

Element-wise Attention Is All You Need

Guoxin Feng¹

Abstract

The self-attention (SA) mechanism has demonstrated superior performance across various domains, yet it suffers from substantial complexity during both training and inference. The next-generation architecture, aiming at retaining the competitive performance of SA while achieving low-cost inference and efficient long-sequence training, primarily focuses on three approaches: linear attention, linear RNNs, and state space models. Although these approaches achieve reduced complexity than SA, they all have built-in performance degradation factors, such as diminished “spikiness” and compression of historical information. In contrast to these approaches, we propose a novel element-wise attention mechanism, which uses the element-wise squared Euclidean distance, instead of the dot product operation, to compute similarity and approximates the quadratic complexity term $\exp(q_i k_{j_c})$ with a Taylor polynomial. This design achieves remarkable efficiency: during training, the element-wise attention has a complexity of $\mathcal{O}(tLD)$, making long-sequence training both computationally and memory efficient, where L is the sequence length, D is the feature dimension, and t is the highest order of the polynomial; during inference, it can be reformulated as recurrent neural networks, achieving a inference complexity of $\mathcal{O}(tD)$. Furthermore, the element-wise attention circumvents the performance degradation factors present in these approaches and achieves performance comparable to SA in both causal and non-causal forms.

1. Introduction

Traditional deep learning methods have significant limitations: Convolutional Neural Networks (CNNs) struggle to

¹College of Information Science and Engineering, Northeastern University, China. Correspondence to: Guoxin Feng <2172064@stu.neu.edu.cn>.

capture long-range dependencies, while traditional Recurrent Neural Networks (RNNs) (Hochreiter, 1997; Chung et al., 2014) do not support parallel training and are prone to exploding or vanishing gradients. In contrast, the self-attention (SA) (Vaswani, 2017) mechanism overcomes these limitations and has demonstrated superior performance across various domains (Dai et al., 2019; Dosovitskiy, 2020; Brown et al., 2020), including natural language processing (NLP), computer vision (CV), and time-series analysis.

Despite its superior performance, SA incurs significant complexity in both training and inference. During inference, the KV-caching (Pope et al., 2023) technique stores historical key and value matrices, leading to an inference complexity of $\mathcal{O}(LD)$, where L is the sequence length and D is the feature dimension. Consequently, the inference cost grows with the sequence length. During training, SA has computational and memory complexities of $\mathcal{O}(L^2)$, which makes training on long sequences both time-consuming and memory-intensive.

Significant efforts have been devoted to developing the next-generation architecture, aiming at simultaneously achieving low-cost inference, efficient long-sequence training, and SA-comparable performance, i.e., breaking the “impossible triangle”. Current research can be classified into three main categories. First, linear attention (LA) (Katharopoulos et al., 2020; Sun et al., 2023; Yang et al., 2024; 2025) employs kernels $\varphi(q_i) \cdot \varphi(k_j^T)$ to approximate the $\exp(q_i k_j^T)$ in SA, achieving linear training complexity and constant inference complexity with respect to the sequence length. However, LA’s performance often lags behind SA, primarily due to its lack of “spikiness” (Zhang et al., 2024). Spikiness refers to the attention mechanisms assigning major weights to a few critical tokens while assigning little weights to others, leading to sharp attention weights distributions, which greatly contributes to the effectiveness of attention mechanisms. SA attains spikiness by using the exponential function to amplify larger dot products and suppress smaller ones. In contrast, LA loses this spikiness by eliminating the exponential function. Secondly, linear RNNs (Bradbury et al., 2017; Orvieto et al., 2023; Peng et al., 2023) eliminate the non-linear functions in traditional RNNs, enabling efficient parallel training while retaining the low-cost inference characteristic in traditional RNNs. However, linear RNNs generally exhibit inferior performance compared to

attention mechanisms. One reason is that eliminating non-linear functions decreases the models’ representational capacity. Another reason is that, unlike attention mechanisms allowing tokens to interact directly, RNNs process data sequentially and compress historical information into a single hidden vector. This process gradually diminishes prior information, limiting the models’ ability to capture long-range dependencies. The third category of research focuses on simulating state space models (SSMs) (Gu et al., 2021; Fu et al., 2023; Gu & Dao, 2023; Poli et al., 2023; Parnichkun et al., 2024). These models exhibit performance degradation factors similar to those in RNNs, and require complex implementations to maintain efficiency during training and inference. None of the three categories of work has yet succeeded in breaking the “impossible triangle”.

Research (Arora et al., 2024) has demonstrated that the Taylor polynomial can preserve the spikiness introduced by the exponential function ($\exp(q_i k_j^\top) = 1 + q_i k_j^\top + \frac{(q_i k_j^\top)^2}{2} + \dots$), leading to the training complexity of $\mathcal{O}(LD^t)$ and the inference complexity of $\mathcal{O}(D^t)$, where t denotes the highest order of the polynomial. Notably, the exponential complexity with respect to the feature dimension D arises from the dot-product operation, which is the standard operation for computing similarity in traditional attention mechanisms. We propose that, by replacing the dot-product operation with element-wise operation, the incredible efficiency (a training complexity of $\mathcal{O}(tLD)$ and an inference complexity of $\mathcal{O}(tD)$) can be achieved. And the proposed element-wise attention avoids the aforementioned performance degradation factors. Specifically, we use the element-wise squared Euclidean distance to measure the similarity between query and key elements at an arbitrary channel c . Then, the Softmax function converts the similarity scores into attention weights. The quadratic complexity term $\exp(q_{ic} k_{jc})$ introduced by the Softmax function is approximated using a Taylor polynomial.

