Accelerating Linear Attention Design by Unifying Forward & Backward Propagation

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

Abstract

The rapid evolution of linear attention has led to the proliferation of various novel methods in recent years. However, the design and implementation process of linear attention mechanisms remains inherently complex and cumbersome. Typically, this process is composed of four essential stages: 1) Formulating the forward recursive expression; 2) Implementing the chunk-parallel approach for forward propagation; 3) Deriving the recursive formulation for backpropagation; and 4) Implementing the chunk-parallel backpropagation and finalizing the kernel implementation. This multifaceted design pipeline represents a significant impediment to the efficient development and exploration of new linear attention variants. In this paper, we demonstrate that both forward and backward propagation in linear attention can be expressed through a unified functional framework. By manipulating the input parameters, the function can yield either forward or backward propagation results. This approach substantially reduces the development effort associated with linear attention kernel implementation. We validate our method across multiple linear attention variants, including constant decay, scalar decay, and vector decay, within the context of language modeling tasks. Despite the reduction in development effort, experimental results demonstrate that the kernels implemented using our approach outperform the original implementations in both speed and memory efficiency.

1. Introduction

The linear-complexity attention mechanism has become a promising alternative to Transformer models in language modeling tasks, effectively reducing their inherent quadratic computational complexity to linear without compromising modeling performance. Multiple variants have been proposed recently, including linear attention (Qin et al., 2024b; Yang et al., 2023; Beck et al., 2024b; Zhang et al., 2024), linear RNN (Qin et al., 2023; Orvieto et al., 2023), and state space models (Gu et al., 2022; Gu & Dao, 2024; Dao & Gu, 2024).

Although these methods provide significant computational benefits, their development and efficient implementation are both complex and time-consuming. According to (Chou et al., 2024; Yang et al., 2023), these methods can be unified under a single framework, wherein each can be expressed in a recursive form of the following structure:

$$\begin{aligned} \mathbf{s}_t &= \mathbf{l}_t \mathbf{s}_{t-1} \mathbf{r}_t + \mathbf{k}_t \mathbf{v}_t^\top, \mathbf{o}_t^\top = \mathbf{q}_t^\top \mathbf{s}_t, \\ \mathbf{q}_t, \mathbf{k}_t, \lambda_t &\in \mathbb{R}^d, \mathbf{s}_t \in \mathbb{R}^{d \times e}, \mathbf{v}_t, \gamma_t \in \mathbb{R}^e, \\ \mathbf{l}_t &= \mathrm{diag}(\lambda_t) \in \mathbb{R}^{d \times d}, \mathbf{r}_t = \mathrm{diag}(\gamma_t) \in \mathbb{R}^{e \times e}. \end{aligned} \tag{1}$$

and $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t$ are query, key, value, λ_t, γ_t are left decay and right decay respectively, d is the q-k dimension, e is the v-o dimension, and all these vectors are computed based on the input $\mathbf{x}_t$. The development of a new linear-complexity attention mechanism fundamentally involves the design of a novel decay λ_t, γ_t (Peng et al., 2023a;b; Qin et al., 2023).

After designing the linear attention, the next step is to design the kernel. Typically, the process of developing a new linear-complexity attention kernel entails four key stages (Yang et al., 2023; Qin et al., 2024a; Dao & Gu, 2024; Beck et al., 2025):

- Formulating a forward recursive expression.
- Implementing the chunk-parallel version of the forward recursive propagation.
- Deriving the backward recursive propagation formulation.
- Implementing the chunk-parallel version of the backward recursive propagation.

This multi-stage pipeline requires researchers not only to propose a novel forward recursive formulation but also to rigorously derive the corresponding backward propagation algorithm. In practice, this process is highly error-prone, as subtle mathematical inaccuracies often go unnoticed until the end-to-end training phase, where they may result in training instability, non-convergence, or degraded model performance.

In this paper, we address these challenges by introducing a unified formulation that seamlessly integrates both forward and backward propagation for linear-complexity attention mechanisms. Specifically, we conduct a detailed analysis of the computational characteristics of a broad class of models, including linear attention,

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

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

linear RNNs, and state space models (SSMs), and observe that all essential computations, whether in recursive or chunk-parallel form and whether for forward or backward propagation, can be expressed within a single unified function. This formulation enables flexible manipulation of input parameters to produce either forward or backward computations and facilitates smooth transitions between recursive and parallel execution modes.

Our approach streamlines the kernel development process by enabling researchers to concentrate on implementing a single core function, rather than managing multiple interdependent algorithms. This simplification significantly reduces development overhead while improving code maintainability, readability, and extensibility. Moreover, it facilitates rapid prototyping of new linear attention variants, thereby promoting innovation in this fast-evolving field of research.

To demonstrate the practical benefits of our approach, we reimplemented the forward and backward kernels for several linear attention mechanisms, including constant decay (Qin et al., 2024b; Sun et al., 2023), scalar decay (Dao & Gu, 2024; Beck et al., 2024a), and vector decay (Yang et al., 2023; Qin et al., 2024c). Experimental results indicate that our unified framework not only reduces development complexity but also delivers improved performance in terms of computational speed and memory efficiency. Comprehensive evaluations on language modeling tasks further confirm that kernels developed using our framework consistently match or surpass the performance of their original counterparts across all tested conditions.

2. Preliminary

In the following discussion, we assume the sequence length is n, the chunk size is c, and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times f}$ represents the matrix composed of vectors $\mathbf{x}_t \in \mathbb{R}^f$, where $\mathbf{X} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{dQ}, \mathbf{dK}, \mathbf{dV}, \mathbf{dO}, \mathbf{\Lambda}, \mathbf{\Gamma}\}$, and $f \in \{d, e\}$ is the corresponding feature dimension. We use $\mathbf{X}_t$ to represent the t-th chunk of $\mathbf{X}$.

2.1. Recursive form

Linear Attention encompasses SSMs (Gu et al., 2022; Gu & Dao, 2024; Dao & Gu, 2024), Linear RNNs (Qin et al., 2023; Orvieto et al., 2023), and various linear attention variants (Katharopoulos et al., 2020; Qin et al., 2024b; Yang et al., 2023; Beck et al., 2024b), which can be expressed in the following unified form:

$$\mathbf{s}_t = \mathbf{l}_t \mathbf{s}_{t-1} \mathbf{r}_t + \mathbf{k}_t \mathbf{v}_t^\top, \mathbf{o}_t^\top = \mathbf{q}_t^\top \mathbf{s}_t. \tag{2}$$

Different methods use different λ_t, γ_t, producing different types of Linear Attention:

- **Constant Decay models** (e.g., TNL, RetNet): $\lambda_t = \lambda$, $\gamma_t = 1$, where $\lambda \in \mathbb{R}$
- **Scalar Decay models** (e.g., Mamba2, RFA, xLSTM): $\lambda_t = \lambda_t$, $\gamma_t = 1$, where $\lambda_t \in \mathbb{R}$
- **Vector Decay models** (e.g., GLA, HGRN2, MetaLA): $\lambda_t = \lambda_t$, $\gamma_t = 1$, where $\lambda_t \in \mathbb{R}^d$

The complete mapping of mainstream methods is shown in Table 5.

