Efficient Decoding for Long Prompts via Low-Rank Attention

Anonymous Authors[1]

Abstract

Tremendous efforts have been seen to allow large language models (LLMs) addressing extremely long documents and background materials. Algorithmically, although low-rank approximation methods have proven to reduce the complexity of bidirectional attention from quadratic to linear growth with respect to sequence length, it is less studied in the context of decoder Transformers, as formally the causal mask within the decoder disables the essential right-multiplication acceleration of the low-rank method. In this paper, we instead explore the potential of applying the overlooked low-rank technique to prompt decoding. Specifically, we first verify that existing low-rank methods, even landmark-based, are applicable to prompt decoding and can enjoy low approximation error, both theoretically and empirically. This observation, along with the emergence of native linear transformers, motivates us to unearth and implement efficient algorithms for decoding from literature in various fields, and to develop FleetAttention, a linear-complexity algorithm that achieves SOTA time and memory efficiency by utilizing the unique properties of the causal mask and IO-aware implementation. Experiments on the MMLU benchmark and TransnormerLLM2 demonstrate that FleetAttention is more memory efficient and faster than other methods on long sequences. The detailed model implementation and the code are attached as supplementary materials, scheduled to be open-sourced upon publication.

1. Introduction

Transformer (Vaswani et al., 2017) architecture, since its inception, has revolutionized the field of natural language processing, particularly in the development of large language models. Its core component, the trainable attention mechanism, captures complex dependencies in input sequences, enabling sophisticated text understanding and generation. Yet, this advancement introduces a computational challenge: the standard transformer model's quadratic scaling with the number of tokens N in the input sequence. This scaling makes the model computationally intensive for long sequences and limits its use in resource-limited settings.

Acknowledging this limitation, recent years have seen an increase in research efforts focused on improving the efficiency of transformer models. These endeavors primarily focus on reducing computational complexity and optimizing memory usage. A promising resolution that emerges in the form of linear attention in this realm is the reimagining of the attention mechanism through the lens of kernelization. By approximating the softmax function integral to the attention mechanism with a linear kernel, researchers have successfully introduced a class of models known as linear attention transformers. These models retain the foundational strengths of the original architecture while significantly reducing computational overhead, thereby enabling the processing of substantially longer sequences without compromising performance.

It is important to highlight that linear attention mechanisms have significantly reduced computation, with the complexity of attention scaling down from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. However, the domain of low-rank methods has not garnered widespread attention, and there are several factors. Firstly, many low-rank approaches are based on "landmarks" and considered leaking future information in the generation phase. This characteristic renders them unfavorable to decoder-only models, such as the GPT series (Brown et al., 2020) and the Llama series (Touvron et al., 2023a;b). Secondly, the computational complexity of low-rank methods escalates to $\mathcal{O}(N^2)$ when trivially implementing causal masking, thereby diminishing their complexity advantage in decoders. Additionally, the earlier limited prompt length did not typically result in issues such as insufficient GPU memory or long prompt processing time. Finally, a number of low-rank approaches focus solely on low-rank decomposition without incorporating efficient subsequent computation strategies, which ultimately limits their performance.

In this work, we explore the potential of applying the over-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Table 1: Overview of the computation algorithms for low-rank causal attention

Low-rank Computation Method	General Low-rank Attention	Time Complexity
vanilla	✓	$\mathcal{O}(N^2)$
causal-dot-product (Vyas et al., 2020)	✓	$\mathcal{O}(N)$
row-based (Choromanski et al., 2020)	✗	$\mathcal{O}(N)$
block-based (Lingle, 2023)	✗	$\mathcal{O}(N)$
recursion (Han et al., 2023)	✗	$\mathcal{O}(N \log N)$
Lightning Attention-2 (Qin et al., 2024)	✗	$\mathcal{O}(N^2)$
FleetAttention	✓	$\mathcal{O}(N)$

looked low-rank technique to prompt decoding. ① Firstly, we theoretically and empirically prove low-rank can be effectively applied in decoding, with well-controlled approximation error. Empirical studies on large language models demonstrate that prediction performance can remain with low-rank approximation to the original attention. ② Moreover, we thoroughly investigate existing linear computation algorithms from various fields for *low-rank causal attention*, expanding and adapting these algorithms for standard low-rank attention, as summarized in Table 1 and Appendix B. To the best of our knowledge, our work is the most comprehensive in this area. ③ Furthermore, we introduce FleetAttention, a linear-complexity algorithm for low-rank causal attention computation that heavily utilizes the special structure of Hadamard product and causal masks. Through careful IO-aware implementation, FleetAttention achieves increased throughput and memory efficiency. ④ Extensive experiments indicate that FleetAttention not only maintains accuracy across different sequence lengths but also achieves higher speedup and memory efficiency compared than existing low-rank computation algorithms.

2. Related Work

Low-rank approximation. It has been extensively studied, particularly for its efficiency in dimension reduction and its application in large-scale data analysis. The seminal work by Eckart and Young laid the foundation for low-rank approximations, providing a mathematical formulation that minimizes the Frobenius norm of the difference between a matrix and its low-rank approximation. This approach involves Singular Value Decomposition (SVD), which traditionally has $\mathcal{O}(N^2 d)$ complexity for an $N \times d$ matrix.

To overcome this, research has pivoted towards more efficient algorithms. For example, randomized low-rank factorization (Liberty et al., 2007; Halko et al., 2010) have gained prominence due to their potential to handle massive datasets with reduced computational overhead while maintaining high accuracy. These methods typically project the data onto a lower-dimensional space using randomized sketching techniques, significantly reducing the computation cost.

In parallel, optimization-based approaches for low-rank approximation have been explored, which aim to solve various formulations of the rank minimization problem. Moreover, the advent of deep learning has brought forth novel methods that integrate low-rank approximations into neural network architectures. These methods leverage low-rank structures to reduce the number of parameters in the model, as seen in the work on compressing fully connected layers (Sainath et al., 2013), thereby enhancing the computational efficiency without compromising the model's performance.

Recently, attention has been directed towards developing low-rank methods that are well-suited for modern transformer models. These efforts aim to mitigate the quadratic complexity associated with the self-attention mechanism, as highlighted by Chen et al. (2022), which introduced low-rank techniques to approximate attention. The dynamic nature of low-rank approximations in adapting to data makes them a powerful tool for improving the scalability of transformer models, particularly for processing long sequences.

Efficient transformers. It can be classified along three axes: model architecture, method type, and compatibility with pre-trained models. ① The first axis encompasses models that adhere to the classic transformer architecture, as well as those employing novel structures, such as RetNet (Sun et al., 2023), TransNormerLLM2 (Qin et al., 2023b;a), RWKV (Peng et al., 2023), and GLA (Yang et al., 2023). RetNet partially eliminates non-linear dependencies in attention calculations to improve its reasoning performance. TransNormerLLM2 completely abandons the Softmax-based attention mechanism and instead uses the newly proposed linear attention. ② The second axis is the difference between sparse methods and low-rank methods. ③ The third axis is the adaptability to pre-trained models.

Sparse methods like top-k Attention (Gupta et al., 2021), focus on enhancing space efficiency without necessarily improving time complexity. Conversely, approaches like Sparse Transformer (Child et al., 2019), Longformer (Beltagy et al., 2020), and ETC (Ainslie et al., 2020) aim to enhance both aspects. Reformer (Kitaev et al., 2020) employs Locality Sensitive Hashing (Andoni & Razenshteyn, 2015) to attend to different token sets dynamically.

Low-rank methods approximate the attention score matrix with a low-rank matrix to reduce the quadratic time-space complexity. For example, Linformer (Wang et al., 2020) and Performer (Choromanski et al., 2020) decompose the attention weight matrix into a product of two rectangular matrices, respectively consisting of learned linear features or random features of the keys and queries. Transformer-VQ (Lingle, 2023) performs "clustering" on the sequence of Key vectors in attention using vector quantization, approximating the original vectors with the centroid of their respective classes. The models that use the above low-rank methods usually better need to be trained with low-rank

approximations built-in, in order to learn to be robust to the associated approximation errors. Some methods can be used directly on pre-trained models without retraining such as Nyströmformer (Xiong et al., 2021), LARA (Zheng et al., 2022), HyperAttention (Han et al., 2023) and Lightning Attention-2 (Qin et al., 2024). These methods accelerate the execution of the standard attention operation without altering the underlying architecture. Nyströmformer employs the Nyström method for efficient computation, while LARA merges randomized attention with random feature attention to effectively reconstruct the attention weight matrix. HyperAttention can approximate an attention mechanism with spectrum guarantee in approximately linear time to improve inference speed. Lightning Attention-2 employs a 'divide and conquer' strategy, segmenting the computation of the attention matrix into diagonal and off-diagonal forms.