We conducted extensive experiments to compare the element-wise attention with SA, and the results demonstrate that the element-wise attention simultaneously achieves low-cost inference, efficient long-sequence training, and SA-comparable performance. In performance comparisons, the element-wise attention achieves results comparable to self-attention in both causal and non-causal forms on time series datasets, particularly when high-order Taylor polynomials are applied. During training, the element-wise attention demonstrates higher training efficiency than SA in terms of memory usage and throughput. And this advantage becomes more pronounced as the sequence length increases. During inference, the element-wise attention can be reformulated as recurrent neural networks, resulting in inference costs that remain constant with respect to the sequence length and are minimally impacted by variations in batch size. In contrast, SA relies on KV-caching, leading to inference costs that

scale with sequence length and are significantly affected by changes in batch size.

2. Notations

Let $X \in \mathbb{R}^{L \times D}$ denote a sequence of L token vectors, each comprising D channels. We introduce the following notation: $X_{i:} \in \mathbb{R}^D$ or $X_i \in \mathbb{R}^D$ denotes the i -th token vector; $X_{::c} \in \mathbb{R}^L$ denotes the elements in the c -th channel across all tokens; $X_{ic} \in \mathbb{R}$ denotes the element at the i -th token vector and c -th channel.

In Self-Attention (SA), dot-product operations are frequently used. Specifically, for $q_i = (q_{i1}, \dots, q_{iD}) \in \mathbb{R}^D$ and $k_j = (k_{j1}, \dots, k_{jD}) \in \mathbb{R}^D$, their dot product is calculated as $q_i k_j^\top = q_{i1} k_{j1} + \dots + q_{iD} k_{jD} \in \mathbb{R}$. In Element-wise Attention (EA), element-wise operations are used. Specifically, the element-wise squaring of q_i is $q_i^2 = (q_{i1}^2, \dots, q_{iD}^2) \in \mathbb{R}^D$; the element-wise multiplication between q_i and k_j is $q_i k_j = (q_{i1} k_{j1}, \dots, q_{iD} k_{jD}) \in \mathbb{R}^D$; and the element-wise division is given by $\frac{q_i}{k_j} = \left(\frac{q_{i1}}{k_{j1}}, \dots, \frac{q_{iD}}{k_{jD}} \right) \in \mathbb{R}^D$.

3. Element-wise Attention

In section 3.1, we formalize the full version of Element-wise Attention (EA), which can serve as a replacement of SA in Transformer architectures. Subsequently, in section 3.2, the EA-series, with only linear training complexity, is derived by approximating the quadratic complexity term in EA using a Taylor polynomial. In section 3.3, we formalize the causal EA-series, which can be reformulated as recurrent neural networks (RNNs) to enable efficient inference. Finally, in section 3.4, EA and EA-series are compared with various methods.

3.1. EA

In this section, we introduce the full version of EA. Figure 1 illustrates the computation process of EA.

Specifically, the input sequence $X \in \mathbb{R}^{L \times D}$ is first projected onto the corresponding representations: *queries* (q), *keys* (k), *values* (v) $\in \mathbb{R}^{L \times D}$. Then, a feature tensor is constructed using q and k . The element $o_{ijc} \in \mathbb{R}$ in the feature tensor, which directly quantifies the similarity between $q_{ic} \in \mathbb{R}$ and $k_{jc} \in \mathbb{R}$, is calculated as follows:

$$o_{ijc} = -(q_{ic} - k_{jc})^2 \quad (1)$$

Subsequently, a Softmax function normalizes $o_{i:c} \in \mathbb{R}^L$, transforming them into weights that are then assigned to $v_{:c} \in \mathbb{R}^L$ to produce $y_{ic} \in \mathbb{R}$.

The complete computational formula for EA is presented below, where squaring, multiplication and division operations

