LEETDECODING: AN EXPERIMENTAL SURVEY AND SCALABILITY EVALUATION FOR PREFILLING WITH CAUSAL LINEAR ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Significant progress has been made in accelerating transformer-based large language models (LLMs). Among various promising approaches, *exponentially decaying causal linear attention* has emerged as a highly effective operator, yet its existing implementations remain fragmented across the literature. In this paper, we present an experimental survey of this fundamental operator for LLM prefill inference. We also introduce `LeetDecoding`, the first comprehensive library and benchmark providing extensive computation routines and a unified testbed for scalability evaluation. `LeetDecoding` addresses multiple critical gaps in the field: the absence of rigorous complexity analyses, the lack of a systematic consolidation of existing computation methods (previously scattered as secondary contributions without direct comparison), and the dearth of extensive empirical evaluations. Our findings reveal that no single computation method universally dominates across diverse configurations, and that hardware-aware optimizations are paramount for substantially improving overall efficiency.

1 INTRODUCTION

The transformer architecture (Vaswani et al., 2017) has become the foundational framework for large language models (LLMs; Yang et al. 2025), a paradigm shift prominently accelerated by the launch of ChatGPT (OpenAI, 2022). Its core component, standard softmax attention, effectively captures local and long-range dependencies (Deng et al., 2022; Zhao et al., 2022; Badaro & Papotti, 2022; Chen et al., 2023; Wu et al., 2023; Lyu et al., 2025; Wang et al., 2025) and supports parallelized training. However, it is fundamentally bottlenecked by a quadratic complexity in sequence length (Zuo et al., 2024).

Linear attention has emerged as a promising alternative, particularly for encoder transformers with *bidirectional attention*. It mitigates this bottleneck by replacing the exponential similarity function with a low-rank dot product over the key and query token vectors (see Appendix 2 for background). Representative methods explore diverse approximation strategies: Performer (Choromanski et al., 2021) and random feature attention (Peng et al., 2021) decompose the attention matrix using learned or random feature maps (Rahimi & Recht, 2007); Skyformer (Chen et al., 2021) and LARA (Zheng et al., 2022) utilize Nyström -based techniques; Linformer (Wang et al., 2020) and Skeinformer (Chen et al., 2022) leverage Johnson-Lindenstrauss projections (Johnson & Lindenstrauss, 1984; Dzahini & Wild, 2024) and randomized sketching; and Transformer-VQ (Lingle, 2024) approximates keys via vector quantization.

Decoder transformers rely on causal attention for autoregressive modeling (see Appendix H.1). While a naive dense implementation of causal linear attention incurs an $\mathcal{O}(N^2)$ cost (for a length-N input sequence), formulating the operator to avoid materializing the dense attention matrix reduces this complexity to $\mathcal{O}(N)$ time for fixed feature dimensions (eq. (1)). As demonstrated in Figure 1, this algorithmic choice profoundly impacts model-level runtime.

To bridge the performance gap between standard causal attention and **causal linear attention**, recent advances have integrated an *exponential decay of attention scores*. This technique has become the de facto standard for *linear transformers* (i.e., linear-attention transformers trained from scratch). Architectures such as RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and TransNormer-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Overview of computation methods for causal linear attention. We detail both time and space complexities, where N denotes the sequence length, and r, d represent the fixed matrix dimensions defined in eq. (1). For block-based methods, the displayed bounds assume the block size $B_c = \mathcal{O}(\min\{r, d\})$. Space bounds account for input, output, and auxiliary tensors; the recursion bound assumes a fixed termination block size.

Computation Method	Time / Space Complexity
`vanilla` (eq. (1))	$\mathcal{O}(N^2(r+d))$ / $\mathcal{O}(N^2 + N(r+d))$
`causal-dot-product` (Vyas et al., 2020)	$\mathcal{O}(Nrd)$ / $\mathcal{O}(N(r+d) + rd)$
`block-based` (Lingle, 2024)	$\mathcal{O}(Nrd)$ / $\mathcal{O}(N(r+d) + rd)$
`Lightning Attention-2` (Qin et al., 2024b)	$\mathcal{O}(Nrd)$ / $\mathcal{O}(N(r+d) + rd)$
`recursion` (Han et al., 2024)	$\mathcal{O}(Nrd \log N)$ / $\mathcal{O}(N(r+d) + rd)$

LLM (Qin et al., 2024a) merge causal masking and relative distance-based exponential decay into a single matrix. This formulation effectively mitigates attention dilution while preserving computational efficiency. Notably, such exponentially decayed models can surpass conventional softmax-attention architectures in both accuracy and speed (Qin et al., 2024a). Furthermore, this paradigm has scaled to massive commercial LLMs, including Minimax-M1 (MiniMax, 2025, 456 billion parameters), Qwen3-Next (QwenTeam, 2025, 80 billion parameters), and Kimi Linear (Kimi Team, 2025, 48 billion parameters).

Despite the rapid development of linear transformers, existing computation methods remain fragmented across the literature (summarized in Table 1). This fragmentation necessitates a unified comparative study and an accessible library for causal linear attention. Notably, most prior works prioritize the modeling capacity of their specific architectural proposals rather than algorithmic efficiency and computational design. The **`recursion`** algorithm, for example, was originally designed to compute sparse approximations of causal attention (Han et al., 2024), whereas the **`causal-dot-product`** (Vyas et al., 2020) and **`block-based`** (Lingle, 2024) algorithms were introduced merely as secondary contributions to support novel attention variants.

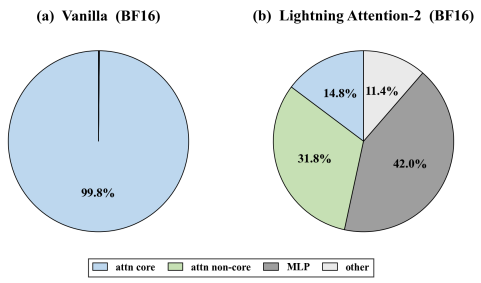


Figure 1: Forward-pass latency breakdown for TransNormerLLM-7B (Qin et al., 2024a) (BF16, batch size 1). Total runtime is partitioned into four components: causal attention core (`attn core`), remaining attention (`attn non-core`), `MLP`, and `other`. While the dense **`vanilla`** implementation devotes 99.8% of total runtime to the `attn core` at long sequences, **`Lightning Attention-2`** reduces this proportion to 14.8%, effectively shifting the computational bottleneck away from the attention mechanism.

Consequently, these methods are typically evaluated only within the narrow context of their parent modules, leaving standardized cross-method comparisons absent from the literature. Furthermore, many existing implementations are tightly coupled with their original attention modules and are not directly compatible with the broader linear-transformer models evaluated here. While recent studies provide IO-aware implementations of chunk-wise linear attention on modern GPUs (Yang et al., 2024), a comprehensive framework remains lacking. Our benchmark bridges this gap by comparing implementations of the common operator in eq. (1) under a strict, unified protocol across diverse configurations, such as varying batch sizes (Section 6). Ultimately, we demonstrate that no single method universally dominates across all tested settings, highlighting the critical need for context-aware evaluations.

To address the absence of a comprehensive survey and a standardized benchmarking framework for exponentially decaying causal linear attention, we systematically evaluate, extend, and adapt existing linear-complexity algorithms into unified, general-purpose operators. Our primary contributions in this paper are:

(1) Algorithmic analysis and survey. We present the first comprehensive survey of computation methods in this domain. We derive an explicit non-iterative matrix formulation (eq. (2)) that elucidates the operator's inherent linearity and provide rigorous complexity analyses, notably proving the suboptimal $\mathcal{O}(N \log N)$ complexity of specific recursive approaches (Han et al., 2024).

(2) `LeetDecoding` library and benchmark. We introduce an open-source, highly optimized PyTorch library that standardizes these scattered algorithms into plug-and-play operators. It features non-trivial algorithmic adaptations, novel CUDA implementations for previously unoptimized methods, and system-level enhancements including explicit data prefetching and Tensor Core utilization. This enables fully reproducible benchmarking across both single-layer modules and full LLM prefill scenarios.

(3) Experimental Findings. Our scalability evaluation reveals that no single method universally dominates; instead, performance heavily depends on operational contexts such as batch size, mask complexity, and data precision. Furthermore, we demonstrate that system-level hardware optimizations can dictate actual efficiency independent of theoretical algorithmic complexity, thereby establishing clear, hardware-aware guidelines for future implementations.