In addition, there are some special studies on efficient transformers such as FlashAttention (Dao et al., 2022), and FlashAttention-2 (Dao, 2023). FlashAttention achieves significant memory saving (from quadratic to linear) and runtime speedup by utilizing tiling techniques to reduce memory reads/writes between GPU high bandwidth memory (HBM) and on-chip SRAM.

3. Preliminaries

3.1. Standard Attention Mechanisms

The dot-product attention (Vaswani et al., 2017) processes three input matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$, wherein N represents the number of tokens in the input sequence and d denotes the head dimension. The bidirectional attention (Devlin et al., 2018) is defined as:

$$\boldsymbol{Attn}_{\leftrightarrow} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V} \tag{1}$$

Here the matrix $\boldsymbol{A} = \exp(\boldsymbol{Q}\boldsymbol{K}^T) \in \mathbb{R}^{N\times N}$ is defined as the element-wise exponential of $\boldsymbol{Q}\boldsymbol{K}^T$. For simplicity, the normalization factor $\sqrt{d}$ has been omitted. The diagonal matrix $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{A}\boldsymbol{1}_N)$ contains the row sums of $\boldsymbol{A}$ along its main diagonal, where $\boldsymbol{1}_N$ is a length N all-ones vector. In this context, matrix $\boldsymbol{A}$ is called the *attention matrix*.

Unidirectional attention (or causal attention) is used for auto-regressive generative modeling, such as the decoder component of Seq2Seq Transformers. It is defined as:

$$\begin{aligned}\boldsymbol{Attn}_{\rightarrow} &= \boldsymbol{D'}^{-1}(\boldsymbol{A}\odot\boldsymbol{M})\boldsymbol{V},\\ \boldsymbol{D'} &= \mathrm{diag}((\boldsymbol{A}\odot\boldsymbol{M})\boldsymbol{1}_N)\end{aligned} \tag{2}$$

where $\boldsymbol{M} \in \{0,1\}^{N\times N}$ is a lower triangular matrix, and

$$\boldsymbol{M}_{i,j} = \delta(i,j) := \begin{cases}1 & \text{if } i \geq j,\\ 0 & \text{otherwise.}\end{cases}$$

The time complexity for computing both Equation (1) and Equation (2) is $\mathcal{O}(N^2d)$. Their space complexity is $\mathcal{O}(N^2 + Nd)$, which is due to the explicit storing of $\boldsymbol{A}$.

3.2. Low-Rank Approximations in Attention Mechanisms

Low-rank approximation methods when applied to bidirectional attention mechanism involve the utilization of two matrices of lower order, denoted as $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N\times r}$, to approximate the attention matrix $\boldsymbol{A}$, where r represents the rank of the matrices. The general low-rank approximation is defined as:

$$\tilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{C}^T \tag{3}$$

By applying the associative law, The time and memory complexity of bidirectional attention can be reduced to $\mathcal{O}(Ndr)$ and $\mathcal{O}(Nr + Nd + rd)$, respectively, as shown:

$$\begin{aligned}\tilde{\boldsymbol{Attn}}_{\leftrightarrow} &= \tilde{\boldsymbol{D}}^{-1}(\boldsymbol{B}\boldsymbol{C}^T)\boldsymbol{V} = \tilde{\boldsymbol{D}}^{-1}\boldsymbol{B}(\boldsymbol{C}^T\boldsymbol{V})\\ \tilde{\boldsymbol{D}} &= \mathrm{diag}(\boldsymbol{B}(\boldsymbol{C}^T\boldsymbol{1}_N))\end{aligned} \tag{4}$$

Numerous low-rank approximations, such as LARA, Performer, and Transformer-VQ, have been used to implement bidirectional attention. These methods control the approximation error is bounded about the norm of $\boldsymbol{A}$ itself: $\|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|_X \leq \varepsilon\|\boldsymbol{A}\|_X$. Here, the subscript X represents either the operator norm induced by the Euclidean norm denoted as $\|\cdot\|_2$, or the Frobenius norm defined as $\|\boldsymbol{A}\|_F^2 = \sum_{i,j} a_{ij}^2$.

Applying low-rank factorization to the decoder, for the general low-rank format, we need to compute the approximated causal attention output $\boldsymbol{O} \in \mathbb{R}^{N\times d}$ as:

$$\begin{aligned}\boldsymbol{O} &= \tilde{\boldsymbol{D}'}^{-1}\boldsymbol{P}, \tilde{\boldsymbol{D}'} = \mathrm{diag}((\boldsymbol{M}\odot\tilde{\boldsymbol{A}})\boldsymbol{1}_N),\\ \boldsymbol{P} &= (\boldsymbol{M}\odot\tilde{\boldsymbol{A}})\boldsymbol{V} = (\boldsymbol{M}\odot\boldsymbol{B}\boldsymbol{C}^T)\boldsymbol{V}\end{aligned} \tag{5}$$

However, due to the application of a mask to $\tilde{\boldsymbol{A}}$, we cannot reduce compute complexity by matrix right-multiplication as in Equation (4). Direct computation of Equation (5) still leads to a time complexity $\mathcal{O}(N^2r + N^2d)$ and memory complexity $\mathcal{O}(N^2 + Nd)$.

In Section 4.2, we introduce a linear computation method, FleetAttention, to address this challenge. Furthermore, in Section 5.1, we review computation algorithms from existing low-rank methods and expand them to suit the linear computation of general low-rank causal attention.

4. Method

In this section, we mainly introduce the low-rank approximation method on the decoder, which includes low-rank approximation causal attention and the use of landmark for

prompt decoding and recursive computation method. Then we introduce our own algorithm FleetAttention and its specific implementation.

4.1. Low-rank Factorization Method in Decoder

Low-rank approximation to causal attention. Given a quality low-rank approximation $\tilde{\boldsymbol{A}}$ to the original attention score matrix $\boldsymbol{A}$, we can show the approximation error for causal attention is well-controlled as well, through the following theorem.

Theorem 4.1. *Consider we have an matrix $\tilde{\boldsymbol{A}}$ approximating the original attention matrix $\boldsymbol{A}$, and we have a small approximation error*

$$\|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|_X,$$

where the subscript X represents either the operator norm induced by euclidean norm $\|\cdot\|_2$ or the Frobenius norm $\|\cdot\|_F$. Then, for the masked target $\boldsymbol{M} \odot \boldsymbol{A}$, we keep enjoying the following approximation guarantee:

$$\|\boldsymbol{M} \odot \boldsymbol{A} - \boldsymbol{M} \odot \tilde{\boldsymbol{A}}\|_X = \tilde{\mathcal{O}}\left(\|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|_X\right),$$

in which $\tilde{\mathcal{O}}(\cdot)$ indicates complexity modulo poly-log term.

The proof is deferred to Appendix C.1 due to space limit. As a sanity check of the theoretical result, we will shortly evaluate the output quality of a large language model after directly replacing the original unidirectional attention with the aforementioned low-rank attention approximations, as shown in Table 2.

Landmark-based method for prompt decoding. Most random-feature-based methods such as LARA and Nyströmformer group different tokens into different clusters, known as landmarks. Implementing causal masking for these landmark-based methods is extremely challenging. This is not only because the masking needs to be applied at the landmark level to prevent the leakage of information from future tokens, but also because an additional set of masks is required to conceal varying numbers of tokens within the same landmark for different queries. The latter functionality is not natively supported and is difficult to implement. We have observed that in the prompt decoding stage, it is feasible to employ a landmark-based method for approximating attention. This is predicated on the fact that all the prompt tokens are indeed known information, and therefore it is safe to compute $\tilde{\boldsymbol{A}}$ itself for prompts with the so-called "information leakage". In the subsequent token generation process, the prompt tokens are known (history) information, and we note there is no leakage concept at all for the history information.

Recursive computation method. An interesting observation of the lower triangular matrix $\boldsymbol{M}$ is that the product $(\boldsymbol{M} \odot \tilde{\boldsymbol{A}})\boldsymbol{V}$ can be rewritten in a partitioned manner:

$$\begin{pmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{M}_2 & \boldsymbol{M}_3, \end{pmatrix} \begin{pmatrix} \boldsymbol{V}_1 \\ \boldsymbol{V}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}_1\boldsymbol{V}_1 \\ \boldsymbol{M}_2\boldsymbol{V}_1 + \boldsymbol{M}_3\boldsymbol{V}_2 \end{pmatrix}, \quad (6)$$

where $\boldsymbol{M}_1, \boldsymbol{M}_3$ are two smaller lower triangular matrices and $\boldsymbol{M}_2$ in the bottom left corner of $\boldsymbol{M} \odot \tilde{\boldsymbol{A}}$ is a low-rank matrix (for simplicity we omit the dependence on $\tilde{\boldsymbol{A}}$ in the notations of $\boldsymbol{M}_i$'s). Realizing the observation, Han et al. (2023) mentioned a scheme to efficiently compute their proposed low-rank attention in decoders: we can normally compute the low-rank matrix product $\boldsymbol{M}_2\boldsymbol{V}_1$ and then recursively address the two smaller matrices $\boldsymbol{M}_1\boldsymbol{V}_1$ and $\boldsymbol{M}_3\boldsymbol{V}_2$, with FlashAttention-2 applied to the base case.