Similar to the forward pass, the recursive backward pass is:

$$\begin{aligned} \mathbf{ds}_t &= \mathbf{l}_t \mathbf{ds}_{t+1} \mathbf{r}_t + \mathbf{q}_t \mathbf{do}_t^\top, \mathbf{dq}_t^\top = \mathbf{do}_t^\top \mathbf{s}_t^\top, \\ \mathbf{dk}_t^\top &= \mathbf{v}_t^\top \mathbf{ds}_t^\top, \mathbf{dv}_t^\top = \mathbf{k}_t^\top \mathbf{ds}_t. \end{aligned} \tag{3}$$

2.2. Chunk form

Although E.q. 2, 3 provide a complete representation of the forward and backward computations in Linear Attention, their practical computational efficiency remains suboptimal (Yang et al., 2023; Qin et al., 2024b). To address this limitation, several studies (Hua et al., 2022; Yang et al., 2023; Qin et al., 2024b; Dao & Gu, 2024; Beck et al., 2025) have proposed a chunk-based approach. The central idea behind this approach is to partition the sequence into multiple chunks and decompose the computation into intra-chunk and inter-chunk components. For the intra-chunk computation, left multiplication is employed (i.e., computing $\mathbf{QK}^\top$), while for the inter-chunk computation, right multiplication is used (i.e., computing $\mathbf{K}^\top \mathbf{V}$). This strategy ultimately achieves linear time and space complexity while fully leveraging GPU parallelism. The computational process is divided into forward and backward phases. Prior to discussing these computations, we first introduce the relevant notation.

$$\begin{aligned} [\mathbf{M}]_{ij} &= \begin{cases} 1, & i \geq j \\ 0, & i < j \end{cases}, \mathbf{\Pi}_{ij} = \prod_{t=(i-1)c+1}^{(i-1)c+j} \lambda_t, \mathbf{\Upsilon}_{ij} = \prod_{t=(i-1)c+1}^{(i-1)c+j} \gamma_t, \\ \bar{\mathbf{\Pi}}_{ij} &= \prod_{t=(i-1)c+j}^{ic} \lambda_t, \bar{\mathbf{\Upsilon}}_{ij} = \prod_{t=(i-1)c+j}^{ic} \gamma_t. \end{aligned} \tag{4}$$

Forward Pass

For the forward pass, we first define the common terms used across computations:

$$\begin{aligned} \bar{\mathbf{Q}}_t = \mathbf{Q}_t \odot \mathbf{\Pi}_t, \bar{\mathbf{K}}_t = \mathbf{K}_t / \mathbf{\Pi}_t, \tilde{\mathbf{K}}_t = \mathbf{K}_t \odot \bar{\mathbf{\Pi}}_\mathbf{t}, \\ \bar{\mathbf{V}}_t = \mathbf{V}_t / \mathbf{\Upsilon}_t, \tilde{\mathbf{V}}_t = \mathbf{V}_t \odot \bar{\mathbf{\Upsilon}}_\mathbf{t}. \end{aligned} \tag{5}$$

Then we compute the output using the following equation:

$$\bar{\mathbf{O}}_t = [[\bar{\mathbf{Q}}_t \bar{\mathbf{K}}_t^\top] \odot \mathbf{M}] \bar{\mathbf{V}}_t + \bar{\mathbf{Q}}_t \mathbf{S}_t, \mathbf{O}_t = \bar{\mathbf{O}}_t \odot \mathbf{\Upsilon}_t. \tag{6}$$

The state update formula is:

$$\mathbf{S}_{t+1} = \mathrm{diag}(\Pi_{t+1,c}) \mathbf{S}_t \mathrm{diag}(\Upsilon_{t+1,c}) + \tilde{\mathbf{K}}_{t+1}^\top \tilde{\mathbf{V}}_{t+1}. \tag{7}$$

We left the backward pass equation in A.

Based on the chunk formula, GLA, Mamba2, Lightning Attention, xLSTM have implemented their respective algorithms. The core idea is to leverage the above chunk-based formulation to reduce the number of iterations from n to n/c, where c is the chunk size. This chunking approach significantly improves computational efficiency by enabling better utilization of GPU tensor cores. By processing multiple tokens simultaneously within each chunk, these methods achieve substantial speedups while maintaining the linear complexity advantage. This optimization is particularly effective for long sequences, where the reduction in iteration count leads to dramatically improved throughput and better hardware utilization.

3. Method

In this section, we present a unified formulation that elegantly combines the forward and backward computation of linear attention into a single function.

3.1. Unified Formulation

Upon careful examination, we observe that the forward and backward computation formulas exhibit remarkable structural similarity. Leveraging this insight, we define the function $f_{\text{recurrence}}$ in Algorithm 3. With this unifying function, we can concisely express both forward and backward computations:

$$\begin{aligned}\mathbf{O} &= f_{\text{recurrence}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{L}, \mathbf{R}, \text{False}),\\ \mathbf{dQ} &= f_{\text{recurrence}}(\mathbf{dO}, \mathbf{V}, \mathbf{K}, \mathbf{R}, \mathbf{L}, \text{False}),\\ \mathbf{dK} &= f_{\text{recurrence}}(\mathbf{V}, \mathbf{dO}, \mathbf{Q}, \mathbf{R}, \mathbf{L}, \text{True}),\\ \mathbf{dV} &= f_{\text{recurrence}}(\mathbf{K}, \mathbf{Q}, \mathbf{dO}, \mathbf{L}, \mathbf{R}, \text{True}).\end{aligned} \tag{8}$$

To further enhance computational efficiency through parallelization, we extend our unified framework to chunked processing. We define a function f_{chunk} in Algorithm 4. Similarly, our unified chunk-parallel function f_{chunk} accommodates both forward and backward computations:

$$\begin{aligned}\mathbf{O} &= f_{\text{chunk}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{L}, \mathbf{R}, \text{False}),\\ \mathbf{dQ} &= f_{\text{chunk}}(\mathbf{dO}, \mathbf{V}, \mathbf{K}, \mathbf{R}, \mathbf{L}, \text{False}),\\ \mathbf{dK} &= f_{\text{chunk}}(\mathbf{V}, \mathbf{dO}, \mathbf{Q}, \mathbf{R}, \mathbf{L}, \text{True}),\\ \mathbf{dV} &= f_{\text{chunk}}(\mathbf{K}, \mathbf{Q}, \mathbf{dO}, \mathbf{L}, \mathbf{R}, \text{True}).\end{aligned} \tag{9}$$

3.2. Efficient Implementation

Based on the unified formulation above, we can efficiently implement the forward and backward computation kernels for Linear Attention, which we call Unified Linear Attention (ULA). Following approaches similar to GLA, Lightning Attention, Mamba2 and xLSTM (Yang et al., 2023; Qin et al., 2024b; Dao & Gu, 2024; Beck et al., 2025), we decompose Algorithm 4 into two distinct kernels: one kernel computes the states, and another kernel computes the output.

