Disentangle to Decay: Linear Attention with Trainable Positional Decay for Length Extrapolation

Anonymous ACL submission

Abstract

Transformer architecture has significantly advanced Natural Language Processing (NLP) by delivering outstanding performance. However, it faces challenges with efficiency and processing long sequences, attributed to its quadratic time complexity. Linear attention offers a more efficient linear time solution but falls short in language modelling and length extrapolation compared with traditional Transformer. To enhance the performance of linear attention and fully leverage its capability in modelling long sequences, we begin with positional encoding, specifying the constraints required for positional encoding by linear attention. Building upon these constraints, we design a positional encoding for linear attention, named **D**isentangle to **D**ecay (**D2D**), which allows for a seamless conversion between absolute positional encoding (APE) and relative positional encoding (RPE). To alleviate the instability of directly training D2D, we disentangle D2D into the combination of RPE and APE, which greatly improves the stability while ensuring the efficiency of model training. Experiments result shows that, application of D2D in linear attention significantly improves performance in language modelling and length extrapolation, demonstrating strong competitiveness with vanilla Transformer and outperforming other positional encodings. [1]

1 Introduction

In recent years, Transformer (Vaswani et al., 2017) has revolutionized the field of Natural Language Processing (NLP). Self-attention in Transformer shows notable abilities on processing sequences. However, a significant limitation of self-attention is the cost in terms of time and storage, which increases quadratically with sequence length. This constraint hinders the extensive application of Transformer to long sequences and enormous parameter sizes, especially in autoregressive natural language generation tasks.

To improve efficiency of Transformer, linear attention (Katharopoulos et al., 2020) replaces $softmax$ calculation with a dot-product of kernel feature maps. It shows recurrent property (Katharopoulos et al., 2020; Kasai et al., 2021; Yang et al., 2023), as it can be formulated into Recurrent Neural Networks (RNN) (Hochreiter and Schmidhuber, 1997) for inference.

However, linear attention suffers from cumulative regularity errors when processing long sequences (Schlag et al., 2021), necessitating the use of specialized mechanisms for information filtering. Positional encodings like ALiBi (Press et al., 2022) and RoPE (Su et al., 2024), integrate relative positional information, mitigate the issue of cumulative errors and extrapolate models to longer sequences (Sun et al., 2023a; Qin et al., 2024; Yang et al., 2023). But, due to structural issues with linear attention, most of these mainstream positional encodings cannot be directly applied. It necessitates certain adjustments to the model structure and fails to achieve satisfactory performance.

Our work proposes **D**isentangle to **D**ecay (**D2D**), an innovative decay factor based positional encoding for linear attention. D2D can freely transition between absolute positional encoding (APE) and relative positional encoding (RPE). During the training process, D2D can simultaneously leverage the advantages of RPE and APE, transforming them into a mixture of mask and trainable positional encoding, thereby enhancing the stability and computational efficiency of the training. In the inference phase, D2D can be converted into APE, thus fulfilling the requirement for transforming linear attention into RNN. Subsequently, D2D supports for further tuning of decay structures and achieves outstanding performance. Moreover, D2D is able for length extrapolation and compatible with

[1] Our code implementation is available at: https://anonymous.4open.science/r/D2D-0CF7/

linear attention in mathematical properties. It provides a solution for simple linear attention models without additional structure design and achieves good performance.

We conduct various experiments on language modelling, length extrapolation and efficient inference. Result shows that D2D enables linear attention to achieve comparable performance in language modelling with vanilla attention. And D2D outperforms existing positional encodings, including RoPE (Su et al., 2024) and ALiBi (Press et al., 2022). We provide an implementation of transformation to RNN and experiment on inference speed, which indicates that D2D is efficient.

The main contributions are as follows:

- We propose a positional encoding that incorporates both APE and RPE in the form of positional decay, and analyze the numerical instability during the decay process.
- By adopting a disentanglement approach, we introduce a new form that combines RPE and APE into D2D during the training process, significantly enhancing the stability of D2D, while still maintaining the efficiency of linear attention during both inference and training phases.
- We test the language modelling and length extrapolation capabilities of D2D on large datasets and explored the performance of various major kernel functions when used in conjunction with D2D.

2 Related Work

2.1 Positional Encoding In Transformer

The calculation of Transformer has been proven to be insensitive to the position of sequence tokens (Zhao et al., 2023). Encoding proper positional information is considered necessary for feature extraction of Transformer and efficient computing, especially for long sequences and large parameters (Kazemnejad et al., 2023). Approaches to integrate positional information into Transformer are known as positional encoding (PE).

PE can be categorized into two groups: APE and RPE. APE exploits absolute positional information, like trigonometric functions (Vaswani et al., 2017) and trainable parameters (Brown et al., 2020; Zhang et al., 2022). RPE exploits relative distance between tokens in the calculation, like RoPE (Su et al., 2024), ALiBi (Press et al., 2022). RPE is common for LLM (Raffel et al., 2020; Chowdhery et al., 2023; Scao et al., 2022)

2.2 Linear Attention

Linear attention mechanism adopts kernel functions to approximate softmax calculation with a simple dot product, aiming at desirably reducing the quadratic space-time complexity. It can be roughly categorized into **kernel-based** and **random-based**. Kernel-based linear attention (Choromanski et al., 2021; Katharopoulos et al., 2020; Qin et al., 2022b,a) processes query and key with kernel functions. Random-based linear attention (Peng et al., 2021; Choromanski et al., 2021) fits expected value through random sampling to approximate softmax.

Kernel-based linear attention can be transformed into recurrent form. (Katharopoulos et al., 2020) provides an approach to transform kernel-based linear attention to a form like RNN. (Yang et al., 2023; Sun et al., 2023a) apply both parallel and serial approaches to construct efficient large language models.

3 Background

3.1 Attention Calculation

Linear attention replaces softmax calculation in vanilla attention with dot product of query and key.[2] (Katharopoulos et al., 2020) provides Eq. 1, as a unified form of linear and vanilla attention.

