, and the input sequence is split into 4 sub-seqs, all initially placed on GPU0. Here, $F_{0,1}$ represents the forward pass of sub-seq1 on GPU0, while $B_{0,1}$ denotes its backward pass. Black arrows indicate the pipeline communication of layer activations via point-to-point send/receive operations, while red arrows represent the local write/read operations for SA on the same device. Memory states for each model shard are accumulated on their respective GPUs without requiring any inter-device communication, facilitating the training of large models with long sequences.

automatically inserted at partition boundaries to allow data transfer between neighboring partitions. During the forward pass, PP first divides every mini-batch into B equal micro-batches, which are pipelined through the P accelerators. During the backward pass, gradients for each micro-batch are computed based on the same model parameters used for the forward pass. At the end of each mini-batch, gradients from all B micro-batches are accumulated and applied to update the model parameters across all accelerators. This process of PP is illustrated in Figure 2.

When integrating SA with PP, the concept of B micro-batches in traditional pipeline parallelism is replaced by S sub-sequences, representing the partitioned segments of the entire input sequence. In this setup, the data flow of SA occurs exclusively within the forward functions on the same accelerator, which is also within the same model shard. For instance, the data flow follows the pattern on device with a index of 0:

$$F\ 0,0 \rightarrow F\ 0,1 \rightarrow \cdots \rightarrow F\ 0,S-1.$$

Since these operations all take place on the same device, there is no need for inter-device communication, making the integration of SA with PP highly efficient with no additional communication overhead. This inherent efficiency allows the combined approach to handle increasingly longer context lengths as the number of sub-sequences S increases. Furthermore, similar to traditional PP, the "bubble time"—the idle time during pipeline execution—decreases as the number of sub-sequences grows, further optimizing the overall training process. This characteristic makes the SAPP particularly well-suited for efficiently training large models on extremely long sequences, leveraging the full capacity of the limited computational resources.

Experiments

We performed thorough evaluations of the SA and SAPP methods, focusing on their convergence behavior, computational speed, and memory efficiency. To test SA's capability of handling infinite context lengths, we first evaluated it on

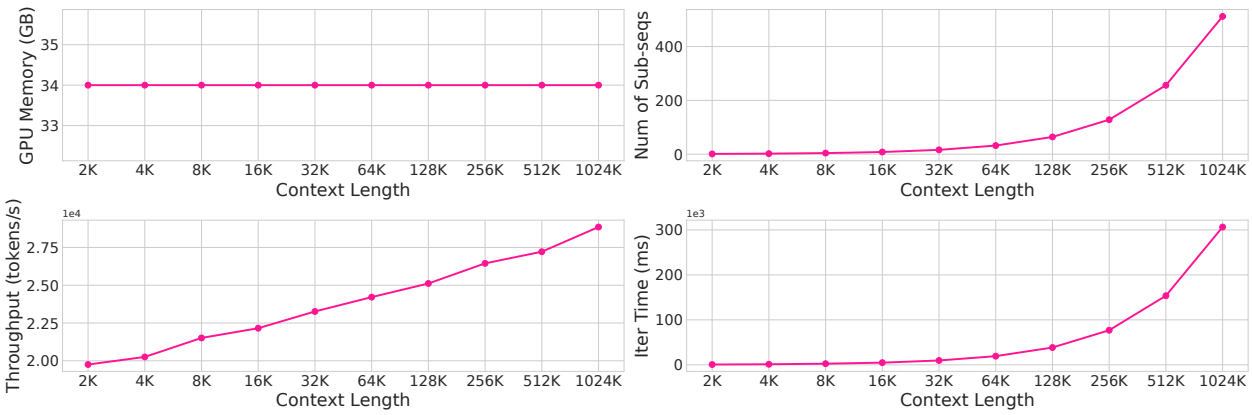

Figure 3: **Performance of SA on Single GPU.** The experiment utilizes a model with 1 billion parameters and a batch size of 4, executed on a single GPU.