2 Notations and Preliminaries

For a single attention head, let $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N \times r}$ and $\boldsymbol{V} \in \mathbb{R}^{N \times d}$, where N is the sequence length, r is the feature rank, and d is the value dimension. The dimensions r and d need not be equal. Our target is the exponentially decaying causal linear attention output

$$\boldsymbol{O} = (\boldsymbol{B}\boldsymbol{C}^\intercal \odot \boldsymbol{M})\boldsymbol{V} \in \mathbb{R}^{N \times d}, \tag{1}$$

where $\odot$ denotes the element-wise product and the causal mask $\boldsymbol{M} \in \mathbb{R}^{N \times N}$ is defined by

$$\boldsymbol{M}_{i,j} = \lambda^{i-j} \mathbf{1}_{i \geq j}.$$

Here, the decaying factor $\lambda \in (0, 1]$ may differ across heads but is fixed across token positions within each head. The indicator $\mathbf{1}_{i \geq j}$ is one when $i \geq j$ and zero otherwise; $\lambda = 1$ recovers the ordinary non-decaying causal mask.

This is the central operator studied in our library and benchmark. TransNormerLLM (Qin et al., 2024a) and RetNet (Sun et al., 2023) instantiate this form using their query and key features. The same formulation also covers attention approximations with compatible low-rank factorizations; background definitions and examples are provided in Appendix H.2.

Without the mask, associativity allows $(\boldsymbol{B}\boldsymbol{C}^\intercal)\boldsymbol{V}$ to be computed as $\boldsymbol{B}(\boldsymbol{C}^\intercal\boldsymbol{V})$. The causal mask prevents directly applying this unmasked reordering, so efficient evaluation must exploit its structure. We compare computation methods that avoid explicitly forming the $N \times N$ attention matrix, using direct dense evaluation as a reference. Their time and space complexities are summarized in Table 1; the following sections present the non-iterative formulation and algorithm taxonomy.

3 Overview of the Computation Methods

We present a non-iterative matrix formulation of eq. (1) and characterize the computation methods collected in `LeetDecoding`.

3.1 A non-iterative matrix formulation

For $1 \leq i \leq r$, let $\boldsymbol{B}_{:,i}, \boldsymbol{C}_{:,i} \in \mathbb{R}^{N \times 1}$ denote the i-th feature columns of $\boldsymbol{B}, \boldsymbol{C}$, respectively. For the non-decaying causal mask, multiplication by $\boldsymbol{M}$ is equivalent to a cumulative sum along the sequence dimension, giving

$$\boldsymbol{O} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}_{:,i})\mathrm{cumsum}(\mathrm{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V}), \tag{2}$$

The cumsum operator is applied independently to each column of $\mathrm{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V}$. Each summand costs $\mathcal{O}(Nd)$, so the total time complexity is $\mathcal{O}(Nrd)$.

For exponential decay (Press et al., 2022; Qin et al., 2022; Peng et al., 2023), define the operator $\lambda\text{-cumsum}(\boldsymbol{X}) = \boldsymbol{Y}$ by $\boldsymbol{Y}_{0,:} = 0$ and $\boldsymbol{Y}_{t,:} = \boldsymbol{X}_{t,:} + \lambda \boldsymbol{Y}_{t-1,:}$ for $t = 1, \ldots, N$. Then the output is

$$\boldsymbol{O} = \sum_{i=1}^{r} \operatorname{diag}(\boldsymbol{B}_{:,i}) \cdot \lambda\text{-cumsum}(\operatorname{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V}).$$

The discounted cumulative sum also takes $\mathcal{O}(Nd)$ time per summand, preserving the total $\mathcal{O}(Nrd)$ complexity. The full derivation is given in Appendix I.1.

3.2 Algorithm Taxonomy

Table 1 summarizes the methods evaluated for the common operator in eq. (1). Beyond the dense reference, we distinguish **iterative** methods (`causal-dot-product`, `block-based`, and `Lightning Attention-2`) from the **recursive** method. The iterative methods maintain a prefix summary, updated token by token or block by block; the recursive method divides the sequence into subproblems and combines their outputs with cross-block contributions.

`vanilla`. This reference directly evaluates eq. (1) using a dense attention matrix, for either non-decaying or exponentially decaying causal masks.

`causal-dot-product`. This method (Vyas et al., 2020; Choromanski et al., 2021) updates a prefix state token by token and computes one output row at each step. It is also referred to as the row-based method.

`block-based`. Adapted from Transformer-VQ (Lingle, 2024), this method organizes the prefix-state computation into blocks. It computes masked attention within each block and accumulates a summary of preceding blocks for cross-block contributions, forming a block-wise version of `causal-dot-product`.

`Lightning Attention-2`. Lightning Attention-2 (Qin et al., 2024b) uses the same block-wise computation structure, separating intra-block and inter-block contributions. Its IO-aware GPU implementation schedules the computation using on-chip memory.

`recursion`. Adapted from the recursive construction in HyperAttention (Han et al., 2024), this method recursively processes the masked diagonal subproblems and separately evaluates unmasked cross-block contributions. Its application to the general low-rank operator is analyzed in Section 4.2.

Further source and adaptation details are provided in Appendix I.2; implementation changes are described in Section 5.

4 Complexity Analysis

This section presents the per-layer forward complexity analysis of causal linear attention, with results summarized in Table 1. Unless otherwise specified, complexity is reported per batch and per head. Because applying a decaying mask introduces additional computational overhead without altering the asymptotic complexity, we focus our analysis on the non-decaying mask $\boldsymbol{M}$. The dense **Vanilla** implementation requires $\mathcal{O}(N^2(r+d))$ time and $\mathcal{O}(N^2 + N(r+d))$ space, including input and output tensors. Our analysis makes the dependence on r, d, and storage explicit for the iterative methods and derives the bounds for the recursive approach.

4.1 Analysis of the Iterative Methods

For token i, let $\boldsymbol{b}_i = \boldsymbol{B}_{i,:}^\intercal, \boldsymbol{c}_i = \boldsymbol{C}_{i,:}^\intercal \in \mathbb{R}^{r\times 1}$, $\boldsymbol{v}_i = \boldsymbol{V}_{i,:}^\intercal \in \mathbb{R}^{d\times 1}$, and $\boldsymbol{O}_i = \boldsymbol{O}_{i,:}^\intercal$. The **causal-dot-product** method maintains a state $\boldsymbol{U}_i \in \mathbb{R}^{r\times d}$, initialized with $\boldsymbol{U}_0 = 0$, and computes

$$\boldsymbol{O}_i^\intercal = \boldsymbol{b}_i^\intercal \boldsymbol{U}_i, \quad \text{where } \boldsymbol{U}_i = \boldsymbol{U}_{i-1} + \boldsymbol{c}_i \boldsymbol{v}_i^\intercal. \tag{3}$$

Each update takes $\mathcal{O}(rd)$ time, yielding $\mathcal{O}(Nrd)$ total time and $\mathcal{O}(N(r+d)+rd)$ space including input and output tensors.

The **block-based** and **Lightning Attention-2** methods update the same prefix summary block by block. For block size B_c, their time complexity is $\mathcal{O}(NB_cr + NB_cd + Nrd)$. When the block

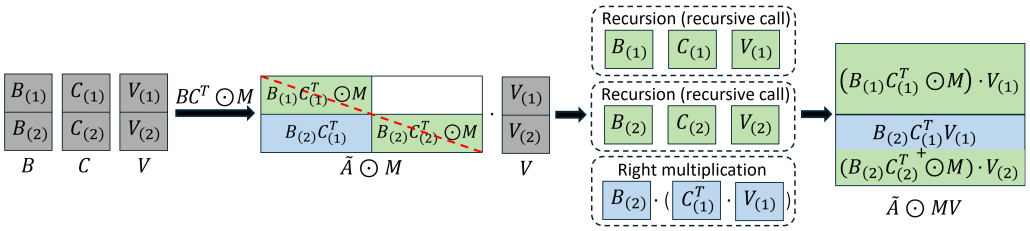


Figure 2: Visualization of the **recursion** algorithm. The causal attention matrix is partitioned into three equal-sized non-zero blocks: $\boldsymbol{B}_{(1)}\boldsymbol{C}_{(1)}^{\intercal} \odot \boldsymbol{M}$ and $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(2)}^{\intercal} \odot \boldsymbol{M}$ are masked diagonal blocks, and $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(1)}^{\intercal}$ is an unmasked cross-block term evaluated efficiently via right-multiplication. Here, $\boldsymbol{M} \in \mathbb{R}^{(N/2)\times(N/2)}$ is the causal mask for each diagonal block.

size $B_c = \mathcal{O}(\min\{r, d\})$, the time and total space complexities become $\mathcal{O}(Nrd)$ and $\mathcal{O}(N(r + d) + rd)$, respectively. Detailed derivations, including final-block handling and pseudocode, are provided in Appendix G.1.

4.2 Analysis of the recursive approach

As illustrated in Figure 2, the recursive method splits the sequence into two halves, indexed by (1) and (2). It recursively computes the two masked diagonal blocks and evaluates the unmasked cross-block contribution as $\boldsymbol{B}_{(2)}(\boldsymbol{C}_{(1)}^{\intercal}\boldsymbol{V}_{(1)})$. The two child calls and $\mathcal{O}(Nrd)$ cross-block computation give $T(N) = 2T(N/2)+\mathcal{O}(Nrd)$, leading to the additional logarithmic factor stated in Lemma 4.1.