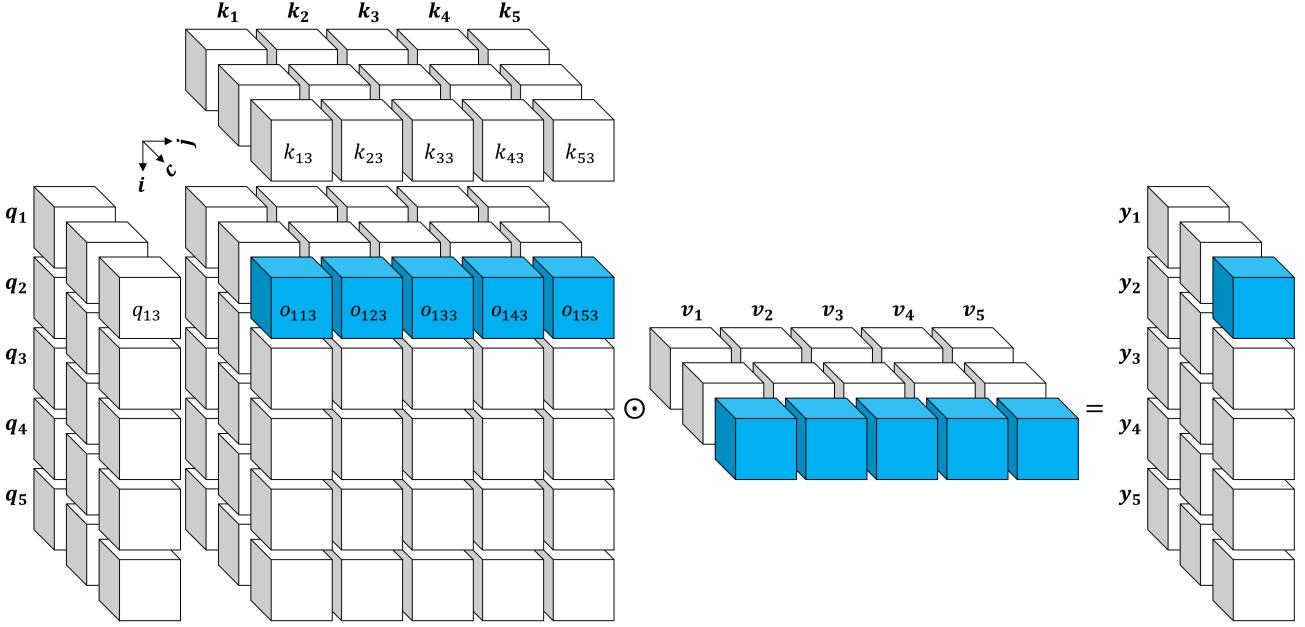


Figure 1. Illustration of EA’s computation process. Specifically, we obtain the similarity scores $o_{ijc} \in \mathbb{R}$ by computing the squared Euclidean distances between the query element $q_{ic} \in \mathbb{R}$ and the key element $k_{jc} \in \mathbb{R}$. Subsequently, the Softmax function converts $o_{i:c} \in \mathbb{R}^L$ into weights, which are assigned to $v_{:c} \in \mathbb{R}^L$ to produce $y_{ic} \in \mathbb{R}$.

are all performed element-wise:

$$y_i = \frac{\sum_{j=1}^L e^{-(q_i - k_j)^2} v_j}{\sum_{j=1}^L e^{-(q_i - k_j)^2}} \quad (2)$$

mulated as follows:

$$\begin{aligned} y_i &= \frac{\sum_{j=1}^L e^{-(q_i - k_j)^2} v_j}{\sum_{j=1}^L e^{-(q_i - k_j)^2}} \\ &= \frac{\sum_{j=1}^L e^{-k_j^2} e^{-q_i^2} e^{2q_i k_j} v_j}{\sum_{j=1}^L e^{-k_j^2} e^{-q_i^2} e^{2q_i k_j}} \\ &= \frac{\sum_{j=1}^L e^{-k_j^2} e^{2q_i k_j} v_j}{\sum_{j=1}^L e^{-k_j^2} e^{2q_i k_j}} \end{aligned} \quad (3)$$

We can find that the quadratic complexity arises from computing $e^{2q_i k_j}$. To reduce the complexity, we approximate $e^{2q_i k_j}$ using a Taylor polynomial:

$$e^{2q_i k_j} = \sum_{n=0}^{+\infty} \frac{(2q_i k_j)^n}{n!} = 1 + 2q_i k_j + \frac{2^2}{2!} q_i^2 k_j^2 + \dots \quad (4)$$

Substituting this polynomial into equation 3, we can obtain the EA-series with only linear complexity:

$$y_i = \frac{\sum_{j=1}^L e^{-k_j^2} e^{2q_i k_j} v_j}{\sum_{j=1}^L e^{-k_j^2} e^{2q_i k_j}}$$

3.2. EA-series

EA construct a feature tensor to meticulously mobilizes the value elements, but it introduces significant complexity. To reduce the complexity and facilitate efficient training and inference, we approximate the quadratic complexity term in EA using a Taylor polynomial, deriving the EA-series. The derivation process is outlined below.

Given that the squaring, multiplication and division operations are performed element-wise, equation 2 can be refor-

```

import torch
import math

# t is the number of terms taken in the Taylor series.
# exponent_tensor: (0, 1, 2,..., t-1), [t, 1, 1]
exponent_tensor = torch.arange(t).reshape(t, 1, 1)
# taylor_coeff: (2^0/0!, 2^1/1!,..., 2^(t-1)/(t-1)!), [t, 1, 1]
taylor_coeff = [2 ** i / math.factorial(i) for i in range(t)]
taylor_coeff = torch.tensor(taylor_coeff).reshape(t, 1, 1)

def EA_series(
    q,                                         # Query matrix, [BS, L, D]
    k,                                         # Key matrix, [BS, L, D]
    v,                                         # Value matrix, [BS, L, D]
    exponent_tensor,                         # [t, 1, 1]
    taylor_coeff,                           # [t, 1, 1]
):
    k = k.unsqueeze(1)                         # [BS, 1, L, D]
    # exp_k: exp(-k^2)
    exp_k = torch.exp(-torch.pow(k, 2))        # [BS, 1, L, D]
    # K: (k^0, k^1,..., k^(t-1))
    K = torch.pow(k, exponent_tensor)          # [BS, t, L, D]

    Den = K * exp_k                          # denominator, [BS, t, L, D]
    Num = Den * v.unsqueeze(1)                # numerator, [BS, t, L, D]
    Num = Num.sum(2).unsqueeze(-2)            # [BS, t, 1, D]
    Den = Den.sum(2).unsqueeze(-2)            # [BS, t, 1, D]

    q = q.unsqueeze(1)                         # [BS, 1, L, D]
    # (q^0, q^1,..., q^(t-1))
    q = torch.pow(q, exponent_tensor)          # [BS, t, L, D]
    # (q^0 * 2^0/0!, q^1 * 2^1/1!,..., q^(t-1) * 2^(t-1)/(t-1)!)
    q = q * taylor_coeff                     # [BS, t, L, D]

    output = torch.sum(q * Num, dim=1) / torch.sum(q * Den, dim=1)
    return output                             # [BS, L, D]

