Various Lengths, Constant Speed: Efficient Language Modeling with Lightning Attention

Zhen Qin [1] **Weigao Sun** [2] **Dong Li** [2] **Xuyang Shen** [2] **Weixuan Sun** [2] **Yiran Zhong** [2]

Abstract

We present Lightning Attention, the first linear attention implementation that maintains a constant training speed for various sequence lengths under fixed memory consumption. Due to the issue with cumulative summation operations (`cumsum`), previous linear attention implementations cannot achieve their theoretical advantage in a casual setting. However, this issue can be effectively solved by utilizing different attention calculation strategies to compute the different parts of attention. Specifically, we split the attention calculation into intra-blocks and inter-blocks and use conventional attention computation for intra-blocks and linear attention kernel tricks for inter-blocks. This eliminates the need for `cumsum` in the linear attention calculation. Furthermore, a tiling technique is adopted through both forward and backward procedures to take full advantage of the GPU hardware. To enhance accuracy while preserving efficacy, we introduce TransNormerLLM (TNL), a new architecture that is tailored to our lightning attention. We conduct rigorous testing on standard and self-collected datasets with varying model sizes and sequence lengths. TNL is notably more efficient than other language models. In addition, benchmark results indicate that TNL performs on par with state-of-the-art LLMs utilizing conventional transformer structures. The source code is released at github.com/OpenNLPLab/TransnormerLLM.

1. Introduction

Linear attention has emerged as a potentially viable alternative to conventional softmax attention over the last five years (Bahdanau et al., 2016; de Brébisson & Vincent, 2016). However, despite its promise, none of the current leading large language models (Touvron et al., 2023a;b; Zeng et al., 2022; Black et al., 2022; Almazrouei et al., 2023; Team et al., 2023; Wang & Komatsuzaki, 2021; Baichuan, 2023; Jiang et al., 2023) have adopted linear attention mechanisms. There are two possible reasons for that: 1). *Inferior performance:* There is a notable performance gap between existing linear attention-based models (Katharopoulos et al., 2020; Qin et al., 2022b) and state-of-the-art softmax attention-based models (Touvron et al., 2023a;b) in language modeling. 2). *Slow training speed:* Existing linear attention models frequently struggle with slow training speeds due to the use of cumulative summation operations (`cumsum`) (Hua et al., 2022). As a result, these models (Hua et al., 2022) often adopt conventional attention computation during practical use, losing the theoretical advantages of linear attention.

In this paper, we address the aforementioned issues of linear attention and propose a new linear attention-based model that outperforms softmax attention-based models in terms of accuracy and efficiency in language modeling.

Training speed. We introduce Lightning Attention, the first linear attention implementation that enables linear attention to realize its theoretical computational benefits. To achieve the linear computational complexities, the core idea is to leverage the "kernel trick" to accelerate the attention matrix computation, *i.e.,* compute the product of keys and values first to circumvent the $n \times n$ query-key matrix multiplication. The slow operation `cumsum` is needed during the calculation in causal language modeling. To solve this dilemma, we apply the concept of "divide and conquer" to perform the calculation. Specifically, our attention calculation is divided into intra-blocks and inter-blocks. The conventional attention calculation is applied to intra-blocks, while the "kernel trick" is utilized for inter-blocks. We also leverage tiling techniques in both forward and backward processes to maximize GPU hardware performance and tailor the technique used in FlashAttention (Dao et al., 2022a; Dao, 2023) to our Lightning Attention to make it IO-friendly. As a result, Lightning Attention maintains a constant training speed with increasing sequence length under fixed memory consumption, as shown in Fig. 1.

Accuracy. As the adage goes, a good horse often needs a

[1]TapTap [2]OpenNLPLab, Shanghai AI Lab. Correspondence to: Yiran Zhong <zhongyiran@gmail.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

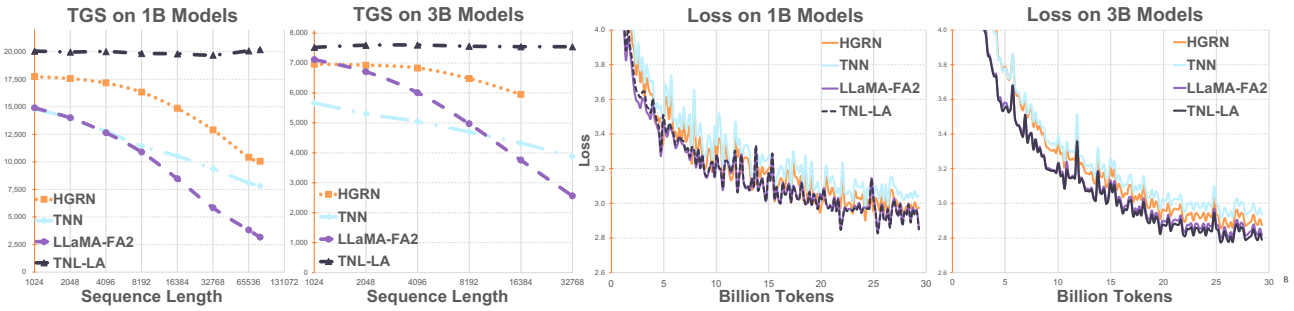

Figure 1. **Training speed and accuracy comparison.** We compare TNL's training speed and losses with state-of-the-art transformer models (LLaMA with FlashAttention-2) and efficient non-transformer models (HGRN (Qin et al., 2023c) and TNN (Qin et al., 2023a)). TNL achieves the lowest training losses and maintains consistent training speed regardless of sequence length.

good spur. We propose a novel architecture, TransNormerLLM (TNL), which is specifically designed for Lightning Attention in order to enhance its performance. TNL evolves from the previous linear attention architecture TransNormer (Qin et al., 2022a) by making advanced modifications that include positional embedding, linear attention acceleration, gating mechanism, tensor normalization. Specifically, we use LRPE (Qin et al., 2023b) together with an exponential decay to avoid attention dilution issues while allowing the model to retain global interactions between tokens. A gating mechanism is utilized to smooth training, and a new tensor normalization scheme is proposed to accelerate the model while preserving its accuracy. We also implement an efficient model parallel schema for TransNormerLLM, enabling seamless deployment on large-scale clusters and facilitating expansion to even more extensive models. As shown in Fig. 1, TNL achieves the lowest training loss among the existing efficient transformer structures (Qin et al., 2023a;c) as well as SOTA transformer models (Touvron et al., 2023b).

We perform a comprehensive evaluation of Lightning Attention across a diverse range of sequence lengths to assess its accuracy and compare its computational speed and memory utilization with FlashAttention-2 (Dao, 2023). Lightning Attention exhibits a notable advantage in computational speed and memory consumption compared to its counterparts without compromising performance. We also validate our model design through a series of ablations and train models with sizes of 44M, 385M, 1B, 7B, and 15B on standard or our self-collected datasets. Benchmark results demonstrate that TNL not only matches the performance of SOTA LLMs with Transformer but is also significantly faster.

2. Related Work

2.1. Efficient Language Modeling

New efficient model architectures are being explored to address the high time complexity of the traditional transformer structure. Four promising alternatives, including linear transformers, state space models, long convolution, and linear recurrence, are being developed to replace self-attention modules for long sequence modeling.

Linear Attention Linear attention decomposes Softmax Attention into the inner product of hidden representations, allowing it to use the "Kernel Trick", where the product of keys and values is computed first to avoid the quadratic $n \times n$ matrix. Different methods utilize various hidden representations. For example, Katharopoulos et al. (2020) use 1+elu as an activation function, Qin et al. (2022b) use the cosine function to approximate the properties of softmax, and Choromanski et al. (2021); Zheng et al. (2022; 2023) approximate softmax through theoretical approaches. Although its theoretical complexity is $O(nd^2)$, the actual computational efficiency of linear attention becomes low when used in causal attention due to the need for *cumsum* operations (Hua et al., 2022). Moreover, most linear attention still exhibits a certain performance gap compared to traditional Transformers (Katharopoulos et al., 2020; Liu et al., 2022).

State Space Model State Space Model is based on the State Space Equation for sequence modeling (Gu et al., 2022b), using special initialization (Gu et al., 2020; 2022c), diagonalization assumptions (Gupta et al., 2022), and mixed techniques (Dao et al., 2022b) to achieve performance comparable to Transformers. Due to the characteristics of the state space equation, inference can be conducted with constant complexity (Gu et al., 2022b), whereas the training speed can be slow compared with FlashAttention.

Long Convolution Long convolution models (Qin et al., 2023a; Fu et al., 2023) utilize a kernel size equal to the input sequence length, facilitating a wider context compared to traditional convolutions. Training these models involves Fast Fourier Transforms (FFT) algorithm, reducing the computational complexities to $O(n \log n)$. However, long convolution models need to cache all historical computations for causal convolution inference, making them less

Algorithm 1 Linear Attention Left Product

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$.
Initialize mask $\mathbf{M} \in \mathbb{R}^{n \times n}$, where $\mathbf{M}_{ts} = 1$, if $t \geq s$, else 0.
Load $\mathbf{Q}, \mathbf{K}, \mathbf{M}$ from HBM, compute $\mathbf{S} = (\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}$, write $\mathbf{S}$ to HBM.
Load $\mathbf{S}, \mathbf{V}$ from HBM, compute $\mathbf{O} = \mathbf{S}\mathbf{V}$, write $\mathbf{O}$ to HBM.
Return $\mathbf{O}$.

ideal for processing long sequences compared to RNNs.

Linear RNN Linear RNNs (Orvieto et al., 2023a; Qin et al., 2023c), in contrast, stand out as more suitable replacements for transformers in long-sequence modeling. A notable example is the HGRN (Qin et al., 2023c) model, a linear RNN-based LLM that has shown competitive performance against similarly scaled GPT models.

2.2. IO-aware Attention

The FlashAttention series (Dao et al., 2022a; Dao, 2023) focuses on system-level optimizations for the efficient implementation of the standard attention operator on GPU platforms. These approaches employ tiling strategies to minimize the volume of memory reads/writes between the GPU's high bandwidth memory (HBM) and on-chip SRAM. Although these methods optimize the IO communication in attention calculation and are faster than previous softmax attention implementations, their theoretical computation complexity remains $O(n^2d)$, making them unsuitable for long sequence language modeling.

3. Lightning Attention

3.1. Preliminary

We first recall the formulation of linear attention and then introduce our proposed Lightning Attention. In the case of NormAttention within TransNormer (Qin et al., 2022a), attention computation deviates from the conventional Transformer structure (Vaswani et al., 2017) by eschewing the costly softmax and scaling operations. The NormAttention mechanism can be expressed as follows:

$$\mathbf{O} = \mathrm{Norm}((\mathbf{Q}\mathbf{K}^\top)\mathbf{V}), \tag{1}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V} \in \mathbb{R}^{n \times d}$ are the query, key, and value matrices, respectively, with n for sequence length and d for feature dimension. The equation can be transformed into its linear variant using right matrix multiplication:

$$\mathbf{O} = \mathrm{Norm}(\mathbf{Q}(\mathbf{K}^\top\mathbf{V})), \tag{2}$$

The linear formulation enables efficient recurrent prediction with $O(nd^2)$ complexity during training. Additionally, linear attention guarantees a constant computation complexity of $O(d^2)$ regardless of the sequence length. This is achieved by recurrently updating $\mathbf{K}^\top\mathbf{V}$, eliminating the need for repeated computation of the entire attention matrix. In contrast, standard softmax attention has a complexity of $O(nd^2)$ during inference.

Algorithm 2 Linear Attention Right Product

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$.
Initialize $\mathbf{kv} = 0 \in \mathbb{R}^{d \times d}$.
for $t = 1, \ldots, n$ **do**
  Load $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t \in \mathbb{R}^{d \times 1}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{kv} = \mathbf{kv} + \mathbf{k}_t\mathbf{v}_t^\top$.
  On chip, compute $\mathbf{o}_t = \mathbf{q}_t^\top\mathbf{kv}$.
  Write $\mathbf{o}_t^\top$ to HBM as the t-th row of $\mathbf{O}$.
end for
Return $\mathbf{O}$.