For state computation, we employ a kernel function that achieves parallelization across batch dimensions, attention heads, and output-value dimensions. Within this kernel function, the algorithm iteratively executes n/c times, updating the state matrix during each iteration. For output computation, we similarly implement an efficient kernel function that enables parallel computation across batch dimensions, attention heads, number of chunks, and output-value dimensions. The internal computational mechanism is systematically decomposed into intra-chunk and inter-chunk components, thereby optimizing computational efficiency.

During the forward propagation phase, our algorithm first computes the state matrices $\mathbf{S}_t$, subsequently derives the output tensors $\mathbf{O}_t$, and strategically preserves $\mathbf{S}_t$ in memory to facilitate the backward propagation process. In the backward propagation phase, the cached state matrices $\mathbf{S}_t$ are utilized to compute the gradients $\mathbf{dQ}_t$, followed by the calculation of backward propagation states $\mathbf{dS}_t$, and ultimately the derivation of the remaining gradient components $\mathbf{dK}_t$ and $\mathbf{dV}_t$. The complete algorithm is presented in the Algorithm 1, 2, and its auxiliary algorithm is in Algorithm 5, 6, B. We implemented four variants: no decay, constant decay, scalar decay and vector decay, and collectively named the algorithm as Unified Linear Attention (ULA).

Algorithm 1 ULA Forward Pass

Require: $\mathbf{Q} \in \mathbb{R}^{n\times d}, \mathbf{K} \in \mathbb{R}^{n\times d}, \mathbf{V} \in \mathbb{R}^{n\times e}, \mathbf{\Lambda} \in \mathbb{R}^{n\times d}, \mathbf{\Gamma} \in \mathbb{R}^{n\times e}$, chunk size c.
Ensure: $\mathbf{O} \in \mathbb{R}^{n\times e}$
1: $\mathbf{\Pi} = f_{\text{preprocess}}(\mathbf{\Lambda}, \text{false}, c)$, $\mathbf{\Upsilon} = f_{\text{preprocess}}(\mathbf{\Lambda}, \text{false}, c)$.
2: $\mathbf{S} = f_{\text{state}}(\mathbf{K}, \mathbf{V}, \mathbf{\Pi}, \mathbf{\Upsilon}, \text{false}, c)$, $\mathbf{O} = f_{\text{intra-inter}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{S}, \mathbf{\Pi}, \mathbf{\Upsilon}, \text{false}, c)$.
3: **return** $\mathbf{O} \in \mathbb{R}^{n\times e}$,$\mathbf{S} \in \mathbb{R}^{n/c\times d\times e}$.

Algorithm 2 ULA Backward Pass

Require: $\mathbf{Q} \in \mathbb{R}^{n\times d}, \mathbf{K} \in \mathbb{R}^{n\times d}, \mathbf{V} \in \mathbb{R}^{n\times e}, \mathbf{\Lambda} \in \mathbb{R}^{n\times d}, \mathbf{\Gamma} \in \mathbb{R}^{n\times e}, \mathbf{dO} \in \mathbb{R}^{n\times e}$, chunk size c.
Ensure: $\mathbf{dQ} \in \mathbb{R}^{n\times d}, \mathbf{dK} \in \mathbb{R}^{n\times d}, \mathbf{dV} \in \mathbb{R}^{n\times e}$.
1: $\mathbf{\Pi} = f_{\text{preprocess}}(\mathbf{\Lambda}, \text{false}, c), \bar{\mathbf{\Pi}} = f_{\text{preprocess}}(\mathbf{\Lambda}, \text{true}, c), \mathbf{\Upsilon} = f_{\text{preprocess}}(\mathbf{\Lambda}, \text{false}, c), \bar{\mathbf{\Upsilon}} = f_{\text{preprocess}}(\mathbf{\Lambda}, \text{true}, c)$.
2: $\mathbf{dS} = f_{\text{state}}(\mathbf{Q}, \mathbf{dO}, \bar{\mathbf{\Pi}}, \bar{\mathbf{\Upsilon}}, \text{false}, c), \mathbf{dQ} = f_{\text{intra-inter}}(\mathbf{dO}, \mathbf{V}, \mathbf{K}, \mathbf{S}^\top, \mathbf{\Upsilon}, \mathbf{\Pi}, c)$.
3: $\mathbf{dK} = f_{\text{intra-inter}}(\mathbf{V}, \mathbf{dO}, \mathbf{Q}, \mathbf{dS}^\top, \bar{\mathbf{\Upsilon}}, \bar{\mathbf{\Pi}}, c), \mathbf{dV} = f_{\text{intra-inter}}(\mathbf{K}, \mathbf{Q}, \mathbf{dO}, \mathbf{dS}, \bar{\mathbf{\Pi}}, \bar{\mathbf{\Upsilon}}, c)$.
4: **return** $\mathbf{O}$.

4. Experiments

Our experiments are divided into two parts. The first part focuses on kernel benchmarking, where we compare the computation time and memory consumption of kernels. We evaluated constant decay, scalar decay, vector decay, and no decay configurations, benchmarking against GLA (Yang et al., 2023; Yang & Zhang, 2024), Mamba2 (Dao & Gu, 2024), Lightning Attention (Qin et al., 2024b), and xLSTM (Beck et al., 2025).

The second part validates the end-to-end convergence properties of the kernel through a series of language modeling experiments using the fineweb-edu-10B (Penedo et al., 2024) dataset. We compare the performance of several models with different kernels, including Linear Transformer (Katharopoulos et al., 2020) (no decay), TNL (Qin et al., 2024b) (constant decay), Mamba2 (Dao & Gu, 2024) (scalar decay), HGRN2 (Qin et al., 2024c) (vector decay).

Our experiments involve training language models of varying parameter sizes, specifically 160M, 1.45B, and 2.8B parameters. Detailed configurations for these models are provided in Table 4. We employ the GPT2-Tokenizer for tokenization.

The training process is governed by several key hyperparameters: a global batch size of 256, a sequence length of 2048, and the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate is set to 3×10^{-4}. We utilize the WSD scheduler (Hu et al., 2024) and train the models for 20,000 steps.

For the kernel part, we implement based on Triton (Tillet et al., 2019), and all benchmarks are conducted on a single A100 GPU. For the language model part, we base our implementation on Flame (Zhang et al., 2025) and PyTorch (Paszke et al., 2019) frameworks. All models are trained using 8 NVIDIA A100 GPUs. Upon completion of training, we evaluate the models using lm-eval-harness (Gao et al., 2021) to perform zero-shot testing.

4.1. Kernel Benchamrk

We conduct a comprehensive evaluation of our proposed method ULA against state-of-the-art linear attention kernels, analyzing both computational efficiency and memory utilization. Our ex-

Table 1. Performance comparison of different model architectures across various model sizes, where implementations without a star represent official implementations, while those with a star represent our implementations. AVG represents the average perplexity value (lower is better) or average correct score (higher is better). As shown in the table, our implemented versions are comparable to official ones in terms of both loss and zero-shot capabilities.