Eq. 1 is constructed with **similarity calculation** $Sim(Q_i, K_j)$, which indicates similarity of i-th token of query and j-th token of key. For vanilla attention, $Sim(Q_i, K_j) = \exp(Q_i K_j^T)$. And for linear attention, similarity is dot product of query and key after **kernel function** [3] ϕ notated in Eq. 2. Transformations of kernel function should always be positive. Additionally, we provide background of vanilla attention in Appendix. A.1 for reference.

$$Att_{i,j} = \frac{Sim(Q_i, K_j)}{\sum_{j=1}^{i} Sim(Q_i, K_j)} \quad (1)$$

$$Sim(Q_i, K_j) = \phi(Q_i)\phi(K_j)^{\mathsf{T}} \quad (2)$$

3.2 Positional Encoding

Positional encoding integrates positional information for queries and keys.

[2] Random-based linear attention is special, and we exclude them for subsequent discussions.

[3] In the majority of works on linear attention, query and key share a common kernel function, but application of different kernel function for query and key is still legal in definition.

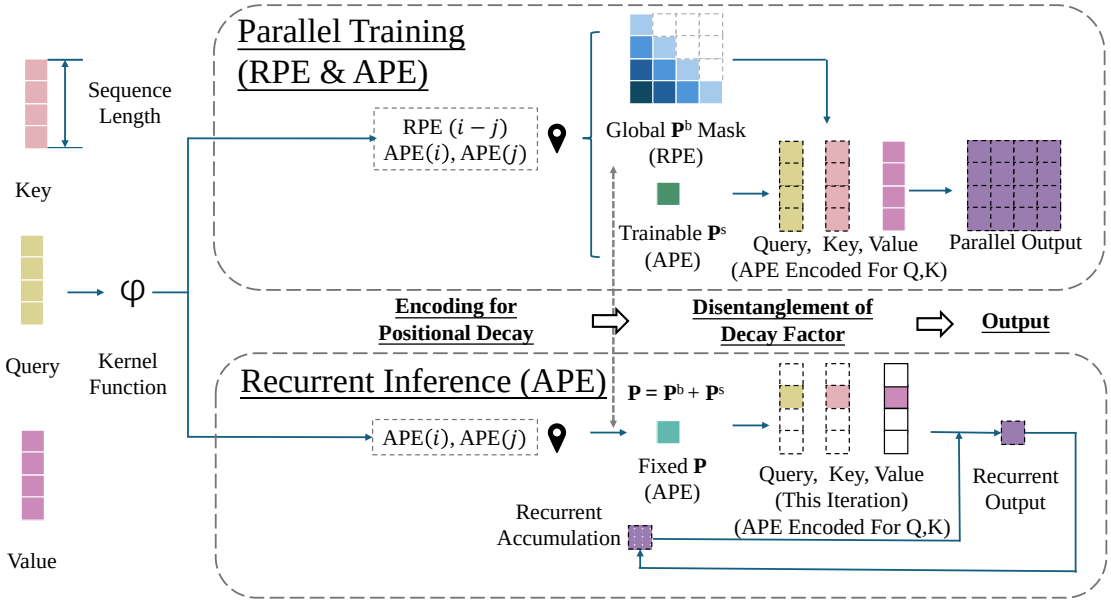

Figure 1: An overview of D2D for linear attention models. The Query (Q) and Key (K) is firstly transformed through kernel function. During the parallel training process, P is disentangled into two parts: $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$. $\boldsymbol{P^b}$ generates mask using RPE form of positional decay, while $\boldsymbol{P^s}$ is integrated into Q and K using APE form of positional decay. In the recurrent inference, $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$ merge into $\boldsymbol{P}$, which integrates positional decay with form of APE.

Absolute Positional Encoding For queries Q and keys K with positional information $\boldsymbol{a} = [1, 2, \ldots, n]$. APE can be represented as functions to add positional information to input sequences, notated as Eq. 3.

$$\tilde{Q} = APE(Q, \boldsymbol{a}), \tilde{K} = APE(K, \boldsymbol{a}) \quad (3)$$

Relative Positional Encoding RPE leverages the positional difference, $i - j$, between the i-th token in the query and the j-th token in the key. Consequently, the similarity calculation as depicted in Eq. 1 incorporates additional relative information, denoted as $g(i - j)$, in Eq. 4. Here, f signifies a novel function designed to integrate relative positional information into the similarity calculation, where common approaches typically involve either adding or multiplying $g(i - j)$ to incorporate RPE, as discussed in (Raffel et al., 2020; Press et al., 2022).

$$Sim(Q_i, K_j) = f(Q_i, K_j, g(i - j)) \quad (4)$$

From APE to RPE Some RPE integrates absolute position to query and key, which shares similar forms as APE in Eq. 3. To achieve this, positional information for tokens with identical relative position should only depend on query Q, key K and relative position $(i - j)$. That is, Eq. 5 holds for $\forall 1 \leq i, j \leq n$ (Su et al., 2024).

$$Sim(APE(Q, i), APE(K, j)) = h(Q, K, i - j) \quad (5)$$

4 Methodology

As demonstrated in Fig. 1, we propose **D2D** to integrate relative positional decay in linear attention. D2D exhibits form of both APE and RPE for positional decay, we exploit a mixture of APE and RPE for training, and use APE for recurrent inference. D2D addresses numeric instability of positional decay through disentanglement of decay factor. Moreover, it enables further tuning for positional encodings.

4.1 Encoding for Positional Decay

Relative position information works as essential information for Transformer calculation, especially for longer or changeable sequences lengths (Chi et al., 2022; Neishi and Yoshinaga, 2019). Positional decay structure like ALiBi (Press et al., 2022), provides an exponential decay item as shown in Eq. 6. The base of exponential is notated as **decay factors**. This enhances models to focus on adjacent tokens and diminish cumulative regularity errors. However, direct integration of RPE is incompatible for kernel-based attention.

$$Sim(Q_i, K_j) = \phi(Q_i)\phi(K_j)^\intercal b^{i-j} \quad (6)$$

To transform linear attention into RNN, similarity calculation can be decomposed into components solely dependent on i and on j. Eq. 7 gives a demonstration, where f_q and f_k can be any functions.

$$Sim(Q_i, K_j) = f_q(Q_i, i) \cdot f_k(K_j, j) \quad (7)$$

More details about transformation are list in Appendix. A.2. Our work designs D2D in form of both APE and RPE. Two forms of PE are identical in mathematical, and we select proper form with subsequent model design.