```

Figure 2. PyTorch implementation of EA-series.

$$\begin{aligned}
 &= \frac{\sum_{j=1}^L \left[e^{-k_j^2} v_j \left(1 + 2q_i k_j + \frac{2^2}{2!} q_i^2 k_j^2 + \dots \right) \right]}{\sum_{j=1}^L \left[e^{-k_j^2} \left(1 + 2q_i k_j + \frac{2^2}{2!} q_i^2 k_j^2 + \dots \right) \right]} \\
 &= \frac{\sum_{j=1}^L \left[e^{-k_j^2} v_j + 2q_i k_j e^{-k_j^2} v_j + \frac{2^2}{2!} q_i^2 k_j^2 e^{-k_j^2} v_j + \dots \right]}{\sum_{j=1}^L \left[e^{-k_j^2} + 2q_i k_j e^{-k_j^2} + \frac{2^2}{2!} q_i^2 k_j^2 e^{-k_j^2} + \dots \right]} \\
 &= \frac{\sum_{j=1}^L e^{-k_j^2} v_j + 2q_i \sum_{j=1}^L k_j e^{-k_j^2} v_j + \dots}{\sum_{j=1}^L e^{-k_j^2} + 2q_i \sum_{j=1}^L k_j e^{-k_j^2} + \dots} \quad (5)
 \end{aligned}$$

Despite showing the complex form, the EA-series can be efficiently implemented using the broadcast mechanism in PyTorch. The PyTorch implementation is shown in figure 2.

t-th order Taylor polynomial There are two considerations when selecting a t -th order Taylor polynomial. Firstly, the $e^{-k_j^2} e^{2q_i k_j}$ in equation 3, which serves as similarity scores, must be positive definite. This requires that the polynomial approximation of $e^{2q_i k_j}$ must be positive definite. Research (Banerjee et al., 2020) has demonstrated that, when t is even, the Taylor polynomial approximation of an exponential function is always positive definite. The second consideration involves the errors between the exponential function and its Taylor polynomial approximations. Figure 3 illustrates e^x alongside its second- and sixth-order Taylor polynomials.

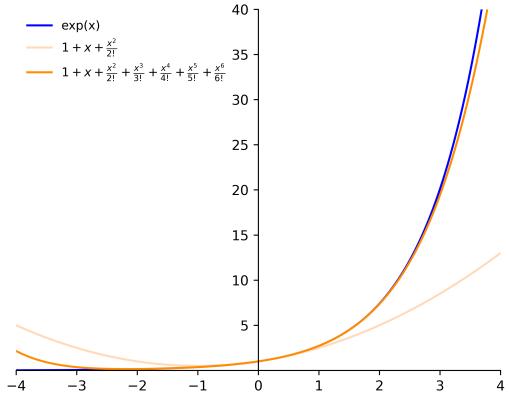


Figure 3. An illustration of e^x alongside its second- and sixth-order Taylor polynomials.

typically remain within a restricted range near the origin due to initialization and normalization techniques.

3.3. Causal EA-series

Attention mechanisms have two forms: causal and non-causal. In non-causal attention, each token can attend to all tokens within the sequence. Equation 5 is the non-causal EA-series. Conversely, causal attention allows each token to attend only to preceding and current tokens, excluding subsequent tokens. Large language models (LLMs) typically use causal attention. The causal EA-series is formally defined in equation 6 and can be implemented by replacing the *sum()* in figure 2 with *cumsum()*.

$$y_i = \frac{\sum_{j=1}^i e^{-k_j^2} v_j + 2q_i \sum_{j=1}^i k_j e^{-k_j^2} v_j + \dots}{\sum_{j=1}^i e^{-k_j^2} + 2q_i \sum_{j=1}^i k_j e^{-k_j^2} + \dots} \quad (6)$$

The causal EA-series can be trained in parallel with linear complexity. When it comes to inference, the causal EA-series can be rewritten as recurrent neural networks:

$$constant = \left(1, 2, \frac{2^2}{2!}, \dots, \frac{2^{t-1}}{(t-1)!} \right) \in \mathbb{R}^t \quad (7)$$

$$s_0 = 0 \in \mathbb{R}^{D \times t} \quad (8)$$

$$z_0 = 0 \in \mathbb{R}^{D \times t} \quad (9)$$

$$K_i = (1, k_i, k_i^2, \dots, k_i^{t-1}) \in \mathbb{R}^{D \times t} \quad (10)$$

$$Q_i = (1, q_i, q_i^2, \dots, q_i^{t-1}) \in \mathbb{R}^{D \times t} \quad (11)$$

$$s_i = s_{i-1} + K_i e^{-k_i^2} v_i \quad (12)$$

$$z_i = z_{i-1} + K_i e^{-k_i^2} \quad (13)$$

$$num = \text{sum}[s_i Q_i \text{constant}] \in \mathbb{R}^D \quad (14)$$

$$den = \text{sum}[z_i Q_i \text{constant}] \in \mathbb{R}^D \quad (15)$$

$$y_i = \frac{num}{den} \quad (16)$$

where squaring, multiplication and division operations are performed element-wise; $K_i, Q_i \in \mathbb{R}^{D \times t}$ and $\text{constant} \in \mathbb{R}^t$ can be efficiently generated using the broadcast mechanism; vectors $e^{-k_i^2} v_i, e^{-k_i^2} \in \mathbb{R}^D$ and $\text{constant} \in \mathbb{R}^t$ can be element-wise multiplied with high-dimensional matrices via the broadcast mechanism. The caches $s_i, z_i \in \mathbb{R}^{D \times t}$ are updated at each step while maintaining constant dimensions, ensuring that the inference cost remains independent of sequence length.