Method	Loss	PPL ↓			Accuracy ↑							
		Wiki	LMB	AVG	PIQA	Hella	Wino	ARC-e	ARC-c	OBQA	SOQA	AVG
						1.45B						
LA	2.63	25.2	34.6	29.9	69.6	43.1	53.4	63.1	30.4	34.0	39.5	47.6
LA*	2.63	25.1	34.6	29.8	68.6	43.4	52.8	63.8	30.1	35.2	39.1	47.6
TNL	2.57	24.1	28.4	27.1	69.8	46.0	51.7	64.9	31.6	34.4	39.6	48.3
TNL*	2.57	24.1	30.1	28.1	69.9	46.0	54.4	65.6	31.9	37.6	40.0	49.3
Mamba2	2.60	23.9	27.5	25.7	69.7	45.0	51.8	63.4	31.2	34.6	39.9	47.9
Mamba2*	2.60	23.9	28.5	26.2	69.2	44.9	51.4	63.6	30.6	34.8	39.7	47.7
HGRN2	2.56	23.3	24.7	24.0	69.4	46.6	51.1	66.3	30.8	36.8	40.4	48.8
HGRN2*	2.56	23.3	24.7	24.0	70.0	46.3	52.2	65.7	30.6	35.0	39.7	48.5

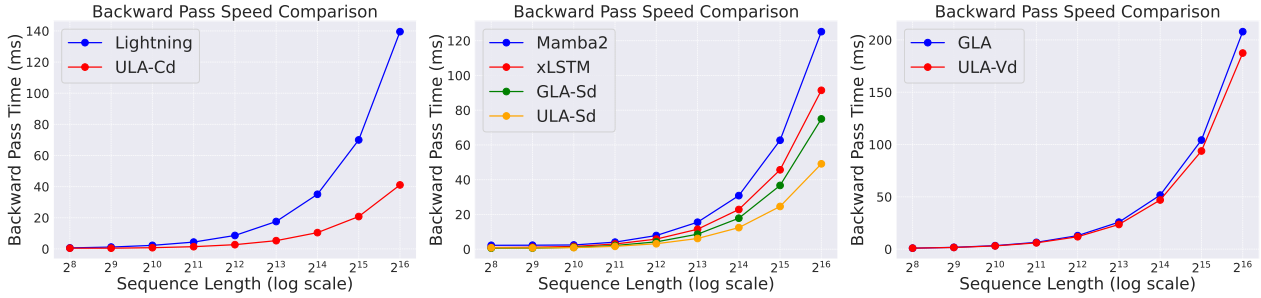

Figure 1. Quantitative analysis of our proposed method with state-of-the-art linear attention kernels, organizing the comparison into three parts: constant decay (Cd) versus Lightning Attention (left figure); scalar decay (Sd) versus Mamba2, xLSTM, and GLA-Sd (middle figure); and vector decay (Vd) versus GLA (right figure). We conducted tests on both the forward pass and the backward pass. As demonstrated by the figures, our method achieves faster speeds in almost all scenarios.

perimental comparison is structured into three distinct categories based on the decay mechanism:

- Constant decay (CD) variants compared against Lightning Attention;
- Scalar decay (SD) implementations benchmarked against Mamba2, xLSTM, and GLA-Sd;
- Vector decay (VD) formulations evaluated against GLA;

For each kernel variant, we rigorously measure both forward pass and backward pass performance to provide a complete assessment of training and inference capabilities. The quantitative results, as shown in Figure 1, 3 and Figure 2, demonstrate that despite simplifying the kernel implementation, our method outperforms current state-of-the-art approaches in both speed and memory usage across all sequence lengths. This further emphasizes the effectiveness of our proposed method.

We notice that our method exhibits only marginal differences in performance compared to GLA in the speed comparison experiment. We hypothesize that this limited impact may stem from the fact that a portion of the vector decay computation does not utilize tensor cores (Yang et al., 2023).

4.2. Language Modeling

We evaluated several Linear Attention methods in language models, specifically examining Linear Transformer (no decay), TNL (constant decay), Mamba2 (scalar decay), and HGRN2 (vector decay) models. Our testing included both official implementations—such as Lightning Attention for Linear Transformer and TNL, the Mamba kernel for Mamba2, and the GLA kernel for HGRN2—as well as custom kernel implementations. The results, presented in Table 1, 3, show that our implementations deliver performance comparable to the official versions in terms of loss and zero-shot effectiveness, with certain models even surpassing their official counterparts. These findings highlight the end-to-end convergence of our kernel implementations.

5. Conclusion

In conclusion, this paper introduces a unified functional framework for both forward and backward propagation in linear complexity attention mechanisms, simplifying the design and implementation process. By consolidating the traditionally complex and time-consuming stages into a single function, our approach significantly reduces development effort. Experimental results across various linear attention variants demonstrate that our method not only streamlines kernel implementation but also improves performance in terms of both speed and memory efficiency, surpassing existing implementations. Through end-to-end training of language models and zero-shot testing, we verified the correctness and robustness of the kernel implementation. This work paves the way for more efficient exploration and development of new linear attention methods.

References

Beck, M., Poppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M. K., Klambauer, G., Brandstetter, J., and Hochreiter, S. xlstm: Extended long short-term memory. *ArXiv*, abs/2405.04517, 2024a. URL https://api.semanticscholar.org/CorpusID:269614336.

Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. xlstm: Extended long short-term memory. In *Thirty-eighth Conference on Neural Information Processing Systems*, 2024b. URL https://arxiv.org/abs/2405.04517.

Beck, M., Pöppel, K., Lippe, P., and Hochreiter, S. Tiled Flash Linear Attention: More efficient linear rnn and xlstm kernels. *arXiv*, 2503.14376, 2025. URL https://arxiv.org/abs/2503.14376.

Chou, Y., Yao, M., Wang, K., Pan, Y., Zhu, R.-J., Wu, J., Zhong, Y., Qiao, Y., XU, B., and Li, G. MetaLA: Unified optimal linear approximation to softmax attention map. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=Y8YVCOMEpz.

Dao, T. and Gu, A. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *ArXiv*, abs/2405.21060, 2024. URL https://api.semanticscholar.org/CorpusID:270199762.

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

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Hu, S., Tu, Y., Han, X., Cui, G., He, C., Zhao, W., Long, X., Zheng, Z., Fang, Y., Huang, Y., Zhang, X., Thai, Z. L., Wang, C., Yao, Y., Zhao, C., Zhou, J., Cai, J., Zhai, Z., Ding, N., Jia, C., Zeng, G., dahai li, Liu, Z., and Sun, M. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=3X2L2TFr0f.

Hua, W., Dai, Z., Liu, H., and Le, Q. V. Transformer quality in linear time. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9099–9117. PMLR, 2022.

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.

Mao, H. H. Fine-tuning pre-trained transformers into decaying fast weights. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 10236–10242, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.697.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gülçehre, Ç., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 26670–26698. PMLR, 2023. URL https://proceedings.mlr.press/v202/orvieto23a.html.

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

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The fineweb datasets: Decanting the web for the finest text data at scale, 2024.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., V., K. K. G., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Mantri, K. S. I., Mom, F., Saito, A., Tang, X., Wang, B., Wind, J. S., Wozniak, S., Zhang, R., Zhang, Z., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R. RWKV: reinventing rnns for the transformer era. *CoRR*, abs/2305.13048, 2023a. doi: 10.48550/ARXIV.2305.13048.