The recursive strategy stands distinct from other low-rank attention computation methods. Although the strategy is attractive at first sight, we note it suffers from several deficiencies, ① higher time complexity and ② inefficient hardware implementation. ① As shortly delineated in Lemma 4.2, even equipped with an efficient $\mathcal{O}(N)$ low-rank attention computation method, the recursive approach is characterized by a computational complexity of $\mathcal{O}(N \log N)$, with its proof detailed in C.3. We note the recursive strategy exhibits a higher complexity than the original $\mathcal{O}(N)$ low-rank attention computation method. ② In addition, GPUs are designed to optimize large-scale, parallel data processing, while recursion involves deeply nested calls and complex control flows, which is inconsistent with the advantages of modern hardware.

The time complexity induced by the recursive method is provided in the following lemma, and the proof is deferred to Appendix C.3.

Lemma 4.2. *Consider the matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N \times r}$, $\boldsymbol{V} \in \mathbb{R}^{N \times d}$, $\tilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{C}^T$, the diagonal matrix $\tilde{\boldsymbol{D}} \in \mathbb{R}^{N \times N}$ with $\tilde{\boldsymbol{D}}_{i,i} = \left\langle \tilde{\boldsymbol{A}}_{i,:}, \boldsymbol{1}_N \right\rangle$, and the causal mask matrix $\boldsymbol{M}$. The time complexity of using the recursive computation method to compute $\tilde{\boldsymbol{D}}^{-1}(\tilde{\boldsymbol{A}} \odot \boldsymbol{M})\boldsymbol{V}$ is $\mathcal{O}(N \log N)$.*

We conclude the recursive computation method is sub-optimal for decoding extremely long prompts on GPUs.

Empirical results. We conducted a comparative analysis of three low-rank methods (FAVOR+, LARA, HyperAttention), including LARA based on the landmark, and one attention computation method (FlashAttention-2). This comparison was performed using the MMLU (Hendrycks et al., 2021b;a) benchmark on the open source large language model vicuna-7b-v1.5-16k (Zheng et al., 2023), which was fine-tuned based on Llama2 (Touvron et al., 2023b). We replace certain layers in the self-attention module using these low-rank methods and the FlashAttention-2 method while ensuring that all other components of each model remain the same as those in the standard transformer. Since these low-rank methods approximate the original attention, their accuracy

Table 2: Accuracy evaluation of using low-rank methods (HyperAttention,FAVOR+,LARA) and FlashAttention-2 to replace part of the self attention layers in the vicuna-7b-v1.5-16k model for testing on the MMLU benchmark. Since FlashAttention-2 only supports model parameters FP16 and BF16, and FAVOR+ only supports FP32, only FlashAttention-2 uses FP16 as the model parameter type here, and the rest use FP32.

Method	Task				
	STEM	Humanities	Social Science	Other	avg
VanillaAttention	39.1	45.5	56.7	55.2	49.1
FlashAttention-2	39.0	44.4	57.1	55.4	49.0
HyperAttention	36.6	40.6	52.5	53.8	45.9
FAVOR+	35.9	43.1	56.1	52.4	46.9
LARA	35.6	41.8	53.4	51.5	45.6

will be lower than vanilla. The data presented in Table 2 indicates that the accuracy deviations of these low-rank approaches, irrespective of their use of landmarks, fall within a 5% margin when compared to conventional attention mechanisms. Consequently, both the low-rank method and the landmark-based approach are viable options for implementation in the decoder. The experimental details are provided in Appendix A.1.

4.2. FleetAttention

In this section, we introduce FleetAttention, which incorporates a linear compute algorithm for causal attention along with IO-awareness optimization.

Linear compute algorithm. Given the input matrices $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$, our objective is to compute the attention output $\boldsymbol{O}$ with linear compute complexity. Here we describe how to compute $\boldsymbol{P}$, and $\tilde{\boldsymbol{D}}'$ can be computed similarly by replacing $\boldsymbol{V}$ by $\mathbf{1}_N$. To ease the derivation, we denote $\boldsymbol{b}_i, \boldsymbol{c}_i$ respectively as the i-th column of $\boldsymbol{B}, \boldsymbol{C}$, and thus $\boldsymbol{B}\boldsymbol{C}^T$ can be represented as $\sum_i \boldsymbol{b}_i \boldsymbol{c}_i^T$. We then have:

$$\begin{aligned}\boldsymbol{P} &= \sum_{i=1}^{r} \left((\boldsymbol{b}_i \boldsymbol{c}_i^T) \odot \boldsymbol{M} \right) \boldsymbol{V} \\ &= \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{b}_i) \boldsymbol{M} \mathrm{diag}(\boldsymbol{c}_i) \boldsymbol{V},\end{aligned}$$

in which the second equation holds due to the known property of Hadamard product. Utilizing the special structure of the mask matrix $\boldsymbol{M}$ that it is equivalent to a cumsum (cumulative sum) operator, we finally obtain

$$\boldsymbol{P} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{b}_i) \mathrm{cumsum}(\mathrm{diag}(\boldsymbol{c}_i) \boldsymbol{V}), \tag{7}$$

where the cumsum operator is applied to the column vectors of matrix $\mathrm{diag}(\boldsymbol{c}_i)\boldsymbol{V}$ in parallel. We note in obtaining each summand the time complexity is $\mathcal{O}(Nd)$, and therefore the total computation complexity will remain $\mathcal{O}(Ndr)$. The algorithm above can be generalized to a popular attention variant with exponentially decaying positional embedding (Press et al., 2022; Qin et al., 2022; Peng et al., 2023), and the corresponding derivation is provided in Appendix C.2.

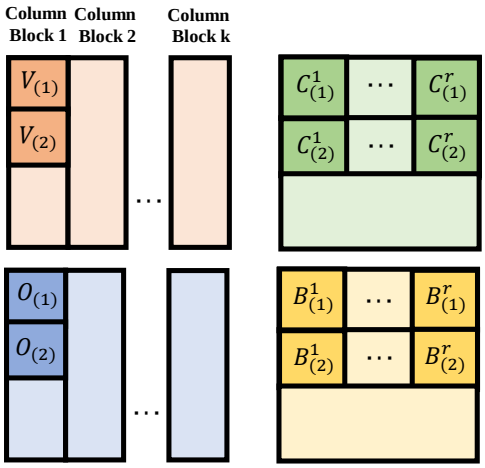

Figure 1: Matrices splitting in FleetAttention

IO-aware implementation. The ideal implementation of the aforementioned algorithm on GPU is non-trivial. With vanilla Einstein sum operations, we will need $\mathcal{O}(Ndr)$ space to store r summands for the final sum up. We need to balance the GPU memory usage and the computation speed, as in matrix multiplication on GPU. To address this, we introduce an IO-awareness implementation optimization. This approach maintains the computational complexity at $\mathcal{O}(Ndr)$ while reducing the space complexity to $\mathcal{O}(Nd)$.

FleetAttention leverages tiling technique (Dao et al., 2022) to enhance memory access efficiency between the GPU High Bandwidth Memory (HBM) and its on-chip Static Random-Access Memory (SRAM). Meanwhile, this method mitigates memory overflow when dealing with long sequence lengths. We demonstrate the time and memory efficiency of this method in Section 5.3.

The core idea of our optimization is that we split the in-

Algorithm 1 FleetAttention

Require: Matrics $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N\times r}$, $\boldsymbol{V} \in \mathbb{R}^{N\times \boldsymbol{d}}$ in HBM, block sizes B_c, B_r.