Nevertheless, when dealing with causal prediction tasks, the effectiveness of the right product is compromised, leading to the requirement for the computation of `cumsum` (Hua et al., 2022). This impediment hinders the potential for highly efficient parallel computation. In this section, we show that the requirement of `cumsum` can be eliminated by leveraging the concept of "divide and conquer" in linear attention calculation. For the convenience of discussion, Norm will be ignored in the subsequent discussion.

There are two computational approaches to handling the causal scenario. One is using conventional attention computation (the Left Product), which involves computing $\mathbf{Q}\mathbf{K}^\top$ first. The complete calculation formula is as follows:

$$\mathbf{O} = [(\mathbf{Q}\mathbf{K}^\top) \odot \mathbf{M}]\mathbf{V} \tag{3}$$

where $\mathbf{M}_{ts} = 1$ if $t \geq s$, otherwise 0. The complete algorithm is detailed in Algorithm 1. Note that this algorithm is parallelizable, but its time complexity is $O(n^2d)$. The other option is to compute the $\mathbf{k}_t\mathbf{v}_t^\top$ first (the Right Product), which leverages a recursive formula for computation:

$$\mathbf{kv}_0 = \mathbf{0}, \mathbf{kv}_t = \mathbf{kv}_{t-1} + \mathbf{k}_t\mathbf{v}_t^\top, \mathbf{o}_t^\top = \mathbf{q}_t^\top\mathbf{kv_t}. \tag{4}$$

The complete algorithm is detailed in Algorithm 2. This algorithm has a time complexity of $O(nd^2)$, but it is not GPU-friendly, making it slower than the first approach.

3.2. Linear Attention with Tiling

We use a tiling technique to compute linear attention in a causal setting. Specifically, we first divide $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ into two blocks by rows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix}, \mathbf{X}_1 \in \mathbb{R}^{m \times d}, \mathbf{X}_2 \in \mathbb{R}^{(n-m) \times d},$$
$$\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}.$$

Then, by unfolding Eq. 3, we get (note that $\mathbf{kv}_0 = 0$):

$$\begin{aligned} \mathbf{kv}_s &= \mathbf{kv}_0 + \sum_{j=1}^{s} \mathbf{k}_j\mathbf{v}_j^\top, s = 1, \ldots, m. \\ \mathbf{o}_s^\top &= \mathbf{q}_s^\top\mathbf{kv}_s = \mathbf{q}_s^\top\mathbf{kv}_0 + \mathbf{q}_s^\top \sum_{j=1}^{s} \mathbf{k}_j\mathbf{v}_j^\top. \end{aligned} \tag{5}$$

Algorithm 3 Lightning Attention Forward Pass

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$, block sizes B.
Divide $\mathbf{X}$ into $T = \frac{n}{B}$ blocks $\mathbf{X}_1, \mathbf{X}_2, ...\mathbf{X}_T$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}\}$.
Initialize mask $\mathbf{M} \in \mathbb{R}^{B \times B}$, where $\mathbf{M}_{ts} = 1$, if $t \geq s$, else 0.
Initialize $\mathbf{KV} = 0 \in \mathbb{R}^{d \times d}$.
for $t = 1, \ldots, T$ **do**
  Load $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{B \times d}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{O}_{\text{intra}} = [(\mathbf{Q}_t \mathbf{K}_t^\top) \odot \mathbf{M}]\mathbf{V}_t$.
  On chip, compute $\mathbf{O}_{\text{inter}} = \mathbf{Q}_t(\mathbf{KV})$.
  On chip, compute $\mathbf{KV} = \mathbf{KV} + \mathbf{K}_t^\top \mathbf{V}_t$.
  Write $\mathbf{O}_t = \mathbf{O}_{\text{intra}} + \mathbf{O}_{\text{inter}}$ to HBM as the t-th block of $\mathbf{O}$.
end for
Return $\mathbf{O}$.

In block form, we have:

$$\begin{aligned} \mathbf{O}_1 &= \mathbf{Q}_1 \mathbf{kv}_0 + [(\mathbf{Q}_1 \mathbf{K}_1^\top) \odot \mathbf{M}]\mathbf{V}_1 \\ &\triangleq \mathbf{Q}_1 \mathbf{KV}_0 + [(\mathbf{Q}_1 \mathbf{K}_1^\top) \odot \mathbf{M}]\mathbf{V}_1. \end{aligned} \tag{6}$$

The above formula shows that the forward causal linear attention can be divided into two parts:

- The computation within the block $[(\mathbf{Q}_1 \mathbf{K}_1^\top) \odot \mathbf{M}]\mathbf{V}_1$ (intra blocks) can use the Left Product;
- The computation between blocks $\mathbf{Q}_1 \mathbf{KV}_0$ (inter blocks) can use the Right Product.

It is worth noting that the second block can be computed using the same idea as follows:

$$\begin{aligned} \mathbf{kv}_{m+t} &= \mathbf{kv}_m + \sum_{j=m+1}^{m+t} \mathbf{k}_j \mathbf{v}_j^\top, t = 1, \ldots, n-m, \\ \mathbf{o}_{m+t}^\top &= \mathbf{q}_{m+t}^\top \mathbf{kv}_{m+t}, \\ \mathbf{O}_2 &= \mathbf{Q}_2 \mathbf{kv}_m + [(\mathbf{Q}_2 \mathbf{K}_2^\top) \odot \mathbf{M}]\mathbf{V}_2 \\ &\triangleq \mathbf{Q}_2 \mathbf{KV}_1 + [(\mathbf{Q}_2 \mathbf{K}_2^\top) \odot \mathbf{M}]\mathbf{V}_2. \end{aligned} \tag{7}$$

Note that to compute the second block, we have to use $\mathbf{KV}_1 = \mathbf{kv}_m$, which can be computed by:

$$\mathbf{KV}_1 = \mathbf{KV}_0 + \sum_{j=1}^{m} \mathbf{k}_m \mathbf{v}_m^\top = \mathbf{KV}_0 + \mathbf{K}_1^\top \mathbf{V}_1. \tag{8}$$

where $\mathbf{KV}_0 = \mathbf{kv}_0$. By using the above strategy to divide the matrix into multiple blocks, we obtain the Lightning Attention Forward Pass. More detailed derivation can be found in the Appendix C.

For the backward propagation, according to (Katharopoulos et al., 2020), we can rewrite the process as:

$$\mathbf{dq}_t^\top = \mathbf{do}_t^\top \mathbf{kv}_t^\top, \mathbf{dk}_t^\top = \mathbf{v}_t^\top \mathbf{dkv}_t^\top, \mathbf{dv}_t^\top = \mathbf{k}_t^\top \mathbf{dkv}_t,$$
$$\mathbf{dkv}_{n+1} = 0 \in \mathbb{R}^{d \times d}, \mathbf{dkv}_{t-1} = \mathbf{dkv}_t + \mathbf{q}_{t-1} \mathbf{do}_{t-1}^\top.$$

Therefore, the calculation of the backward propagation is consistent with the forward Eq. 4, and the Lightning Attention Backward Pass can also be obtained using the tiling technique. A detailed proof can be found in the Appendix C.

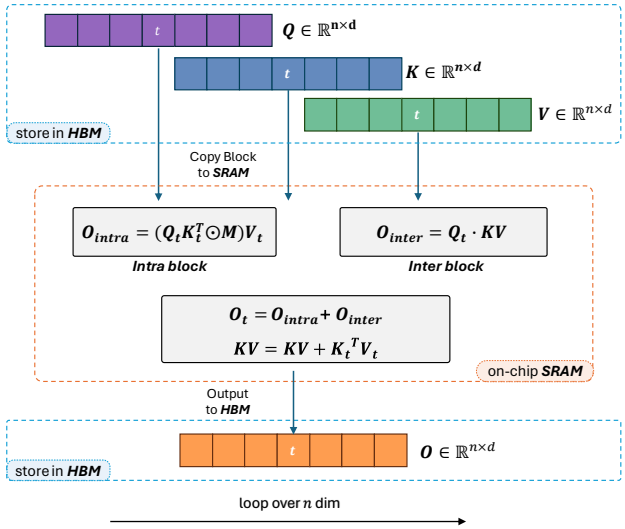

Figure 2. **Structural framework of Lightning Attention** is detailed in its algorithmic schematic. During the t-th iteration, the tiling blocks of matrices $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t$ are transferred from High Bandwidth Memory (HBM) to Static Random-Access Memory (SRAM). Within the SRAM, the outputs $\mathbf{O}_{\text{intra}}$ and $\mathbf{O}_{\text{inter}}$ are computed independently, followed by an update to the $\mathbf{KV}$ matrix. Subsequently, the final output $\mathbf{O}_t$, which is the sum of $\mathbf{O}_{\text{intra}}$ and $\mathbf{O}_{\text{inter}}$, is written back from SRAM to HBM.

3.3. Complexity analysis

Theorem 3.1. The time complexity of Lightning Attention is $O(nd^2 + nBd)$[1].

Proof of Theorem 3.1. For the forward pass, according to Algorithm 3, each intra part's time complexity is $O(B^2 d)$, each inter part's time complexity is $O(Bd^2)$, the time complexity of updating $\mathbf{KV}$ is $O(Bd^2)$, so each the time complexity in each loop is $O(B^2 d + Bd^2)$, since we loop for $T = n/B$ times, the total time complexity is $O((B^2 d + Bd^2)n/B) = O(nd^2 + nBd)$. Because the computation of the backward pass is similar to that of the forward pass, the time complexity of the backward pass is also $O(nd^2 + nBd)$. □

3.4. Exact IO-aware Implementation

Lightning Attention employs the above tiling methodology throughout its whole computation process and leverages distinct approaches to optimize the utilization of memory bandwidth between HBM and SRAM within a GPU. Specifically, in each iteration t, matrices $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t$ undergo segmentation into blocks, subsequently transferred to SRAM for computation. The intra- and inter-block operations are segregated, with intra-blocks employing the left product and inter-blocks utilizing the right product. This approach

[1] we choose $B \approx d$ in practice, the time complexity is $O(nd^2)$.

Algorithm 4 Lightning Attention Backward Pass

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{dO} \in \mathbb{R}^{n \times d}$, block sizes B.
Divide $\mathbf{X}$ into $T = \frac{n}{B}$ blocks $\mathbf{X}_1, \mathbf{X}_2, ...\mathbf{X}_T$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$.
Divide $\mathbf{dX}$ into $T = \frac{n}{B}$ blocks $\mathbf{dX}_1, \mathbf{dX}_2, ...\mathbf{dX}_T$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}\}$.
Initialize mask $\mathbf{M} \in \mathbb{R}^{B \times B}$, where $\mathbf{M}_{ts} = 1$, if $t \geq s$, else 0.
Initialize $\mathbf{KV} = 0, \mathbf{dKV} = 0 \in \mathbb{R}^{d \times d}$.
for $t = 1, \ldots, T$ **do**
  Load $\mathbf{K}_t, \mathbf{V}_t, \mathbf{O}_t, \mathbf{dO}_t \in \mathbb{R}^{B \times d}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{dQ}_{\text{intra}} = [(\mathbf{dO}_t \mathbf{V}_t^\top) \odot \mathbf{M}]\mathbf{K}_t$.
  On chip, compute $\mathbf{dQ}_{\text{inter}} = \mathbf{dO}_t \mathbf{KV}^\top$.
  On chip, compute $\mathbf{KV} = \mathbf{KV} + \mathbf{K}_t^\top \mathbf{V}_t$.
  Write $\mathbf{dQ}_t = \mathbf{dQ}_{\text{intra}} + \mathbf{dQ}_{\text{inter}}$ to HBM as the t-th block of $\mathbf{dQ}$.
end for
for $t = T, \ldots, 1$ **do**
  Load $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t, \mathbf{O}_t, \mathbf{dO}_t \in \mathbb{R}^{B \times d}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{dK}_{\text{intra}} = [(\mathbf{dO}_t \mathbf{V}_t^\top) \odot \mathbf{M}]^\top \mathbf{Q}_t$.
  On chip, compute $\mathbf{dK}_{\text{inter}} = \mathbf{V}_t \mathbf{dKV}^\top$.
  On chip, compute $\mathbf{dV}_{\text{intra}} = [(\mathbf{Q}_t \mathbf{K}_t^\top) \odot \mathbf{M}]^\top \mathbf{dO}_t$.
  On chip, compute $\mathbf{dV}_{\text{inter}} = \mathbf{K}_t \mathbf{dKV}$.
  On chip, compute $\mathbf{dKV} = \mathbf{dKV} + \mathbf{Q}_t^\top \mathbf{dO}_t$.
  Write $\mathbf{dK}_t = \mathbf{dK}_{\text{intra}} + \mathbf{dK}_{\text{inter}}, \mathbf{dV}_t = \mathbf{dV}_{\text{intra}} + \mathbf{dV}_{\text{inter}}$ to HBM as the t-th block of $\mathbf{dK}, \mathbf{dV}$.
end for
Return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

optimally exploits the computational and memory efficiencies associated with the right product, enhancing overall execution speed. The intermediate activation $\mathbf{KV}$ is iteratively saved and accumulated within SRAM. Subsequently, the outputs of intra-blocks and inter-blocks are summed within SRAM, and the results are written back to HBM. The structure of Lightning Attention is illustrated in Fig. 2. The intricate details of the Lightning Attention implementation are explained through Algorithm 3 for the forward pass and Algorithm 4 for the backward pass.