Method	SA-PP-DP	Num of GPUs (DP × PP)	Sub-seqs per Iter (DP × SA)	Iterations	Loss
Baseline	1-1-1	1	1	512K	3.710
SA	8-1-1	1	8	64K	3.705
PP	1-8-1	8	1	512K	3.709
DP	1-1-8	8	8	64K	3.714
PP & DP	1-2-4	8	4	128K	3.711
SA & DP	2-1-4	4	8	64K	3.708
SAPP	2-4-1	4	2	256K	3.712
SAPP & DP	4-4-2	8	8	64K	3.709

Table 2: **Convergence Results of SA and SAPP.** All experiments were conducted on a single node with 8x A100 80GB GPUs, using a basic linear transformer model with 0.4 billion parameters. The sequence length was set to 16K (16,384 tokens), with a batch size of 4. Each experiment utilized a specific number of GPUs, determined by DP size × PP size. Additionally, the total number of sub-seqs processed in each iteration was calculated as DP size × SA size. The baseline experiment, which did not involve DP, PP, or SA, required 512K iterations. To ensure that all experiments processed an equivalent amount of data tokens, we adjusted the number of iterations accordingly.

a single GPU, ensuring its effectiveness in scenarios with extremely long sequences. We then expanded our evaluation to multiple GPUs, integrating SA with DP to explore its scalability. For SAPP, we assessed its performance in both single-node and multi-node configurations, further demonstrating its flexibility and efficiency when combined with various parallel training techniques such as DP, TP, and CP. These experiments were conducted on a state-of-the-art GPU cluster comprising 64 A100 GPUs, each with 80GB of memory. The implementation was built on the Megatron-LM framework (Shoeybi et al. 2019; Korthikanti et al. 2022), which provided a robust foundation for testing the scalability and applicability of our methods on large-scale models and extensive computational resources.

Experimental Setup

In all experiments presented in this paper, we employ the Adam optimizer, configured with beta values of 0.9 and 0.999, to ensure efficient training while mitigating potential issues such as gradient instability (Tang et al. 2023). To further enhance regularization and prevent overfitting, a weight decay rate of 0.01 is applied. We set the learning rate at 0.0005, accompanied by a warmup phase spanning 2000 updates, which gradually increases the learning rate to stabilize the early stages of training (Zhou et al. 2020; Sun et al. 2024b). The total number of updates varies depending on the specific training configurations used in each experiment. For pretraining, we utilize the Pile (Gao et al. 2020) dataset, a comprehensive and diverse open-source language modeling dataset consisting of 825 GB of text data (Qin et al. 2023a).

Convergence

Both SA and SAPP are theoretically accurate training methods. To validate their convergence, we conducted experiments using a 0.4B parameter basic linear transformer model with a fixed context length of 16K. The results are summarized in Table 2. In this experiment, we utilized a total of 8 GPUs, applying various configurations of SA, PP, and DP to explore different method combinations. The baseline model, which does not employ DP, PP, or SA, required 512K iterations to

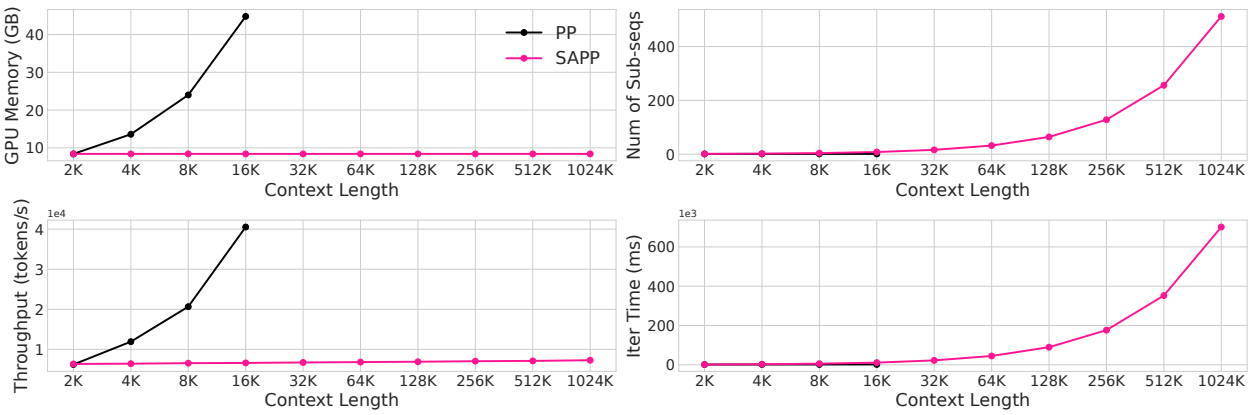

Figure 4: **Performance Comparison of PP and SAPP.** This experiment uses a model with 1 billion parameters and a batch size of 4 across four GPUs, which also represents the size of PP configuration. It is observed that PP encounters OOM when the context length reaches 32K.