Divide $\boldsymbol{V}$ into $T_c = \left\lceil \frac{d}{B_c} \right\rceil$ blocks $\boldsymbol{V}^1, \ldots, \boldsymbol{V}^{T_c}$ of size $N \times B_c$ each, then devide $\boldsymbol{V}^k$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\boldsymbol{V}_1^k, \ldots, \boldsymbol{V}_{T_r}^k$ of size $B_r \times B_c$ each.
Divide $\boldsymbol{B}$ into $T_r \times r$ column blocks $\boldsymbol{B}_1^1, \ldots, \boldsymbol{B}_1^r, \ldots, \boldsymbol{B}_{T_c}^1, \ldots, \boldsymbol{B}_{T_r}^r$ of size $B_r \times 1$ each.
Divide $\boldsymbol{C}$ into $T_r \times r$ column blocks $\boldsymbol{C}_1^1, \ldots, \boldsymbol{C}_1^r, \ldots, \boldsymbol{C}_{T_c}^1, \ldots, \boldsymbol{C}_{T_r}^r$ of size $B_r \times 1$ each.
Divide the output $\boldsymbol{O}$ into T_c blocks $\boldsymbol{O}^1, \ldots, \boldsymbol{O}^{T_c}$ of size $N \times B_c$ each, then devide $\boldsymbol{O}^k$ into T_r blocks $\boldsymbol{O}_1^k, \ldots, \boldsymbol{O}_{T_r}^k$ of size $B_r \times B_c$ each.
for $1 \le k \le T_c$, parallel computation of $\boldsymbol{O}^k$ **do**
  $\boldsymbol{l}^0, \ldots, \boldsymbol{l}^r = (\boldsymbol{0})_{B_c} \in \mathbb{R}^{B_c}$, $m^0, \ldots, m^r = 0 \in \mathbb{R}$.
  for $1 \le i \le T_r$ **do**
    Load $\boldsymbol{V}_i^k$ from HBM to on-chip SRAM.
    On chip, initialize $\boldsymbol{O}_i = (\boldsymbol{0})_{B_r \times B_c} \in \mathbb{R}^{B_r \times B_c}$, $\boldsymbol{d}_i = (\boldsymbol{0})_{B_r} \in \mathbb{R}^{B_r}$.
    for $1 \le j \le r$ **do**
      Load $\boldsymbol{B}_i^j, \boldsymbol{C}_i^j$ from HBM to on-chip SRAM.
      On chip, compute $\boldsymbol{O}_i \leftarrow \boldsymbol{O}_i + \mathrm{diag}(\boldsymbol{B}_i^j)(\mathrm{cumsum}(\mathrm{diag}(\boldsymbol{C}_i^j)\boldsymbol{V}_i^k) + \boldsymbol{l}^j)$.
      On chip, compute $\boldsymbol{d}_i \leftarrow \boldsymbol{d}_i + \mathrm{diag}(\boldsymbol{B}_i^j)(\mathrm{cumsum}(\boldsymbol{C}_i^j) + m^j)$.
      On chip, compute $\boldsymbol{l}^j \leftarrow \mathrm{sum}(\mathrm{diag}(\boldsymbol{C}_i^j)\boldsymbol{V}_i^k) + \boldsymbol{l}^j$, $m^j \leftarrow \mathrm{sum}(\boldsymbol{C}_i^j) + m^j$.
    end for
    On chip, compute $\boldsymbol{O}_i \leftarrow \mathrm{diag}(\boldsymbol{d}_i)^{-1}\boldsymbol{O}_i$.
    Write $\boldsymbol{O}_i$ to HBM as the block of $\boldsymbol{O}_i^k$.
  end for
end for
Return the output $\boldsymbol{O}$.

put matrices $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ into blocks. These blocks are then loaded from the slow HBM into the fast SRAM, where the attention outputs are computed w.r.t. these blocks. We simultaneously apply the appropriate normalization factors to the outputs of each block, guaranteeing the accuracy of the results. Specifically, since the computation for each column of $\boldsymbol{V}$ in Equation (7) is independent, we can split $\boldsymbol{V}$ into column blocks for parallel computation. Moreover, an entire column block may still exceed the SRAM memory, we further subdivide each column block into row blocks, as illustrated in Figure 1. This ensures that the sub-matrices can fit entirely into SRAM, allowing us to complete all operations for a given sub-matrix consecutively. The computation process is described as follows.

To simplify, we focus on a single column block of the value matrix $\boldsymbol{V}$, split into $\begin{bmatrix}\boldsymbol{V}_{(1)}\\ \boldsymbol{V}_{(2)}\end{bmatrix}$ for some matrices $\boldsymbol{V}_{(1)}, \boldsymbol{V}_{(2)} \in \mathbb{R}^{B_r \times B_c}$, where B_r and B_c are the row and column block sizes respectively. Our objective is to calculate the attention output $\boldsymbol{O}$ for this column block, which takes a similar form $\begin{bmatrix}\boldsymbol{O}_{(1)}\\ \boldsymbol{O}_{(2)}\end{bmatrix}$ for some metrics $\boldsymbol{O}_{(1)}, \boldsymbol{O}_{(2)} \in \mathbb{R}^{B_r \times B_c}$. We have low-rank approximation matrices $\boldsymbol{B}$ and $\boldsymbol{C}$, of the form $\boldsymbol{C} = \begin{bmatrix}\boldsymbol{C}_{(1)}^1, \ldots, \boldsymbol{C}_{(1)}^r\\ \boldsymbol{C}_{(2)}^1, \ldots, \boldsymbol{C}_{(2)}^r\end{bmatrix}$, $\boldsymbol{B} = \begin{bmatrix}\boldsymbol{B}_{(1)}^1, \ldots, \boldsymbol{B}_{(1)}^r\\ \boldsymbol{B}_{(2)}^1, \ldots, \boldsymbol{B}_{(2)}^r\end{bmatrix}$, where $\boldsymbol{B}_{(1)}^i$, $\boldsymbol{B}_{(2)}^i, \boldsymbol{C}_{(1)}^i, \boldsymbol{C}_{(2)}^i \in \mathbb{R}^{B_r \times 1}$ are columns of matrices. For the first block $\boldsymbol{V}_{(1)}$, according to Equation (7), we can obtain

$$\boldsymbol{P}_{(1)} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}_{(1)}^i)\mathrm{cumsum}(\mathrm{diag}(\boldsymbol{C}_{(1)}^i)\boldsymbol{V}_{(1)}),$$

where $\mathrm{cumsum}(\cdot)$ returns a matrix of the same shape as the input, representing the cumulative sum of elements along the matrix's vertical direction. According to Equation (5), we can obtain the corresponding $\boldsymbol{d}_{(1)}$ and $\boldsymbol{O}_{(1)}$:

$$\boldsymbol{d}_{(1)} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}_{(1)}^i)\mathrm{diag}(\mathrm{cumsum}(\boldsymbol{C}_{(1)}^i))$$
$$\boldsymbol{O}_{(1)} = \mathrm{diag}(\boldsymbol{d}_{(1)})^{-1}\boldsymbol{P}_{(1)}$$

The prefix sums of this block are cached to facilitate the cumsum operation in the subsequent block:

$$\boldsymbol{l}_{(1)}^i = \mathrm{sum}(\mathrm{diag}(\boldsymbol{C}_{(1)}^i)\boldsymbol{V}_{(1)})$$
$$m_{(1)}^i = \mathrm{sum}(\boldsymbol{C}_{(1)}^i)$$

For subsequent blocks, we add prefix sums at the corresponding positions to ensure the accuracy of the results and

Table 3: Accuracy of five low-rank computation methods (causal-dot-product, row-based, block-based, recursion, FleetAttention (this work)) and vanilla using TransNormerLLM2-3B-300B (performance significantly inferior to vicuna-7b-v1.5-16k) on the MMLU benchmark. All of these methods are low-rank computation methods, which are mathematically equivalent.

Low-rank Computation Method	Task				
	STEM	humanities	social sciences	Other	avg
vanilla	23.0	OOM	25.8	26.0	NA
causal-dot-product	23.0	25.2	25.8	26.0	25.0
row-based	23.0	25.2	25.8	26.0	25.0
block-based	23.0	25.2	25.8	26.0	25.0
recursion	23.0	25.2	25.8	26.0	25.0
FleetAttention	23.0	25.2	25.8	26.0	25.0

Table 4: Throughput of low-rank computation methods

method	vanilla	causal-dot-product	row-based	block-based	recursion	FleetAttention
throughput	338	869	120	680	853	966

continue updating the prefix sums.

$$\boldsymbol{P}_{(2)} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}^i_{(2)})(\mathrm{cumsum}(\mathrm{diag}(\boldsymbol{C}^i_{(2)})\boldsymbol{V}_{(2)}) + \boldsymbol{l}^i_{(1)})$$

$$\boldsymbol{d}_{(2)} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}^i_{(2)})(\mathrm{cumsum}(\boldsymbol{C}^i_{(2)}) + m^i_{(1)})$$

$$\boldsymbol{l}^i_{(2)} = \mathrm{sum}(\mathrm{diag}(\boldsymbol{C}^i_{(2)})\boldsymbol{V}_{(2)}) + \boldsymbol{l}^i_{(1)}$$

$$m^i_{(2)} = \mathrm{sum}(\boldsymbol{C}^i_{(2)}) + m^i_{(1)}$$

The full procedure of FleetAttention is given in Algorithm 1 and implemented in Triton.