4. TransNormerLLM

4.1. The Overall Structure

Our structure is based on the findings of TransNormer (Qin et al., 2022a) but has custom modifications to balance efficiency and performance. We illustrate the overall structure in Fig. 3. The input $\mathbf{X}$ is updated through two consecutive steps: 1). It undergoes Gated Linear Attention (GLA) with the application of SimpleRMSNorm (SRMSNorm) normalization. 2). It goes through the Simple Gated Linear Unit (SGLU) with SRMSNorm normalization. We apply the Pre-norm for both modules.

4.2. Custom Modification

In this section, we outline the key designs and inspiration behind each custom modification, including positional encoding, gating mechanisms, and tensor normalization.

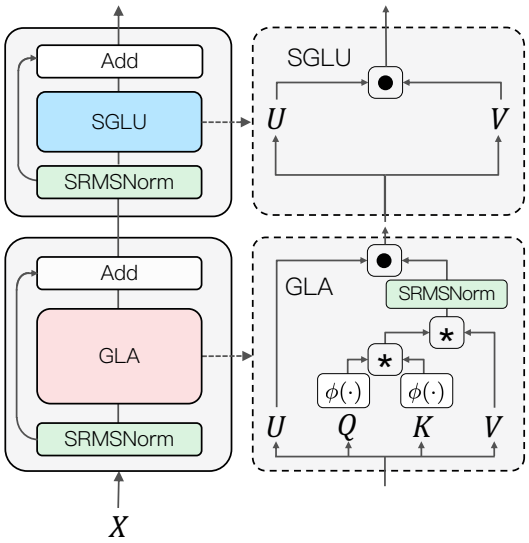

Figure 3. Architecture overview of TransNormerLLM (TNL). Each transformer block is composed of a Gated Linear Attention (GLA) for token mixing and a Simple Gated Linear Unit (SGLU) for channel mixing. We apply Pre-norm for both modules.

Position Encoding In TransNormer, DiagAttention is used at the lower layers to avoid dilution issues. However, this leads to a lack of global interaction between tokens. In TNL, we leverage LRPE (Qin et al., 2023b) with exponential decay (Press et al., 2022; Qin et al., 2023a; Peng et al., 2023b) to address this issue, retaining full attention at the lower layers. The expression of our position encoding is as follows:

$$a_{ts} = \mathbf{q}_t^\top \mathbf{k}_s \lambda^{t-s} \exp^{i\theta(t-s)} . \tag{9}$$

which we call LRPE-d - Linearized Relative Positional Encoding with exponential decay. Similar to the original LRPE, we set θ to be learnable. We empirically find that rather than applying LRPE-d to every layer, applying it to the first layer and keeping other layers with exponential decay can speed up training by approximately 15-20% but only with a subtle effect on the performance.

Note that this position encoding is fully compatible with Linear Attention, as it can be decomposed with respect to s and t separately. The value of λ for the h-th head in the l-th layer (assuming there are a total of H heads and L layers) is given by:

$$\lambda = \exp\left(-\frac{8h}{H} \times \left(1 - \frac{l}{L}\right)\right) . \tag{10}$$

Here, $\frac{8h}{H}$ corresponds to the decay rate of the h-th head, while $\left(1 - \frac{l}{L}\right)$ corresponds to the decay rate of the l-th layer. The term $\left(1 - \frac{l}{L}\right)$ ensures that the Theoretical Receptive Fields (TRF) (Qin et al., 2024) at the lower layers is smaller compared to the higher layers, which aligns with TransNormer's motivation. We choose λ to be non-learnable since we empirically found that gradients become unstable

when λ is learnable, leading to NaN values. Note that this positional encoding is still compatible with Lightning Attention, with the specific algorithm detailed in Appendix A B.

Gating Mechanism Gate can enhance the performance of the model and smooth the training process. In TNL, we adopt the approach from Flash (Hua et al., 2022) and use Gated Linear Attention (GLA) in token mixing:

$$\begin{aligned}\mathbf{O} &= \mathrm{Norm}(\mathbf{Q}\mathbf{K}^\top\mathbf{V}) \odot \mathbf{U}, \mathbf{Q} = \phi(\mathbf{X}\mathbf{W}_q), \\ \mathbf{K} &= \phi(\mathbf{X}\mathbf{W}_k), \mathbf{V} = \mathbf{X}\mathbf{W}_v, \mathbf{U} = \mathbf{X}\mathbf{W}_u.\end{aligned} \tag{11}$$

We choose ϕ to be Swish (Ramachandran et al., 2017) activation function as we empirically find that it outperforms other activation functions.

To further accelerate the model, we propose Simple GLU (SGLU), which removes the activation function from the original GLU structure as the gate itself can introduce nonlinearity. Therefore, our channel mixing becomes:

$$\mathbf{O} = [\mathbf{V} \odot \mathbf{U}]\mathbf{W}_o, \mathbf{V} = \mathbf{X}\mathbf{W}_v, \mathbf{U} = \mathbf{X}\mathbf{W}_u. \tag{12}$$

We empirically find that not using an activation function in GLU will not lead to any performance loss.

Tensor Normalization The origin NormAttention introduced in TransNormer (Qin et al., 2022a) is as follows:

$$\mathbf{O} = \mathrm{Norm}(\mathbf{Q}\mathbf{K}^\top\mathbf{V}) \tag{13}$$

In TransNormerLLM, we replace the origin RMSNorm with a new simple normalization function called SimpleRMSNorm, abbreviated as SRMSNorm:

$$\mathrm{SRMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\|\mathbf{x}\|_2/\sqrt{d}}. \tag{14}$$

We empirically find that using SRMSNorm does not lead to any performance loss.

5. Experiments

We carried out thorough experiments on TNL models and lightning attention. We implemented our models on the Metaseq framework (Zhang et al., 2022) with Pytorch (Paszke et al., 2019). The Lightning Attention was executed through Triton (Tillet et al., 2019). All the experiments were conducted on A100 80G GPU clusters. The assessment of our work is divided into three main sections: I) We evaluated the efficiency and accuracy of the Lightning Attention module; II) We further benchmarked our TNL models' performance on standard small-scale corpus and LLM benchmarks and compared their training and inference speeds with STOA models; III) We also provide an ablation study on the design of TNL.

5.1. Lightning Attention Evaluation

Since our Lightning Attention is an exact implementation of norm linear attention (Qin et al., 2022a), we compared the speed and memory usage between its original pytorch implementation (named Vanilla) and our Lightning Attention. As a reference, we have also included FlashAttention-2 (Dao, 2023) (named Flash2), which is currently the SOTA implementation of softmax attention. As shown in Fig. 4, Lightning Attention shows remarkable linear growth of processing time in both forward and backward passes, whereas Vanilla and Flash2 exhibit quadratic growth. In terms of memory footprint, Vanilla tends to rapidly exhaust memory resources. Lightning Attention shows a similar trend to Flash2 but requires less memory.

Table 1. **Results on Wikitext-103** (TNN(Qin et al., 2023a)'s setting). ↓ means *lower is better*.

	Model	PPL (val)↓	PPL (test)↓	Params (M)
Attn-based	Transformer	24.40	24.78	44.65
	FLASH	25.92	26.70	42.17
	1+elu	27.44	28.05	44.65
	Performer	62.50	63.16	44.65
	cosFormer	26.53	27.06	44.65
	TN1	24.43	25.00	44.64
	TN2	24.50	25.05	44.64
MLP-based	Syn(D)	31.31	32.43	46.75
	Syn(R)	33.68	34.78	44.65
	gMLP	28.08	29.13	47.83
RNN-based	S4	38.34	39.66	45.69
	DSS	39.39	41.07	45.73
	GSS	29.61	30.74	43.84
	RWKV	24.31	25.07	46.23
	LRU	29.86	31.12	46.24
	HGRN	24.14	24.82	46.25
FFT-based	TNN	23.98	24.67	48.68
Ours	TNL	23.46	24.03	45.45

5.2. TNL Evaluation

Performance Evaluation In Table 1, we present an evaluation across various 40M models on a standard dataset. This includes models based on attention/linear attention mechanisms (Vaswani et al., 2017; Dao et al., 2022a; Katharopoulos et al., 2020; Qin et al., 2022b;a), MLPs (Multi-Layer Perceptrons) (Tay et al., 2021; Liu et al., 2021), RNNs (Recurrent Neural Networks) (Gu et al., 2022a; Gupta et al., 2022; Mehta et al., 2022; Peng et al., 2023b; Orvieto et al., 2023b), FFTs (Fast Fourier Transforms) (Qin et al., 2023a), and our model. TNL records the lowest perplexity on test set after trained on the Wikitext-103 dataset.

We also scaled up our model to 1B and 3B parameters and compared its training loss with top-tier LLM structures such as LLaMA-FA2 (Touvron et al., 2023a; Dao, 2023), HGRN (Qin et al., 2023c), and TNN (Qin et al., 2023a). For a fair comparison, we retrain all models on the same 30B corpus and plot the training losses in Fig. 1. TNL achieved the lowest training losses in both 1B and 3B parameters.

Efficiency Evaluation In Fig. 1, we present a comparative analysis of training speeds under the same corpora and

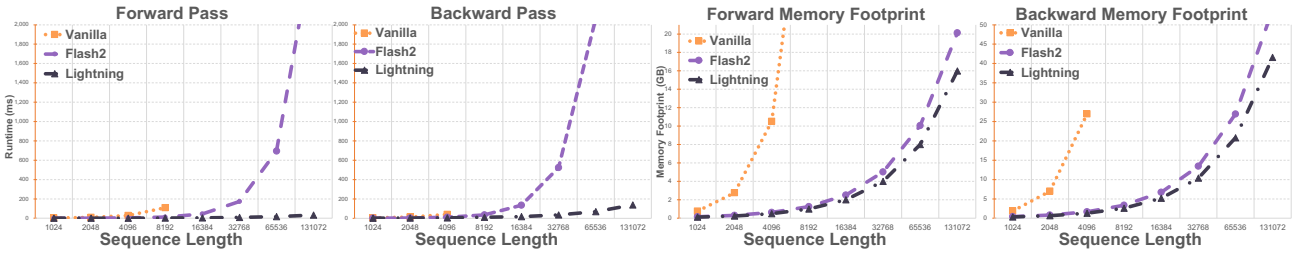

Figure 4. **Comparative Analysis of Speed and Memory Usage**: Vanilla represents norm linear attention in pytorch (Qin et al., 2022a), Flash2 represents FlashAttention-2. Left two sub-figures: Runtime in milliseconds for the forward and backward pass across varying sequence lengths. Right two sub-figures: Memory utilization (in GB) during the forward and backward pass at different sequence lengths.