3.4. Relation to and Differences from Previous Methods

Table 1 compares the EA-series with typical attention mechanisms from various perspectives, including computational and memory complexity during training, and inference complexity. Generally, the sequence length L is much greater than the feature dimension D , and D is significantly larger than the highest order t in the Taylor polynomial. The EA-series demonstrates reduced memory complexity and the lowest computational and inference complexity. Detailed comparisons with specific methods are provided below:

SA In SA, each token vector of dimension D is divided into H heads, where each head is a sub-vector containing D/H channels. The output at each head is computed as follows, with the scaling factor omitted for simplicity:

$$y_i = \frac{\sum_{j=1}^L e^{q_i k_j^T} v_j}{\sum_{j=1}^L e^{q_i k_j^T}} \quad (17)$$

Four key comparisons can be made. (1) The SA uses dot-product operations to compute the similarity, while the EA uses element-wise squared Euclidean distance. (2) The SA generates a total of H head-level feature maps, while the EA generates D channel-level feature maps, allowing for more comprehensive interactions between tokens. (3) The exponential function introduces “spikiness” which contributes much to attention’s effectiveness, while the exponential function incurs significant complexity. We use a Taylor polynomial to approximate the exponential function, preserving its effectiveness while reducing complexity. (4) During inference, SA caches all previous keys and values, causing its inference cost to scale with sequence length. Conversely, the EA-series achieves constant inference complexity with respect to the sequence length, enabling efficient long-sequence inference.

LA As shown in equation 18, Linear Attention (LA) approximates the $\exp(q_i k_j^T)$ in SA with kernels $\varphi(q_i) \cdot \varphi(k_j^T)$, thereby achieving linear training complexity and constant

Table 1. EA-series compares with typical attention mechanisms in terms of computational and memory complexity during training, and inference complexity.

	COMPUTATIONAL	MEMORY	INFERENCE
SA	$\mathcal{O}(L^2 D)$	$\mathcal{O}(L^2)$	$\mathcal{O}(LD)$
LA	$\mathcal{O}(LD^2)$	$\mathcal{O}(LD)$	$\mathcal{O}(D^2)$
AFT	$\mathcal{O}(L^2 D)$	$\mathcal{O}(LD)$	$\mathcal{O}(LD)$
EA-SERIES	$\mathcal{O}(tLD)$	$\mathcal{O}(tLD)$	$\mathcal{O}(tD)$

inference complexity with respect to the sequence length.

$$y_i = \frac{\sum_{j=1}^L \varphi(q_i) \varphi(k_j^T) v_j}{\sum_{j=1}^L \varphi(q_i) \varphi(k_j^T)} = \frac{\varphi(q_i) \sum_{j=1}^L \varphi(k_j^T) v_j}{\varphi(q_i) \sum_{j=1}^L \varphi(k_j^T)} \quad (18)$$

Two key comparisons can be made. (1) The LA loses the “spikiness” when approximating the exponential function with kernels, leading to performance degradation. In contrast, the EA-series avoids such limitations. (2) Due to the large feature dimension D in LLMs, the $\mathcal{O}(D^2)$ inference complexity of LA results in substantial inference cost. In contrast, the EA-series achieves significantly lower inference cost of an $\mathcal{O}(tD)$ complexity.

AFT AFT (Zhai et al., 2021) defines an attention mechanism as follows, where multiplication and division operations are performed element-wise; $w \in \mathbb{R}^{L \times L}$ is learned positional biases.

$$y_i = \frac{\sum_{j=1}^L e^{k_j + w_{ij}} v_j}{\sum_{j=1}^L e^{k_j + w_{ij}}} \quad (19)$$

Both EA and AFT use element-wise operations. Comparing equation 19 with equation 2, the key difference is that AFT computes attention weights using the position-bias-corrected k , while EA computes attention weights based on the similarity between q and k .

RNNs Four key comparisons can be made. (1) Traditional RNNs cannot be trained in parallel, resulting in low training efficiency. Linear RNNs enable parallel training by removing nonlinear functions in traditional RNNs, while the removal compromises their representational capacity. In contrast, the EA-series achieves high training efficiency while preserving representational capacity by approximating the nonlinear function with the Taylor polynomial. (2) Unlike direct token interactions in attention mechanisms, RNNs process tokens sequentially, compressing historical information into hidden states. This sequential processing in RNNs gradually diminishes prior information, limiting the model’s ability to capture long-range dependencies. (3) Although the autoregressive abilities of RNNs are comparable to those of causal attention mechanisms, RNNs lack the global interaction abilities in non-causal attention mech-

anisms. (4) Although the inference cost of attention mechanisms is generally significantly higher than those of RNNs, EA-series can be reformulated as recurrent neural networks during inference, achieving an $\mathcal{O}(tD)$ inference cost, which approaches the $\mathcal{O}(D)$ inference cost of RNNs.