5. Empirical Results

In this section, we introduce several low-rank computation methods. These methods, along with FleetAttention, have been applied to a large language model and evaluated using the MMLU benchmark. Furthermore, we also analyzed their performance with long prompts on a single self-attention layer. All experiments were conducted on a single NVIDIA RTX A6000 with 48GB of memory.

5.1. Low-Rank Causal Attention Baseline Methods

We unearth the algorithms for low-rank causal attention from the literature in various fields (we generalize some methods from their original settings to low-rank causal attention), and prepare the PyTorch implementation if originally unavailable. Table 1 presents a comprehensive overview of existing computation algorithms. We give a brief introduction to each algorithm in the following list; more detailed description, including how to generalize them from the original setting, are provided in Appendix B.

vanilla: Directly computing Equation (5) using PyTorch.

causal-dot-product: The CausalLinearAttention class in the Fast Transformers library provides an official CUDA implementation of causal-dot-product. It utilizes the dot product between feature maps and leverages the triangular structure of the causal masking to achieve linear time complexity.

row-based: Choromanski et al. (2020) introduced a linear causal attention algorithm that can be applied to general low-rank formats. Their algorithm computes the output matrix row-by-row, which we refer to as the row-based algorithm. Since there is currently no official implementation available, we provide the full details of the algorithm in Appendix B.1 and have implemented it in PyTorch.

block-based: Lingle (2023) proposed an algorithm to accelerate linear causal attention computation through a block-by-block approach. It is uniquely tailored to the vector-quantized keys and applied in a gated activation unit (GAU). We have extended this to a more general low-rank factorization format as Equation (3). The extended algorithm is detailed in Appendix B.2 and implemented using PyTorch.

recursion: Han et al. (2023) proposed a recursive method for computing causal attention by dividing the attention matrix into masked and unmasked submatrices. In their original work, they utilized sparse attention to accelerate computation of the unmasked submatrix, while continuing to recursively divide the masked submatrices. We have expanded the recursive technique to low-rank causal attention by combining it with causal-dot-product. When the length of the recursive matrix is less than 32, causal-dot-product is used for calculation. The specific algorithm is detailed in Appendix B.3, alongside our PyTorch implementation.

5.2. Effectiveness on Linear Transformers

Datasets and metrics. We evaluate TransNormerLLM2-3B-300B equipped with FleetAttention and other different low-rank computation methods on the MMLU (Hendrycks

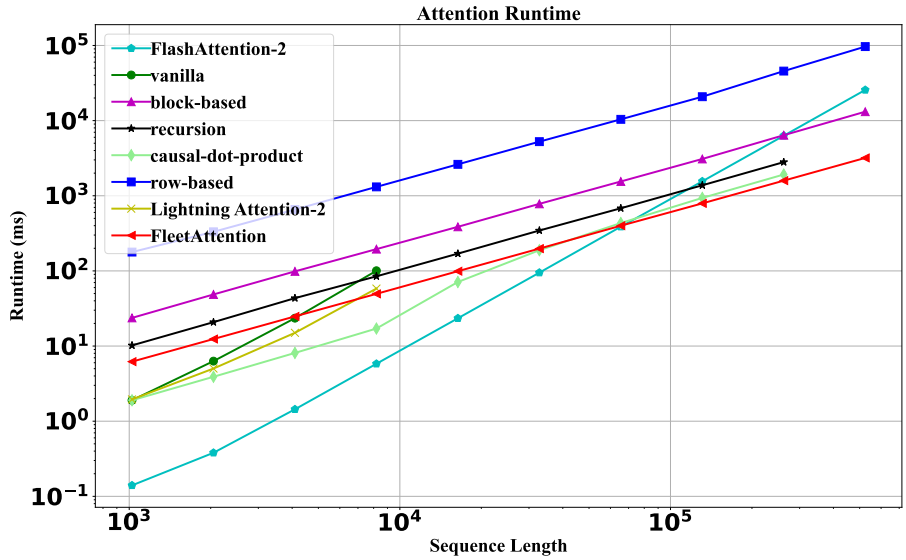

Figure 2: Speedup of the exact GPU computation time during forward operations using low-rank computation methods which include row-based, block-based, recursion (based on causal-dot-product), causal-dot-product, FleetAttention (this work), Lightning Attention-2. We use vanilla attention as a baseline and FlashAttention-2 as a reference for temporal performance analysis. When the sequence length exceeds 8k, the vanilla computation method and Lightning Attention-2 are out-of-memory. When the sequence length exceeds 256k, causal-dot-product and recursion will also be out-of-memory.

et al., 2021b;a) benchmark. MMLU is a mainstream LLM evaluation dataset. It is a massive multitask test consisting of multiple-choice questions from various branches of knowledge. The questions are divided into four categories: STEM, humanities, social sciences, and other. In this experiment, we use its official evaluation approach, with all evaluation results being conducted with a 5-shot setting, and the maximum prompt length is set to 4096. We report the accuracy for each category. For detailed information about the experiment, please refer to Appendix A.2. Besides, we evaluate the throughput of these methods on randomly generated tokens using GPT with lengths of 2k-3k prompts.

Base large language model. We evaluate FleetAttention to improve the capability of prompt processing in LLMs. We choose TransNormerLLM2-3B-300B (performance significantly inferior to vicuna-7b-v1.5-16k) which is an open-source large language model and stands out as the first LLM within the linear transformer architecture. TransNormerLLM2-3B-300B uses a special casual mask. For more detailed information, please refer to Appendix C.2.

Result analysis. In Table 3, we observe all five low-rank computation methods exhibit the same level of accuracy as the vanilla method because they are mathematically equivalent. Additionally, on the humanities task, the vanilla method experienced an out-of-memory (OOM) issue; however, none of the low-rank computation methods encountered OOM, indicating these methods consume less memory compared to the standard implementation. This allows Large Language Models (LLMs) to process longer prompts with the same GPU memory. In Table 4, we observe that FleetAttention has a higher throughput than all other low-rank computation methods.

5.3. Decoding Extremely Long Prompts

We further explore the efficiency of FleetAttention (this work) with varying sequence lengths from 1024 to 524288 and longer sequences will cause the GPU memory to overflow. Experiments were repeated 8 times and average times were taken. We measured the GPU time for the forward operations of FleetAttention and 5 other low-rank computation methods, including row-based, block-based, recursion, causal dot-product, and Lightning Attention-2. We used the vanilla computation method and FlashAttention-2 as the baseline for performance measures. FlashAttention-2 here is only used as a calculation formula to measure the speed of other methods because it does not use low-rank approximation. We refer to the parameters of vicuna-7b-v1.5-16k (Zheng et al., 2023). So all inputs $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ have the same sequence length, dimensions, and the number of attention heads where their dimensions are set to d=128, the number of attention heads is set to h=32 and the batch size of the prompt is set to batch=1.

From Figure 2 we can see that when the sequence length reaches 8k length, the vanilla computation method and Lightning Attention-2 have reached the upper limit of GPU

memory. When the sequence length exceeds 256k, causal-dot-product and recursion will also be out-of-memory (OOM). We think that the recursion algorithm encounters OOM issues and suboptimal speed partly due to its recursive implementation not being GPU friendly. While the row-based algorithm and block-based algorithm avoid OOM issues, their lack of IO-awareness optimizations results in very low efficiency as they remain IO-bounded.

As the sequence length increases, the superiority of the linear algorithms gradually emerges. We can observe that when the sequence length exceeds 64k, FleetAttention is faster than the other methods. In addition, we can also find that FleetAttention uses little GPU memory so it can handle a sequence length of 512k and the Runtime of FleetAttention has a linear relationship with sequence length.

6. Conclusions and Future Directions

In this work, we explore the application the low-rank approximation to the decoder and demonstrated its effectiveness through both theoretical analysis of the approximation error bound and empirical experiments on large language models. We have also extended and summarized existing computation algorithms that enable the low-rank method to achieve linear time and memory complexity on causal attention. Moreover, we propose an efficient algorithm, FleetAttention, and empirically proved its capability to significantly speed up processing for long sequence lengths while maintaining memory efficiency.

From our perspective, our method is proposed for efficient computation on GPU. In this sense, FleetAttention might underperform on CPU or other distinct hardware. In addition, FleetAttention currently only supports ordinary causal masks and does not support special causal masks such as mask with decay. Our future direction is to solve these problems and further optimize the implementation on the CUDA level for finer manipulation of the GPU cores and cache.