Lemma 4.1. *Consider matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N\times r}$, $\boldsymbol{V} \in \mathbb{R}^{N\times d}$, $\tilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{C}^{\intercal}$, and the causal mask matrix $\boldsymbol{M}$. Assume that a linear-complexity causal linear attention algorithm can compute the base case. Then, the recursive computation of $(\tilde{\boldsymbol{A}} \odot \boldsymbol{M})\boldsymbol{V}$ has time complexity $\mathcal{O}(Nrd \log N)$. Moreover, if the termination block size is fixed, dense attention blocks are not materialized beyond the base case, and the recursive calls execute sequentially, the peak space complexity, including the input and output tensors, is $\mathcal{O}(N(r + d) + rd)$.*

The full partitioned formulation, pseudocode, and proof, including the peak-space analysis, are provided in Appendix G.2.

5 Generalizing and Optimizing Existing Implementations

Table 2 summarizes the PyTorch, native CUDA, and Triton (Tillet et al., 2019) implementations provided by `LeetDecoding`. We adapt **block-based** and **recursion** from their original module-specific settings to the general operator in eq. (1), while **Vanilla** provides a direct dense reference. Both adapted methods are implemented in PyTorch, preserving their block-wise and recursive computation structures. Additional implementation details are provided in Appendix A.2.

Causal-dot-product extensions. The original CUDA implementation of **causal-dot-product** (Vyas et al., 2020) supported FP32 inputs and non-decaying causal masks. We extend it with FP16/BF16 support and exponentially decaying masks, and provide a PyTorch counterpart for comparison. These extensions enable evaluation under the mask types and input precisions used by the benchmark without altering the underlying algorithm.

Lightning Attention-2 adaptations and optimizations. The original Triton implementation (Qin et al., 2024b) uses configurations tailored to NVIDIA A100 GPUs. We adapt tile sizes and kernel launch configurations to the shared-memory limits of the target GPU and additionally support non-decaying causal masks. We make two further changes to the implementation:

- **Explicit prefetching.** We load the first K/V tile before entering the loop, then issue loads for the next tile before computing with the current tile. The prefetched K/V tile is reused in the next iteration, allowing memory access to overlap with computation.

Table 2: Implementation languages for each method in `LeetDecoding`. A dash ('-') indicates that no CUDA implementation is provided, as PyTorch is sufficient for that method.

Method name (Abbreviation)	PyTorch impl.	CUDA impl.
`vanilla` (`Vanilla`)	✓	-
`causal-dot-product` (`Cdotp`)	✓	Native CUDA
`block-based` (`Bb`)	✓	-
`Lightning Attention-2` (`LA`)	✓	OpenAI Triton
`recursion` (`Recur`)	✓	-

- **Half-precision computation with FP32 state.** The baseline implementation converts input matrices to FP32 for subsequent computation. We retain FP16/BF16 operands for matrix products to use the corresponding Tensor Core paths, while maintaining the state used for aggregation across iterations in FP32. Further precision details are given in Appendix A.1.

The single-layer ablation in Section 6.1 compares the baseline, the prefetching variant, and the final optimized implementation.

PyTorch comparison. We also adapt the native PyTorch implementation of **Lightning Attention-2** from TransNormerLLM (Qin et al., 2024a) to support non-decaying causal masks, denoted as **LA_torch**. **LA_torch** and **Bb** retain distinct PyTorch realizations of the block-wise strategy; both are included in the benchmark, with implementation differences described in Appendix A.2.

6 Empirical Results

We report the empirical results of our library and benchmark under two settings. In Section 6.1, we benchmark the computational performance of implementations in our library when handling *extremely long prompts*. In Section 6.2, we examine the computation methods on pretrained linear transformers. The latency measurements compare the implementations listed in Table 2, whose programming frameworks and optimization levels differ. Throughout this section, we adopt the abbreviations in Table 2.

6.1 Prefilling Extremely Long Prompts

Setup and metrics. We benchmark single-layer causal linear attention on an NVIDIA RTX 5090 GPU with BF16 inputs. We use $h = 32$, $r = d = 128$, sequence lengths from 128 to 102,400, and batch sizes of 1 and 16. Input matrices are sampled from $\mathcal{N}(0, 1)$, and both non-decaying and head-specific exponentially decaying masks are tested. Latency measures the total time to compute eq. (1) for the entire input batch; we report the mean and standard deviation over 15 measurements[1]. **Vanilla** is the dense reference implementation. Precision details are given in Appendix A.1; decay settings and additional latency comparisons are provided in Appendix C, and memory measurements in Appendix C.3.

Result analysis. We vary the sequence length N from 128 to 102,400 and report the following observations.

1. **Runtime scaling**. We investigate how the latency of the evaluated implementations scales with sequence length. The methods have linear arithmetic complexity for fixed feature dimensions, except for **Vanilla** and the additional logarithmic factor in **Recur**. The tables also show that **Vanilla** encounters OOM at shorter sequences, while other methods also encounter OOM at the longest tested sequence with batch size 16.
2. **Latency for single-layer computation**. Table 3 (a), (b) indicate that, with a batch size of 1 and tested sequence lengths greater than 512, **`Lightning Attention-2`** (**LA**) is the fastest among the evaluated methods.

   Additionally, for most evaluated configurations where both runs complete, increasing the batch size from 1 to 16 in Table 3 (c), (d) results in less than a 16× increase in computational time.

[1] Along all the tables in this paper, boldface represents the best result, and "OOM" denotes "out-of-memory".

Table 3: Measured latency (s) of single-layer causal linear attention on an NVIDIA RTX 5090 GPU with BF16 inputs. We compare performance across batch sizes (B) of 1 and 16, both with and without an exponentially decaying causal mask. Entries report the mean and standard deviation over 15 measurements. The lowest mean latency in each column per setting is highlighted in bold.