Hedgehog Hedgehog (Zhang et al., 2024) approximates the exponential function in SA using a Taylor polynomial as follows:

$$e^{q_i k_j^T} = 1 + q_i k_j^T + \frac{(q_i k_j^T)^2}{2} + \dots \quad (20)$$

This approximation reduces the complexity from quadratic to linear with respect to the sequence length. And the experimental results show that the approximation preserves key properties of the exponential function, such as "spikiness" and monotonicity.

Due to the dot-product operations in SA, approximating $e^{q_i k_j^T}$ introduces exponential complexity with respect to the feature dimension D . Specifically, using a t -order Taylor polynomial to approximate $e^{q_i k_j^T}$ results in a training complexity of $\mathcal{O}(LD^t)$. This implies that employing high-order Taylor polynomials significantly increases complexity. In contrast, the EA-series, which relies on element-wise operations, achieves a training complexity of $\mathcal{O}(tLD)$, allowing to incorporate sufficient Taylor terms to ensure the effectiveness while maintaining low complexity.

4. Experiment

Section 4.1 evaluates the performance of EA-2, EA-6, and SA in both causal and non-causal forms, where "2" and "6" indicate the highest order in the Taylor polynomial. Section 4.2 analyzes their training efficiency based on memory usage, the BS-L curves, and throughput. Section 4.3 compares their inference cost, focusing on memory usage and latency.

4.1. Performance Comparisons

We compare the performance of EA-2, EA-6, and SA in both non-causal and causal forms. The models are constructed using standard Transformer blocks, where Post-Layer Normalization (Post-LN) (Lei Ba et al., 2016) is used; absolute positional embedding (Devlin et al., 2019), which is designed based on the sampling times of the data, is used to incorporate temporal information. To enable direct comparisons between EA-2, EA-6, and SA, we modify only the attention mechanisms within the models while keeping all other components unchanged. The implementation follows the Time Series Library repository (Wu et al., 2023), adhering to its data processing procedures and the specified training, validation, and test splits. To ensure fairness, all models are trained under identical conditions, and the optimal model is selected based on performance on the validation set. Full implementation details are provided on

Table 2. The characteristics of multivariate time series classification datasets, including the number of time series, the length of each time series and the number of labels.

	JAP	SCP1	SCP2	UWG
# OF SERIES	12	6	7	3
LENGTH OF SERIES	29	896	1152	315
# OF LABELS	9	2	2	8

Table 3. Multivariate time series classification results. Bolded values indicate the best results.

	JAP	SCP1	SCP2	UWG
EA-2	0.957	0.887	0.500	0.794
EA-6	0.973	0.904	0.533	0.847
SA	0.970	0.894	0.528	0.822

GitHub.

Non-causal form In non-causal attention mechanisms, each token can attend to all tokens in the sequence. Consequently, non-causal attention is well-suited for tasks involving understanding, summarization, and classification. We compare the performance of the non-causal EA-2, EA-6 and SA on multivariate time series classification (MTSC). In MTSC, each sample consists of multiple interrelated time series, with each series containing observations recorded at sampling times. The objective is to predict the label of each sample. Specifically, a sample $S = (S_1, \dots, S_n) \in \mathbb{R}^{L \times n}$ comprises n time series, each of length L . The model's goal is to predict the label associated with S . We choose four real-world MTSC datasets from the UEA Time Series Classification Archive (Bagnall et al., 2018): JapaneseVowels, SelfRegulationSCP1, SelfRegulationSCP2 and UWaveGesture. Table 2 provides the characteristics of these datasets, including the number of time series, the length of each time series and the number of labels. Table 3 reports the models' prediction accuracy on these datasets, where higher accuracy indicates better performance. The results show that the EA-series underperforms with few Taylor terms. With a sufficient number of terms, the EA-series demonstrates strong performance, surpassing that of SA.

Causal form In causal attention mechanisms, each token attends only to preceding and current tokens. Consequently, causal attention is well-suited for tasks involving generation and forecasting. We compare the performance of the causal EA-2, EA-6 and SA on time series forecasting (TSF). In TSF, the model's objective is to predict the future series based on a given series. Specifically, for a time series $S = (s_1, \dots, s_L) \in \mathbb{R}^L$, the model aims to forecast the future series $S' = (s_{L+1}, \dots, s_{L+L'}) \in \mathbb{R}^{L'}$. We evaluate the models using three real-world TSF datasets: ETTh2, ETTm2, and Traffic. In the experiments, we set $L = 6$ and

Table 4. Time series forecasting results. Bolded values indicate the best results

	ETTH2				ETTM2				TRAFFIC			
	6		12		6		12		6		12	
	MAE	RMSE										
EA-2	1.43	2.01	2.14	2.90	0.44	0.63	0.73	1.08	0.12	0.15	0.18	0.28
EA-6	1.23	1.79	1.90	2.67	0.36	0.53	0.64	0.98	0.09	0.13	0.16	0.25
SA	1.24	1.79	1.92	2.71	0.36	0.54	0.64	0.99	0.10	0.13	0.16	0.27