Broader Impact

As a computation method, we consider our work itself to have a low ethical risk since the outcomes of the algorithm mainly depend on the downstream applications. The method might be used in some applications to accelerate the regression tasks, such as text generation and image generation tasks. Our method can be applied to more architectures employing linear attention, including low-rank attention approximation methods, Retnet and so on. We hope our paper can be a meaningful step in developing the de facto "FlashAttention" for linear attention. Also, our method doesn't assume some specific structure of the input and thus doesn't leverage biases in the data. We can safely conclude that it's not likely for our work to have a negative societal impact.

References

Ainslie, J., Ontanon, S., Alberti, C., Cvicek, V., Fisher, Z., Pham, P., Ravula, A., Sanghai, S., Wang, Q., and Yang, L. Etc: Encoding long and structured inputs in transformers, 2020.

Andoni, A. and Razenshteyn, I. Tight lower bounds for data-dependent locality-sensitive hashing, 2015.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer, 2020.

Bhatia, R. Pinching, trimming, truncating, and averaging of matrices. *The American Mathematical Monthly*, 107(7): 602–608, 2000.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

Chen, Y., Zeng, Q., Hakkani-Tur, D., Jin, D., Ji, H., and Yang, Y. Sketching as a tool for understanding and accelerating self-attention for long sequences. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5187–5199, Seattle, United States, July 2022. Association for Computational Linguistics.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers, 2019.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

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

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

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Gupta, A., Dar, G., Goodman, S., Ciprut, D., and Berant, J. Memory-efficient transformers via top-k attention, 2021.

Halko, N., Martinsson, P.-G., and Tropp, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010.

Han, I., Jayaram, R., Karbasi, A., Mirrokni, V., Woodruff, D. P., and Zandieh, A. Hyperattention: Long-context attention in near-linear time, 2023.

Hendrycks, D., Burns, C., Basart, S., Critch, A., Li, J., Song, D., and Steinhardt, J. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021a.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021b.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Liberty, E., Woolfe, F., Martinsson, P.-G., Rokhlin, V., and Tygert, M. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

Lingle, L. D. Transformer-vq: Linear-time transformers via vector quantization. *arXiv preprint arXiv:2309.16354*, 2023.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Derczynski, L., Du, X., Grella, M., Gv, K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Lin, J., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wind, J., Woźniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R.-J. RWKV: Reinventing RNNs for

the transformer era. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, Singapore, December 2023. Association for Computational Linguistics.

Press, O., Smith, N., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Qin, Z., Sun, W., Deng, H., Li, D., Wei, Y., Lv, B., Yan, J., Kong, L., and Zhong, Y. cosformer: Rethinking softmax in attention. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=Bl8CQrx2Up4.

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

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

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

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6655–6659, 2013.

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

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023b.

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

Vyas, A., Katharopoulos, A., and Fleuret, F. Fast transformers with clustered attention. 2020.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity, 2020.

Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. Nyströmformer: A nyström-based algorithm for approximating self-attention, 2021.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training, 2023.

Zheng, L., Wang, C., and Kong, L. Linear complexity randomized self-attention mechanism. In *International Conference on Machine Learning*, pp. 27011–27041. PMLR, 2022.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

A. Experiment Details

A.1. vicuna-7b-v1.5-16k Experiment on MMLU benchmark

We used vicuna-7b-v1.5-16k as the base model to test on the MMLU benchmark. We use FlashAttention-2, HyperAttention, FAVOR+, and LARA to replace the self-attention layers of 23-30 layers in the vicuna model, a total of 8 layers. vicuna has a total of 32 layers of transformer blocks. Because the longest prompt length of MMLU is 3480. So we set the experimental max_sequence_length=4096. For different methods, we also set their hyperparameters, see Table 5.

Table 5: Hyperparameter settings for different low-rank methods

Method	hyper-parameter	Value
HyperAttention	lsh_num_projs	7
	block_size	64
	sample_size	64
	min_seq_len	64
FAVOR+	nb_features	256
LARA	nb_features	256

A.2. TransNormerLLM-3B-300B Experiment on MMLU benchmark

We used TransNormerLLM2-3B-300B as the base model to test on the MMLU benchmark. We use causal-dot-product, row-based, block-based, recursion, and FleetAttention to replace the attention calculation in TransNormerLLM2-3B-300B. The mask matrix used in TransNormerLLM2-3B-300B is not a causal mask with only 0 and 1 elements. It is an attention mask that enables lower triangular causal masking and positional encoding and Lightning Attention-2 also uses calculations for this kind of mask. Currently, our algorithm only implements the causal mask, so in this experiment, we uniformly used the causal mask instead of the attention mask of TransNormerLLM2-3B-300B and we did not compare us with Lightning Attention-2. Appendix C.2 gives proof that our method can also be extended to the attention mask of TransNormerLLM2-3B-300B without any change in complexity. To increase the prompt length as much as possible, we also set max_sequence_length=4096 for this experiment.

A.3. License Information

MMLU is licensed under the MIT License.

vicuna-7b-v1.5-16k is under the Llama 2 Community License Agreement.

TransNormerLLM model is licensed under the Apache 2.0.

B. Algorithms for Linear Causal Attention

B.1. Row-based Algorithm

Choromanski et al. (2020, Equation (11)) introduce a prefix-sum algorithm for causal attention. In each iteration, it can compute the value of one row of the output matrix, which we refer to as the row-based algorithm. Vectors $\boldsymbol{c}_j, \boldsymbol{b}_j \in \mathbb{R}^{1\times r}$, and $\boldsymbol{v}_j \in \mathbb{R}^{1\times d}$ represent the j-th row-vectors in matrices $\boldsymbol{C}, \boldsymbol{B}, \boldsymbol{V}$, respectively.

A single row of the output is computed as:

$$\boldsymbol{O}_i = \sum_{j=1}^{i} \boldsymbol{b}_i \boldsymbol{c}_j^T \boldsymbol{v}_j = \boldsymbol{b}_i \sum_{j=1}^{i} \boldsymbol{c}_j^T \boldsymbol{v}_j \tag{8}$$

Let $\boldsymbol{U}_i = \sum_{j=1}^{i} \boldsymbol{c}_j^T \boldsymbol{v}_j \in \mathbb{R}^{r\times d}$, then Equation (8) can be realized through recursion:

$$\boldsymbol{O}_i = \boldsymbol{b}_i \boldsymbol{U}_i,\ \boldsymbol{U}_i = \boldsymbol{U}_{i-1} + \boldsymbol{c}_i^T \boldsymbol{v}_i \tag{9}$$

The matrix $\boldsymbol{D}$ Is calculated by substituting $\boldsymbol{V}$ with 1_N. The attention output is then calculated as Equation (5). The full row-based algorithm is described in Algorithm 2.

Algorithm 2 Row-based Algorithm

Require: Matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N \times r}$, $\boldsymbol{V} \in \mathbb{R}^{N \times d}$.
Initialize $\boldsymbol{U} = (\boldsymbol{0})_{r \times d} \in \mathbb{R}^{r \times d}$, $\boldsymbol{Z} = (\boldsymbol{0})_{1 \times r} \in \mathbb{R}^{1 \times r}$, $\boldsymbol{O} = (\boldsymbol{0})_{N \times d} \in \mathbb{R}^{N \times d}$.
for $1 \leq i \leq N$ **do**
  Extract row vectors $\boldsymbol{b}_i \in \mathbb{R}^{1 \times r}$, $\boldsymbol{c}_i \in \mathbb{R}^{1 \times r}$, $\boldsymbol{v}_i \in \mathbb{R}^{1 \times d}$ from $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ respectively.
  Update $\boldsymbol{U} \leftarrow \boldsymbol{U} + \boldsymbol{c}_i^T \boldsymbol{v}_i$.
  Compute $\boldsymbol{o}_i = \boldsymbol{b}_i \boldsymbol{U}$.
  Update $\boldsymbol{Z} \leftarrow \boldsymbol{Z} + \boldsymbol{c}_i$.
  Compute $\boldsymbol{d}_i = \boldsymbol{b}_i \boldsymbol{Z}^T$.
  Store $\boldsymbol{O}_i = \boldsymbol{o}_i / \boldsymbol{d}_i$.
end for
Return the output $\boldsymbol{O}$.

The time complexity for processing each row of $[\tilde{\boldsymbol{A}}\boldsymbol{V}]$ is $\mathcal{O}(rd)$, leading to a total time complexity of $\mathcal{O}(Nrd)$. The space complexity is $\mathcal{O}(Nd + rd)$.

B.2. Block-based Algorithm

The row-by-row computation algorithm described in Equation (9) is considered to be inefficient. Transformer-VQ (Lingle, 2023, Equation (28)) proposed a block-by-block computation algorithm to accelerate the process. The computation method proposed in Transformer-VQ is designed for its specific low-rank form. We extend this method to a general low-rank form represented as $\tilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{C}$.