Peng, B., Alcaide, E., Anthony, Q. G., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M. N., Derczynski, L., Du, X., Grella, M., GV, K. 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. S., Woźniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R.-J. RWKV: Reinventing RNNs for the transformer era. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023b. URL https://openreview.net/forum?id=7SaXczaBpG.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Qin, Z., Yang, S., and Zhong, Y. Hierarchically gated recurrent neural network for sequence modeling. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/694be3548697e9cc8999d45e8d16fe1e-Abstract-Conference.html.

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

Qin, Z., Sun, W., Li, D., Shen, X., Sun, W., and Zhong, Y. Various lengths, constant speed: Efficient language modeling with lightning attention. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=Lwm6TiUP4X.

Qin, Z., Yang, S., Sun, W., Shen, X., Li, D., Sun, W., and Zhong, Y. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024c.

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.

Tillet, P., Kung, H., and Cox, D. D. Triton: an intermediate language and compiler for tiled neural network computations. In Mattson, T., Muzahid, A., and Solar-Lezama, A. (eds.), *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019*, pp. 10–19. ACM, 2019. doi: 10.1145/3315508.3329973.

Yang, S. and Zhang, Y. FLA: A Triton-Based Library for Hardware-Efficient Implementations of Linear Attention Mechanism, January 2024. URL https://github.com/sustcsonglin/flash-linear-attention.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention transformers with hardware-efficient training. *CoRR*, abs/2312.06635, 2023. doi: 10.48550/ARXIV.2312.06635. URL https://doi.org/10.48550/arXiv.2312.06635.

Zhang, Y., Yang, S., Zhu, R.-J., Zhang, Y., Cui, L., Wang, Y., Wang, B., Shi, F., Wang, B., Bi, W., Zhou, P., and Fu, G. Gated slot attention for efficient linear-time sequence modeling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=jY4PhQibmg.

Zhang, Y., Yang, S., Guo, H., rakkit, and Han, J. *fla-org/flame*. 3 2025. URL https://github.com/fla-org/flame.

Appendix

Table 2. **Mapping of mainstream linear attention methods to our unified formulation.** This table shows how different linear complexity sequence models can be expressed using the unified form $\mathbf{s}_t = \mathbf{l}_t \mathbf{s}_{t-1} \mathbf{r}_t + \mathbf{k}_t \mathbf{v}_t^\top, \mathbf{o}_t^\top = \mathbf{q}_t^\top \mathbf{s}_t$. For each method, we show the corresponding Query ($\mathbf{q}_t$), Key ($\mathbf{k}_t$), Value ($\mathbf{v}_t$), Left Decay ($\mathbf{l}_t$), and Right Decay ($\mathbf{r}_t$) components.

Methods	Query	Key	Value	Left Decay	Right Decay
Linear Attention (Katharopoulos et al., 2020)	$\mathbf{q}_t$	$\mathbf{k}_t$	$\mathbf{v}_t$	1	1
TNL/RetNet (Qin et al., 2024b; Sun et al., 2023)	$\mathbf{q}_t$	$\mathbf{k}_t$	$\mathbf{v}_t$	λ	1
RFA/xLSTM (Beck et al., 2025; Peng et al., 2021)	$\mathbf{q}_t$	$\mathbf{k}_t$	$\mathbf{v}_t$	λ_t	1
Mamba2 (Dao & Gu, 2024)	$\mathbf{C}_t$	$\mathbf{B}_t$	$\mathbf{x}_t$	$\mathbf{A}_t$	1
RWKV (Peng et al., 2023b)	$\mathbf{R}_t$	k_t	$\mathbf{v}_t$	$\mathrm{diag}(\mathbf{w}_t)$	1
GLA (Yang et al., 2023)	$\mathbf{q}_t$	$\mathbf{k}_t$	$\mathbf{v}_t$	$\mathrm{diag}(\gamma_t)$	1
HGRN2/MetaLA (Chou et al., 2024; Qin et al., 2024c)	$\mathbf{q}_t$	$1-\lambda_t$	$\mathbf{v}_t$	$\mathrm{diag}(\lambda_t)$	1
DFW (Mao, 2022)	$\mathbf{q}_t$	$\mathbf{k}_t$	$\mathbf{v}_t$	$\mathrm{diag}(\lambda_t)$	$\mathrm{diag}(\gamma_t)$

A. Linear Attention Backward Pass

Similarly, for the backward pass, we first the common terms used across all gradient computations:

$$\mathbf{d}\bar{\mathbf{O}}_t = \mathbf{d}\mathbf{O}_t \odot \boldsymbol{\Upsilon}_t, \quad \mathbf{d}\tilde{\mathbf{O}}_t = \mathbf{d}\mathbf{O}_t / \bar{\boldsymbol{\Upsilon}}_t, \tilde{\mathbf{Q}}_t = \mathbf{Q}_t / \bar{\boldsymbol{\Pi}}_t. \tag{10}$$

We then compute the gradients with respect to $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ using these terms:

$$\begin{aligned}
\mathbf{d}\bar{\mathbf{Q}}_t &= [[\mathbf{d}\bar{\mathbf{O}}_t \bar{\mathbf{V}}_t^\top] \odot \mathbf{M}]\bar{\mathbf{K}}_t + \mathbf{d}\bar{\mathbf{O}}_t \mathbf{S}_t^\top, \mathbf{d}\mathbf{Q}_t = \mathbf{d}\bar{\mathbf{Q}}_t \odot \boldsymbol{\Pi}_t, \\
\mathbf{d}\bar{\mathbf{K}}_t &= [[\mathbf{d}\bar{\mathbf{O}}_t \bar{\mathbf{V}}_t^\top] \odot \mathbf{M}]^\top \tilde{\mathbf{Q}}_t + \tilde{\mathbf{V}}_t \mathbf{d}\mathbf{S}_t^\top, \mathbf{d}\mathbf{K}_t = \mathbf{d}\bar{\mathbf{K}}_t \odot \bar{\boldsymbol{\Pi}}_t, \\
\mathbf{d}\bar{\mathbf{V}}_t &= [[\tilde{\mathbf{Q}}_t \tilde{\mathbf{K}}_t^\top] \odot \mathbf{M}]^\top \mathbf{d}\tilde{\mathbf{O}}_t + \tilde{\mathbf{K}}_t \mathbf{d}\mathbf{S}_t, \mathbf{d}\mathbf{V}_t = \mathbf{d}\bar{\mathbf{V}}_t \odot \bar{\boldsymbol{\Upsilon}}_t.
\end{aligned} \tag{11}$$

The state gradient is computed recursively:

$$\mathbf{d}\mathbf{S}_i = \mathrm{diag}(\bar{\boldsymbol{\Pi}}_{i,c}) \mathbf{d}\mathbf{S}_{i+1} \mathrm{diag}(\bar{\boldsymbol{\Gamma}}_{i,c}) + \bar{\mathbf{Q}}_i \mathbf{d}\bar{\mathbf{O}}_i^\top. \tag{12}$$