For APE form, we introduce a pair of positional functions to represent absolute positions respectively for query and key $\alpha, \beta : \mathbb{N}^+ \to \mathbb{R}^{1\times d_{model}}$, and a **decay factor** $\boldsymbol{P} \in \mathbb{R}^{1\times d_{model}}$ as base of positional decay, as Eq. 8 shows. d_{model} is the dimension of features for Transformer, known as *hidden size*. For further similarity calculation, $\tilde{Q}, \tilde{K}$ with absolute positional information is exploited.

$$\begin{gathered}\tilde{Q}_i = Q_i \exp(\alpha(i)), \tilde{K}_j = K_j \exp(\beta(j)) \\ \alpha(i) = -i\boldsymbol{P}, \beta(j) = j\boldsymbol{P}\end{gathered} \quad (8)$$

And Eq. 8 can be transformed into RPE form, as shown in Eq. 9. In this context, $\boldsymbol{P}_d$ indicates a scalar for d-dimension of vector $\boldsymbol{P}$. And Σ indicates a serial summation. Unfortunately, Eq. 9 is not parallel parallelizable. We provide solutions in Sec. 4.3.

$$\begin{aligned}Sim(Q_i, K_j) &= \phi(\tilde{Q}_i)\phi(\tilde{K}_j)^\intercal \\ &= \sum_{d=0}^{d_{model}-1} \frac{\phi(Q_{i,d})\phi(K_{j,d})^\intercal}{\exp(\boldsymbol{P}_d)^{i-j}}\end{aligned} \quad (9)$$

4.2 Instability In Decay Factor

Similar positional decay structures in Sec. 4.1 appear in previous works (Sun et al., 2023b; Press et al., 2022; Sun et al., 2023a). However, the decay factors in these approaches are typically fixed hyperparameters. Since such exponential calculation with relative position is unstable in numeric. Reasons include large gradients and extreme value for exponential calculation. It prevents models from further training and learn more positional information. Moreover, decay structures will be more unstable in long sequences because range of $i - j$ is larger.

Positional decay will cause inconsistency of APE and RPE, which limits models transformation to RNN in inference. Taking the case in $i = j, i \to +\infty$ as an example, $\exp(-\boldsymbol{P}_d)^{i-j}$ in Eq. 9 should be 1. However, result of $\tilde{Q}_i$ and $\tilde{K}_j$ approaches $+\infty$ and 0 respectively, which leads to unexpected numerics in actual computations. Practically, directly training of decay factor encounters issue of numerical underflow. Fig. 2 demonstrates the value of $\boldsymbol{P}$ under direct training. $\boldsymbol{P}$ experiences truncation at approximately 0.12, inhibiting any further tuning. Normalization methods are

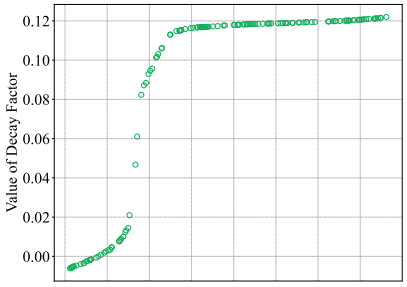

Figure 2: The distribution of decay factor in direct training (sorted). It is observed that most values truncated around 0.12 without further tuning.

not effective for this instability, because of the following reasons: (1) In order to transform linear attention to RNN, normalization based on data distribution is disabled because it violates calculations orders and dependencies in Sec. 4.1, for example, batch-normalization (Ioffe and Szegedy, 2015). (2) Scaling methods like L1-normalization will limit data into a thinner range as a result of stricter stability requirements. This will cause performance drop. (3) Clamping decay factor $\boldsymbol{P}$ into a certain range can limit range of $i\boldsymbol{P}$, but this brings little performance enhancement since ranges for tuning is too thin.

4.3 Disentanglement of Decay Factor

Sec. 4.1 proposes RPE and APE form of D2D. However, it is not available in actual computation. Since decay factor does not support for direct tuning due to numeric instability in Sec. 4.2. Moreover, as shown in Eq. 9 RPE form of D2D exhibits a serial summation, and this is not acceptable for parallel training. In this section, we propose a novel approach to disentangle decay factor $\boldsymbol{P}$. We will introduce methods and calculation implementation of disentanglement for decay factor.

Disentanglement To address numeric instability and achieve parallelism, we provide a disentanglement solution as shown in Fig. 3. And we aim to demonstrate ideal decay factor distributions

through disentangled $\boldsymbol{P}$. We decompose $\boldsymbol{P}$ into h attention heads, as $\boldsymbol{P} = concat(\boldsymbol{P}_1, \ldots, \boldsymbol{P}_h)$, where $concat$ stands for vector concentration (Headwise split is based on multi head attention (Vaswani et al., 2017), more backgrounds can be found in Appendix. A.1.) For l-th attention head, we disentangle $\boldsymbol{P}_l \in \mathbb{R}^{1\times d_h}$ into $\boldsymbol{P}_l = \boldsymbol{P}_l^b + \boldsymbol{P}_l^s$. Here, $\boldsymbol{P}_l^b \in \mathbb{R}^{1\times d_h} = [P_l^b, \ldots, P_l^b]$ consists of a scalar $P_l^b = 2^{-\frac{h}{l}}$, delineating a rough range for $\boldsymbol{P}_l^b$. And $\boldsymbol{P}_l^s \in \mathbb{R}^{1\times d_h}$ is a trainable vector of small numerical values, to enable further tuning.And Eq. 10 shows calculation after disentanglement, where $Sim(Q_i, K_j)[l]$ stands for similarity calculation of l-th attention head.