Let the block size be denoted as B_c. The block slice $[iB_c : (i+1)B_c]$ is abbreviated as $[i]$. Let $\hat{\boldsymbol{K}} \in \mathbb{R}^{N \times k}$ be the approximation of $\boldsymbol{K}$ obtained through the low-rank method of Transformer-VQ. Let $\boldsymbol{M} \in \{0,1\}^{B_c \times B_c}$ be a lower triangular causal mask, where $\boldsymbol{M}_{ij} = 1$ if $j \leq i$ and $\boldsymbol{M}_{ij} = 0$ otherwise.

The derivation from the original computation method to a general computation method is as follows:

$$\boldsymbol{O}_{[i]} = (\exp(\boldsymbol{Q}_{[i]} \hat{\boldsymbol{K}}_{[i]}^T) \odot \boldsymbol{M}) V_{[i]} + \sum_{j=1}^{i} \exp(\boldsymbol{Q}_{[i]} \hat{\boldsymbol{K}}_{[j]}^T) \boldsymbol{V}_j \tag{10}$$

$$= (\boldsymbol{B}_{[i]} \boldsymbol{C}_{[i]}^T \odot \boldsymbol{M}) \boldsymbol{V}_{[i]} + \sum_{j=1}^{i} \boldsymbol{B}_{[i]} \boldsymbol{C}_{[j]}^T V_j \tag{11}$$

$$= (\boldsymbol{B}_{[i]} \boldsymbol{C}_{[i]}^T \odot \boldsymbol{M}) \boldsymbol{V}_{[i]} + \boldsymbol{B}_{[i]} \sum_{j=1}^{i} \boldsymbol{C}_{[j]}^T \boldsymbol{V}_j \tag{12}$$

Let $\boldsymbol{U}_{[i]} = \sum_{j=1}^{i} \boldsymbol{C}_{[j]}^T \boldsymbol{V}_{[j]} \in \mathbb{R}^{r \times d}$. Then, we obtain:

$$\boldsymbol{O}_{[i]} = (\boldsymbol{B}_{[i]} \boldsymbol{C}_{[i]}^T \odot \boldsymbol{M}) \boldsymbol{V}_{[i]} + \boldsymbol{B}_{[i]} \boldsymbol{U}_{[i-1]} \tag{13}$$

$$\boldsymbol{U}_{[i]} = \boldsymbol{U}_{[i-1]} + \boldsymbol{C}_{[j]}^T \boldsymbol{V}_{[j]} \tag{14}$$

Matrix $\boldsymbol{D}$ can be calculated by replacing $\boldsymbol{V}$ with 1_N:

$$\boldsymbol{D}_{[i]} = (\boldsymbol{B}_{[i]} \boldsymbol{C}_{[i]}^T \odot \boldsymbol{M}) \boldsymbol{1}_N + \boldsymbol{B}_{[i]} \boldsymbol{Z}_{[i-1]} \tag{15}$$

$$\boldsymbol{Z}_i = \boldsymbol{Z}_{[i-1]} + \boldsymbol{C}_{[i]}^T \boldsymbol{1}_N \tag{16}$$

The full block-based algorithm is described in Algorithm 3. The time complexity for processing each block $\boldsymbol{O}_{[i]}$ is $\mathcal{O}(B_c^2 r + B_c^2 d + B_c r d)$, there are $\left\lceil \frac{N}{B_c} \right\rceil$ blocks, leading to a total time complexity of $\mathcal{O}(NB_c r + NB_c d + Nrd)$. If we let $1 < B_c \leq d$, the time and space complexity are $\mathcal{O}(Nrd)$ and $\mathcal{O}(Nd + rd)$, respectively.

Algorithm 3 Block-based Algorithm

Require: Matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N \times r}$, $\boldsymbol{V} \in \mathbb{R}^{N \times d}$, block size B_c.

Divide $\boldsymbol{V}$ into $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\boldsymbol{V}_{[1]}, \ldots, \boldsymbol{V}_{[T_c]}$ of size $B_c \times d$ each.
Divide $\boldsymbol{B}, \boldsymbol{C}$ into T_c blocks $\boldsymbol{B}_{[1]}, \ldots, \boldsymbol{B}_{[T_c]}; \boldsymbol{C}_{[1]}, \ldots, \boldsymbol{C}_{[T_c]}$; of size $B_c \times r$ each.
Initialize $\boldsymbol{U} = (\boldsymbol{0})_{r \times d} \in \mathbb{R}^{r \times d}$, $\boldsymbol{Z} = (\boldsymbol{0})_{r \times 1} \in \mathbb{R}^{r \times 1}$, $\boldsymbol{O} = (\boldsymbol{0})_{N \times d} \in \mathbb{R}^{N \times d}$.
Let $\boldsymbol{M} \in \{0,1\}^{B_c \times B_c}$ be a lower triangular causal mask.
for $0 \leq i < T_c$ **do**
  Compute $\boldsymbol{o} = (\boldsymbol{B}_{[i]} \boldsymbol{C}_{[i]}^T \odot \boldsymbol{M}) \boldsymbol{V}_{[i]} + \boldsymbol{B}_{[i]} \boldsymbol{U}$.
  Update $\boldsymbol{U} \leftarrow \boldsymbol{U} + \boldsymbol{C}_{[i]}^T \boldsymbol{V}_{[i]}$.
  Compute $\boldsymbol{d} = (\boldsymbol{B}_{[i]} \boldsymbol{C}_{[i]}^T \odot \boldsymbol{M}) \boldsymbol{1}_{B_c} + \boldsymbol{B}_{[i]} \boldsymbol{Z}$.
  Update $\boldsymbol{Z} \leftarrow \boldsymbol{Z} + \boldsymbol{C}_{[i]}^T \boldsymbol{1}_N$.
  Compute and store $\boldsymbol{O}_{[i]} = \mathrm{diag}(\boldsymbol{d})^{-1} \boldsymbol{o}$.
end for
Return the output $\boldsymbol{O}$.

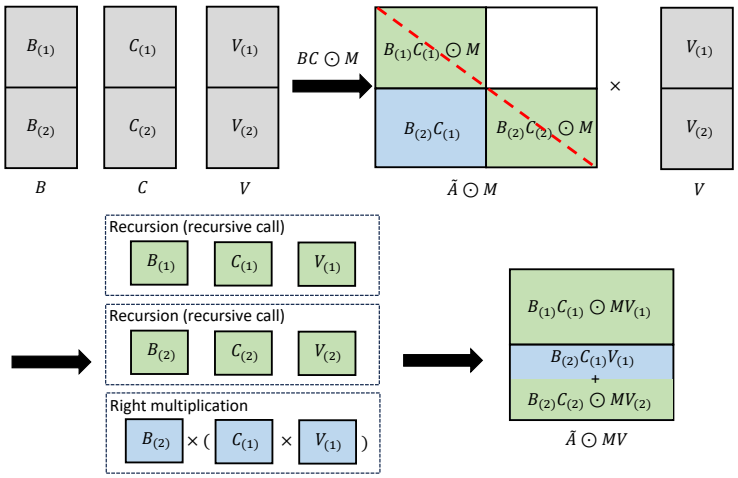

Figure 3: Causal attention matrix can be divided into three equal-sized non-zero sections: $\boldsymbol{B}_{(1)}\boldsymbol{C}_{(1)} \odot \boldsymbol{M}$ and $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(2)} \odot \boldsymbol{M}$ are both causal attention matrices and $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(1)}$ is an unmasked attention matrix.

B.3. Resursion Algorithm

HyperAttention (Han et al., 2023, Algorithm (4)) proposes a recursive method to compute causal attention. This approach involves initially partitioning matrices recursively and then approximating the computations for each partitioned component. We have extended the recursive approach to general low-rank matrix forms $\tilde{\boldsymbol{A}} = \boldsymbol{BC}$. We divide $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ recursively to enable more efficient computation of the causal attention.

As shown in Figure 3, the masked attention matrix $\tilde{\boldsymbol{A}} \odot \boldsymbol{M}$ can be decomposed into three distinct non-zero submatrices, each half the size of the original attention matrix. Notably, the block $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(1)}$, located entirely below the diagonal is unmasked attention. We can efficiently compute this part of the result as $\boldsymbol{B}_{(2)}(\boldsymbol{C}_{(1)}\boldsymbol{V}_{(1)})$, enabling linear computation through right multiplication. The two diagonal blocks $\boldsymbol{B}_{(1)}\boldsymbol{C}_{(1)} \odot \boldsymbol{M}$ and $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(2)} \odot \boldsymbol{M}$ in Fig. 3 represent causal attentions with half the original size. To compute these, we recursively partition them into even smaller blocks and repeat this decomposition procedure. The full recursion algorithm is described in Algorithm 4.