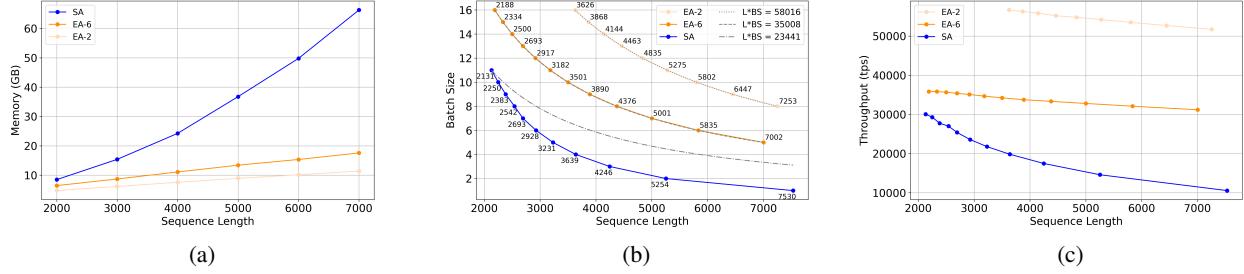


Figure 4. Training cost of EA-2, EA-6, and SA. Specifically, (a) illustrates their memory usage, (b) presents their BS-L curves, and (c) shows their throughput.

assess the models' performance of $L' = 6$ and $L' = 12$ using the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics, where lower metric values indicate better performance. Table 4 presents the time series forecasting results. The results show that the EA-series underperforms with few Taylor terms. As more terms are incorporated, the EA series exhibits superior performance, outperforming the SA.

4.2. Training Cost

In this section, we compare the training cost of EA-2, EA-6, and SA. All models use the BERT-base (Devlin et al., 2019) configuration with 12 layers, a hidden dimension $D = 768$, a head dimension of 64, and an intermediate dimension of $4D$ in the FFN. The models are implemented in PyTorch without additional optimizations, and experiments are conducted on an A800-80GB GPU.

Memory Figure 4(a) illustrates the memory usage of the models at different sequence lengths with a batch size of 1. The memory usage of the EA-series increases linearly with the sequence length, due to its linear memory complexity with respect to the sequence length. In contrast, the memory usage of SA grows exponentially due to its quadratic memory complexity. Overall, the EA-series requires substantially less memory than SA for long-sequence modeling.

BS-L Curve Maximizing GPU memory utilization is important for model training. For a given model and GPU, memory utilization is primarily influenced by the batch

size (BS) and sequence length (L). In figure 4(b), the solid curves, referred to as the BS-L curves, depict the maximum sequence lengths the models can handle on the GPU for various BS. Clearly, BS and L have an inverse relationship: as the sequence length increases, the batch size that fits on the GPU decreases. The dashed curves are inverse proportional curves. The product of BS and L, representing the number of tokens processed per step, remains constant along each dashed curve.

Key observations from the figure are as follows. (1) The EA-series can process more tokens per step than SA due to the lower memory usage of EA-series, making EA-series advantageous for processing large datasets. (2) The BS-L curves of the EA-series closely align with the dashed curves, whereas that of SA deviates downward from the dashed curve as the sequence length increases. This implies that the total number of tokens SA can process decreases as sequence length increases, leading to reduced efficiency in long-sequence modeling.

Throughput Throughput refers to the amount of data a model can process within a given time frame, typically measured in tokens per second (tps). Figure 4(c) presents the throughput at various points along the BS-L curves. Key observations from the figure are as follows. (1) The EA-series achieves significantly higher throughput than SA. (2) The EA-series maintains high throughput, while SA's throughput decreases notably with longer sequences.

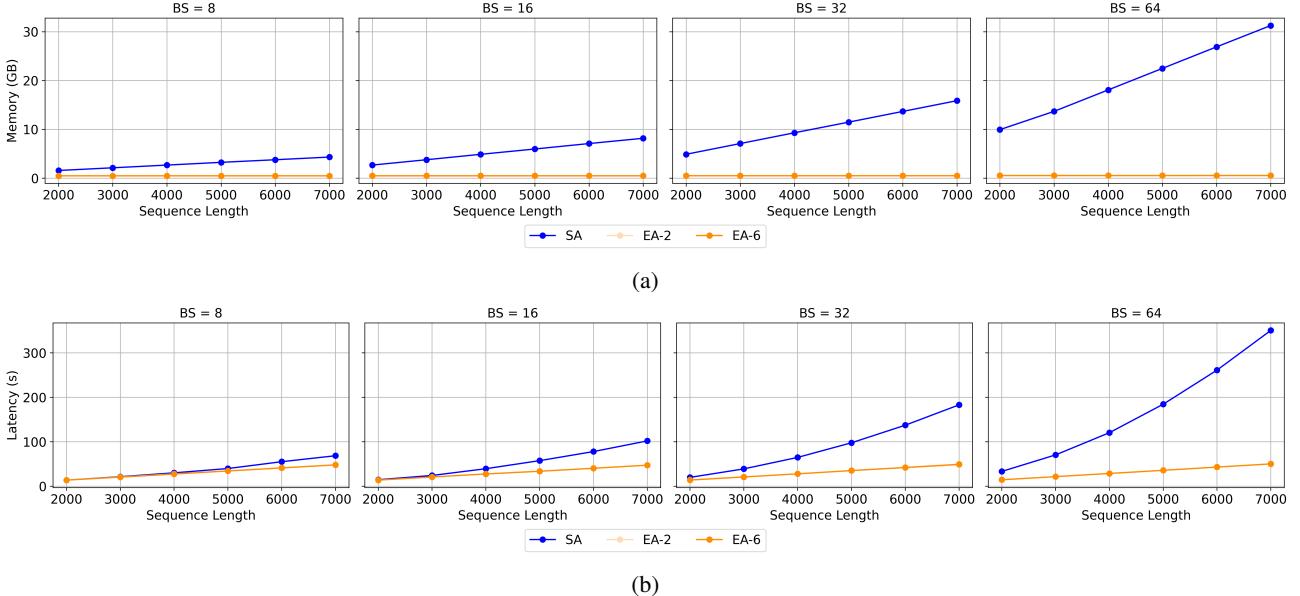


Figure 5. Inference cost of EA-2, EA-6, and SA. Specifically, (a) illustrates their memory usage, and (b) presents their latency.