Context Length	SA-PP-DP-TP-CP	GPU Memory (GB)	Throughput (tokens/s)	Iteration Time (s)
2K	1-2-4-2-4	18.2	11256	2.4
4K	2-2-4-2-4	18.2	11984	4.6
8K	4-2-4-2-4	18.2	12565	9.3

Table 3: **Compatibility with Hybrid Parallelism.** This experiment involves a 32-layer model with 7 billion parameters and utilizes a distributed setup across 64 GPUs. The length of each sub-sequence is fixed at 2K.

achieve a loss value of 3.710. For the other configurations, the number of GPUs used was determined by the product of the DP size and PP size, while the number of sub-sequences processed in each iteration was calculated as the product of the DP size and SA size. To ensure a fair comparison, we adjusted the number of iterations across different experiments so that all models processed the same amount of data tokens, thus targeting a similar loss value. As shown in the table, regardless of whether SA, SAPP, or their combinations with DP were used, all methods achieved comparable loss values, demonstrating the convergence accuracy of these approaches.

Memory and Speed

We conducted a series of experiments to evaluate the memory usage and speed performance of both SA and SAPP. The SA method was tested on a single GPU, while SAPP was tested on a node with 4 GPUs, both using a linear transformer model containing 1 billion parameters. The experiments employed a batch size of 4, and the context length was progressively increased from 2K to 1024K to assess the changes in GPU memory usage, the number of sub-seqs, throughput, and the time per iteration.

When testing SA on a single GPU, as depicted in Figure 3, the context length of each sub-seq was fixed at 2K. Since SA accumulates the KV states of all sub-seqs, the memory usage per GPU remained constant as the context length increased. However, the number of sub-sequences and the time per iteration increased rapidly with longer context lengths. Throughput saw only a modest increase, which was expected, given that SA sequentially computes all sub-sequences and accumulates their KV states on a single device.

In contrast, Figure 4 illustrates the performance of SAPP on 4 GPUs within a single node and compares it to the standard PP approach. The results show that with PP, both the memory usage per GPU and throughput increased sharply as the context length doubled, eventually leading to an out-of-memory (OOM) error when the context length reached 32K. On the other hand, SAPP maintained a constant memory usage per GPU and exhibited a more gradual increase in throughput as the context length grew, demonstrating its efficiency in handling long sequences without overwhelming the available memory resources.

Hybrid Parallelism: SA-PP-DP-TP-CP

To showcase the capabilities of SAPP when integrated with existing distributed training methods, we evaluate SAPP in conjunction with DP, TP, and CP. As detailed in Table 3, we maintained a fixed sub-sequence length of 2K and varied the SA sizes to $\{1, 2, 4\}$, corresponding to total context lengths of $\{2\text{K}, 4\text{K}, 8\text{K}\}$. The results reveal that, similar to our previous findings with SA and SAPP, while the iteration time increases significantly with longer context lengths, the memory usage remains consistent. Throughput shows a moderate increase. This stability in memory usage and gradual improvement in throughput are attributed to the way SA accumulates intermediate computation states across multiple sub-sequences.

Performance on Benchmarks

We have performed additional benchmark experiments, with the results summarized in Table 4. To ensure consistency, we utilized the pretrained models from the Baseline, SA, and SAPP experiments, as outlined in Table 2. The results indicate that both SA and SAPP achieve generalization performance that is very close to that of the Baseline across standard benchmarks.

Method	LMB.	PIQA	Hella.	Wino.	ARC-e	ARC-c
Baseline	30.9	64.8	36.1	50.5	42.9	23.6
SA	30.7	65.3	36.5	49.6	42.2	23.7
SAPP	30.6	65.6	35.8	49.9	42.7	23.9

Table 4: **Performance Comparison Across Various Benchmarks.** Both SA and SAPP achieve very close generalization performance comparing with that of the Baseline.