Algorithm 4 Recursion Algorithm

Require: Matrices $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$.
if $\text{len}(\boldsymbol{B}) \leq$ termination block size **then**
  Return $(\boldsymbol{M} \odot \boldsymbol{B}\boldsymbol{C})\boldsymbol{V}$
else
  Split $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ into equal sized sub-matrices: $\boldsymbol{B}_1, \boldsymbol{B}_2; \boldsymbol{C}_1, \boldsymbol{C}_2; \boldsymbol{V}_1, \boldsymbol{V}_2$.
  $\boldsymbol{O}_1 \leftarrow \text{Recursion}(\boldsymbol{B}_1, \boldsymbol{C}_1, \boldsymbol{V}_1)$
  $\boldsymbol{O}_2 \leftarrow \text{Recursion}(\boldsymbol{B}_2, \boldsymbol{C}_2, \boldsymbol{V}_2)$
  $\boldsymbol{O}_3 \leftarrow \boldsymbol{B}_2(\boldsymbol{C}_1\boldsymbol{V}_1)$
  Return $\begin{bmatrix} \boldsymbol{O}_1 & \boldsymbol{0} \\ \boldsymbol{O}_3 & \boldsymbol{O}_2 \end{bmatrix}$
end if

C. Derivations Omitted in the Main Text

C.1. Proof of Theorem 4.1

Proof. We first recall $\boldsymbol{M}$ is a 0-1 matrix with elements in the lower triangle all one. For the ease of analysis, we also denotes $\boldsymbol{A} = [a_{ij}]$ and $\tilde{\boldsymbol{A}} = [\tilde{a}_{ij}]$.

We then start with the case for the Frobenius norm. As per the definition of the Frobenius norm, we can deduce:

$$\|\boldsymbol{M} \odot \boldsymbol{A} - \boldsymbol{M} \odot \tilde{\boldsymbol{A}}\|_F^2 = \sum_{j \leq i} \sum_{i=1}^{n} \|a_{ij} - \tilde{a}_{ij}\|^2$$
$$\leq \sum_{j=1}^{n} \sum_{i=1}^{n} \|a_{ij} - \tilde{a}_{ij}\|^2 = \|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|_F^2,$$

which directly reads $\|\boldsymbol{M} \odot \boldsymbol{A} - \boldsymbol{M} \odot \tilde{\boldsymbol{A}}\|_F \leq \|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|_F$.

The derivation for the operator norm case depends on an existing result (Bhatia, 2000, Equation (15)) for the triangular truncation operator Δ_U, which replaces all entries of a matrix below the main diagonal by zero. Bhatia (2000) proves that for any n-by-n real matrix $\boldsymbol{X}$, we can attain $\|\Delta_U(\boldsymbol{X})\| \leq L_n \|\boldsymbol{X}\|$, where L_n is a constant with scale $\mathcal{O}(\log n)$. We set $\boldsymbol{X} = (\boldsymbol{A} - \tilde{\boldsymbol{A}})^T$ and rewrite $\|\boldsymbol{M} \odot \boldsymbol{A} - \boldsymbol{M} \odot \tilde{\boldsymbol{A}}\|_2$ as $\|\Delta_U(\boldsymbol{X})\|$. We then immediately have

$$\|\Delta_U(\boldsymbol{X})\| \leq L_n \|\boldsymbol{X}\| = L_n \left\|(\boldsymbol{A} - \tilde{\boldsymbol{A}})^T\right\|_2$$
$$= L_n \left\|(\boldsymbol{A} - \tilde{\boldsymbol{A}})\right\|_2 = \tilde{\mathcal{O}}\left(\|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|_2\right),$$

which completes the proof. ♢

C.2. Generalization of FleetAttention to Attention with Exponentially Decaying Positional Embedding

In the specific attention variant with exponentially decaying positional embedding, we have a slightly different mask matrix

$$\boldsymbol{M} = [\lambda^{i-j} \cdot \delta(i, j)], \quad \text{with the discount factor } \lambda \in (0, 1],$$

which is still a lower triangular matrix. With the notation above, the original causal attention matrix can be taken as a special case with $\lambda = 1$.

We first study the i-th row of the product $\boldsymbol{O} = \boldsymbol{M}\boldsymbol{V}$, and have

$$\boldsymbol{O}_{i,:} = \sum_{j=1}^{i} \lambda^{i-j} \boldsymbol{V}_{j,:} = \boldsymbol{V}_{i,:} + \sum_{j=1}^{i-1} \lambda^{i-j} \boldsymbol{V}_{j,:} = \boldsymbol{V}_{i,:} + \lambda \left(\sum_{j=1}^{i-1} \lambda^{i-1-j} \boldsymbol{V}_{j,:} \right)$$
$$= \boldsymbol{V}_{i,:} + \lambda \boldsymbol{O}_{i-1,:}.$$

We note the operator $\boldsymbol{M}$ is now equivalent to the so-called discounted cumsum operator, which enjoys the linear $\mathcal{O}(N)$ complexity as well.

Denoting the discounted cumsum operator as λ-cumsum, we combine the pieces above and have

$$\left(\boldsymbol{M} \odot \tilde{\boldsymbol{A}}\right) \boldsymbol{V} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{b}_i) \boldsymbol{M} \mathrm{diag}(\boldsymbol{c}_i) \boldsymbol{V} = \left((\boldsymbol{B}\boldsymbol{C}^T) \odot \boldsymbol{M}\right) \boldsymbol{V} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{b}_i) \cdot \lambda\text{-}\mathrm{cumsum}(\mathrm{diag}(\boldsymbol{c}_i)\boldsymbol{V}).$$

Therefore, the complexity analysis of our algorithm in Section 4.2 still applies, and we conclude the generalized version of FleetAttention maintains $\mathcal{O}(N)$ time complexity.

C.3. Proof of Lemma 4.2

Proof. We first call $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N \times r}$, $\boldsymbol{V} \in \mathbb{R}^{N \times d}$ which satisfies $N \gg d$ and $N \gg r$. $\boldsymbol{M}$ is a 0-1 matrix with elements in the lower triangle all one. $\tilde{\boldsymbol{A}}\boldsymbol{V} = \boldsymbol{B}\boldsymbol{C} \odot \boldsymbol{M}\boldsymbol{V}$ can be rewritten in a partitioned manner:

$$\begin{pmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2, \end{pmatrix} (\boldsymbol{C}_1, \boldsymbol{C}_2) \odot \boldsymbol{M} \begin{pmatrix} \boldsymbol{V}_1 \\ \boldsymbol{V}_2, \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}_1\boldsymbol{C}_1 \odot \boldsymbol{M}, \boldsymbol{0} \\ \boldsymbol{B}_2\boldsymbol{C}_1, \boldsymbol{B}_2\boldsymbol{C}_2 \odot \boldsymbol{M} \end{pmatrix} \begin{pmatrix} \boldsymbol{V}_1 \\ \boldsymbol{V}_2, \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}_1\boldsymbol{C}_1 \odot \boldsymbol{M}\boldsymbol{V}_1 \\ \boldsymbol{B}_2\boldsymbol{C}_1\boldsymbol{V}_1 + \boldsymbol{B}_2\boldsymbol{C}_2 \odot \boldsymbol{M}\boldsymbol{V}_2 \end{pmatrix}$$

For the formula $\boldsymbol{B}_1\boldsymbol{C}_1 \odot \boldsymbol{M}\boldsymbol{V}_1$ and $\boldsymbol{B}_2\boldsymbol{C}_2 \odot \boldsymbol{M}\boldsymbol{V}_2$ we can use the method just now to further recurse. For the formula $\boldsymbol{B}_2\boldsymbol{C}_1\boldsymbol{V}_1$ we can use right multiplication which has a time complexity of $\mathcal{O}(N)$.
According to the above recursion relationship, we can get the time complexity expression:

$$\begin{aligned} T(N) &= 2T(N/2) + N \\ Master \quad theorem: T(N) &= aT(N/b) + f(N) \\ So \quad a = 2, b = 2, f(N) &= N. \\ f(N) &= \Theta(N^{log_b a}) = \Theta(N) \\ then \quad T(N) &= \Theta(NlgN) \end{aligned}$$

♢

In the same way, we can also calculate $\tilde{\boldsymbol{D}'}^{-1} = \mathrm{diag}^{-1}(\boldsymbol{B}\boldsymbol{C} \odot \boldsymbol{M}\boldsymbol{1})$ in $\mathcal{O}(N)$. Compute $\tilde{\boldsymbol{D}'}^{-1}\tilde{\boldsymbol{A}'}\boldsymbol{V}$ only need $\mathcal{O}(N)$ time because $\tilde{\boldsymbol{D}'}^{-1}$ is a diag matrix. Then complete the proof.