As depicted in Fig. 3, $\boldsymbol{P}^b$ constitutes the major component of $\boldsymbol{P}$. Compared to $\boldsymbol{P}^s$, $\boldsymbol{P}^b$ has larger numerical values and a broader range. The values of this part are fixed during training. By setting the magnitude of $\boldsymbol{P}^b$, the numerical range of $\boldsymbol{P}$ can be limited. And the numerical range of $\boldsymbol{P}^s$ is significantly smaller than that of $\boldsymbol{P}$ itself, which endows D2D training with enhanced stability and avoids the truncation phenomenon depicted in Fig. 2.

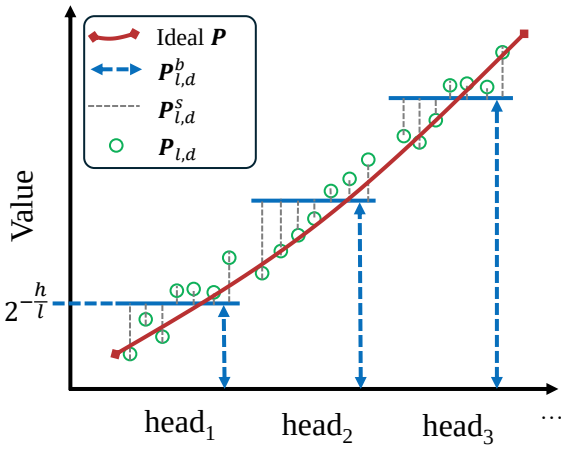

Figure 3: Illustration of disentanglement. Green circle stands for each index of $\boldsymbol{P}$ is sum of fixed $\boldsymbol{P^b}$ and trainable $\boldsymbol{P^s}$, aiming to fit ideal distribution of $\boldsymbol{P}$ (board line with squares).

$$
\begin{aligned}
Sim(Q_i, K_j)[l] &= \Theta_b \times \Theta_s \\
\Theta_b &= \exp\left(-P_l^b\right)^{i-j} \\
\Theta_s &= \frac{\phi(Q_i)}{\exp(i\boldsymbol{P_l^s})}\left(\frac{\phi(K_j)}{\exp(-j\boldsymbol{P_l^s})}\right)^{\mathsf{T}}
\end{aligned}
\tag{10}
$$

Calculation Implementation Eq. 10 enables D2D to be compatible with efficient parallel calculation implementation. $Sim(Q_i, K_j)[l]$ is decomposed into two parts. The first part is Θ_b, which

1	0	0	0
P	1	0	0
P^2	P	1	0
P^3	P^2	P	1

Figure 4: An instance of decay mask (length $n = 4$). The element in the i-th row and j-th column of the matrix corresponds $M_{i,j}$. The part where $j > i$ is assigned a value of 0 due to the presence of the causal mask, to ensure attention is unidirectional in autoregressive language modelling tasks.

is same for all computations within the head. So it can be calculated once and applied in all subsequent calculation, with a mask matrix M like Fig. 4. And this mask can be integrated into attention score along with causal mask in vanilla attention. This is implemented by element-wise product of matrices, which is efficient in calculation. The second part is Θ_s, which varies for each dimension within the head. The $\boldsymbol{P^s}$ in Θ_s are similar to $\boldsymbol{P}$ in Eq. 8, directly acting on Q and K in the form of APE, and are ultimately expressed in the form of matrix multiplication. This enables the computation of Θ_s to fully leverage the GPU's acceleration for matrix operations, ensuring efficiency during training.

Algorithm. 1 and Algorithm. 2 demonstrate algorithmic process of linear attention when employing D2D. **Par** refers to the process of model parallel training, while **Rec** pertains to the process of serial model inference. For input, $Q, K, V, M, \boldsymbol{P}^s, n$ remain the same meaning as former part of paper. $\mathbf{0}_{d_h\times d_h}$ indicates a zero matrix with size $d_h \times d_h$. For calculation, `splithead` function represents the operation of splitting the query, key, and value into multiple heads, which is used for the multi-head attention mechanism. `mergehead` function represents the operation of merging heads, combining the output results of the multi-head attention mechanism into a single head for subsequent calculations. S, Z are intermediate variables generated in recurrent computation, details can be found in Appendix. A.2. $\div$ stands for element-wise division for matrices, while $\odot$ stands for element-wise product for matrices.

5 Experiments and Analysis

In this section, we apply our design of PE and linear attention into vanilla Transformer for language

Algorithm 1 Parallel Training

procedure PAR($Q, K, V, M, \boldsymbol{P}^s, n$)
  $K \leftarrow K^\intercal$
  $Q, K \leftarrow \phi(Q), \phi(K)$
  $\boldsymbol{a} \leftarrow [0, 1, \ldots, n-1]$
  $C \leftarrow \exp(\boldsymbol{a} \cdot \boldsymbol{P}^s)$
  $Q \leftarrow Q \div C$
  $K \leftarrow K \odot C$
  $Q, K, V \leftarrow$ splithead (Q, K, V)
  $Att \leftarrow Q \cdot K \odot M$
  for $i \leftarrow 0$, **to** $n-1$ **do**
    $Att_i \leftarrow Att_i / \sum_{j=0}^{n-1} (Att_{i,j})$
  end for
  $O \leftarrow Att \cdot V$
  $O \leftarrow$ mergehead (O)
  return O
end procedure

Algorithm 2 Recurrent Inference

procedure REC($Q, K, V, \boldsymbol{P}^b, \boldsymbol{P}^s, n$)
  $K \leftarrow K^\intercal$
  $\boldsymbol{P} \leftarrow \boldsymbol{P}^b + \boldsymbol{P}^s$
  $\boldsymbol{P} \leftarrow \exp(\boldsymbol{P})$
  $\boldsymbol{S}, \boldsymbol{Z} \leftarrow \mathbf{0}_{d_h \times d_h}, \mathbf{0}_{d_h \times 1}$
  $Q, K, V \leftarrow$ splithead (Q, K, V)
  for $i \leftarrow 0$ **to** $n-1$ **do**
    $Q_i, K_i \leftarrow \phi(Q_i), \phi(K_i)$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} \odot \boldsymbol{P} + K_i \cdot V_i$
    $\boldsymbol{Z} \leftarrow \boldsymbol{Z} \odot \boldsymbol{P} + K_i$
    $O_i \leftarrow (Q_i \cdot \boldsymbol{S}) / (Q_i \cdot \boldsymbol{Z})$
  end for
  $O \leftarrow concat(O_1, \ldots, O_n)$
  $O \leftarrow$ mergehead (O)
  return O
end procedure

modelling. We contrast our methods with mainstream PE on linear attention and vanilla Transformer. Additionally, we conduct length extrapolation experiments for these models. Experiments demonstrate that our designed position embedding offers a significant advantage for linear attention on autoregressive language models. Moreover, we provide an implementation to transform linear attention based on D2D to RNN and compare its inference speed with vanilla GPT in Appendix. A.6. Experiment show that our model is efficient when sequence length grows.