Method	Sequence Length						
	128	512	2,048	8,192	12,800	25,600	102,400
(a) Batch Size 1, Without Decay							
Vanilla	5.64e−5 ± 1e−6	6.61e−5 ± 3.3e−7	9.61e−4 ± 7e−7	1.8e−2 ± 2.9e−5	4.43e−2 ± 8.9e−5	OOM	OOM
Cdotp	8.13e−5 ± 2.6e−7	3.05e−4 ± 5.7e−7	1.66e−3 ± 5.1e−4	7.42e−3 ± 8.3e−6	1.16e−2 ± 1.6e−5	2.33e−2 ± 2.3e−5	9.32e−2 ± 4.8e−5
Cdotp_torch	1.12e−2 ± 7.1e−5	4.45e−2 ± 1.1e−4	1.91e−1 ± 2.1e−4	7.05e−1 ± 8e−3	1.11 ± 5.1e−3	2.24 ± 7e−3	8.98 ± 2.7e−2
Bb	5.89e−4 ± 3.8e−6	2.25e−3 ± 3.4e−4	8.59e−3 ± 7.1e−5	3.38e−2 ± 3.7e−4	5.26e−2 ± 1.3e−3	1.08e−1 ± 8.8e−4	4.18e−1 ± 3.3e−3
LA (original)	**5.52e−5** ± 2.7e−6	6.36e−5 ± 3.1e−6	2.32e−4 ± 2.6e−6	1.18e−3 ± 1.6e−6	1.83e−3 ± 1.6e−6	3.67e−3 ± 5.3e−6	1.46e−2 ± 1.6e−5
LA (prefetch)	6.82e−5 ± 1.5e−5	**5.84e−5** ± 2.3e−6	1.42e−4 ± 1.9e−6	6.77e−4 ± 2.7e−6	1.04e−3 ± 3.6e−6	2.07e−3 ± 1.8e−6	8.27e−3 ± 1.4e−5
LA	6.28e−5 ± 2.8e−6	6.49e−5 ± 3.4e−6	**1.16e−4** ± 1.5e−6	**5.64e−4** ± 3.1e−6	**8.79e−4** ± 1.6e−6	**1.74e−3** ± 2.2e−6	**6.92e−3** ± 2.8e−6
LA_torch	5.26e−3 ± 4.1e−3	5.89e−3 ± 3.8e−3	9.96e−3 ± 2.9e−3	2.85e−2 ± 5.1e−3	3.11e−2 ± 3.7e−3	4.75e−2 ± 3.1e−3	7.77e−2 ± 3.1e−3
Recur	4.41e−4 ± 3e−6	2.07e−3 ± 1.3e−4	7.59e−3 ± 7.1e−5	3.02e−2 ± 9.8e−5	5.99e−2 ± 1.9e−4	1.18e−1 ± 1.9e−4	4.95e−1 ± 3.9e−3
(b) Batch Size 1, With Decay							
Vanilla	1.58e−4 ± 2.2e−6	1.67e−4 ± 2.2e−6	4.56e−3 ± 2.6e−6	7.79e−2 ± 3.4e−5	1.99e−1 ± 6.6e−5	8.03e−1 ± 3.8e−4	OOM
Cdotp	8.05e−5 ± 3.5e−7	5.31e−4 ± 8.5e−4	1.4e−3 ± 7.7e−4	7.43e−3 ± 9.2e−6	1.16e−2 ± 1.6e−5	2.33e−2 ± 2.4e−5	9.32e−2 ± 5.7e−5
Cdotp_torch	1.21e−2 ± 1.1e−4	4.78e−2 ± 1.8e−4	1.91e−1 ± 2.5e−3	7.75e−1 ± 2.7e−3	1.2 ± 5.4e−3	2.41 ± 9.9e−3	9.52 ± 2.4e−2
Bb	1.2e−1 ± 2.1e−2	1.2e−1 ± 9.3e−3	1.06e−1 ± 1.3e−2	1.44e−1 ± 1.4e−2	1.47e−1 ± 9.9e−3	2.04e−1 ± 1.3e−2	6.02e−1 ± 8.6e−3
LA (original)	5.78e−5 ± 3.5e−6	6.78e−5 ± 2.2e−6	2.49e−4 ± 1.6e−6	1.3e−3 ± 2.3e−6	2.03e−3 ± 5.2e−6	4.04e−3 ± 1.5e−6	1.62e−2 ± 1.4e−5
LA (prefetch)	**5.74e−5** ± 3.3e−6	**5.88e−5** ± 2.5e−6	1.51e−4 ± 1.7e−6	7.08e−4 ± 2.9e−6	1.09e−3 ± 2.2e−6	2.17e−3 ± 1.9e−6	8.69e−3 ± 1.2e−5
LA	6.57e−5 ± 2.4e−6	6.75e−5 ± 2.6e−6	**1.19e−4** ± 1.7e−6	**5.84e−4** ± 2.2e−6	**9.09e−4** ± 1.4e−6	**1.79e−3** ± 3.4e−6	**7.15e−3** ± 5.2e−6
LA_torch	2.92e−3 ± 3.5e−3	4.52e−3 ± 3.8e−3	1.19e−2 ± 3.4e−3	2.79e−2 ± 4.5e−3	3.82e−2 ± 4.1e−3	5.19e−2 ± 4.1e−3	8.3e−2 ± 1e−3
Recur	8.9e−3 ± 1.8e−4	3.76e−2 ± 4.8e−4	1.42e−1 ± 1e−3	5.68e−1 ± 9.6e−3	1.13 ± 3e−3	2.28 ± 9.6e−3	9.02 ± 2.8e−2
(c) Batch Size 16, Without Decay							
Vanilla	**6.67e−5** ± 5.2e−7	1.17e−3 ± 1.2e−6	1.57e−2 ± 4.8e−5	OOM	OOM	OOM	OOM
Cdotp	1.16e−3 ± 1.5e−3	3.16e−3 ± 1.7e−5	1.29e−2 ± 5.1e−5	5.34e−2 ± 1.5e−4	8.41e−2 ± 2.6e−4	1.69e−1 ± 5.6e−4	OOM
Cdotp_torch	1.13e−2 ± 9.4e−5	4.47e−2 ± 2e−4	1.78e−1 ± 4e−4	7.36e−1 ± 4.9e−3	1.2 ± 2.1e−3	2.71 ± 7.5e−3	OOM
Bb	9.15e−4 ± 1.2e−6	3.69e−3 ± 4.2e−5	1.53e−2 ± 9.6e−5	6.23e−2 ± 1.5e−4	9.76e−2 ± 3.5e−4	1.96e−1 ± 7.1e−4	OOM
LA (original)	1.59e−4 ± 2.4e−6	6.99e−4 ± 1.2e−6	2.76e−3 ± 1.9e−5	1.1e−2 ± 3.2e−5	1.73e−2 ± 4e−5	3.48e−2 ± 6e−5	OOM
LA (prefetch)	9.03e−5 ± 1.8e−6	4.44e−4 ± 2.2e−6	1.81e−3 ± 2.2e−6	7.25e−3 ± 6e−6	1.14e−2 ± 1e−5	2.28e−2 ± 1.5e−5	OOM
LA	6.9e−5 ± 3.1e−6	**2.92e−4** ± 1.6e−6	**1.21e−3** ± 2.3e−6	**6.56e−3** ± 3.6e−5	**1.07e−2** ± 2.7e−5	**2.22e−2** ± 7.7e−5	OOM
LA_torch	3.86e−2 ± 5.2e−3	4.27e−2 ± 3.6e−3	5.8e−2 ± 4.9e−3	9.13e−2 ± 1.1e−2	9.49e−2 ± 5.8e−3	1.4e−1 ± 1.5e−3	OOM
Recur	8.65e−4 ± 1.3e−6	3.97e−3 ± 4.1e−5	1.85e−2 ± 1.4e−4	8.31e−2 ± 1.5e−4	1.4e−1 ± 3.5e−4	2.94e−1 ± 1.2e−3	OOM
(d) Batch Size 16, With Decay							
Vanilla	1.65e−4 ± 1.9e−6	2.99e−3 ± 2.8e−6	4.53e−2 ± 8.2e−6	OOM	OOM	OOM	OOM
Cdotp	7.35e−4 ± 3.5e−6	3.46e−3 ± 9.5e−4	1.3e−2 ± 6.6e−5	5.36e−2 ± 1e−4	8.42e−2 ± 2.2e−4	1.69e−1 ± 7.1e−4	OOM
Cdotp_torch	1.27e−2 ± 5.9e−4	5.18e−2 ± 9.6e−5	1.92e−1 ± 4e−4	8.2e−1 ± 2.2e−3	1.25 ± 6.6e−3	2.95 ± 3.8e−2	OOM
Bb	1.15e−1 ± 8.6e−3	1.54e−1 ± 5.1e−2	1.09e−1 ± 1.4e−2	1.45e−1 ± 1.1e−2	1.42e−1 ± 5.1e−3	2.03e−1 ± 6.4e−3	OOM
LA (original)	1.66e−4 ± 1.7e−6	7.29e−4 ± 2.2e−6	2.86e−3 ± 9.6e−6	1.15e−2 ± 2.3e−5	1.8e−2 ± 2.7e−5	3.62e−2 ± 3.9e−5	OOM
LA (prefetch)	9.46e−5 ± 1.7e−6	4.5e−4 ± 2.5e−6	1.83e−3 ± 2.2e−6	7.3e−3 ± 6.9e−6	1.15e−2 ± 1.6e−5	2.31e−2 ± 2.3e−5	OOM
LA	**7.16e−5** ± 3.1e−6	**3.37e−4** ± 2.1e−6	**1.31e−3** ± 4.9e−6	**5.21e−3** ± 5.7e−5	**8.17e−3** ± 1.2e−4	**1.68e−2** ± 1.5e−4	OOM
LA_torch	3.73e−2 ± 3e−3	4.31e−2 ± 4.3e−3	6.23e−2 ± 6.1e−3	9.37e−2 ± 7e−3	9.76e−2 ± 5.6e−3	1.54e−1 ± 1.4e−3	OOM
Recur	8.94e−3 ± 8.5e−5	3.65e−2 ± 1.4e−3	1.43e−1 ± 2.7e−3	5.81e−1 ± 5.2e−3	1.16 ± 7.7e−3	2.35 ± 1e−2	OOM

3. **System-level optimizations**. We compare **LA (original)**, the baseline implementation without our optimizations, with **LA (prefetch)**, which adds explicit K/V prefetching, and **LA**, our final optimized implementation. For the tested BF16 configurations with sequence lengths of at least 2,048 where all three variants complete, **LA (prefetch)** improves on **LA (original)**, and **LA** further reduces latency. The improvements are not uniform for short sequences.
4. **Numerical error**. Due to space limits, additional single-layer numerical comparisons are provided in Appendix E. In short, with BF16 inputs and exponential decay, **Cdotp** and **Cdotp_torch** show larger deviations than **LA**.
5. **Performance of CUDA and Triton implementations**. For **Cdotp** and **LA**, the evaluated CUDA and Triton implementations achieve lower latency than their PyTorch counterparts in these BF16 benchmarks. The gains depend on batch size and sequence length.

Insight 1: No single method dominates across all settings. With BF16 inputs on the RTX 5090, **LA** is substantially faster than **Cdotp** at batch size 16 with exponentially decaying masks. With FP32 inputs, however, **Cdotp** is faster on the RTX 5090 and has similar latency to **LA** on the A800 (Appendices C.1 and C.2).

6.2 Linear-transformer large language models

We evaluate all library methods on end-to-end prefill with two real linear-transformer LLMs. Benchmarks are conducted on an NVIDIA A800 (80 GB) GPU; we additionally evaluate on an NVIDIA RTX 5090 (Blackwell architecture), with results in Appendix D.2. Experimental setup and results are as follows.