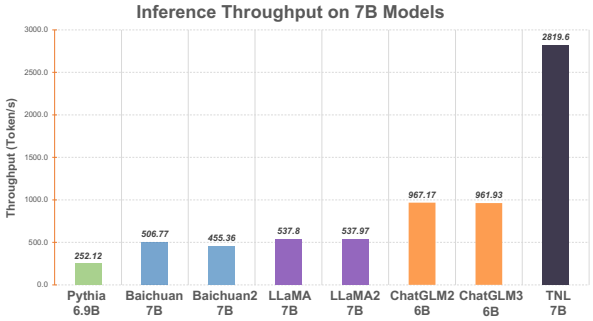

Figure 5. **Inference Throughput Comparison.** We measure the inference throughput of various 7B LLM models on a A100 80G GPU. Batch sizes for models are chosen to optimize GPU utilization without exceeding memory limits. Each model is tested with a 512-token input prompt and can generate up to 1024 new tokens. Reported throughput is averaged from 20 attempts.

hardware setups. This comparison encompasses four variants: TNL, LLaMA-FA2 (Touvron et al., 2023a; Dao, 2023), HGRN (Qin et al., 2023c) , and TNN (Qin et al., 2023a). Our findings show that during both the forward and backward passes, the TGS (tokens per GPU per second) for TNL remains consistently high, while the other three models exhibit a rapid decline when sequence length is scaled from 1K to 128K. This pattern suggests that Lightning Attention offers a significant advancement in managing extremely long sequence lengths in LLM.

Inference Evaluation We conduct an inference throughput comparison on various 7B large language models using their standard codebase from Huggingface, as detailed in Fig. 5. TNL with Lightning Attention demonstrates a significant advantage, achieving a throughput rate that up to 11$\times$ higher than transformer structure models.

Benchmark Results In order to validate the effectiveness of TNL, we pretraining 385M, 1B, 7B, and 15B models on self-collected datasets, the details of the data are in the Appendix D, and tested on Commonsense Reasoning Task, MMLU(Hendrycks et al., 2021), C-Eval(Huang et al., 2023), and SCROLLS (Shaham et al., 2022). For comparison, we selected several open-source models as competitors, including Transformer-based models such as OPT (Zhang et al., 2022), Pythia (Biderman et al., 2023), BLOOM (Workshop et al., 2023), GPT-Neo (Black et al., 2022), Falcon (Almazrouei et al., 2023), LLaMA (Touvron et al., 2023a;b), OpenLLAMA (Geng & Liu, 2023), Baichuan (Baichuan, 2023), ChatGLM (Zeng et al., 2022; Du et al., 2022), and non-Transformer model RWKV (Peng et al., 2023a). It can be observed in Table 2 and Table 3 that, compared to these models, TNL remains highly competitive.

- We report BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2019), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2019), ARC easy and challenge (Clark et al., 2018) and OpenBookQA (Mihaylov et al., 2018). We report 0-shot results for all benchmarks using LM-Eval-Harness (Gao et al., 2021). All of our models achieve competitive performance compared to existing state-of-the-art LLMs, showcasing a remarkable ability to comprehend and apply commonsense reasoning.
- We report the overall results for MMLU (Hendrycks et al., 2021), C-Eval (Huang et al., 2023). Official scripts were used for evaluating MMLU and C-Eval, with all evaluation results being conducted with a 5-shot setup. In comparison to top-tier open-source models available in the industry, our models have demonstrated matched performance in both English and Chinese benchmarks.
- On SCROLLS (Shaham et al., 2022) benchmark, we assess the large language models trained on a 1 billion parameter and pre-trained using a sequence length of 2048. We present zero-shot performance results for all benchmarks using the LM-Eval-Harness (Gao et al., 2021). For generation tasks within SCROLLS, we employ a greedy search with hyper-parameters top_k set to 5 and top_p set to 1. Our models consistently match or surpass the performance of existing state-of-the-art LLMs in these tasks.

Table 2. **Performance Comparison on Commonsense Reasoning and Aggregated Benchmarks.** For a fair comparison, we report competing methods' results reproduced by us using their released models. Official results are denoted in *italics*. PS: parameter size (billion). T: tokens (billion). HS: HellaSwag. WG: WinoGrande.

Model	PS	T	BoolQ	PIQA	HS	WG	ARC-e	ARC-c	OBQA	MMLU	C-Eval
	B	B	acc	acc	acc_norm	acc	acc	acc_norm	acc_norm	acc-5shot	acc-5shot
OPT	0.35	0.30	57.74	64.58	36.69	52.49	44.02	23.89	28.20	26.02	25.71
Pythia	0.40	0.30	60.40	67.08	40.52	53.59	51.81	24.15	29.40	25.99	24.81
RWKV	0.43	-	-	*67.52*	*40.90*	*51.14*	*52.86*	*25.17*	*32.40*	24.85	-
TNL	0.39	1.0	62.14	66.70	46.27	54.46	55.43	27.99	32.40	25.90	25.24
OPT	1.3	0.3	57.77	71.71	53.70	59.35	57.24	29.69	33.20	24.96	25.32
Pythia	1.4	0.3	60.73	70.67	47.18	53.51	56.99	26.88	31.40	26.55	24.25
RWKV	1.5	-	-	*72.36*	*52.48*	*54.62*	*60.48*	*29.44*	*34.00*	25.77	-
Falcon	1.0	0.35	61.38	75.14	61.50	60.30	63.38	32.17	35.60	25.28	25.66
TNL	1.0	1.2	63.27	72.09	56.49	60.38	63.68	35.24	36.60	27.10	26.01
OPT	6.7	0.3	66.18	76.22	67.21	65.19	65.66	34.64	37.20	24.57	25.32
Pythia	6.9	0.3	63.46	75.14	63.92	60.77	67.34	35.41	37.00	24.64	26.40
RWKV	7.4	-	-	*76.06*	*65.51*	*61.01*	*67.80*	*37.46*	*40.20*	24.96	-
Falcon	7.2	1.5	73.73	79.38	76.3	67.17	74.62	43.60	43.80	27.79	22.92
Baichuan2	7.0	2.6	72.72	76.50	72.17	68.35	75.17	42.32	39.60	*54.16*	*54.00*
ChatGLM2	7.1	1.4	77.65	69.37	50.51	57.62	59.13	34.30	37.00	*45.46*	*52.55*
OpenLLaMAv2	6.7	1.0	72.20	78.84	74.51	65.67	72.39	41.30	41.00	41.29	30.01
LLaMA1	6.7	1.0	*76.50*	*79.80*	*76.10*	*70.10*	*72.80*	*47.60*	*57.20*	*35.10*	25.72
LLaMA2	6.7	2.0	*77.68*	*78.07*	*76.02*	*68.98*	*76.30*	*46.33*	*44.20*	*45.30*	33.20
TNL	6.8	1.4	75.87	80.09	75.21	66.06	75.42	44.40	63.40	43.10	43.18
OPT	13	0.3	65.93	75.84	69.83	65.19	67.00	35.75	38.80	24.68	22.23
Pythia	12	0.3	65.72	76.17	68.85	66.22	70.62	38.23	41.00	25.51	22.99
RWKV	14	-	70.12	78.51	71.49	64.48	72.35	40.87	41.00	26.49	26.49
Baichuan2	13	2.6	79.20	77.31	75.27	70.01	77.36	47.01	43.80	57.02	59.63
OpenLLaMAv2	13	1.0	72.29	77.58	72.07	70.09	75.42	43.86	43.00	43.43	25.95
LLaMA1	13	1.0	77.95	79.16	79.06	72.61	77.40	47.70	44.80	47.62	32.13
LLaMA2	13	2.0	80.61	79.11	79.35	72.38	79.34	48.98	35.20	55.70	38.34
TNL	15	2.0	76.64	81.56	82.18	75.61	77.61	50.51	46.40	60.06	53.01

5.3. TNL Ablation

We conducted an extensive ablation analysis on various components of TNL, including positional encoding, gating mechanisms, GLA activation functions, GLU activation functions, and normalization functions.

Table 4. **Exploration of Positional Encoding.** LRPE-d leads to the most optimal outcome.

PE Methods	Params	Updates	Loss	PPL
Mix	385M	100K	2.248	4.770
APE	386M	100K	2.387	5.253
Exp-Decay	385M	100K	2.267	4.834
LRPE	385M	100K	2.287	4.899
LRPE-d	385M	100K	2.236	4.728

Positional Encoding: in our experiment comparing various PE strategies—Mix, Absolute Positional Encoding (APE), LRPE, Exponential Decay, and LRPE-d—our approach and LRPE-d demonstrated superior performance. We chose the Mix method for its ability to enhance training speed by up to 20%, despite being slightly less effective than LRPE-d.

Table 5. **Ablations on decay temperature.** The results of decay temperature proved to be superior.

Temperature	Params	Updates	Loss	PPL
w/ temperature	385M	100K	2.248	4.770
w/o temperature	385M	100K	2.258	4.804

We also perform ablations on the decay temperature $\left(1-\frac{l}{L}\right)$ in Eq. 10. The perplexity of the TNL is reduced by adding the decay temperature, as shown in Table 5.

Table 6. **Ablations on gating mechanism.** The performance with the gate proved to be superior.

Gate	Params	Updates	Loss	PPL
w/ gate	385M	100K	2.248	4.770
w/o gate	379M	100K	2.263	4.820

Gating Mechanism: we further investigate the impact of integrating a gating mechanism. According to the data presented in Table 6, enabling the gate decreased the loss value from 2.263 to 2.248.

Table 3. **Performance Comparison on SCROLLS (Shaham et al., 2022):** A review of models up to 1 billion parameters on 2048 pre-training sequence length. PS: parameter size (billion). T: tokens (billion).

Model	PS	T	GovRep	SumScr	QMSum	Qspr	Nrtv	QALT	CNLI	Avg
	B	B	ROUGE-1/2/L	ROUGE-1/2/L	ROUGE-1/2/L	F1	F1	EM	EM	
OPT	0.35	0.30	2.52/0.53/2.24	7.72/0.68/6.52	8.05/1.79/6.6	13.13	10.13	29.05	9.16	7.55
Pythia	0.40	0.30	4.96/1.19/4.06	2.03/0.2/1.79	7.51/1.43/6.08	15.27	8.24	28.57	15.24	7.43
RWKV	0.43	-	1.63/0.4/1.49	0.94/0.11/0.76	10.19/2.26/8.06	13.16	9.76	26.32	16.49	7.04
TNL	0.39	1.0	3.67/1.16/3.14	8.27/0.82/6.91	13.62/3.29/10.95	14.29	11.69	28.14	17.36	9.48
OPT	1.3	0.3	5.7/2.09/4.41	10.17/0.82/8.29	12.36/3.15/9.85	18.37	13.42	29.15	12.44	10.02
Pythia	1.4	0.3	4.03/1.25/3.33	8.34/0.87/6.97	13.17/3.4/10.92	16.09	11.91	28.72	9.06	9.08
Falcon	1.0	0.35	2.74/0.67/2.37	10.95/1.28/8.66	13.29/3.09/10.58	16.17	12.91	29.19	14.75	9.74
TNL	1.0	1.2	6.81/2.30/5.25	12.28/1.23/9.27	14.60/3.51/11.62	15.02	14.66	28.72	37.32	12.51

Table 7. **Exploration of Normalization Function.** The deviation in results among the bellowing normalization functions is minimal.

Norm Type	Params	Updates	Loss	PPL
SRMSNorm	385M	100K	2.248	4.770
RMSNorm	385M	100K	2.247	4.766
LayerNorm	385M	100K	2.247	4.765

Normalization Functions: our study involved testing various normalization techniques—SRMSNorm, RMSNorm, and LayerNorm—on TNL, finding little difference in their effectiveness. However, we enhanced SRMSNorm using Triton, resulting in notable improvements in processing speed for larger dimensions.

GLA Activation Functions: in our study on the GLA (Gated Linear Attention) mechanism, we evaluated activation functions, finding Swish and 1+elu to perform similarly, as detailed in Table 8. However, due to NaN issues with 1+elu in our 7B model, we opted for Swish.

Table 8. **Ablations on GLA activation functions.** The results obtained from different activation functions were virtually identical.

GLA Act	Params	Updates	Loss	PPL
Swish	385M	100K	2.248	4.770
No Act	385M	100K	2.283	4.882
1+elu	385M	100K	2.252	4.767

GLU Activation Functions: our experiment additionally involved removing the activation function from the Gated Linear Units (GLU), showing minimal effect on outcomes as per Table 9. Therefore, we opted for the Simple Gated Linear Units (SGLU) configuration in our model.