B. Algorithm

Algorithm 3 Recurrence Linear Attention Function $f_{\text{recurrence}}$

Input: $\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{n \times e}, \boldsymbol{\Lambda} \in \mathbb{R}^{n \times d}, \boldsymbol{\Gamma} \in \mathbb{R}^{n \times e}$, reverse $\in \{\text{True}, \text{False}\}$
Output: $\mathbf{O} \in \mathbb{R}^{n \times e}$

Initialize $\mathbf{s}_0$ or $\mathbf{s}_{n+1}$ as zero matrix;
$\mathbf{l}_t = \mathrm{diag}(\lambda_t), \mathbf{r}_t = \mathrm{diag}(\gamma_t)$;
if reverse $=$ False **then**
    Initialize $\mathbf{s}_0$ as zero matrix, $\Delta = 1, t = 1$.
else
    Initialize $\mathbf{s}_{n+1}$ as zero matrix, $\Delta = -1, t = n$.
end if
for $i = 1$ to n **do**
    $\mathbf{s}_t = \mathbf{l}_t \mathbf{s}_{t-\Delta} \mathbf{r}_t + \mathbf{k}_t \mathbf{v}_t^\top, \mathbf{o}_t^\top = \mathbf{q}_t^\top \mathbf{s}_t, t = t + \Delta$.
end for
return $\mathbf{O}$

Algorithm 4 Chunk-Parallel Linear Attention Function f_{chunk}

Require: $\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{n \times e}, \mathbf{\Lambda} \in \mathbb{R}^{n \times d}, \mathbf{\Gamma} \in \mathbb{R}^{n \times e}$, reverse $\in \{\text{True}, \text{False}\}$
Ensure: $\mathbf{O} \in \mathbb{R}^{n \times e}$

Divide sequence into chunks of size c: $\{\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i, \mathbf{\Lambda}_i, \mathbf{\Gamma}_i\}$ for $i = 1, \ldots, m, m = n/c$.
for all $i = 1, \ldots, m$ in parallel **do**
  $\mathbf{\Pi}_i = \exp(\text{cumsum}(\log \mathbf{\Lambda}_i, \text{reverse}))$, $\mathbf{\Upsilon}_i = \exp(\text{cumsum}(\log \mathbf{\Gamma}_i, \text{reverse}))$,
  $\bar{\mathbf{Q}}_i = \mathbf{Q}_i \odot \mathbf{\Pi}_i, \bar{\mathbf{K}}_i = \mathbf{K}_i / \mathbf{\Pi}_i, \bar{\mathbf{V}}_i = \mathbf{V}_i / \mathbf{\Upsilon}_i, \tilde{\mathbf{K}}_i = \mathbf{K}_i \odot (\Pi_{i,c} / \mathbf{\Pi}_i), \tilde{\mathbf{V}}_i = \mathbf{V}_i \odot (\Upsilon_{i,c} / \mathbf{\Upsilon}_i)$.
end for
if reverse $=$ False **then**
  Initialize $\mathbf{S}_0$ as zero matrix, $\mathbf{M}_{ij} = 1, i \geq j; 0, j < i, \Delta = 1, t = 1$.
else
  Initialize $\mathbf{S}_{m+1}$ as zero matrix, $\mathbf{M}_{ij} = 1, i \leq j; 0, j > i, \Delta = -1, t = m$.
end if
for $i = 1, \ldots, m$ **do**
  $\mathbf{S}_t = \text{diag}(\Pi_{t,c}) \mathbf{S}_{t-\Delta} \text{diag}(\Upsilon_{t,c}) + \tilde{\mathbf{K}}_t^\top \tilde{\mathbf{V}}_t, t = t + \Delta$.
end for
for all $t = 1, \ldots, m$ in parallel **do**
  $\bar{\mathbf{O}}_t = \bar{\mathbf{Q}}_t \mathbf{S}_t + [[\bar{\mathbf{Q}}_t \bar{\mathbf{K}}_t^\top] \odot \mathbf{M}] \bar{\mathbf{V}}_t, \mathbf{O}_t = \bar{\mathbf{O}}_t \odot \mathbf{\Upsilon}_t$.
end for
return $\mathbf{O} = [\mathbf{O}_1^\top; \ldots; \mathbf{O}_m^\top]^\top$.

Algorithm 5 Chunk Decay Preprocess Function $f_{\text{preprocess}}$

Require: $\mathbf{\Lambda} \in \mathbb{R}^{n \times d}$, reverse $\in \{\text{True}, \text{False}\}$, chunk size c.
Ensure: $\mathbf{\Pi} \in \mathbb{R}^{n \times d}$

Divide sequence into chunks of size c: $\{\mathbf{\Lambda}_t\}$ for $t = 1, \ldots, m, m = n/c$.
for all $t = 1, \ldots, m$ in parallel **do**
  $\mathbf{\Pi}_t = \exp(\text{cumsum}(\log \mathbf{\Lambda}_t, \text{reverse}))$.
end for
return $\mathbf{\Pi}$.

Algorithm 6 Chunk State Function f_{state}

Require: $\mathbf{K} \in \mathbb{R}^{n \times d}, \mathbf{V} \in \mathbb{R}^{n \times e}, \mathbf{\Pi} \in \mathbb{R}^{n \times d}, \mathbf{\Upsilon} \in \mathbb{R}^{n \times e}$, reverse $\in \{\text{True}, \text{False}\}$, chunk size c.
Ensure: $\mathbf{State} \in \mathbb{R}^{n/c \times d \times e}$.

Divide sequence into chunks of size c: $\{\mathbf{K}_t, \mathbf{V}_t, \mathbf{\Pi}_t, \mathbf{\Upsilon}_t\}$ for $t = 1, \ldots, m, m = n/c$.
if reverse $=$ False **then**
  Initialize $\mathbf{S}_0$ as zero matrix.
else
  Initialize $\mathbf{S}_{m+1}$ as zero matrix.
end if
for $i = 1, \ldots, m$ **do**
  $\tilde{\mathbf{K}}_t = \mathbf{K}_t \odot (\Pi_{t,c} / \mathbf{\Pi}_t), \tilde{\mathbf{V}}_t = \mathbf{V}_t \odot (\Upsilon_{t,c} / \mathbf{\Upsilon}_t)$,
  $\mathbf{S}_t = \text{diag}(\Pi_{t,c}) \mathbf{S}_{t-\Delta} \text{diag}(\Upsilon_{t,c}) + \tilde{\mathbf{K}}_t^\top \tilde{\mathbf{V}}_t, t = t + \Delta$.
end for
return $\mathbf{S} = [\mathbf{S}_1, \ldots, \mathbf{S}_m]$.