Linear-transformer large language models. We evaluate two open-source linear-transformer LLMs, TransNormerLLM-7B and toy-retnet-1.3B. TransNormerLLM-7B has 32 heads and a hid-

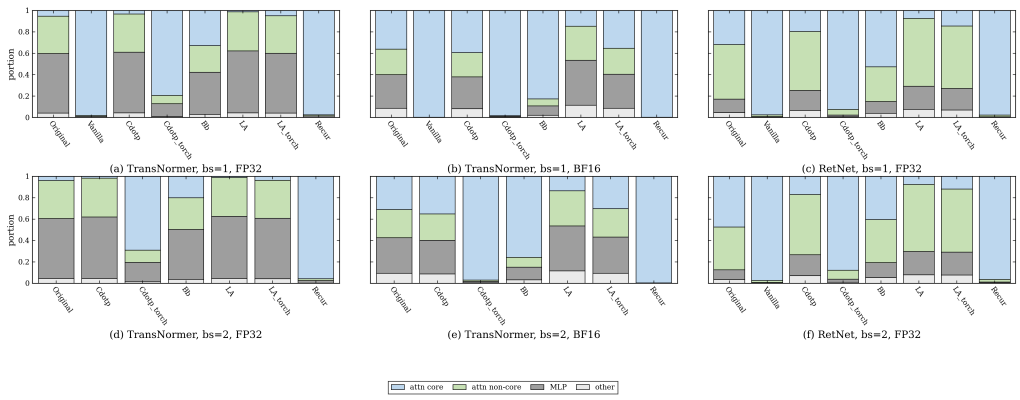


Figure 3: Per-component latency breakdown for a single forward pass, for TransNormerLLM-7B as well as the RetNet results in Appendix D.1. Each bar segments total runtime into causal attention core (`attn core`, the computation in eq. (1)), remaining attention (`attn non-core`), `MLP`, and `other` modules. Best viewed zoomed in.

den dimension 4096 (with rank $r = d = 128$), while toy-retnet-1.3B has 8 heads and a hidden dimension 2048, with rank $r = d = 256$. The two models both employ exponentially decaying causal masks in their linear attention modules, and we can therefore insert our function, `causal_linear_decoder` within `LeetDecoding`, into those two LLMs. In particular, after the associated matrices $\mathbf{B}, \mathbf{C}, \mathbf{V}$ are ready in the module, we call the function `causal_linear_decoder` to calculate $(\mathbf{B}\mathbf{C}^\intercal \odot \mathbf{M})\mathbf{V}$. These models instantiate eq. (1), allowing computation methods to be compared without changing their attention rule. Gated delta-rule models such as Qwen3-Next (QwenTeam, 2025) and Kimi Linear (Kimi Team, 2025) use different recurrent updates and require separate evaluation of their corresponding operators.

For TransNormerLLM-7B, we evaluate methods under both FP32 (PyTorch default) and the model's native BF16; for toy-retnet-1.3B, experiments use its native FP32 precision.

Datasets, protocols, and metrics. To prepare long prompts, we extracted samples from the TREC (Li & Roth, 2002) dataset in LongBench (Bai et al., 2024). This dataset contains 200 samples with an average sequence length $5,177$.

We evaluate each method on an A800 GPU with batch sizes of 1 and 2. For each run, we process all 200 TREC samples and sum the measured prefill times across input batches, counting each batch once. We refer to this dataset-level metric as *cumulative prefill time*. The timed region includes the model forward pass and first-token selection, while prompt tokenization and subsequent autoregressive decoding are excluded. We perform four runs, discard the first, and report the mean and sample standard deviation of the cumulative prefill time over the remaining three runs. We additionally profile the runtime contributions of individual model components on fixed input batches.

Result analysis. We make the following observations from our experiments.

1. **Overall latency**. In both Table 4 and Appendix D.1, **LA** is significantly faster than the **Original** method (the original implementations of causal linear attention in those two LLMs). For TransNormerLLM-7B, its original code is shared with the commercial Minimax-M1 LLM (MiniMax, 2025), making it a production-level baseline. For toy-retnet-1.3B, we further compare **LA** with a production-level reference implementation of RetNet "**LA_ref**" with chunk-wise computation (i.e., **`Lightning Attention-2`**; here, we employ the advanced `fused_chunk` mode to address the input $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$), and observe that **LA** achieves comparable performance. Full RetNet results are reported in Appendix D.1 (Table 13). **Cdotp**, implemented in native CUDA, performed competitively under FP32 with batch size greater than 1 (on par with **LA** for both LLMs), which aligns with the A800 FP32 observation in Table 12.
2. **Latency breakdown**. Figure 3 illustrates the latency breakdown for LLMs equipped with different causal linear attention methods. Since non-attention modules are shared across methods,

Table 4: Cumulative prefill time (s) for one pass over all 200 TREC samples from LongBench using TransNormerLLM-7B on an A800 GPU. We compare FP32 and the model's original BF16 precisions across batch sizes (B) of 1 and 2. Entries report the mean and sample standard deviation across three runs after discarding the first of four; each run sums the measured time across input batches. The "Original" method denotes the original attention computation method implemented in TransNormerLLM-7B. Methods with cumulative times exceeding 1 hour are denoted as $> 1h$.

Method	FP32 (s)		BF16 (s)	
	$B = 1$	$B = 2$	$B = 1$	$B = 2$
Original	1400.89 ± 1.17	1539.35 ± 0.10	212.48 ± 0.06	185.88 ± 0.44
Vanilla	$> 1h$	OOM	$> 1h$	OOM
Bb	1933.78 ± 7.83	1869.94 ± 2.53	778.41 ± 0.80	515.07 ± 0.24
Cdotp	1358.27 ± 2.28	1504.13 ± 0.01	229.96 ± 0.23	197.05 ± 0.42
Cdotp_torch	$> 1h$	$> 1h$	$> 1h$	$> 1h$
LA	**1336.10** ± 3.65	**1495.69** ± 0.09	**184.40** ± 0.26	**151.42** ± 0.26
LA_torch	1379.71 ± 1.48	1536.24 ± 0.07	213.59 ± 0.14	183.89 ± 0.30
Recur	$> 1h$	$> 1h$	$> 1h$	$> 1h$

the `attn core` fraction directly reflects the efficiency of the attention implementation. Notably, for LLMs using the **Vanilla** or **Recur** methods, the core attention computation dominates the overall latency. For the smaller RetNet model, attention accounts for a larger share of total runtime, so an efficient attention implementation yields proportionally greater gains.

3. **Numerical behavior under implementation substitution**. We additionally evaluate model-level numerical differences against each model's native attention implementation on 200 TREC prompts (Appendix F). For RetNet-1.3B with FP32 inputs, the tested methods yield small deviations and no greedy next-token disagreements. For TransNormerLLM-7B with BF16 inputs, the differences are more method-dependent: **LA** yields $\Delta\text{PPL} \approx 0.032$ and 5% next-token disagreement, while **Cdotp** and **Recur** exhibit larger deviations.

Insight 2: Hardware-level optimizations significantly affect efficiency. As shown in Table 4, **LA** is significantly faster than the **Original** method under the BF16 data type, despite both employing the same chunk-wise algorithm. In contrast, for FP32 operations, the performance gap is less pronounced. We attribute this efficiency gain to intensive use of A800 Tensor Cores specifically designed for BF16 operations, as described in Section 5. This principle, established in prior work on FlashAttention (Zadouri et al., 2026), applies equally to causal linear attention. The RTX 5090 results further corroborate this finding: **LA** retains a clear advantage over the original TransNormerLLM implementation for BF16 inputs (Appendix D.2).

7 Conclusions

In this paper, we present `LeetDecoding`, a comprehensive library and benchmark for exponentially decaying causal linear attention, complemented by an experimental survey and an extensive scalability evaluation. By deriving an explicit non-iterative matrix formulation, we systematically adapt and implement diverse computation methods from various domains, consolidating techniques that previously lacked direct comparison, alongside providing rigorous complexity analyses and formal proofs. To enable fair and reproducible assessments, we extend these routines to support both decaying and non-decaying causal masks across diverse data precisions. Crucially, we incorporate hardware-aware optimizations for CUDA-based implementations, such as explicit data prefetching and memory access streamlining, to isolate algorithmic efficiency from system-level bottlenecks. Comprehensive benchmarks across single-layer modules and full language model prefilling scenarios reveal that no single evaluated method dominates across all tested configurations. Instead, each method exhibits configuration-dependent performance characteristics heavily influenced by factors like batch size scaling, sequence length, and mask complexity. Beyond establishing a standardized and open-source benchmarking framework, we expect these empirical insights to inform future hardware-aware algorithm design, guide practical deployment decisions, and facilitate the development of highly efficient linear transformers.