5.1 Experiment Settings

Models Implementation We implement all of our models in the *Huggingface* framework [4]. We select GPT-2 [5] (Brown et al., 2020) for backbone of auto-regressive language models. All models are trained and validated on 4xNVIDIA V100 GPUs. More details are record in Appendix A.3

Datasets For language modelling, we select OpenWebText (Gokaslan and Cohen, 2019) dataset for experiments. OpenWebText corpus is sourced from Reddit-linked URLs, covering a wide range of topics from news to blogs. It's designed for training large-scale language models with diverse internet text. For efficiency and limitation on computations, we sample part of OpenWebText for experiments. Column of texts for experiments are described in the following Table 1.

Train	13,288,030
Valid	1,662,260

Table 1: Statistics of columns for datasets in experiments.

Baselines We select linear attention with ALiBi (Press et al., 2022), RoPE (Su et al., 2024), Positional Embedding in vanilla Transformer (Vaswani et al., 2017). Additionally, we select vanilla Transformer with softmax attention for contrastive concerns. Details about initialization and calculation for PE can be found in Appendix A.4.

Kernel Selection For linear attention, there is no one-size-fits-all criterion for selecting the kernel function. Thus, we select some commonly used kernel functions and test them for experiments.

Exponential (EXP) Kernel : $\phi(x) = \exp(x)$.

ELU kernel: $\phi(x) = elu(x) + 1$. elu is ELU function in (Clevert et al., 2016).

Metrics For the language model task, we use perplexity (PPL) (Brown et al., 1993) to measure performance. Lower PPL for language modelling reflects better language modelling ability.

5.2 Language Modelling

Language modelling capabilities for various baselines are demonstrated in Table. 2. Firstly, D2D outperforms other PE and vanilla attention in language modelling task. Compared to ALiBi and

[4] https://huggingface.co/
[5] https://huggingface.co/openai-community/gpt2

PE	Kernels	PPL(Train)	PPL(Valid)
Vanillia APE	EXP	49.30	50.75
	ELU	49.40	50.86
RoPE	EXP	44.66	47.85
	ELU	44.59	47.80
ALiBi	EXP	45.24	48.18
	ELU	44.88	47.85
D2D	EXP	44.70	47.80
	ELU	<u>43.82</u>	<u>46.90</u>
Attention w/o Linear		45.74	47.66

Table 2: Perplexity of language modelling tasks, lower ppl shows better performance. Values underlined are denoted as optimal results.

PE	Kernel	PPL
ALiBi	EXP	49.80
	ELU	49.37
RoPE	EXP	50.81
	ELU	51.25
D2D	EXP	49.23
	ELU	<u>48.54</u>

Table 3: Result on length extrapolation tests. Values underlined are denoted as optimal results.

RoPE, D2D provides effective improvement for linear attention. We also discover that the impact of our designed position embedding varies with different types of kernel functions. Language modelling performance varies with kernel functions, and we provide more detailed experiments to discuss this in Appendix. A.7. Additionally, linear attention models with extra PE shows a wider gap between performance on training and valid sets than vanilla GPT. Linear attention still has limitations on generalization. In Appendix A.5,we provide further discussion with PPL during training process.

5.3 Length Extrapolation

Models based on linear attention exhibit substantial competitiveness in generating long texts. It is imperative that we ensure the position encoding we design possesses adequate length extrapolation capability to fully leverage the advantages of linear attention.

"Train short and test long" is the key for length extrapolation test. More consciously, we train models with normal length in datasets (512). Then, we obtain longer texts by repeating and concatenating the same text 4 times.

The following table describes perplexity of repeating texts. Vanilla APE in GPT-2 is not able for length extrapolation, so that we do not mention it in the experiments. As shown in Table. 3, our position encoding demonstrates superior length extrapolation capability than other RPE. Compared to ALiBi, our position encoding strategy introduces more trainable parameters, offering stronger representational power. This enables the model to fit more appropriate position encoding during training. Compared to RoPE, the constant positivity and smoothness of our position encoding are the main reasons for its better performance. Although RoPE exhibits decay for larger relative positions, the oscillatory circumstances of trigonometric functions can lead to significant numerical jitter during this decay. Additionally, the introduction of RoPE cannot guarantee that the values of $Sim(Q, K)$ are positive, and normalization is weakened for this reason in Eq. 14. We believe this affects the generalization ability of the position encoding on long texts and the normalization during training for linear attention.

5.4 Ablation Study

In designing D2D, we experiment with various methods. Table. 4 presents the performance of these methods in language modelling, with evaluation criteria consistent with Section 5.1.

PE	Kernel	PPL(Train)	PPL(Valid)
Vanilla APE	EXP	49.30	50.75
	ELU	49.40	50.86
D2D	ELU	<u>43.82</u>	<u>46.90</u>
	EXP	44.70	47.80
D2D w/o $\boldsymbol{P^b}$	EXP	45.24	48.33
D2D w/o $\boldsymbol{P^s}$	EXP	45.24	48.23
D2D w/ Vanilla APE	EXP	45.51	48. 57

Table 4: Results for ablation study. w/o means without and w/ means with. Values underlined are denoted as optimal results.