Discussion

The limitations of linear attention in comparison to softmax attention can be summarized in two main points: 1) Modeling performance: Linear attention sometimes fails to match the performance of softmax attention, particularly in tasks that require high recall. 2) Support technologies: The supporting technologies for linear attention models, such as CUDA kernels, parallel training, quantization, inference acceleration, and post-training enhancements, are not as mature as those available for softmax attention models.

When comparing linear attention with methods like RWKV, both are sequence modeling techniques that operate with linear complexity, utilizing similar RNN-like recurrence structures. These methods benefit from the advantages of linear-time training and constant memory usage during inference. However, their underlying designs are distinct: linear attention is a variant of softmax attention, whereas RWKV is rooted in linear RNN principles. The primary advantage of linear attention lies in its simplicity, as it only modifies standard attention by removing the softmax operation and reordering matrix multiplications. On the other hand, RWKV, based on linear RNNs, introduces a more complex architecture with advanced mechanisms for token mixing, time mixing, and channel mixing.

The proposed SA shares similarities with the gradient accumulation (GA) approach, both in terms of technical idea and practical application. GA has proven useful for simulating larger batch sizes on devices with limited memory, making it possible to conduct large (theoretically infinite) batch training on a single GPU. Similarly, SA enables training on long (theoretically infinite) sequences with restricted memory resources. For users with limited computational resources, such as a few GPUs, personal computers, or even mobile devices, SA allows for model fine-tuning on very-long-context inputs (which is infeasible without SA). Meanwhile, for those with access to extensive resources, like large GPU clusters or data centers, SAPP provides a solution for training large models on lengthy input sequences.

Related Work

Distributed Training

Distributed training techniques enhance model training efficiency across multiple devices (Team 2023). Pipeline parallelism (Huang et al. 2019; Kim et al. 2020; Narayanan et al. 2019) divides the model into stages, allowing simultaneous computation and communication across devices. Sequence or context parallelism (Li et al. 2021; Korthikanti et al. 2022; Jacobs et al. 2023; Liu, Zaharia, and Abbeel 2023; Sun et al. 2024a) processes long sequences in smaller segments concurrently to manage memory and speed up training. Gradient accumulation (You et al. 2019) are often utilized to accumulates gradients over several mini-batches before updating model parameters, simulating large batch training with limited resources.

Linear Sequence Modeling

Linear Transformer models avoid using Softmax attention by employing various approximation techniques (Katharopoulos et al. 2020; Choromanski et al. 2020; Peng et al. 2021; Qin et al. 2022b,a). These methods leverage the "kernel trick" to accelerate the computation of attention matrix. The chunk-wise approach to linear attention was initially introduced in FLASH (Hua et al. 2022), who utilized a hybrid model that combined linear and nonlinear attention mechanisms. This chunk-wise linear attention also has been independently developed in several other studies (Kacham, Mirrokni, and Zhong 2023; Sun et al. 2023; Yang et al. 2023; Qin et al. 2024b,c). GLA (Yang et al. 2023) and Lightning Attention (Qin et al. 2024b,c) focus on optimizing I/O operations for chunk-wise linear attention, while LASP (Sun et al. 2024a) extends these methods to multi-node distributed training environments. Additionally, Mamba2 (Dao and Gu 2024) provides a comprehensive theoretical framework connecting Structured Semi-Separable Matrices with linear attention through matrix de-compositions. And DeltaNet (Yang et al. 2024) presents hardware-efficient DeltaNet training.

Conclusion

In this paper, we have introduced SA and its extension SAPP, to address the challenge of managing extremely long sequences with limited computational resources. SA leverages the recurrence feature inherent in linear sequence modeling methods to handle infinite context lengths efficiently, even on a single GPU. By dividing long sequences into fixed-length sub-sequences and accumulating intermediate states sequentially, SA achieves constant memory consumption, making it feasible to work with extremely contexts without excessive memory usage. Furthermore, we have extended SA by integrating it with PP to develop SAPP, which allows for the training of large models with infinite context lengths on large GPU clusters without additional communication costs in the sequence dimension. Our extensive experiments demonstrate the effectiveness of both SA and SAPP across various context lengths and GPU configurations. The results highlight that SA and SAPP not only enable the training of infinite context lengths under constrained resources but also integrate seamlessly with existing distributed training methods.