AI USE STATEMENT

In this work, we used generative AI tools to assist with checking and revising mathematical formulations and proofs, providing feedback on experimental methodology, and interpreting experimental results. We did not use generative AI tools to implement experimental methods or generate experimental datasets. Moreover, we used these tools to draft and edit manuscript text, revise code examples, and organize experimental results into tables. We have reviewed the AI-assisted revisions, including checking mathematical consistency, comparing descriptions and code examples against the implementation, and cross-checking reported values against the source tables. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Our algorithms, implementation details, and experimental protocols are detailed in Sections 3 through 6; detailed complexity analyses and proofs are provided in Appendix G. Usage examples and numerical precision details appear in Appendices A and A.1, while the supplementary material contains further experimental results. License information is provided in Appendix B. Following the official ICLR guidelines, we will supply an anonymous repository containing our code and experiment scripts via a comment directed to the reviewers and area chairs. All materials will be open-sourced upon publication.

REFERENCES

Gilbert Badaro and Paolo Papotti. Transformers for tabular data representation: A tutorial on models and applications. *Proc. VLDB Endow.*, 15(12):3746–3749, 2022.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Yifan Chen, Qi Zeng, Heng Ji, and Yun Yang. Skyformer: remodel self-attention with gaussian kernel and nyström method. *Advances in Neural Information Processing Systems*, 2021.

Yifan Chen, Qi Zeng, Dilek Hakkani-Tur, Di Jin, Heng Ji, and Yun Yang. Sketching as a tool for understanding and accelerating self-attention for long sequences. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5187–5199, Seattle, United States, July 2022. Association for Computational Linguistics.

Yuqi Chen, Hanyuan Zhang, Weiwei Sun, and Baihua Zheng. Rntrajrec: Road network enhanced trajectory recovery with spatial-temporal transformer. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pp. 829–842. IEEE, 2023.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

Tri Dao. Flashattention-2: faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: fast and memory-efficient exact attention with io-awareness. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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, 2019.

Kwassi Joseph Dzahini and Stefan M. Wild. A class of sparse Johnson–Lindenstrauss transforms and analysis of their extreme singular values. *SIAM Journal on Matrix Analysis and Applications*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024.

Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David Woodruff, and Amir Zandieh. Hyperattention: long-context attention in near-linear time. In *International Conference on Learning Representations*, volume 2024, pp. 39310–39325, 2024.

William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984.

Kimi Team. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pp. 1–7, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1072228.1072378.

Lucas D Lingle. Transformer-vq: linear-time transformers via vector quantization. In *International Conference on Learning Representations*, volume 2024, pp. 6121–6150, 2024.

Chenghao Lyu, Guillaume Lachaud, Gabriel Lozano, and Yanlei Diao. Graph transformers for query plan representation: Potentials and challenges. *Proceedings of the VLDB Endowment*, 18 (13):5716–5730, 2025.

MiniMax. Minimax-m1: ccaling test-time compute efficiently with lightning attention, 2025. URL https://arxiv.org/abs/2506.13585.

OpenAI. Introducing ChatGPT. https://openai.com/research/techniques-for-training-large-neural-networks, 2022.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, Singapore, December 2023. Association for Computational Linguistics.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *International Conference on Learning Representations*, 2021.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: rethinking softmax in attention. In *International Conference on Learning Representations*, 2022.

Zhen Qin, Dong Li, Weigao Sun, Weixuan Sun, Xuyang Shen, Xiaodong Han, Yunshen Wei, Baohong Lv, Xiao Luo, Yu Qiao, and Yiran Zhong. Transnormerllm: a faster and better large language model with improved transnormer. *arXiv preprint arXiv:2307.14995*, 2024a.

Zhen Qin, Weigao Sun, Dong Li, Xuyang Shen, Weixuan Sun, and Yiran Zhong. Lightning Attention-2: a free lunch for handling unlimited sequence lengths in large language models. *arXiv preprint arXiv:2401.04658*, 2024b.

QwenTeam. Qwen3-next: towards ultimate training & inference efficiency, 2025. URL https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

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

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. *Advances in Neural Information Processing Systems*, 33:21665–21674, 2020.

Meng Wang, Jintao Yang, Bin Yang, Hui Li, Tongxin Gong, Bo Yang, and Jiangtao Cui. Towards lightweight time series forecasting: A patch-wise transformer with weak data enriching. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 1278–1291. IEEE, 2025.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

Yongtao Wu, Fanghui Liu, Grigorios Chrysos, and Volkan Cevher. On the convergence of encoder-only shallow transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Forty-first International Conference on Machine Learning*, 2024.

Tiankai Yang, Yi Nian, Li Li, Ruiyao Xu, Yuangang Li, Jiaqi Li, Zhuo Xiao, Xiyang Hu, Ryan A Rossi, Kaize Ding, et al. Ad-llm: Benchmarking large language models for anomaly detection. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1524–1547, 2025.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. Flashattention-4: algorithm and kernel pipelining co-design for asymmetric hardware scaling, 2026.

Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. Queryformer: A tree transformer model for query plan representation. *Proceedings of the VLDB Endowment*, 15(8):1658–1670, 2022.

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

Rundong Zuo, Guozhong Li, Rui Cao, Byron Choi, Jianliang Xu, and Sourav S Bhowmick. Darker: efficient transformer with data-driven attention mechanism for time series. *Proc. VLDB Endow.*, 17(11):3229–3242, July 2024. ISSN 2150-8097. doi: 10.14778/3681954.3681996.

Supplement to "LeetDecoding: An Experimental Survey and Scalability Evaluation for Prefilling With Causal Linear Attention"

A How to Use LeetDecoding

We describe how `LeetDecoding` is designed to serve various LLM practitioners. At the end of this section, we also provide usage examples to demonstrate the library's interface.

For **application developers** using linear transformers for efficient inference, `LeetDecoding` integrates into transformer-based models as a drop-in replacement for the attention computation module. Users can directly replace the original attention computation interface with `LeetDecoding`. The library expects $\boldsymbol{B}, \boldsymbol{C}$ in `(batch, heads, sequence, rank)` layout and $\boldsymbol{V}$ in `(batch, heads, sequence, dim)` layout.

For **researchers**, `LeetDecoding` provides a benchmarking framework for new linear attention algorithms (see Listing 1). `LeetDecoding` is designed for easy extension: new computation methods can be added by implementing the corresponding class and registering it in the library.

Listing 1: An example of how to use LeetDecoding Library to benchmark a new implementation.

```python
import torch
from leetDecoding.test.test_method import benchmark_forward
from leetDecoding.methods.lightningAttention2_optimized import (
    lightning_attn2_optimized,
)

def attn(q, k, v, slope):
    # Replace this body with the implementation to benchmark.
    return lightning_attn2_optimized(q, k, v, slope)

shape = (1, 32, 1024, 128)
q = torch.randn(shape, device='cuda', dtype=torch.float32)
k = torch.randn_like(q)
v = torch.randn_like(q)
slope = torch.full((32,), 0.1, device='cuda')

_, elapsed_s = benchmark_forward(attn, q, k, v, slope)
print(f"Forward latency: {elapsed_s:.6f} s")
```

Listing 2: Usage of the `causal_linear_decoder` function, requiring only one line for regular causal linear attention with binary masks or the one with exponentially decaying causal masks.

```python
import torch
from leetDecoding.efficient_linear_decoding import
    causal_linear_decoder

batch_size, heads, seqlen, rank, dim = 1, 32, 4096, 128, 256
B = torch.randn((batch_size, heads, seqlen, rank),
                device='cuda', dtype=torch.float32)
C = torch.randn_like(B)
V = torch.randn((batch_size, heads, seqlen, dim),
                device='cuda', dtype=torch.float32)
method = 'lightningAttention2_optimized'

# Regular causal mask.
output = causal_linear_decoder(B, C, V, attn_method=method)

# Exponentially decaying mask: per-step decay is exp(-gamma).
gamma = torch.full((heads,), 0.1, device='cuda')
output = causal_linear_decoder(
    B, C, V, is_mask_weight=True, gamma=gamma,
    is_need_exp=True, attn_method=method,
```

```
)

# Select another computation method.
output = causal_linear_decoder(B, C, V, attn_method='block-based')
```

Listing 2 gives three usage examples, covering input argument formats, mask type selection, and computation method choice. The examples use FP32 inputs with batch size 1 and explicitly select the computation method. In the decaying example, `gamma` specifies a per-head slope, with per-step decay $\exp(-\texttt{gamma})$.

A.1 Numerical precision

In the single-layer benchmarks, FP32 and BF16 denote input tensor dtypes. LA uses the `lightningAttention2_optimized` implementation. For FP32 inputs, matrix operands remain FP32 and `tl.dot` uses Triton's default multiplication precision. For FP16/BF16 inputs, matrix products return FP32 results, with selected intermediates converted back to the input dtype for subsequent products. The inter-block state is maintained in FP32, and outputs use the input dtype. Cdotp also uses FP32 state accumulation and converts its outputs back to the input dtype. In its BF16 decay path, the decay coefficient is rounded before being repeatedly applied to the state; this can contribute to the larger observed error.

A.2 Additional implementation details

Recursive base case. The recursive implementation terminates on small sequence blocks, which are evaluated directly without further recursive splitting.