4.3. Inference Cost

In this section, we compare the inference cost of EA-2, EA-6, and SA in terms of memory usage and latency. All models use the BERT-base configuration, and all experiments are conducted on an A800-80GB GPU. During inference, SA utilizes KV-caching to improve efficiency, resulting in an $\mathcal{O}(LD)$ inference complexity; EA-series employs the recurrent representation described in section 3.3, achieving an $\mathcal{O}(tD)$ inference complexity. Figure 5 illustrates the performance of the models. From the figure, we observe that EA-2 and EA-6 exhibit very similar performance, despite EA-6 containing more terms than EA-2; EA-series achieves significantly better inference efficiency than SA. Detailed comparisons are presented below.

Memory Figure 5(a) illustrates the memory usage of the models under varying batch sizes and sequence lengths. The memory usage of SA increases linearly with both sequence length and batch size. In contrast, the memory usage of EA-series remains constant with respect to the sequence length, because the size of the caches $s_i, z_i \in \mathbb{R}^{D \times t}$ is independent of the sequence length; the memory usage of EA-series is minimally affected by changes in batch size, because the size of the caches remains negligible compared to the model parameters, even for large batch sizes.

Latency Figure 5(b) illustrates the latency of the models under varying batch sizes and sequence lengths. Key observations from the figure are as follows. (1) The latency of the EA-series increases linearly with the number of tokens, while the latency of the SA increases exponentially. This occurs because the cache size of the EA-series remains

constant with respect to the sequence length, resulting in a fixed generation time per token. In contrast, the cache size of the SA grows with the sequence length, leading to progressively longer generation time for subsequent tokens. (2) The latency of the EA-series is minimally affected by batch size changes, whereas the latency of the SA is significantly influenced. This is because the EA-series maintains a small cache size even with large batch sizes, resulting in minimal variation in computation time. In contrast, the SA has a large cache size, and increasing the batch size significantly increases computation time.

5. Conclusion

In this work, we propose an element-wise attention mechanism as a replacement for SA. The mechanism substitutes the dot-product operation with element-wise operation for similarity computation and approximates the quadratic complexity term using a Taylor polynomial. By doing so, it simultaneously achieves SA-comparable performance and incredible efficiency (a training complexity of $\mathcal{O}(tLD)$ and an inference complexity of $\mathcal{O}(tD)$). The proposed model offers a foundation for the next-generation architecture and may have significant impact across various domains.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

- Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zou, J., Rudra, A., and Re, C. Simple linear attention language models balance the recall-throughput tradeoff. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 1763–1840. PMLR, 21–27 Jul 2024. URL <https://proceedings.mlr.press/v235/arora24a.html>.
- Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.
- Banerjee, K., Gupta, R. R., Vyas, K., Mishra, B., et al. Exploring alternatives to softmax function. *arXiv preprint arXiv:2011.11538*, 2020.
- Bradbury, J., Merity, S., Xiong, C., and Socher, R. Quasi-recurrent neural networks. In *International Conference on Learning Representations*, 2017.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.
- Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. Transformer-XL: Attentive language models beyond a fixed-length context. In Korhonen, A., Traum, D., and Márquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL <https://aclanthology.org/P19-1285/>.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423/>.
- Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Re, C. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*, 2023.
- Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.
- Hochreiter, S. Long short-term memory. *Neural Computation MIT-Press*, 1997.
- 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.
- Lei Ba, J., Kiros, J. R., and Hinton, G. E. Layer normalization. *ArXiv e-prints*, pp. arXiv–1607, 2016.
- Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org, 2023.
- Parnichkun, R. N., Massaroli, S., Moro, A., Smith, J. T., Hasani, R., Lechner, M., An, Q., Ré, C., Asama, H., Ermon, S., et al. State-free inference of state-space models: The transfer function approach. *arXiv preprint arXiv:2405.06147*, 2024.
- Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Derdzynski, L., Du, X., Grella, M., Gv, K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Lin, J., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wind, J., Woźniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R.-J. RWKV: Reinventing RNNs for the transformer era. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, Singapore, December 2023. Association for Computational

Linguistics. doi: 10.18653/v1/2023.findings-emnlp.936. URL <https://aclanthology.org/2023.findings-emnlp.936/>.

Poli, M., Massaroli, S., Nguyen, E. Q., Fu, D. Y., Dao, T., Baccus, S. A., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, 2023. URL <https://api.semanticscholar.org/CorpusID:257050308>.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

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

Vaswani, A. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Wu, H., Hu, T., Liu, Y., Zhou, H., Wang, J., and Long, M. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The Eleventh International Conference on Learning Representations*, 2023.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24. JMLR.org, 2025.

Zhai, S., Talbott, W., Srivastava, N., Huang, C., Goh, H., Zhang, R., and Susskind, J. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.

Zhang, M., Bhatia, K., Kumbong, H., and Re, C. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *The Twelfth International Conference on Learning Representations*, 2024.