Table 9. **Ablations on GLU activation functions.** The exclusion of the activation function had no negative impact on the results.

GLU Act	Params	Updates	Loss	PPL
No Act	385M	100K	2.248	4.770
Swish	385M	100K	2.254	4.788

6. Conclusion

We introduced Lightning Attention, the first linear attention implementation that unleashed the full power of linear attention. As a result, our Lightning Attention can handle various sequence lengths with a constant speed under a constant memory footprint. The main concept is to divide the calculation of attention into intro-blocks and inter-blocks, while applying distinct computation techniques to perform the calculation. A new architecture, TNL, that is tailored for Lightning Attention is presented. TNL outperforms existing efficient language models in terms of both efficiency and accuracy and achieves competitive performance compared to state-of-the-art large language models using conventional transformer architectures.

Acknowledgement

This work is partially supported by the National Key R&D Program of China (NO.2022ZD0160100). We thank Songlin Yang for the helpful discussions.

Impact Statement

The introduction of Lightning Attention and its accompanying architecture TNL, heralds significant shifts in machine learning, particularly in language model efficiency and accessibility. By addressing the limitations of linear attention in varying sequence lengths without increasing memory consumption, this advancement democratizes access to state-of-the-art language models, potentially reducing the computational and environmental footprint of large-scale AI systems. Ethically, it underscores a move towards more sustainable AI practices, yet raises questions about the proliferation of powerful language models and their societal impacts, including concerns over privacy, misinformation, and the digital divide.

References

Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Goffinet, E., Heslow, D., Launay, J., Malartic, Q., et al. Falcon-40b: an open large language model with state-of-the-art performance. Technical report, Technical report, Technology Innovation Institute, 2023.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate, 2016.

Baichuan. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023. URL https://arxiv.org/abs/2309.10305.

Biderman, S., Schoelkopf, H., Anthony, Q., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., Skowron, A., Sutawika, L., and van der Wal, O. Pythia: A suite for analyzing large language models across training and scaling, 2023.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language, 2019.

Black, S., Biderman, S., Hallahan, E., Anthony, Q., Gao, L., Golding, L., He, H., Leahy, C., McDonell, K., Phang, J., et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018.

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

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022a.

Dao, T., Fu, D. Y., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. Hungry hungry hippos: Towards language modeling with state space models. *CoRR*, abs/2212.14052, 2022b. doi: 10.48550/arXiv.2212.14052. URL https://doi.org/10.48550/arXiv.2212.14052.

de Brébisson, A. and Vincent, P. A cheap linear attention mechanism with fast lookups and fixed-size representations, 2016.

Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., and Tang, J. Glm: General language model pretraining with autoregressive blank infilling, 2022.

Fu, D. Y., Epstein, E. L., Nguyen, E., Thomas, A. W., Zhang, M., Dao, T., Rudra, A., and Ré, C. Simple hardware-efficient long convolutions for sequence modeling. *CoRR*, abs/2302.06646, 2023. doi: 10.48550/arXiv.2302.06646. URL https://doi.org/10.48550/arXiv.2302.06646.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 2021.

Geng, X. and Liu, H. Openllama: An open reproduction of llama. *URL: https://github. com/openlm-research/open_llama*, 2023.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. Hippo: Recurrent memory with optimal polynomial projections, 2020.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022a.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022b. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Gu, A., Gupta, A., Goel, K., and Ré, C. On the parameterization and initialization of diagonal state space models, 2022c.

Gupta, A., Gu, A., and Berant, J. Diagonal state spaces are as effective as structured state spaces, 2022.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding, 2021.

Hua, W., Dai, Z., Liu, H., and Le, Q. V. Transformer quality in linear time. *arXiv preprint arXiv:2202.10447*, 2022.

Huang, Y., Bai, Y., Zhu, Z., Zhang, J., Zhang, J., Su, T., Liu, J., Lv, C., Zhang, Y., Lei, J., Fu, Y., Sun, M., and He, J. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7b, 2023.

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

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

Liu, H., Dai, Z., So, D., and Le, Q. V. Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34: 9204–9215, 2021.

Liu, Z., Li, D., Lu, K., Qin, Z., Sun, W., Xu, J., and Zhong, Y. Neural architecture search on efficient transformers and beyond. *arXiv preprint arXiv:2207.13955*, 2022.

Mehta, H., Gupta, A., Cutkosky, A., and Neyshabur, B. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences, 2023a.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gülçehre, Ç., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. *CoRR*, abs/2303.06349, 2023b. doi: 10.48550/arXiv.2303.06349. URL https://doi.org/10.48550/arXiv.2303.06349.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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. Rwkv: Reinventing rnns for the transformer era, 2023a.

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. Rwkv: Reinventing rnns for the transformer era, 2023b.

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., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., and Zhong, Y. The devil in linear transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 7025–7041, Abu Dhabi, United Arab Emirates, December 2022a. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.473.

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*, 2022b. URL https://openreview.net/forum?id=Bl8CQrx2Up4.

Qin, Z., Han, X., Sun, W., He, B., Li, D., Li, D., Dai, Y., Kong, L., and Zhong, Y. Toeplitz neural network for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=IxmWsm4xrua.

Qin, Z., Sun, W., Lu, K., Deng, H., Li, D., Han, X., Dai, Y., Kong, L., and Zhong, Y. Linearized relative positional encoding. *Transactions on Machine Learning Research*, 2023b.

Qin, Z., Yang, S., and Zhong, Y. Hierarchically gated recurrent neural network for sequence modeling. In *NeurIPS*, 2023c.

Qin, Z., Zhong, Y., and Deng, H. Exploring transformer extrapolation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions, 2017.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale, 2019.

Sap, M., Rashkin, H., Chen, D., LeBras, R., and Choi, Y. Socialiqa: Commonsense reasoning about social interactions, 2019.

Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., et al. Scrolls: Standardized comparison over long language sequences. *arXiv preprint arXiv:2201.03533*, 2022.

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

Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., and Zheng, C. Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning*, pp. 10183–10192. PMLR, 2021.

Team, M. N. et al. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. *URL www. mosaicml. com/blog/mpt-7b. Accessed*, pp. 05–05, 2023.

Tillet, P., Kung, H.-T., and Cox, D. D. Triton: an intermediate language and compiler for tiled neural network computations. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019.

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. *arXiv preprint arXiv:2302.13971*, 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.

Wang, B. and Komatsuzaki, A. Gpt-j-6b: A 6 billion parameter autoregressive language model, 2021.

Workshop, B., :, Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., Tow, J., Rush, A. M., Biderman, S., Webson, A., Ammanamanchi, P. S., Wang, T., Sagot, B., Muennighoff, N., del Moral, A. V., Ruwase, O., Bawden, R., Bekman, S., McMillan-Major, A., Beltagy, I., Nguyen, H., Saulnier, L., Tan, S., Suarez, P. O., Sanh, V., Laurençon, H., Jernite, Y., Launay, J., Mitchell, M., Raffel, C., Gokaslan, A., Simhi, A., Soroa, A., Aji, A. F., Alfassy, A., Rogers, A., Nitzav, A. K., Xu, C., Mou, C., Emezue, C., Klamm, C., Leong, C., van Strien, D., Adelani, D. I., Radev, D., Ponferrada, E. G., Levkovizh, E., Kim, E., Natan, E. B., Toni, F. D., Dupont, G., Kruszewski, G., Pistilli, G., Elsahar, H., Benyamina, H., Tran, H., Yu, I., Abdulmumin, I., Johnson, I., Gonzalez-Dios, I., de la Rosa, J., Chim, J., Dodge, J., Zhu, J., Chang, J., Frohberg, J., Tobing, J., Bhattacharjee, J., Almubarak, K., Chen, K., Lo, K., Werra, L. V., Weber, L., Phan, L., allal, L. B., Tanguy, L., Dey, M., Muñoz, M. R., Masoud, M., Grandury, M., Šaško, M., Huang, M., Coavoux, M., Singh, M., Jiang, M. T.-J., Vu, M. C., Jauhar, M. A., Ghaleb, M., Subramani, N., Kassner, N., Khamis, N., Nguyen, O., Espejel, O., de Gibert, O., Villegas, P., Henderson, P., Colombo, P., Amuok, P., Lhoest, Q., Harliman, R., Bommasani, R., López, R. L., Ribeiro, R., Osei, S., Pyysalo, S., Nagel, S., Bose, S., Muhammad, S. H., Sharma, S., Longpre, S., Nikpoor, S., Silberberg, S., Pai, S., Zink, S., Torrent, T. T., Schick, T., Thrush, T., Danchev, V., Nikoulina, V., Laippala, V., Lepercq, V., Prabhu, V., Alyafeai, Z., Talat, Z., Raja, A., Heinzerling, B., Si, C., Taşar, D. E., Salesky, E., Mielke, S. J., Lee, W. Y., Sharma, A., Santilli, A., Chaffin, A., Stiegler, A., Datta, D., Szczechla, E., Chhablani, G., Wang, H., Pandey, H., Strobelt, H., Fries, J. A., Rozen, J., Gao, L., Sutawika, L., Bari, M. S., Al-shaibani, M. S., Manica, M., Nayak, N., Teehan, R., Albanie, S., Shen, S., Ben-David, S., Bach, S. H., Kim, T., Bers, T., Fevry, T., Neeraj, T., Thakker, U., Raunak, V., Tang, X., Yong, Z.-X., Sun, Z., Brody, S., Uri, Y., Tojarieh, H., Roberts, A., Chung, H. W., Tae, J., Phang, J., Press, O., Li, C., Narayanan, D., Bourfoune, H., Casper, J., Rasley, J., Ryabinin, M., Mishra, M., Zhang, M., Shoeybi, M., Peyrounette, M., Patry, N., Tazi, N., Sanseviero, O., von Platen, P., Cornette, P., Lavallée, P. F., Lacroix, R., Rajbhandari, S., Gandhi, S., Smith, S., Requena, S., Patil, S., Dettmers, T., Baruwa, A., Singh, A., Cheveleva, A., Ligozat, A.-L., Subramonian, A., Névéol, A., Lovering, C., Garrette, D.,

Tunuguntla, D., Reiter, E., Taktasheva, E., Voloshina, E., Bogdanov, E., Winata, G. I., Schoelkopf, H., Kalo, J.-C., Novikova, J., Forde, J. Z., Clive, J., Kasai, J., Kawamura, K., Hazan, L., Carpuat, M., Clinciu, M., Kim, N., Cheng, N., Serikov, O., Antverg, O., van der Wal, O., Zhang, R., Zhang, R., Gehrmann, S., Mirkin, S., Pais, S., Shavrina, T., Scialom, T., Yun, T., Limisiewicz, T., Rieser, V., Protasov, V., Mikhailov, V., Pruksachatkun, Y., Belinkov, Y., Bamberger, Z., Kasner, Z., Rueda, A., Pestana, A., Feizpour, A., Khan, A., Faranak, A., Santos, A., Hevia, A., Unldreaj, A., Aghagol, A., Abdollahi, A., Tammour, A., HajiHosseini, A., Behroozi, B., Ajibade, B., Saxena, B., Ferrandis, C. M., McDuff, D., Contractor, D., Lansky, D., David, D., Kiela, D., Nguyen, D. A., Tan, E., Baylor, E., Ozoani, E., Mirza, F., Ononiwu, F., Rezanejad, H., Jones, H., Bhattacharya, I., Solaiman, I., Sedenko, I., Nejadgholi, I., Passmore, J., Seltzer, J., Sanz, J. B., Dutra, L., Samagaio, M., Elbadri, M., Mieskes, M., Gerchick, M., Akinlolu, M., McKenna, M., Qiu, M., Ghauri, M., Burynok, M., Abrar, N., Rajani, N., Elkott, N., Fahmy, N., Samuel, O., An, R., Kromann, R., Hao, R., Alizadeh, S., Shubber, S., Wang, S., Roy, S., Viguier, S., Le, T., Oyebade, T., Le, T., Yang, Y., Nguyen, Z., Kashyap, A. R., Palasciano, A., Callahan, A., Shukla, A., Miranda-Escalada, A., Singh, A., Beilharz, B., Wang, B., Brito, C., Zhou, C., Jain, C., Xu, C., Fourrier, C., Periñán, D. L., Molano, D., Yu, D., Manjavacas, E., Barth, F., Fuhrimann, F., Altay, G., Bayrak, G., Burns, G., Vrabec, H. U., Bello, I., Dash, I., Kang, J., Giorgi, J., Golde, J., Posada, J. D., Sivaraman, K. R., Bulchandani, L., Liu, L., Shinzato, L., de Bykhovetz, M. H., Takeuchi, M., Pàmies, M., Castillo, M. A., Nezhurina, M., Sänger, M., Samwald, M., Cullan, M., Weinberg, M., Wolf, M. D., Mihaljcic, M., Liu, M., Freidank, M., Kang, M., Seelam, N., Dahlberg, N., Broad, N. M., Muellner, N., Fung, P., Haller, P., Chandrasekhar, R., Eisenberg, R., Martin, R., Canalli, R., Su, R., Su, R., Cahyawijaya, S., Garda, S., Deshmukh, S. S., Mishra, S., Kiblawi, S., Ott, S., Sang-aroonsiri, S., Kumar, S., Schweter, S., Bharati, S., Laud, T., Gigant, T., Kainuma, T., Kusa, W., Labrak, Y., Bajaj, Y. S., Venkatraman, Y., Xu, Y., Xu, Y., Xu, Y., Tan, Z., Xie, Z., Ye, Z., Bras, M., Belkada, Y., and Wolf, T. Bloom: A 176b-parameter open-access multilingual language model, 2023.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence?, 2019.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. Opt: Open pre-trained transformer language models, 2022.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 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., Yuan, J., Wang, C., and Kong, L. Efficient attention via control variates. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=G-uNfHKrj46.