Kernel We explore different choices for the kernel function in Eq. 9, primarily testing $\phi(x) = \exp(x)$ and $\phi(x) = elu(x) + 1$ as the kernel functions. These two are selected as representatives mainly due to their significant differences in mathematical properties (detailed discussion can be

found in Appendix. A.7), making them suitable for our tests. Table. 4 shows that, without special positional encoding, there is no significant difference in language modelling capability between two kernel functions. However, after applying D2D, the model using $elu(x)+1$ demonstrates significantly better language modelling performance than the one using $\exp(x)$. We believe this is because D2D carries more prior information, encouraging the model to focus on several adjacent tokens. This leads to the $\exp(x)$, which has better focus ability (Han et al., 2023), having more difficulty in fully capturing the dataset's information during training. This will result in a decrease in final language modelling capability. Appendix. A.8 provides a feasible approach to measure the focus ability of kernel functions.

Disentangled components of decay factor Decay factor $\boldsymbol{P}$ primarily consists of two components [6]: $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$. As shown in Table. 4, when used individually, both components demonstrate similar language modelling capabilities. However, using $\boldsymbol{P^b}$ alone achieves better performance on the validation set, indicating that $\boldsymbol{P^b}$ endows the model with stronger generalization performance. When both components are utilized together, the model's language modelling capability significantly improves. We believe that $\boldsymbol{P^b}$ exhibits stronger prior information during training, encouraging the model to pay more attention to adjacent token parts, thus enhancing the model's generalization ability in long sequence modelling. On the other hand, $\boldsymbol{P^s}$ provides stronger representational capacity on this foundation, leading to an overall enhancement in the model's language modelling capability.

Moreover, we demonstrate outcomes of $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$ in the first layer of the model after full training. We extract $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$ from the trained model, sum them, and then sort the combined values for a clearer presentation of the training results as shown in Fig. 5. The figure illustrates that disentanglement is effective for information representative.

Absolute Positional Encoding In constructing the model, we also consider adding a trainable absolute positional encoding at the input, similar to GPT2-small. However, as shown in Table. 4, models augmented with absolute positional encoding demonstrate inferior performance in language modelling capability.

[6]For simplicity, we concatenate headwise vectors into a whole one.

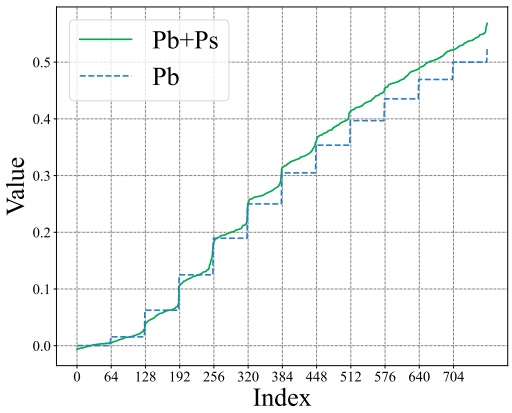

Figure 5: Outcomes of $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$ in the first layer. The dashed line represents $\boldsymbol{P^b}$, and the solid line indicates the sum of $\boldsymbol{P^b}$ and $\boldsymbol{P^s}$.

6 Conclusion

In this paper, we design a positional encoding, D2D, for models based on linear attention. By analyzing the conditions under which linear attention can be transformed into RNN, we ascertain that D2D needs to facilitate the conversion between absolute and relative positional encoding. Leveraging this characteristic, we disentangle D2D during the training process, transforming it into a combination of APE and RPE to enhance training stability. In the inference process, we fully convert D2D into APE, enabling the transformation of linear attention into an RNN form. This fully leverages the advantages of linear attention in terms of time complexity and space complexity during the inference process. Models utilizing D2D linear attention have demonstrated commendable performance in language modelling and length extrapolation.

7 Limitation

Our positional encoding demonstrates effectiveness across various kernel functions, though the extent of the effect is somewhat dependent on the choice of kernel function. Based on our experiments, we find that $elu(x)+1$ is a good choice for the kernel function, but we cannot provide a very systematic theoretical explanation for this choice. Additionally, the initialization of $\boldsymbol{P^b}$ significantly impacts the stability of training and the final results. Manually adjusting $\boldsymbol{P^b}$ is quite labor-intensive, which is not conducive to the rapid and simple application of our positional encoding in other models. Moreover, due to computational resource limitations, we

are unable to verify whether D2D achieves better performance in models with a larger number of parameters.

References

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ta-Chung Chi, Ting-Han Fan, Peter J. Ramadge, and Alexander Rudnicky. 2022. KERPLE: kernelized relative positional embedding for length extrapolation. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`.

Dongchen Han, Xuran Pan, Yizeng Han, Shiji Song, and Gao Huang. 2023. Flatten transformer: Vision transformer using focused linear attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5961–5971.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org.

Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A. Smith. 2021. Finetuning pretrained transformers into RNNs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10630–10643, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan, Payel Das, and Siva Reddy. 2023. The impact of positional encoding on length generalization in transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Masato Neishi and Naoki Yoshinaga. 2019. On the relation between position information and sentence length in neural machine translation. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 328–338, Hong Kong, China. Association for Computational Linguistics.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. 2021. Random feature attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Ofir Press, Noah A. Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. 2022a. The devil in linear transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7025–7041, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, and Yiran Zhong. 2024. Transnormerllm: A faster and better large language model with improved transnormer.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. 2022b. cosformer: Rethinking softmax in attention. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733. Version 2.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9355–9366. PMLR.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023a. Retentive network: A successor to transformer for large language models. *CoRR*, abs/2307.08621.

Yutao Sun, Li Dong, Barun Patra, Shuming Ma, Shaohan Huang, Alon Benhaim, Vishrav Chaudhary, Xia Song, and Furu Wei. 2023b. A length-extrapolatable transformer. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14590–14604, Toronto, Canada. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2023. Gated linear attention transformers with hardware-efficient training. *CoRR*, abs/2312.06635.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.

Liang Zhao, Xiaocheng Feng, Xiachong Feng, Bing Qin, and Ting Liu. 2023. Length extrapolation of transformers: A survey from the perspective of position encoding. *CoRR*, abs/2312.17044.