Block-wise PyTorch implementations. **Bb** and **LA_torch** use distinct PyTorch realizations of the same block-wise computation strategy, adapted from Transformer-VQ (Lingle, 2024) and TransNormerLLM (Qin et al., 2024a), respectively. The **Bb** implementation expresses block products using `einsum`, whereas **LA_torch** uses block-wise matrix multiplications.

B License information

`LeetDecoding` is released under the MIT License.

TransNormerLLM-7B model is licensed under the Apache 2.0 License.

toy-retnet-1.3B model is licensed under the Apache 2.0 License.

LongBench is licensed under the MIT License.

C Additional single-layer comparisons

Decay settings for the RTX 5090 experiments. For the 32 attention heads, the benchmark generates slopes $s_h = 2^{-(h+1)/4}$ for $h = 0, \ldots, 31$, corresponding to causal weights $\exp[-s_h(i-j)]$ for $i \geq j$. The slopes are cast to the input dtype and remain fixed across token positions.

C.1 RTX 5090, FP32 inputs

With FP32 inputs, **LA** has the lowest mean latency among the compared methods for batch size 1 and tested sequence lengths greater than 512. For batch size 16, **Cdotp** is faster than **LA** wherever both complete, showing that the BF16 ranking does not directly carry over to FP32.

C.2 A800, FP32 inputs

The A800 FP32 results likewise favor **LA** for batch size 1 and tested sequence lengths greater than 512. For batch size 16 with exponentially decaying masks, **Cdotp** and **LA** have similar latencies, whereas **LA** is substantially faster in the RTX 5090 BF16 results.

Table 5: Measured latency (s) of single-layer causal linear attention on an NVIDIA RTX 5090 GPU with FP32 inputs, batch size 1, and no exponentially decaying causal mask. Entries report the mean and standard deviation over 15 measurements. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	**5.55e-5** ± 7.9e−7	1.61e−4 ± 3.2e−7	2.78e−3 ± 6.4e−6	4.82e−2 ± 1.4e−4	OOM	OOM	OOM
Cdotp	9.18e−5 ± 3.8e−7	3.45e−4 ± 6.7e−7	1.32e−3 ± 8.2e−6	5.28e−3 ± 7.8e−5	8.28e−3 ± 1.3e−4	1.67e−2 ± 1.3e−4	6.9e−2 ± 2.6e−4
Cdotp_torch	8.92e−3 ± 1.4e−4	3.5e−2 ± 4e−4	1.4e−1 ± 2.1e−3	5.64e−1 ± 1.3e−3	8.79e−1 ± 2.4e−3	1.74 ± 1.3e−2	6.96 ± 3.2e−2
Bb	5.15e−4 ± 2.6e−6	1.88e−3 ± 6.2e−6	7.81e−3 ± 5.7e−5	2.97e−2 ± 4.2e−4	4.57e−2 ± 1.6e−4	9.17e−2 ± 7.3e−4	3.68e−1 ± 8.4e−4
LA	6.45e−5 ± 5.2e−6	**7.95e-5** ± 3.6e−6	**2.99e-4** ± 3.1e−6	**1.17e-3** ± 7.3e−6	**1.83e-3** ± 3.7e−6	**3.67e-3** ± 1.3e−5	**1.53e-2** ± 3.7e−5
LA_torch	6e−4 ± 5.6e−5	7.93e−4 ± 9.2e−5	1.51e−3 ± 5.7e−5	4.59e−3 ± 4.8e−5	7.24e−3 ± 4.7e−4	1.37e−2 ± 8.8e−5	5.31e−2 ± 1.6e−4
Recur	4.42e−4 ± 3.6e−5	1.66e−3 ± 7.8e−5	6.47e−3 ± 1.7e−4	2.61e−2 ± 9.8e−5	5.21e−2 ± 4.4e−4	1.04e−1 ± 1.8e−3	4.12e−1 ± 4.3e−3

Table 6: Measured latency (s) of single-layer causal linear attention on an NVIDIA RTX 5090 GPU with FP32 inputs, batch size 1, and an exponentially decaying causal mask. Entries report the mean and standard deviation over 15 measurements. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	6.37e−2 ± 3e−4	2.76e−1 ± 2.9e−3	1.57 ± 2.3e−2	1.18e1 ± 3.2e−1	OOM	OOM	OOM
Cdotp	**5.05e-5** ± 7.1e−7	1.87e−4 ± 3.7e−7	1.06e−3 ± 1.4e−6	4.33e−3 ± 2e−6	6.74e−3 ± 4.8e−6	1.35e−2 ± 1e−5	5.42e−2 ± 3.2e−5
Cdotp_torch	9.67e−3 ± 4.1e−5	3.81e−2 ± 8.5e−5	1.55e−1 ± 1.1e−3	6.23e−1 ± 7.7e−3	9.56e−1 ± 7.5e−3	1.91 ± 8.4e−3	7.89 ± 1.1e−1
Bb	1.7e−2 ± 2.7e−4	1.93e−2 ± 1.8e−4	2.81e−2 ± 1.3e−4	6.06e−2 ± 1.3e−4	8.81e−2 ± 2.5e−4	1.58e−1 ± 5.5e−4	5.97e−1 ± 6.9e−3
LA	6.92e−5 ± 1.1e−5	**8.89e-5** ± 3.9e−6	**3.29e-4** ± 2.9e−6	**1.28e-3** ± 3.7e−6	**2.01e-3** ± 1.4e−5	**4.02e-3** ± 1.7e−5	**1.72e-2** ± 3.6e−5
LA_torch	7.07e−4 ± 8.6e−5	8.94e−4 ± 5.8e−5	1.87e−3 ± 8.9e−5	6.14e−3 ± 2.3e−4	9.06e−3 ± 2.3e−4	1.74e−2 ± 9.3e−5	6.93e−2 ± 1.3e−3
Recur	9.27e−2 ± 1.7e−3	3.76e−1 ± 3.1e−3	1.48 ± 1.5e−2	6.07 ± 5.6e−2	9.99 ± 8e−2	2.08e1 ± 4.1e−1	8.15e1 ± 5.5e−1

Table 7: Measured latency (s) of single-layer causal linear attention on an NVIDIA RTX 5090 GPU with FP32 inputs, batch size 16, and no exponentially decaying causal mask. Entries report the mean and standard deviation over 15 measurements. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	**1.67e-4** ± 5.5e−7	2.68e−3 ± 6.4e−6	4.51e−2 ± 1.2e−4	OOM	OOM	OOM	OOM
Cdotp	1.69e−4 ± 2.7e−7	**6.98e-4** ± 9e−7	**2.79e-3** ± 4.9e−6	**1.12e-2** ± 3.2e−5	**1.76e-2** ± 4.8e−5	**3.53e-2** ± 4.6e−5	OOM
Cdotp_torch	9.08e−3 ± 2.6e−4	3.53e−2 ± 2.7e−4	1.45e−1 ± 1.2e−3	5.68e−1 ± 7.6e−3	9.62e−1 ± 1.9e−2	2.24 ± 4e−2	OOM
Bb	6.19e−4 ± 5.1e−7	2.43e−3 ± 5.3e−6	9.64e−3 ± 2.8e−5	3.88e−2 ± 6.2e−5	6.06e−2 ± 4.4e−4	1.22e−1 ± 7.6e−5	OOM
LA	2.09e−4 ± 4.1e−6	9.17e−4 ± 3.3e−6	3.54e−3 ± 6.9e−6	1.41e−2 ± 2.3e−5	2.22e−2 ± 3.8e−5	4.52e−2 ± 1.2e−4	OOM
LA_torch	6.48e−3 ± 4.5e−4	9.52e−3 ± 1.7e−4	1.92e−2 ± 3.6e−4	5.59e−2 ± 1.2e−3	8.4e−2 ± 9.3e−4	1.55e−1 ± 1.2e−3	OOM
Recur	4.01e−4 ± 2.7e−7	2.69e−3 ± 2.1e−4	1.47e−2 ± 4.6e−5	7.68e−2 ± 2.8e−4	1.39e−1 ± 7.3e−4	OOM	OOM