Appendix

A. Linear Attention with decay

TransNormerLLM uses LRPE-d positional encoding, which has the following format:

$$a_{ts} = \mathbf{q}_t^\top \mathbf{k}_s \lambda^{t-s} \exp^{i\theta(t-s)}. \tag{15}$$

According to (Qin et al., 2023b), Lrpe can be decomposed into $\mathbf{q}$ and $\mathbf{k}$, so we consider the following simplified form:

$$\begin{aligned} a_{ts} &= \mathbf{q}_t^\top \mathbf{k}_s \lambda^{t-s}, \\ \mathbf{o}_t^\top &= \sum_{s=1}^{t} a_{ts} \mathbf{v}_t^\top \\ &= \sum_{s=1}^{t} \mathbf{q}_t^\top \mathbf{k}_s \lambda^{t-s} \mathbf{v}_s^\top \\ &= \mathbf{q}_t^\top \sum_{s=1}^{t} \mathbf{k}_s \lambda^{t-s} \mathbf{v}_s^\top \\ &\triangleq \mathbf{q}_t^\top \overline{\mathbf{kv}}_t. \end{aligned} \tag{16}$$

We call this Linear Attention with decay and prove it's equivalent to the recurrence form:

$$\mathbf{kv}_0 = 0, \mathbf{kv}_t = \lambda \mathbf{kv}_{t-1} + \mathbf{k}_t \mathbf{v}_t^\top, \mathbf{o}_t^\top = \mathbf{q}_t^\top \mathbf{kv}_t. \tag{17}$$

We will use induction to prove $\overline{\mathbf{kv}}_t = \mathbf{kv}_t$.

Base Case ($n = 1$)**:**

$$\overline{\mathbf{kv}}_1 = \mathbf{k}_1 \mathbf{v}_1^\top = \mathbf{kv}_1. \tag{18}$$

Assume the statement holds for $n = m - 1$, i.e., $\overline{\mathbf{kv}}_{m-1} = \mathbf{kv}_{m-1}$. Then, when $n = m$:

$$\begin{aligned} \overline{\mathbf{kv}}_m &= \sum_{s=1}^{m} \mathbf{k}_s \lambda^{m-s} \mathbf{v}_s^\top \\ &= \lambda \sum_{s=1}^{m-1} \mathbf{k}_s \lambda^{m-1-s} \mathbf{v}_s^\top + \mathbf{k}_m \mathbf{v}_m^\top \\ &= \lambda \overline{\mathbf{kv}}_{m-1} + \mathbf{k}_m \mathbf{v}_m^\top \\ &= \lambda \mathbf{kv}_{m-1} + \mathbf{k}_m \mathbf{v}_m^\top \\ &= \mathbf{kv}_m, \end{aligned} \tag{19}$$

the statement holds. Therefore, by induction, the statement holds for all $n \geq 1$.

B. Lightning Attention with decay

We extended Lightning Attention to accommodate Linear Attention with decay. The complete algorithm can be found in Algorithm 5, 6, and the proof of correctness is provided in C.

C. Proofs

Here we discuss linear attention with decay directly, because vanilla linear attention is the case of $\lambda = 1$.

Algorithm 5 Lightning Attention(with decay) Forward Pass

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$, decay rate $\lambda \in \mathbb{R}^+$, block sizes B.
Divide $\mathbf{X}$ into $T = \frac{n}{B}$ blocks $\mathbf{X}_1, \mathbf{X}_2, ...\mathbf{X}_T$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}\}$.
Initialize mask $\mathbf{M} \in \mathbb{R}^{B \times B}$, where $\mathbf{M}_{ts} = \lambda^{t-s}$, if $t \geq s$, else 0.
Initialize $\Lambda = \text{diag}\{\lambda, \lambda^2, \ldots, \lambda^B\} \in \mathbb{R}^{B \times B}$.
Initialize $\mathbf{KV} = 0 \in \mathbb{R}^{d \times d}$.
for $t = 1, \ldots, T$ **do**
  Load $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{B \times d}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{O}_{\text{intra}} = [(\mathbf{Q}_t \mathbf{K}_t^\top) \odot \mathbf{M}]\mathbf{V}_t$.
  On chip, compute $\mathbf{O}_{\text{inter}} = \Lambda \mathbf{Q}_t (\mathbf{KV})$.
  On chip, compute $\mathbf{KV} = \lambda^B \mathbf{KV} + (\lambda^B \Lambda^{-1} \mathbf{K}_t)^\top \mathbf{V}_t$.
  Write $\mathbf{O}_t = \mathbf{O}_{\text{intra}} + \mathbf{O}_{\text{inter}}$ to HBM as the t-th block of $\mathbf{O}$.
end for
return $\mathbf{O}$.

C.0.1. Forward Pass

During forward pass of Linear attention with decay, the t-th output can be formulated as

$$\mathbf{o}_t^\top = \mathbf{q}_t^\top \sum_{s \leq t} \lambda^{t-s} \mathbf{k}_s \mathbf{v}_s^\top. \tag{20}$$

In a recursive form, the above equation can be rewritten as

$$\begin{aligned} \mathbf{kv}_0 &= 0 \in \mathbb{R}^{d \times d}, \\ \mathbf{kv}_t &= \lambda \mathbf{kv}_{t-1} + \mathbf{k}_t \mathbf{v}_t^\top, \\ \mathbf{o}_t^\top &= \mathbf{q}_t^\top (\mathbf{kv}_t), \end{aligned} \tag{21}$$

where

$$\mathbf{kv}_t = \sum_{s \leq t} \lambda^{t-s} \mathbf{k}_s \mathbf{v}_s^\top. \tag{22}$$

To perform tiling, let us write the equations in block form. Given the total sequence length n and block size B, $\mathbf{X}$ is divided into $T = \frac{n}{B}$ blocks $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T\}$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}\}$.

We first define

$$\mathbf{KV}_0 = \mathbf{0} \in \mathbb{R}^{d \times d}, \mathbf{KV}_t = \sum_{s \leq tB} \lambda^{tB-s} \mathbf{k}_s \mathbf{v}_s^\top. \tag{23}$$

Given $\mathbf{KV}_t$, the output of $(t+1)$-th block, i.e., $tB + r$, with $1 \leq r \leq B$ is

$$\begin{aligned} &\mathbf{o}_{tB+r}^\top \\ =&\mathbf{q}_{tB+r}^\top \sum_{s \leq tB+r} \lambda^{tB+r-s} \mathbf{k}_s \mathbf{v}_s^\top \\ =&\mathbf{q}_{tB+r}^\top \left(\sum_{s=tB+1}^{tB+r} \lambda^{tB+r-s} \mathbf{k}_s \mathbf{v}_s^\top + \lambda^r \sum_{s \leq tB} \lambda^{tB-s} \mathbf{k}_s \mathbf{v}_s^\top \right) \\ =&\mathbf{q}_{tB+r}^\top \sum_{s=tB+1}^{tB+r} \lambda^{tB+r-s} \mathbf{k}_s \mathbf{v}_s^\top + \lambda^r \mathbf{q}_{tB+r} \mathbf{kv}_{tB}^\top. \end{aligned} \tag{24}$$

Algorithm 6 Lightning Attention(with decay) Backward Pass

Input: $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{dO} \in \mathbb{R}^{n \times d}$, decay rate $\lambda \in \mathbb{R}^+$, block sizes B.
Divide $\mathbf{X}$ into $T = \frac{n}{B}$ blocks $\mathbf{X}_1, \mathbf{X}_2, ...\mathbf{X}_T$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$.
Divide $\mathbf{dX}$ into $T = \frac{n}{B}$ blocks $\mathbf{dX}_1, \mathbf{dX}_2, ...\mathbf{dX}_T$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}\}$.
Initialize mask $\mathbf{M} \in \mathbb{R}^{B \times B}$, where $\mathbf{M}_{ts} = \lambda^{t-s}$, if $t \geq s$, else 0.
Initialize $\Lambda = \mathrm{diag}\{\lambda, \lambda^2, \ldots, \lambda^B\} \in \mathbb{R}^{B \times B}$.
Initialize $\mathbf{KV} = 0, \mathbf{dKV} = 0 \in \mathbb{R}^{d \times d}$.
for $t = 1, \ldots, T$ **do**
  Load $\mathbf{K}_t, \mathbf{V}_t, \mathbf{O}_t, \mathbf{dO}_t \in \mathbb{R}^{B \times d}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{dQ}_{\text{intra}} = [(\mathbf{dO}_t \mathbf{V}_t^\top) \odot \mathbf{M}]\mathbf{K}_t$.
  On chip, compute $\mathbf{dQ}_{\text{inter}} = \Lambda \mathbf{dO}_t (\mathbf{KV})^\top$.
  On chip, compute $\mathbf{KV} = \lambda^B \mathbf{KV} + (\lambda^B \Lambda^{-1} \mathbf{K}_t)^\top \mathbf{V}_t$.
  Write $\mathbf{dQ}_t = \mathbf{dQ}_{\text{intra}} + \mathbf{dQ}_{\text{inter}}$ to HBM as the t-th block of $\mathbf{dQ}$.
end for
for $t = T, \ldots, 1$ **do**
  Load $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t, \mathbf{O}_t, \mathbf{dO}_t \in \mathbb{R}^{B \times d}$ from HBM to on-chip SRAM.
  On chip, compute $\mathbf{dK}_{\text{intra}} = [(\mathbf{dO}_t \mathbf{V}_t^\top) \odot \mathbf{M}]^\top \mathbf{Q}_t$.
  On chip, compute $\mathbf{dK}_{\text{inter}} = (\lambda^B \Lambda^{-1} \mathbf{V}_t)(\mathbf{dKV})^\top$.
  On chip, compute $\mathbf{dV}_{\text{intra}} = [(\mathbf{Q}_t \mathbf{K}_t^\top) \odot \mathbf{M}]^\top \mathbf{dO}_t$.
  On chip, compute $\mathbf{dV}_{\text{inter}} = (\lambda^B \Lambda^{-1} \mathbf{K}_t)\mathbf{dKV}$.
  On chip, compute $\mathbf{dKV} = \lambda^B \mathbf{dKV} + (\Lambda \mathbf{Q}_t)^\top \mathbf{dO}_t$.
  Write $\mathbf{dK}_t = \mathbf{K}_{\text{intra}} + \mathbf{K}_{\text{inter}}, \mathbf{dV}_t = \mathbf{V}_{\text{intra}} + \mathbf{V}_{\text{inter}}$ to HBM as the t-th block of $\mathbf{dK}, \mathbf{dV}$.
end for
return $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$.