A Appendix

A.1 Notations Of Vanilla Transformer

In the transformer architecture, X is transformed into three distinct sequences, namely **query** (Q), **key** (K), and **value** (V), through separate linear projections. This projection is split into h attention heads, known as **Multi Head Attention**. As shown in Eq. 11, l-th head transform Q, K, V into d_h dimension, obtaining Q_l, K_l, V_l.

$$\begin{aligned} Q_l &= QW_l^Q \\ K_l &= KW_l^K \\ V_l &= VW_l^V \\ W_l^Q, W_l^K, W_l^V &\in \mathbb{R}^{d_{model} \times d_h} \end{aligned} \tag{11}$$

Attention calculation is defined as Eq. 12, where Att is known as attention score.

$$\begin{aligned} Att &= \text{softmax}\left(\frac{QK^\intercal}{\sqrt{d_h}}\right) \\ Attention(Q, K, V) &= Att \cdot V \end{aligned} \tag{12}$$

And final output of attention needs to concatenate (notated as $concat$ in equations) each head and apply a linear projection.

$$\begin{aligned} &MultiHead(Q, K, V) \\ &= concat(head_1, \ldots, head_h)W_O, \\ &head_l = Attention(QW_l^Q, KW_l^K, VW_l^V) \\ &W_O \in \mathbb{R}^{d_{model} \times d_{model}} \end{aligned} \tag{13}$$

A.2 Conversion of Kernel-Based Linear Attention to RNN

The process of converting kernel-based linear attention to an RNN framework hinges on the ability to decompose the similarity calculation into independent functions of queries and keys. Here, we delve into the mathematical underpinnings of this conversion, starting with the general form of linear attention:

$$Att_{i,j} = \frac{\phi(Q_i)\phi(K_j)^\intercal}{\sum_{j=1}^{i} \phi(Q_i)\phi(K_j)^\intercal}$$

The computation of the updated representation V_i' involves weighting by the attention scores:

$$V_i' = \frac{\sum_{j=1}^{i} \phi(Q_i)\phi(K_j)^\intercal V_j}{\sum_{j=1}^{i} \phi(Q_i)\phi(K_j)^\intercal}$$

Parameter	Value
Number of Layers	12
Attention Heads	12 per layer
Hidden Dimension	64 per attention head
Batch Size	640
Training Text Length	512 tokens
Learning Rate	5e-4
Learning Rate Schedule	Cosine
Warmup Rounds	1000
Epochs	1
Gradient Optimizer	Adam (Kingma and Ba, 2015)
Total Parameters	137M

Table 5: Training Configuration and Model Parameters

This equation can be simplified by recognizing that $\phi(Q_i)$ can be factored out, leading to a recursive form that mirrors RNN computations:

$$V_i' = \frac{\phi(Q_i)(S_{i-1} + \phi(K_i)^\intercal V_i)}{\phi(Q_i)(Z_{i-1} + \phi(K_i)^\intercal)}$$

with S_{i-1} and Z_{i-1} representing cumulative sums over j up to $i-1$, allowing for an RNN-like iterative update mechanism. This section will provide a detailed walkthrough of the derivation of these equations, underlining the critical role of the absence of cross terms in enabling the RNN conversion. We will illustrate through examples and further algebraic manipulation how this condition facilitates the transformation, ensuring the feasibility of maintaining cumulative variables akin to S and Z for sequential processing.

A.3 Implementation Details of Experiments

The specific model parameters and training settings are presented in Table. 5.

A.4 Calculation and Initialization of Positional Encoding

ALiBi (Press et al., 2022) integrates decay for relative positions in the following Eq. 6. The base of decay b is fixed and initialized headwise.

RoPE (Su et al., 2024) exploits APE to catch relative Positional information. We select implementation for linear attention as Eq. 14, where R_i stands for RoPE positional encoding for position i. RoPE cancels applications of APE in normalization of similarity calculation.

$$\begin{aligned} Sim(Q_i, K_j) &= (R_i\phi(Q_i))(R_j\phi(K_j)^\intercal) \\ Att_{i,j} &= \frac{Sim(Q_i, K_j)}{\sum_{j=1}^{i} \phi(Q_i)\phi(K_j)^\intercal} \end{aligned} \tag{14}$$

Vanilla APE of Transformer (Vaswani et al., 2017) applies a trainable embedding [7] for absolute positional information $E(\boldsymbol{a}), \boldsymbol{a} = [1, 2, \ldots, n]$. The embedding is initialized randomly.

$$Sim(Q_i, K_j) = \phi(Q_i + E(\boldsymbol{a})_i)\phi(K_j + E(\boldsymbol{a})_j)^\intercal \tag{15}$$

For D2D, we initialize $\boldsymbol{P_l^s}$ for each head l with a zero vector $\mathbf{0} \in \mathbb{R}^{1\times d_h}$. $\boldsymbol{P_l^b}$ is initialized with scalar P_l^b in Eq. 16, where h indicates the number of heads, and then fill the vector $\boldsymbol{P_l^b}$ with the scalar.

$$P_l^b = 2^{-\frac{h}{l}} \tag{16}$$

A.5 Fitting Process Analysis

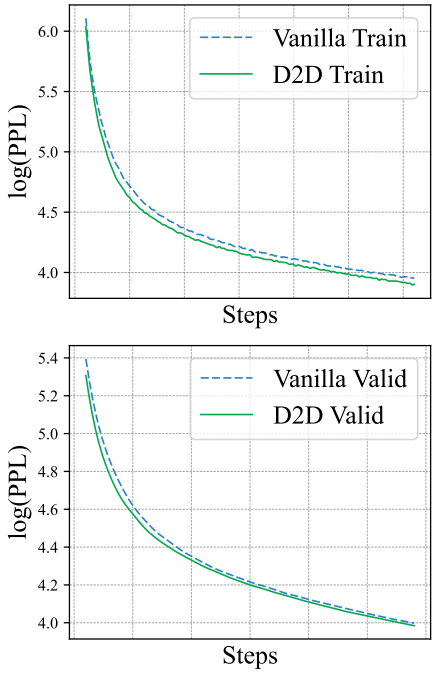

Figure 6: log PPL on the former 60% steps for Vanilla GPT and linear attention using D2D.