Algorithm 7 Chunk Intra Inter Function $f_{\text{intra}-\text{inter}}$

Require: $\mathbf{Q} \in \mathbb{R}^{n\times d}, \mathbf{K} \in \mathbb{R}^{n\times d}, \mathbf{V} \in \mathbb{R}^{n\times e}, \mathbf{S} \in \mathbb{R}^{n/c\times d\times e}, \mathbf{\Pi} \in \mathbb{R}^{n\times d}, \mathbf{\Upsilon} \in \mathbb{R}^{n\times e}$.
Ensure: $\mathbf{O} \in \mathbb{R}^{n\times e}$

Divide sequence into chunks of size c: $\{\mathbf{Q}_t, \mathbf{S}_t, \mathbf{\Pi}_t, \mathbf{\Upsilon}_t\}$ for $t = 1, \ldots, m, m = n/c$.
if reverse = False, $\mathbf{M}_{ij} = 1, i \geq j; 0, j < i$, else $\mathbf{M}_{ij} = 1, i \leq j; 0, j > i$.
for all $t = 1, \ldots, m$ in parallel **do**
    $\bar{\mathbf{Q}}_t = \mathbf{Q}_t \odot \mathbf{\Pi}_t, \bar{\mathbf{K}}_t = \mathbf{K}_t / \mathbf{\Pi}_t, \bar{\mathbf{V}}_t = \mathbf{V}_t / \mathbf{\Upsilon}_t,$
    $\bar{\mathbf{O}}_t = [[\bar{\mathbf{Q}}_t \bar{\mathbf{K}}_t^\top] \odot \mathbf{M}]\bar{\mathbf{V}}_t + \bar{\mathbf{Q}}_t \mathbf{S}_t, \mathbf{O}_t = \bar{\mathbf{O}}_t \odot \mathbf{\Upsilon}_t.$
end for
return $\mathbf{O} = [\mathbf{O}_1^\top; \ldots; \mathbf{O}_m^\top]^\top$.

C. More Experiments

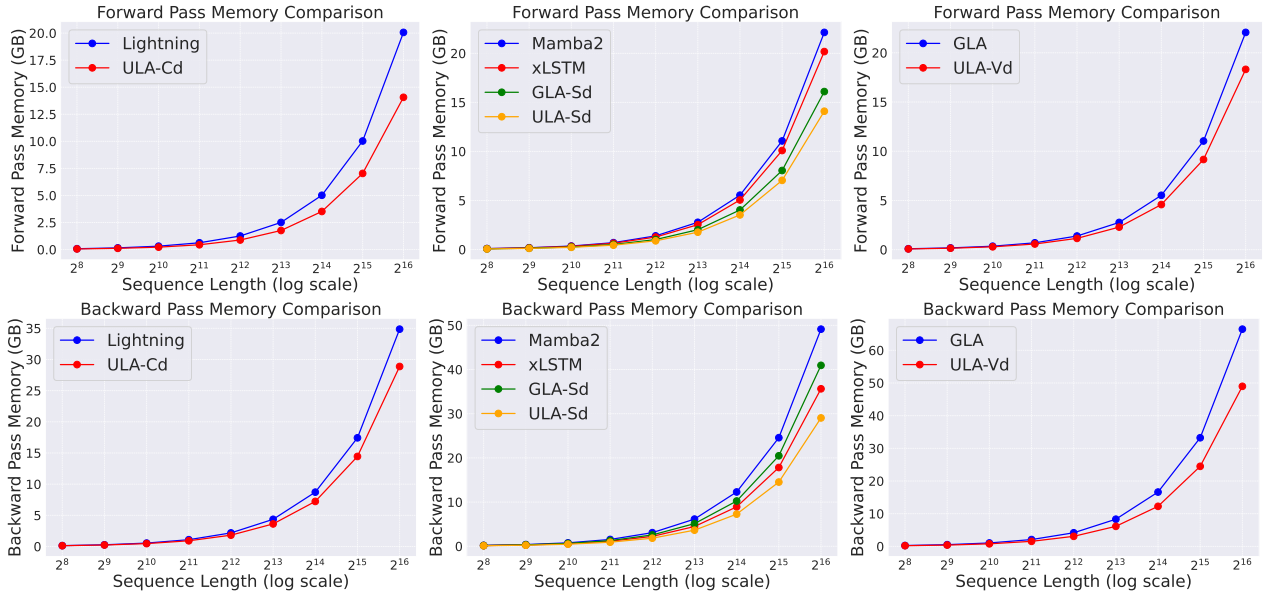

Figure 2. Quantitative analysis of our proposed method with state-of-the-art linear attention kernels in terms of memory efficiency, dividing the analysis into three parts: constant decay (Cd) versus Lightning Attention (left figure); scalar decay (Sd) versus Mamba2, xLSTM, and GLA-Sd (middle figure); and vector decay (Vd) versus GLA (right figure). We measured memory consumption during both forward pass and backward pass operations. The results clearly demonstrate that our method achieves lower memory usage in nearly all scenarios.

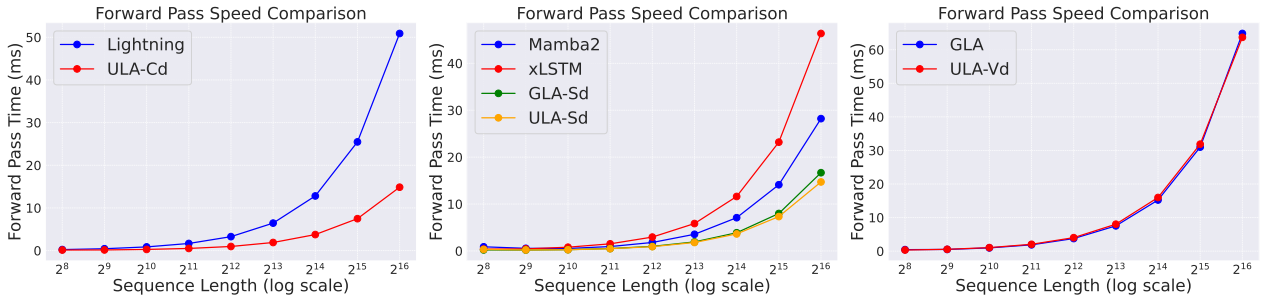

Figure 3. Quantitative analysis of our proposed method with state-of-the-art linear attention kernels, organizing the comparison into three parts: constant decay (Cd) versus Lightning Attention (left figure); scalar decay (Sd) versus Mamba2, xLSTM, and GLA-Sd (middle figure); and vector decay (Vd) versus GLA (right figure). We conducted tests on both the forward pass and the backward pass. As demonstrated by the figures, our method achieves faster speeds in almost all scenarios.

Table 3. Performance comparison of different model architectures across various model sizes, where implementations without a star represent official implementations, while those with a star represent our implementations. AVG represents the average perplexity value (lower is better) or average correct score (higher is better). As shown in the table, our implemented versions are comparable to official ones in terms of both loss and zero-shot capabilities.