Rewritten in matrix form, we have

$$\begin{aligned}\mathbf{O}_{t+1} =& \underbrace{[(\mathbf{Q}_{t+1}\mathbf{K}_{t+1}^\top) \odot \mathbf{M}]\mathbf{V}_{t+1}}_{\text{Intra Block}} \\ &+ \underbrace{\Lambda \mathbf{Q}_{t+1}(\mathbf{KV}_t)}_{\text{Inter Block}},\end{aligned} \tag{25}$$

where

$$\begin{aligned}\mathbf{M}_{ts} &= \begin{cases} \lambda^{t-s} & t \geq s \\ 0 & t < s \end{cases}, \\ \Lambda &= \mathrm{diag}\{1, \ldots, \lambda^{B-1}\}.\end{aligned} \tag{26}$$

And the $\mathbf{KV}$ at $(t+1)$-th block can be written as

$$\begin{aligned}\mathbf{KV}_{t+1} &= \sum_{s \leq (t+1)B} \lambda^{(t+1)B-s} \mathbf{k}_s^\top \mathbf{v}_s \\ &= \lambda^B \sum_{s \leq tB} \lambda^{tB-s} \mathbf{k}_s^\top \mathbf{v}_s + \sum_{s=tB+1}^{(t+1)B} \lambda^{(t+1)B-s} \mathbf{k}_s^\top \mathbf{v}_s \\ &= \lambda^B \mathbf{KV}_t + \left(\mathrm{diag}\{\lambda^{B-1}, \ldots, 1\}\mathbf{K}_t\right)^\top \mathbf{V}_t \\ &= \lambda^B \mathbf{KV}_t + \left(\lambda^B \Lambda^{-1} \mathbf{K}_t\right)^\top \mathbf{V}_t.\end{aligned} \tag{27}$$

The complete expression of the forward pass of Lightning Attention with decay can be found in Algorithm 5.

C.0.2. Backward Pass

For backward pass, let us consider the reverse process. First given $\mathbf{do}_t$, we have

$$\begin{aligned}\mathbf{dq}_t^\top &= \mathbf{do}_t^\top \mathbf{kv}_t^\top \in \mathbb{R}^{1 \times d}, \\ \mathbf{dk}_t^\top &= \mathbf{v}_t^\top \mathbf{dkv}_t^\top \in \mathbb{R}^{1 \times d}, \\ \mathbf{dv}_t^\top &= \mathbf{k}_t^\top \mathbf{dkv}_t \in \mathbb{R}^{1 \times d}, \\ \mathbf{dkv}_t &= \sum_{s \geq t} \lambda^{s-t} \mathbf{q}_s \mathbf{do}_s^\top \in \mathbb{R}^{d \times d}.\end{aligned} \tag{28}$$

By writing $\mathbf{dkv}_t$ in a recursive form, we get

$$\begin{aligned}\mathbf{dkv}_{n+1} &= 0 \in \mathbb{R}^{d \times d}, \\ \mathbf{dkv}_{t-1} &= \lambda \mathbf{dkv}_t + \mathbf{q}_{t-1}\mathbf{do}_{t-1}^\top.\end{aligned} \tag{29}$$

To facilitate the understanding of tiling, let us consider the above equations in block style. Given the total sequence length n and block size B, $\mathbf{X}$ is divided into $T = \frac{n}{B}$ blocks $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T\}$ of size $B \times d$ each, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dO}\}$.

We first define

$$\begin{aligned}\mathbf{dKV}_{T+1} &= \mathbf{0} \in \mathbb{R}^{d \times d}, \\ \mathbf{dKV}_t &= \sum_{s > tB} \lambda^{s-tB} \mathbf{q}_s \mathbf{do}_s^\top.\end{aligned} \tag{30}$$

Then for the $(t+1)$-th block, i.e., $tB + r, 0 \leq r < B$, we have

$$\begin{aligned}&\mathbf{dq}_{tB+r}^\top \\ =&\mathbf{do}_{tB+r}^\top \sum_{s \leq tB+r} \lambda^{tB+r-s} \mathbf{v}_s \mathbf{k}_s^\top \\ =&\mathbf{do}_{tB+r}^\top \left(\sum_{s=tB+1}^{tB+r} \lambda^{tB+r-s} \mathbf{v}_s \mathbf{k}_s^\top + \lambda^r \sum_{s \leq tB} \lambda^{tB-s} \mathbf{v}_s \mathbf{k}_s^\top \right) \\ =&\mathbf{do}_{tB+r}^\top \sum_{s=tB+1}^{tB+r} \lambda^{tB+r-s} \mathbf{v}_s \mathbf{k}_s^\top + \lambda^r \mathbf{do}_{tB+r} \mathbf{kv}_{tB}^\top.\end{aligned} \tag{31}$$

In matrix form, we have

$$\begin{aligned}\mathbf{dQ}_{t+1} =& \underbrace{[(\mathbf{dO}_{t+1}\mathbf{V}_{t+1}^\top) \odot \mathbf{M}]\mathbf{K}_{t+1}}_{\text{Intra Block}} \\ &+ \underbrace{\Lambda \mathbf{dO}_{t+1}(\mathbf{KV}_t^\top)}_{\text{Inter Block}}.\end{aligned} \tag{32}$$

Since the recursion of $\mathbf{dK}_t$ steps from $t+1$ to t, given $\mathbf{KV}_{t+1}, \mathbf{dK}_t$ for the t-th block, i.e., at positions $(t-1)B +$

$r, 0 < r \leq B$ is

$$\begin{aligned}
&\mathbf{dk}_{(t-1)B+r}^\top \\
=&\mathbf{v}_{(t-1)B+r}^\top \sum_{s\geq(t-1)B+r} \lambda^{s-(t-1)B-r}\mathbf{do}_s\mathbf{q}_s^\top \\
=&\mathbf{v}_{(t-1)B+r}^\top \left(\sum_{s=(t-1)B+r}^{tB} \lambda^{tB+r-s}\mathbf{do}_s\mathbf{q}_s^\top\right) \\
&+ \mathbf{v}_{(t-1)B+r}^\top \left(\lambda^{B-r}\sum_{s>tB} \lambda^{s-tB}\mathbf{do}_s\mathbf{q}_s^\top\right) \\
=&\mathbf{v}_{(t-1)B+r}^\top \sum_{s=(t-1)B+r}^{tB} \lambda^{tB+r-s}\mathbf{do}_s\mathbf{q}_s^\top \\
&+ \lambda^{B-r}\mathbf{v}_{(t-1)B+r}^\top \mathbf{dKV}_t^\top .
\end{aligned} \tag{33}$$

In matrix form, we get

$$\begin{aligned}
\mathbf{dK}_{t-1} =& \underbrace{[(\mathbf{dO}_{t-1}\mathbf{V}_{t-1}^\top) \odot \mathbf{M}]^\top \mathbf{Q}_{t-1}}_{\text{Intra Block}} \\
&+ \underbrace{\lambda^B \Lambda^{-1}\mathbf{V}_{t-1}(\mathbf{dKV}_t^\top)}_{\text{Inter Block}} .
\end{aligned} \tag{34}$$

Considering $\mathbf{dV}_t$ for the t-th block, i.e., at positions $(t-1)B + r, 0 < r \leq B$, we have

$$\begin{aligned}
&\mathbf{dv}_{(t-1)B+r}^\top \\
=&\mathbf{k}_{(t-1)B+r}^\top \sum_{s\geq(t-1)B+r} \lambda^{s-(t-1)B-r}\mathbf{q}_s\mathbf{do}_s^\top \\
=&\mathbf{k}_{(t-1)B+r}^\top \left(\sum_{s=(t-1)B+r}^{tB} \lambda^{tB+r-s}\mathbf{q}_s^\top\mathbf{do}_s\right) \\
&+ \mathbf{k}_{(t-1)B+r}^\top \left(\lambda^{B-r}\sum_{s>tB} \lambda^{s-tB}\mathbf{q}_s\mathbf{do}_s^\top\right) \\
=&\mathbf{k}_{(t-1)B+r}^\top \sum_{s=(t-1)B+r}^{tB} \lambda^{tB+r-s}\mathbf{q}_s\mathbf{do}_s^\top \\
&+ \lambda^{B-r}\mathbf{k}_{(t-1)B+r}^\top \mathbf{dKV}_t.
\end{aligned} \tag{35}$$

In matrix form, we get

$$\begin{aligned}
\mathbf{dV}_{t-1} =& \underbrace{[(\mathbf{Q}_{t-1}\mathbf{K}_{t-1}^\top) \odot \mathbf{M}]^\top \mathbf{dO}_t}_{\text{Intra Block}} \\
&+ \underbrace{\lambda^B \Lambda^{-1}\mathbf{K}_{t-1}(\mathbf{dKV}_t)}_{\text{Inter Block}} .
\end{aligned} \tag{36}$$

Finally, the recursive relation for $\mathbf{dKV}_t$ is

$$\begin{aligned}
\mathbf{dKV}_t =& \sum_{s>tB} \lambda^{s-tB}\mathbf{q}_s\mathbf{do}_s^\top \\
=& \lambda^B \sum_{s>(t+1)B} \lambda^{s-(t+1)B}\mathbf{q}_s\mathbf{do}_s^\top \\
&+ \sum_{s=tB+1}^{(t+1)B} \lambda^{s-tB}\mathbf{q}_s\mathbf{do}_s^\top \\
=& \lambda^B\mathbf{dKV}_{t+1} + (\Lambda\mathbf{Q}_t)^\top \mathbf{dO}_t.
\end{aligned} \tag{37}$$

Algorithm 6 describes the backward pass of Lightning Attention with decay in more detail.

D. Corpus

We gather an extensive corpus of publicly accessible text from the internet, totaling over 700TB in size. The collected data are processed by our data preprocessing procedure as shown in Fig. 6, leaving a 6TB cleaned corpus with roughly 2 trillion tokens. We categorize our data sources to provide better transparency and understanding. The specifics of these categories are outlined in Table 10.

D.1. Data Preprocessing

Our data preprocessing procedure consists of three steps: 1). rule-based filtering, 2). deduplication, and 3). a self-cleaning scheme. Before being added to the training corpus, the cleaned corpus needs to be evaluated by humans.

Rule-based filtering The rules we used to filter our collected data are listed as follows:

- *Removal of HTML Tags and URLs:* The initial step in our process is the elimination of HTML tags and web URLs from the text. This is achieved through regular expression techniques that identify these patterns and remove them, ensuring the language model focuses on meaningful textual content.
- *Elimination of Useless or Abnormal Strings:* Subsequently, the cleaned dataset undergoes a second layer of refinement where strings that do not provide value, such as aberrant strings or garbled text, are identified and excised. This process relies on predefined rules that categorize certain string patterns as non-contributing elements.
- *Deduplication of Punctuation Marks:* We address the problem of redundant punctuation marks in the data. Multiple consecutive punctuation marks can distort the natural flow and structure of sentences when training the model. We employ a rule-based system that trims these duplications down to a single instance of each punctuation mark.