Acknowledgments

This work was supported by the National Key R&D Program of China (No. 2022ZD0160201) and Shanghai Artificial Intelligence Laboratory.

References

Choromanski, K.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; Sarlós, T.; Hawkins, P.; Davis, J.; Mohiuddin, A.; Kaiser, L.; Belanger, D.; Colwell, L. J.; and Weller, A. 2020. Rethinking Attention with Performers. *ArXiv*, abs/2009.14794.

Dao, T. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Dao, T.; Fu, D.; Ermon, S.; Rudra, A.; and Ré, C. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359.

Dao, T.; and Gu, A. 2024. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

Gao, L.; Biderman, S.; Black, S.; Golding, L.; Hoppe, T.; Foster, C.; Phang, J.; He, H.; Thite, A.; Nabeshima, N.; Presser, S.; and Leahy, C. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. arXiv:2101.00027.

Gu, A.; and Dao, T. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Hua, W.; Dai, Z.; Liu, H.; and Le, Q. V. 2022. Transformer Quality in Linear Time. *arXiv preprint arXiv:2202.10447*.

Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.; Lee, H.; Ngiam, J.; Le, Q. V.; Wu, Y.; et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32.

Jacobs, S. A.; Tanaka, M.; Zhang, C.; Zhang, M.; Song, S. L.; Rajbhandari, S.; and He, Y. 2023. DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. arXiv:2309.14509.

Kacham, P.; Mirrokni, V.; and Zhong, P. 2023. Polysketchformer: Fast transformers via sketches for polynomial kernels. *arXiv preprint arXiv:2310.01655*.

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

Kim, C.; Lee, H.; Jeong, M.; Baek, W.; Yoon, B.; Kim, I.; Lim, S.; and Kim, S. 2020. torchgpipe: On-the-fly Pipeline Parallelism for Training Giant Models.

Korthikanti, V.; Casper, J.; Lym, S.; McAfee, L.; Andersch, M.; Shoeybi, M.; and Catanzaro, B. 2022. Reducing Activation Recomputation in Large Transformer Models. arXiv:2205.05198.

Li, S.; Xue, F.; Baranwal, C.; Li, Y.; and You, Y. 2021. Sequence parallelism: Long sequence training from system perspective. *arXiv preprint arXiv:2105.13120*.

Liu, H.; Zaharia, M.; and Abbeel, P. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889.

Narayanan, D.; Harlap, A.; Phanishayee, A.; Seshadri, V.; Devanur, N. R.; Ganger, G. R.; Gibbons, P. B.; and Zaharia, M. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM symposium on operating systems principles*, 1–15.

Peng, B.; Alcaide, E.; Anthony, Q.; Albalak, A.; Arcadinho, S.; Cao, H.; Cheng, X.; Chung, M.; Grella, M.; GV, K. K.; He, X.; Hou, H.; Kazienko, P.; Kocon, J.; Kong, J.; Koptyra, B.; Lau, H.; Mantri, K. S. I.; Mom, F.; Saito, A.; Tang, X.; Wang, B.; Wind, J. S.; Wozniak, S.; Zhang, R.; Zhang, Z.; Zhao, Q.; Zhou, P.; Zhu, J.; and Zhu, R.-J. 2023. RWKV: Reinventing RNNs for the Transformer Era. arXiv:2305.13048.

Peng, B.; Goldstein, D.; Anthony, Q.; Albalak, A.; Alcaide, E.; Biderman, S.; Cheah, E.; Ferdinan, T.; Hou, H.; Kazienko, P.; et al. 2024. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*.

Peng, H.; Pappas, N.; Yogatama, D.; Schwartz, R.; Smith, N. A.; and Kong, L. 2021. Random Feature Attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Qin, Z.; Han, X.; Sun, W.; Li, D.; Kong, L.; Barnes, N.; and Zhong, Y. 2022a. The Devil in Linear Transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 7025–7041. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics.