Method	**Loss**	**PPL ↓**			**Accuracy ↑**							
		Wiki	**LMB**	**AVG**	**PIQA**	**Hella**	**Wino**	**ARC-e**	**ARC-c**	**OBQA**	**SOQA**	**AVG**
					0.16B							
LA	3.06	45.9	174.5	110.2	63.2	30.8	50.9	52.4	25.3	30.0	36.1	41.3
LA*	3.06	45.7	156.3	101.0	63.3	30.7	51.0	52.7	23.5	29.8	35.8	41.0
TNL	2.98	41.4	112.0	79.6	63.8	32.3	48.7	53.7	25.6	30.6	36.2	41.6
TNL*	2.98	40.6	108.3	75.1	63.9	32.4	48.9	53.2	24.7	29.8	37.3	41.4
Mamba2	2.96	39.1	95.0	67.0	64.3	33.0	50.1	51.8	25.9	32.6	36.3	42.0
Mamba2*	2.96	38.8	103.4	71.1	63.4	32.4	51.9	52.7	25.3	31.6	37.4	42.1
HGRN2	2.98	41.4	108.8	75.1	65.1	32.4	51.5	53.2	26.6	31.4	37.7	42.6
HGRN2*	2.98	41.4	111.5	76.4	64.0	32.2	49.7	53.2	25.9	30.8	38.1	42.0
					2.8B							
LA	2.52	22.6	26.5	24.6	70.3	47.5	50.7	67.0	33.2	36.0	40.2	49.3
LA*	2.51	22.3	25.6	24.0	70.2	47.5	52.4	66.9	32.2	36.4	39.9	49.4
TNL	2.47	21.5	22.0	21.9	70.3	50.0	54.5	68.4	34.0	35.6	41.8	50.6
TNL*	2.46	21.6	22.3	21.7	71.1	49.6	55.3	66.9	32.6	37.2	39.6	50.3
Mamba2	2.53	21.9	23.6	22.8	71.1	48.6	53.4	64.8	32.2	38.2	39.2	49.6
Mamba2*	2.53	21.9	23.5	22.7	69.5	48.5	54.7	66.7	31.9	35.4	39.8	49.5
HGRN2	2.45	20.9	19.7	20.3	70.6	50.5	53.0	68.9	34.8	39.0	40.4	51.0
HGRN2*	2.45	20.9	20.0	20.4	70.7	50.6	52.8	68.8	32.7	38.4	40.1	50.6

Table 4. Model configurations for different parameter sizes, where we show the parameters for 0.16B, 1.45B, and 2.8B models, in which L.R. represents learning rate.

Params(B)	**Layers**	**Hidden Dim**	**Num Heads**	**L.R.**	**Batch Size**	**SeqLen**	**GPUs**
0.16	12	768	12	3E-04	32	2048	8
1.45	24	2048	16	3E-04	32	2048	8
2.8	32	2560	20	3E-04	32	2048	8

D. Illustration

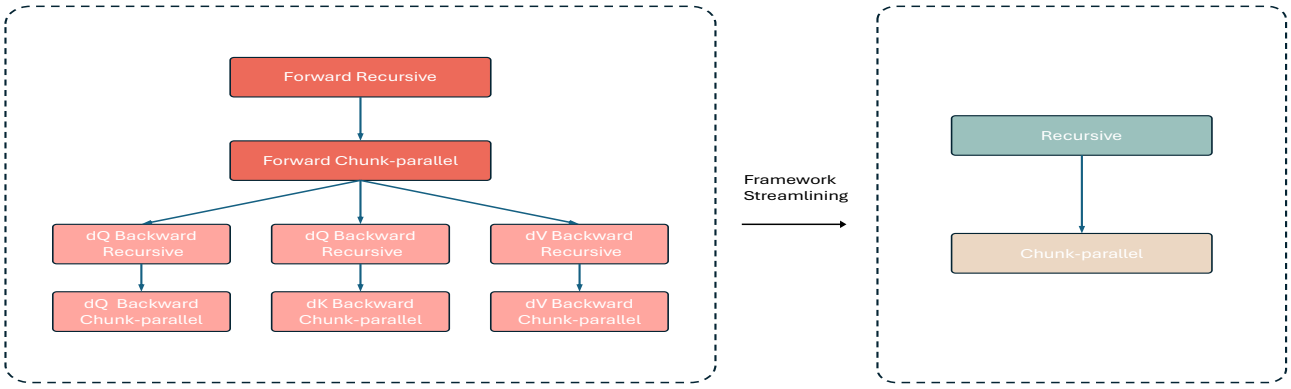

Figure 4. **Kernel development workflow diagram**. Left: Traditional design pipeline, progressing from forward recursive, to O's forward chunk-parallel, followed by backward recursive, and finally to the backward chunk-parallel for dQ, dK, and dV. Right: Proposed design pipeline, transitioning from recursion to chunk-parallelism, streamlining both forward and backward propagation within a unified function.

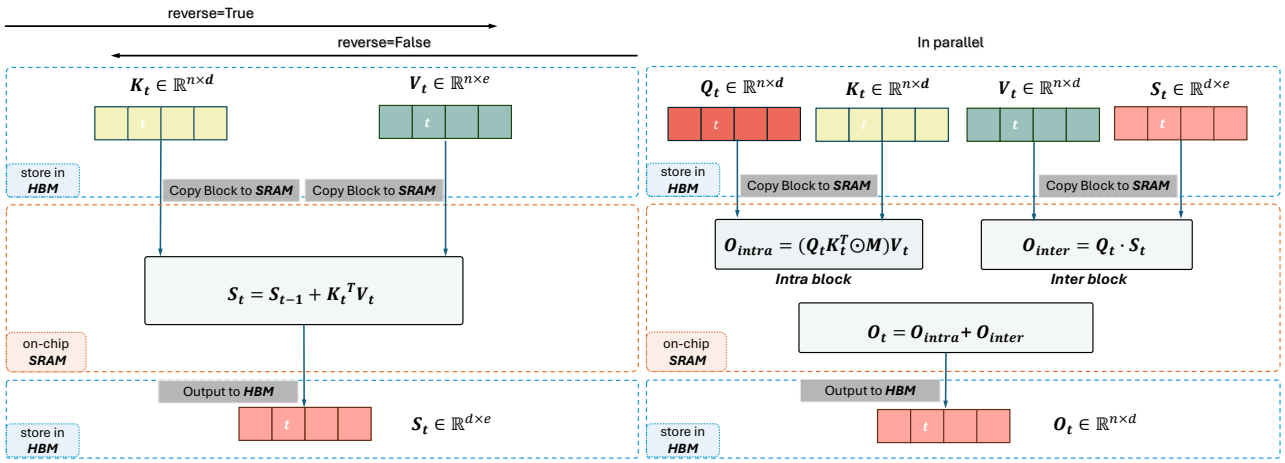

Figure 5. **ULA is detailed in its algorithmic schematic**. For ease of understanding, here we only show the version without decay. reverse=False indicates iteration from small to large indices, while reverse=True indicates iteration from large to small indices. Left figure: During the t-th iteration, the tiling blocks of matrices $\mathbf{K}_t$ and $\mathbf{V}_t$ are transferred from High Bandwidth Memory (HBM) to Static Random-Access Memory (SRAM). Within the SRAM, the State $\mathbf{S}_t$ is updated, then written back from SRAM to HBM. Right figure: After computing and storing the States, the matrix $\mathbf{Q}_t$ is loaded in parallel from High Bandwidth Memory (HBM) to Static Random-Access Memory (SRAM). $\mathbf{O}_{intra}$ and $\mathbf{O}_{inter}$ are computed independently, finally producing the output $\mathbf{O}_t$, which is written back from SRAM to HBM.