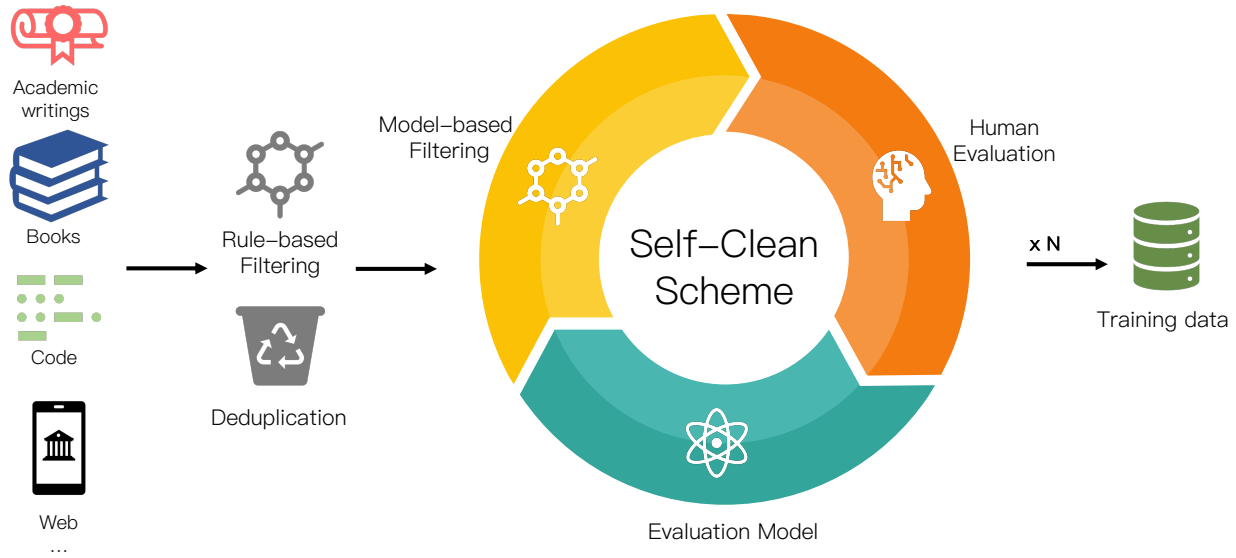

Figure 6. **Data Preprocess Procedure.** The collected data undergoes a process of rule-based filtering and deduplication, followed by our self-clean data processing strategy: model-based filtering, human evaluation, and evaluation model. After several iterations of the above cycle, we obtain high-quality training data at around 2T tokens.

- *Handling Special Characters:* Unusual or special characters that are not commonly part of the language's text corpus are identified and either removed or replaced with a standardized representation.
- *Number Standardization:* Numerical figures may be presented in various formats across different texts. These numbers are standardized into a common format to maintain consistency.
- *Preservation of Markdown/LaTeX Formats:* While removing non-textual elements, exceptions are made for texts in Markdown and LaTeX formats. Given their structured nature and ubiquitous use in academia and documentation, preserving these formats can enhance the model's ability to understand and generate similarly formatted text.

Deduplication To ensure the uniqueness of our data and avert the risk of overfitting, we employ an efficient deduplication strategy at the document or line level using MinHash and Locality-Sensitive Hashing (LSH) algorithms. This combination of MinHash and LSH ensures a balance between computational efficiency and accuracy in the deduplication process, providing a robust mechanism for data deduplication and text watermark removal.

Self-cleaning scheme Our data self-cleaning process involves an iterative loop of the following three steps to continuously refine and enhance the quality of our dataset. An issue of using model-based data filters is that the filtered data will have a similar distribution as the evaluation model, which may have a significant impact on the diversity of the training data. Assuming that the majority of the pre-processed data is of high quality, we can train an evaluation model on the entire set of pre-processed data, and the model will automatically smooth the data manifold distribution and outlet low-quality data while retaining the majority of the diversities.

The self-cleaning scheme unfolds as follows:

- *Evaluation Model:* We train a 385M model on the pre-processed corpus to act as a data quality filter.
- *Model-Based Data Filtering:* We use the evaluation model to assess each piece of data with perplexity. Only data achieving a score above a certain threshold is preserved for the next step. Low-quality data are weeded out at this stage.
- *Human Evaluation:* We sample a small portion of the filtered data and manually evaluate the quality.

These steps are repeated in cycles, with each iteration improving the overall quality of the data and ensuring the resulting model is trained on relevant, high-quality text. This self-cleaning process provides a robust mechanism for maintaining data integrity, thereby enhancing the performance of the resulting language model.

D.2. Tokenization

We tokenize the data with the Byte-Pair Encoding (BPE) algorithm. Notably, to enhance compatibility with Chinese

Table 10. **Statistics of our corpus.** For each category, we list the number of epochs performed on the subset when training on the 2 trillion tokens, as well as the number of tokens and disk sizes. We also list the table on the right according to the language distribution.

Dataset	Epochs	Tokens	Disk size
Academic Writings	1.53	200 B	672 GB
Books	2.49	198 B	723 GB
Code	0.44	689 B	1.4 TB
Encyclopedia	1.51	5 B	18 GB
Filtered Webpages	1.00	882 B	3.1 TB
Others	0.63	52 B	154 GB
Total	-	2026 B	6 TB

Language	Tokens	Disk size
English	743 B	2.9 TB
Chinese	555 B	1.7 TB
Code	689 B	1.4 TB
Others	39 B	89 GB
Total	2026 B	6 TB

language content, a significant number of common and uncommon Chinese characters have been incorporated into our vocabulary. In cases where vocabulary items are not present in the dictionary, the words are broken down into their constituent UTF-8 characters. This strategy ensures comprehensive coverage and flexibility for diverse linguistic input during model training.

E. Distributed System Optimization

We optimize our system to execute large-scale pre-training for TNL effectively. We employ fully sharded data parallelism (FSDP) (Zhao et al., 2023), activation checkpointing (Shoeybi et al., 2019), and automatic mixed precision (AMP) (Micikevicius et al., 2017) techniques to reduce memory footprint and expedite computational speed. We used BFloat16 (Kalamkar et al., 2019) to enhance training stability. We implemented model parallelism tailored to Lightning Attention. Inspired by Megatron-LM (Shoeybi et al., 2019) model parallelism, which independently addresses self-attention and MLP blocks, we apply model parallelism to SGLU and GLA separately. The details of our model parallelism strategies are elaborated below.

SGLU Model Parallelism Recall SGLU structure in (12):

$$\mathbf{O} = [(\mathbf{X}\mathbf{W}_v) \odot (\mathbf{X}\mathbf{W}_u)]\mathbf{W}_o, \tag{38}$$

The model parallelism adaptation of SGLU is as follows:

$$\begin{aligned}[\mathbf{O}_1', \mathbf{O}_2'] &= \mathbf{X}[\mathbf{W}_v^1, \mathbf{W}_v^2] \odot \mathbf{X}[\mathbf{W}_u^1, \mathbf{W}_u^2] \\ &= [\mathbf{X}\mathbf{W}_v^1, \mathbf{X}\mathbf{W}_v^2] \odot [\mathbf{X}\mathbf{W}_u^1, \mathbf{X}\mathbf{W}_u^2],\end{aligned} \tag{39}$$

which splits the weight matrices $\mathbf{W}_v$ and $\mathbf{W}_u$ along their columns and obtains an output matrix splitting along its columns too. Then the split output $[\mathbf{O}_1, \mathbf{O}_2]$ is multiplied by another matrix which is split along its rows as:

$$\mathbf{O} = [\mathbf{O}_1', \mathbf{O}_2'][\mathbf{W}_o^1, \mathbf{W}_o^2]^\top = \mathbf{O}_1'\mathbf{W}_o^1 + \mathbf{O}_2'\mathbf{W}_o^2 \tag{40}$$

Similar to model parallelism in Megatron-LM, this whole procedure splits three general matrix multiplies (GEMMs) inside the SGLU block across multiple GPUs and only introduces a single *all-reduce* collective communication operation in both the forward and backward passes, respectively.

GLA Model Parallelism Recall the GLA block in (11), its model parallelism version is:

$$[\mathbf{O_1}, \mathbf{O_2}] = \mathrm{SRMSNorm}(\mathbf{Q}\mathbf{K}^\top\mathbf{V}) \odot \mathbf{U}, \tag{41}$$

where:

$$\begin{aligned}&\mathbf{Q} = [\phi(\mathbf{X}\mathbf{W}_q^1), \phi(\mathbf{X}\mathbf{W}_q^2)], \mathbf{K} = [\phi(\mathbf{X}\mathbf{W}_q^1), \phi(\mathbf{X}\mathbf{W}_q^2)], \\ &\mathbf{V} = \mathbf{X}[\mathbf{W}_v^1, \mathbf{W}_v^2], \mathbf{U} = \mathbf{X}[\mathbf{W}_u^1, \mathbf{W}_u^2],\end{aligned} \tag{42}$$

Note that in our implementation, we use the combined QKVU projection to improve computation efficiency for linear attention. The obtained split output matrix $[\mathbf{O_1}, \mathbf{O_2}]$ again is multiplied by a weight matrix split along its columns which is similar to (40).

F. Additional TNL Ablation

Transformer *vs* TNL We carried out a meticulous series of comparative tests between our TNL and Transformer, spanning over an array of disparate sizes. The comparative performance of these models is clearly illustrated in Table 11. Under identical configurations, it becomes evident that our TNL exhibits a superior performance profile compared to Transformer. We observed that TNL outperformed Transformer by a remarkable 5% at the size of 385M. More importantly, as the size reached 1B, this superiority became even more pronounced, with an advantage of 9% for TNL over Transformer.

Table 11. **Transformer *vs* TNL.** TNL performs better than Transformer in size of 385M and 1B under identical configurations by 5% and 9%, respectively.

Method	Updates	Loss	PPL
Transformer-385M	100K	2.362	5.160
TNL-385M	100K	2.248	4.770
Transformer-1B	100K	2.061	4.765
TNL-1B	100K	1.896	3.729

Table 12. **TransNormer *vs* TNL.** TNL performs better than TransNormer.

Method	Params	Updates	Loss	PPL
TNL	385M	100K	2.248	4.770
TransNormer-T1	379M	100K	2.290	4.910
TransNormer-T2	379M	100K	2.274	4.858

We compare the original TransNormer and the improved TNL and the results are shown in Table 12. TNL exhibited an enhancement of 2% and 1% respectively.

Speed Normalization Fucntions We enhanced SRMSNorm using Triton, resulting in notable improvements in processing speed for larger dimensions, as shown in Fig. 7, outperforming conventional PyTorch implementations.

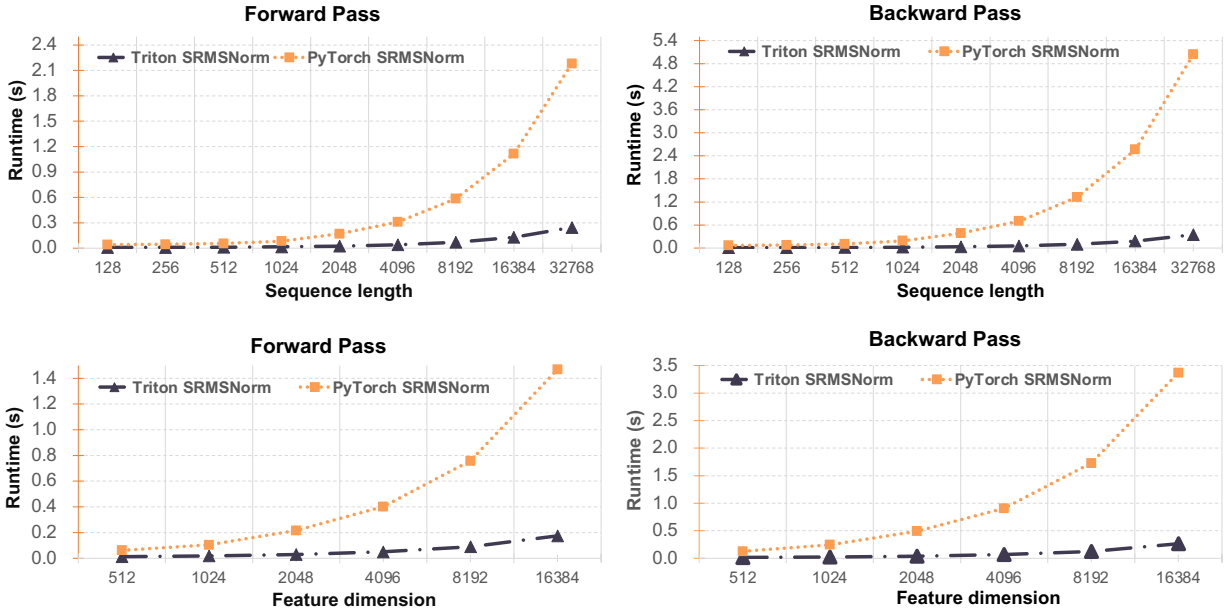

Figure 7. **Performance Evaluation of SRMSNorm Implementation.** The upper figures exhibit the runtime comparison of the forward pass (left section) and backward pass (right section) for different sequence lengths, with a fixed feature dimension of 3072. The lower two figures illustrate the runtime comparison for various feature dimensions, with a fixed sequence length of 4096.