Qin, Z.; Li, D.; Sun, W.; Sun, W.; Shen, X.; Han, X.; Wei, Y.; Lv, B.; Luo, X.; Qiao, Y.; et al. 2023a. Transnormerllm: A faster and better large language model with improved transnormer.

Qin, Z.; Li, D.; Sun, W.; Sun, W.; Shen, X.; Han, X.; Wei, Y.; Lv, B.; Yuan, F.; Luo, X.; et al. 2023b. Scaling transnormer to 175 billion parameters. *arXiv preprint arXiv:2307.14995*.

Qin, Z.; Shen, X.; Sun, W.; Li, D.; Birchfield, S.; Hartley, R.; and Zhong, Y. 2024a. Unlocking the Secrets of Linear Complexity Sequence Model from A Unified Perspective. *arXiv preprint arXiv:2405.17383*.

Qin, Z.; Sun, W.; Deng, H.; Li, D.; Wei, Y.; Lv, B.; Yan, J.; Kong, L.; and Zhong, Y. 2022b. cosFormer: Rethinking Softmax In Attention. In *International Conference on Learning Representations*.

Qin, Z.; Sun, W.; Li, D.; Shen, X.; Sun, W.; and Zhong, Y. 2024b. Lightning attention-2: A free lunch for handling unlimited sequence lengths in large language models. *arXiv preprint arXiv:2401.04658*.

Qin, Z.; Sun, W.; Li, D.; Shen, X.; Sun, W.; and Zhong, Y. 2024c. Various Lengths, Constant Speed: Efficient Language Modeling with Lightning Attention. *arXiv preprint arXiv:2405.17381*.

Qin, Z.; Yang, S.; Sun, W.; Shen, X.; Li, D.; Sun, W.; and Zhong, Y. 2024d. HGRN2: Gated linear RNNs with state expansion. *arXiv preprint arXiv:2404.07904*.

Qu, X.; Dong, D.; Hu, X.; Zhu, T.; Sun, W.; and Cheng, Y. 2024. LLaMA-MoE v2: Exploring Sparsity of LLaMA from Perspective of Mixture-of-Experts with Post-Training. *arXiv preprint arXiv:2411.15708*.

Shah, J.; Bikshandi, G.; Zhang, Y.; Thakkar, V.; Ramani, P.; and Dao, T. 2024. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*.

Shen, X.; Li, D.; Leng, R.; Qin, Z.; Sun, W.; and Zhong, Y. 2024. Scaling laws for linear complexity language models. *arXiv preprint arXiv:2406.16690*.

Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; and Catanzaro, B. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Sun, W.; Qin, Z.; Li, D.; Shen, X.; Qiao, Y.; and Zhong, Y. 2024a. Linear Attention Sequence Parallelism. *arXiv preprint arXiv:2404.02882*.

Sun, W.; Qin, Z.; Sun, W.; Li, S.; Li, D.; Shen, X.; Qiao, Y.; and Zhong, Y. 2024b. CO2: Efficient distributed training with full communication-computation overlap. *arXiv preprint arXiv:2401.16265*.

Sun, Y.; Dong, L.; Huang, S.; Ma, S.; Xia, Y.; Xue, J.; Wang, J.; and Wei, F. 2023. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*.

Tang, X.; Sun, W.; Hu, S.; Sun, Y.; and Guo, Y. 2023. MS-Net: A Multi-Path Sparse Model for Motion Prediction in Multi-Scenes. *IEEE Robotics and Automation Letters*.

Team, I. 2023. Internlm: A multilingual language model with progressively enhanced capabilities.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yang, S.; Wang, B.; Shen, Y.; Panda, R.; and Kim, Y. 2023. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*.

Yang, S.; Wang, B.; Zhang, Y.; Shen, Y.; and Kim, Y. 2024. Parallelizing Linear Transformers with the Delta Rule over Sequence Length. *arXiv preprint arXiv:2406.06484*.

You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; and Hsieh, C.-J. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*.

Zhou, B.; Liu, J.; Sun, W.; Chen, R.; Tomlin, C. J.; and Yuan, Y. 2020. pbSGD: Powered Stochastic Gradient Descent Methods for Accelerated Non-Convex Optimization. In *IJCAI*, 3258–3266.