As shown in Fig. 6, during training, we find that D2D and position encodings like ALiBi make linear attention more aggressive throughout the training process. Specifically, the perplexity (PPL) decreases faster during the training process. A similar trend occurs in the validation set, but gap between vanilla GPT and D2D is narrowed. For other PE like RoPE or ALiBi, vanilla GPT can reach a lower PPL at last even with a higher start.

We speculate that this may be due to the introduction of relative positional information, allowing the model to learn sequence information more quickly. However, the original version of GPT has a slower start in terms of Perplexity (PPL) decrease, but maintains a longer optimization process, and finally reaches comparable performance as linear attention. We analyze that linear attention, by removing the softmax, still has many shortcomings in terms of normalization. D2D and other Relative Positional Encodings (RPE) provide a shortcut, endowing the model with the prior knowledge of "focusing on nearby tokens", thereby achieving better performance. The softmax operation still possesses good properties and stronger learning capabilities.

A.6 Experiments For Effective Inference

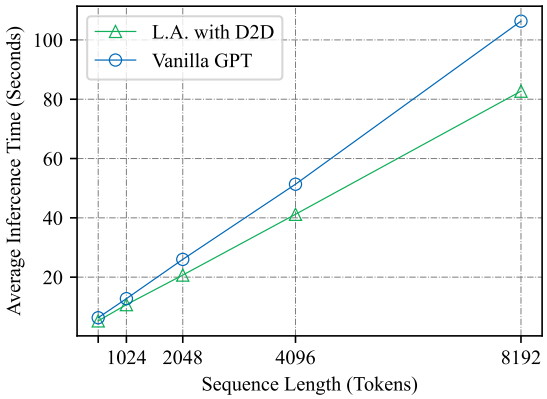

Figure 7: Average infercene time for sequence with different length. L.A. with D2D stands for linear attention with D2D.

To ensure that D2D exhibits superiority in terms of inference speed compared to the vanilla model, we conduct speed tests for language generation at the inference stage. We transform our method into RNN-form to achieve $O(n)$ time complexity. We eliminate the "End of Sequence" (EOS) token from the vocabulary to guarantee the production of texts that conform to specified length criteria. We conducte ten experiments for each model at each length and took the average as the generation time. The weights of the model are subjected to random initialization, given that this has no impact on the assessment of generation speed.

Results indicate that inference time complexity of our method is lower than that of the vanilla GPT, and as the inference length increases, the advantages of our method become increasingly pronounced. When the sequence length is relatively short, the improvement in time is not very pronounced, as the fundamental computations and data

[7] Trainable embedding is only added in the first layer of GPT-2 in vanilla implementation.

copying still require a certain amount of time.

A.7 Detailed Result of Kernel Selection

For linear attention, there is no one-size-fits-all criterion for selecting the kernel function; thus, it is essential to choose the most appropriate kernel function based on the position encoding we use. As demonstrated in Table. 2, for the position encoding we employ, using $elu(x)+1$ as the kernel function achieves a lower perplexity (ppl) compared to using $\exp(x)$.

To further ascertain which kernel function holds an advantage, we employ a new method to validate that $elu(x)+1$ is the better choice. The specific experimental scheme is as follows: We first train a linear attention model with $\exp(x)$ as the kernel function on 10% of the training data from OpenWebText. After the training is completed, we replace the kernel function with a three-layer Multilayer Perceptron (MLP) network, retaining the rest of the trained parameters and setting them to be non-trainable. We then conduct extensive fitting on 1% of the training data from OpenWebText. Finally, we extract the trained kernel function from this model and plot its function graph. As shown

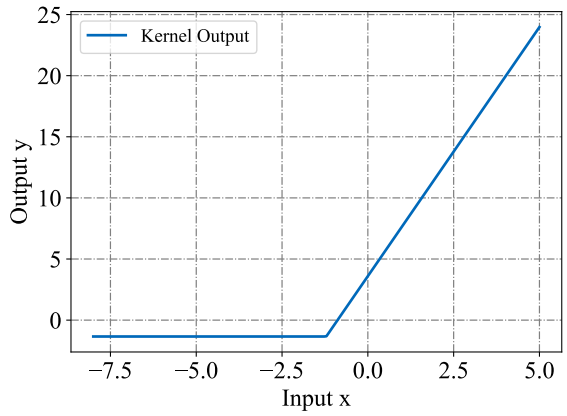

Figure 8: We fully train a three-layer MLP network in place of the kernel function on 1% of OpenWebText's training dataset, which is ultimately extracted to draw the function image. This is the image of the kernel function for the first layer of the entire network, and the images for the remaining layers are similar to this one.

in Fig. 8, the fully trained three-layer MLP network is close to a linear function when x is large, which means the trained three-layer MLP network is closer to $elu(x)+1$ than to the $\exp(x)$. $relu(x)$ also possesses similar mathematical properties as $elu(x)+1$, but $relu(x)$'s gradient of 0 at x<0 is unfavorable for training, so we prefer to choose $elu(x)+1$ as the kernel function rather than choose $relu(x)$.

A.8 Kernel Function Concentration Analysis

We refer to the ability of a kernel function to focus on one or several tokens as the function's focus ability (Han et al., 2023). To more conveniently assess a kernel function's focus ability from a mathematical perspective, we use the **coefficient of variation** of the kernel function under a specific distribution as a mathematical indicator to judge the kernel function's focus ability.

Coefficient of Variation The coefficient of variation is a normalized measure of the dispersion of a probability distribution, defined as the ratio of the standard deviation to the mean. The expression is as follows:

$$c_v = \frac{\sigma}{\mu} \times 100\% \tag{17}$$

Where σ is the standard deviation of the distribution, μ is the mean of the distribution, and c_v is the coefficient of variation of the distribution. Since the calculation of attention scores involves normalization, the coefficient of variation, as opposed to mean or variance, can avoid the impact of overall numerical scaling. This aligns with the purpose of normalization and is more suitable for measuring the focus ability of kernel functions.