Table 8: Measured latency (s) of single-layer causal linear attention on an NVIDIA RTX 5090 GPU with FP32 inputs, batch size 16, and an exponentially decaying causal mask. Entries report the mean and standard deviation over 15 measurements. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	6.15e−2 ± 1.2e−3	2.81e−1 ± 1.5e−3	1.64 ± 2.3e−2	OOM	OOM	OOM	OOM
Cdotp	**1.93e-4** ± 3.7e−7	**7.71e-4** ± 1e−6	**3.09e-3** ± 1.3e−5	**1.25e-2** ± 4.6e−5	**1.95e-2** ± 7e−5	**3.93e-2** ± 1.1e−4	OOM
Cdotp_torch	9.6e−3 ± 5.6e−5	3.82e−2 ± 7e−5	1.57e−1 ± 1.4e−3	6.2e−1 ± 7.3e−3	1.04 ± 6.8e−3	2.38 ± 7e−3	OOM
Bb	1.7e−2 ± 1.3e−4	1.86e−2 ± 8.4e−5	2.79e−2 ± 3e−4	6.35e−2 ± 2.6e−4	9.48e−2 ± 1.8e−3	1.87e−1 ± 3.7e−3	OOM
LA	2.27e−4 ± 2.8e−6	9.83e−4 ± 3.2e−6	3.87e−3 ± 9.2e−6	1.51e−2 ± 1.5e−5	2.26e−2 ± 2.4e−5	4.76e−2 ± 8e−5	OOM
LA_torch	6.65e−3 ± 3.5e−4	9.66e−3 ± 5.4e−4	2.13e−2 ± 1.6e−4	6.5e−2 ± 6.3e−4	9.45e−2 ± 3.1e−4	1.77e−1 ± 6.3e−4	OOM
Recur	9.24e−2 ± 7.4e−4	3.72e−1 ± 4.3e−3	1.53 ± 1.5e−2	5.97 ± 4.1e−2	1.01e1 ± 9.9e−2	OOM	OOM

Table 9: Measured latency (s) of single-layer causal linear attention on an NVIDIA A800 GPU for FP32 inputs at batch size 1 without exponentially decaying causal masks. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	**9.56e-5** ± 3.7e-7	3.92e-4 ± 2.7e-7	5.91e-3 ± 7.7e-6	9.4e-2 ± 1e-4	OOM	OOM	OOM
Cdotp	2.36e-4 ± 2.2e-7	9.32e-4 ± 2.7e-6	3.7e-3 ± 6.8e-6	1.6e-2 ± 2.4e-5	2.69e-2 ± 5e-5	1.07e-1 ± 7.6e-5	5.66e-1 ± 7e-3
Cdotp_torch	1.43e-2 ± 3.6e-5	5.61e-2 ± 3.5e-5	2.25e-1 ± 8.6e-5	1.21 ± 4.1e-4	1.39 ± 5.5e-4	2.9 ± 8.2e-2	1.11e1 ± 8.9e-2
Bb	1.06e-3 ± 4.5e-5	4.8e-3 ± 5.1e-4	1.84e-2 ± 4.7e-4	7.08e-2 ± 1.1e-3	1.15e-1 ± 1.8e-3	2.11e-1 ± 1.9e-3	8.9e-1 ± 2.4e-2
LA	1.15e-4 ± 4.5e-6	**1.28e-4** ± 4.2e-6	**2.42e-4** ± 1.8e-6	**9.21e-4** ± 1.5e-6	**1.43e-3** ± 1.7e-6	**2.85e-3** ± 1.3e-5	**1.12e-2** ± 5.8e-5
LA_torch	1e-3 ± 9.7e-6	1.35e-3 ± 8e-6	2.78e-3 ± 3.2e-5	8.84e-3 ± 5.2e-5	1.35e-2 ± 2.1e-4	2.65e-2 ± 3.7e-4	9.96e-2 ± 2.6e-4
Recur	6.44e-4 ± 7e-7	2.8e-3 ± 5.7e-6	1.15e-2 ± 1.5e-3	4.57e-2 ± 6.1e-5	9.12e-2 ± 4.6e-5	1.82e-1 ± 1.2e-4	7.35e-1 ± 1e-3

Table 10: Measured latency (s) of single-layer causal linear attention on an NVIDIA A800 GPU for FP32 inputs at batch size 1 and exponentially decaying causal masks. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	1.12e-1 ± 1.7e-3	5.08e-1 ± 1.7e-3	2.593 ± 8.2e-3	1.86e1 ± 1.5e-1	OOM	OOM	OOM
Cdotp	2.36e-4 ± 1.3e-7	9.28e-4 ± 2.9e-6	3.68e-3 ± 7.5e-6	1.66e-2 ± 2.3e-5	2.7e-2 ± 7.7e-5	1.06e-1 ± 6e-5	5.64e-1 ± 7.4e-3
Cdotp_torch	1.59e-2 ± 3.2e-5	8.37e-2 ± 5.5e-5	2.51e-1 ± 5.7e-5	1.36 ± 4.2e-4	1.54 ± 8.6e-4	3.46 ± 1.2e-1	1.26e1 ± 1.6e-1
Bb	5.53e-1 ± 6.4e-2	3.25e-1 ± 4.8e-2	5.4e-1 ± 5.1e-2	5.56e-1 ± 2.8e-2	8.53e-1 ± 3.3e-2	9.26e-1 ± 4.2e-2	1.69 ± 3.7e-2
LA	**1.23e-4** ± 1.3e-5	**1.37e-4** ± 5.1e-6	**4.83e-4** ± 3.6e-6	**1.85e-3** ± 7.3e-6	**2.91e-3** ± 1.2e-5	**5.67e-3** ± 1.4e-5	**2.22e-2** ± 4.4e-5
LA_torch	1.21e-3 ± 6.5e-6	1.56e-3 ± 5e-6	3.45e-3 ± 8.6e-5	1.18e-2 ± 4.2e-4	4.66e-3 ± 2.5e-6	3.21e-2 ± 4.1e-4	1.54e-1 ± 3.7e-3
Recur	2e-1 ± 2.6e-3	8.1e-1 ± 6.4e-3	3.17 ± 2e-2	1.18e1 ± 9.7e-2	2.0e1 ± 8.6e-3	4.01e1 ± 7.45e-2	1.59e2 ± 6.86e-1

Table 11: Measured latency (s) of single-layer causal linear attention on an NVIDIA A800 GPU for FP32 inputs at batch size 16 and without exponentially decaying causal masks. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	5.46e-4 ± 3.4e-7	5.33e-3 ± 1.2e-5	9.65e-2 ± 2e-4	OOM	OOM	OOM	OOM
Cdotp	4.42e-4 ± 1.5e-7	1.82e-3 ± 7.9e-6	7.50e-3 ± 2.4e-5	3.15e-2 ± 6.6e-5	4.71e-2 ± 2.1e-5	9.97e-2 ± 3.1e-4	OOM
Cdotp_torch	3.31e-2 ± 2.9e-6	1.32e-1 ± 1.6e-5	5.31e-1 ± 4.5e-5	2.12 ± 1.0e-4	3.38 ± 2.1e-4	7.04 ± 4.5e-4	OOM
Bb	2.54e-3 ± 1e-4	9.33e-3 ± 1.1e-4	3.68e-2 ± 8.6e-5	1.46e-1 ± 4.6e-5	2.31e-1 ± 1e-4	4.65e-1 ± 8.9e-5	OOM
LA	**2.43e-4** ± 2.1e-6	**9.28e-4** ± 4.0e-6	**3.69e-3** ± 7.9e-6	**1.48e-2** ± 6.1e-5	**2.31e-2** ± 9.5e-5	**4.63e-2** ± 2.0e-4	OOM
LA_torch	1.18e-2 ± 2.3e-4	1.82e-2 ± 3.2e-5	3.9e-2 ± 4.3e-5	1.18e-1 ± 1.9e-4	1.78e-1 ± 2.6e-4	3.47e-1 ± 1e-4	OOM
Recur	1.49e-3 ± 6.8e-5	8.6e-3 ± 9.3e-5	4.36e-2 ± 4.4e-5	2.12e-1 ± 1e-4	3.81e-1 ± 2.8e-4	8.2e-1 ± 3.3e-4	OOM

Table 12: Measured latency (s) of single-layer causal linear attention on an NVIDIA A800 GPU for FP32 inputs at batch size 16 and exponentially decaying causal masks. The lowest mean latency in each column is highlighted in bold.

Method \ Seqlen	128	512	2,048	8,192	12,800	25,600	102,400
Vanilla	1.12e-1 ± 8.9e-4	5.43e-1 ± 1.7e-2	2.77 ± 2.8e-2	OOM	OOM	OOM	OOM
Cdotp	**4.88e-4** ± 7.3e-7	**2.03e-3** ± 1.1e-5	**8.36e-3** ± 2.2e-5	3.50e-2 ± 1.1e-4	**5.25e-2** ± 3.3e-5	1.12e-1 ± 2.3e-4	OOM
Cdotp_torch	4.58e-2 ± 4.8e-6	1.83e-1 ± 2e-5	7.34e-1 ± 5.7e-5	2.93 ± 8.5e-5	4.64 ± 3.2e-5	9.59 ± 4.8e-4	OOM
Bb	4.52e-2 ± 2e-3	6.29e-2 ± 7.8e-3	8.55e-2 ± 6.2e-4	2.93e-1 ± 1.8e-2	3.39e-1 ± 7.2e-4	1.31 ± 3.3e-2	OOM
LA	6.63e-4 ± 3.4e-6	2.21e-3 ± 2.0e-5	8.72e-3 ± 7.5e-6	**3.49e-2** ± 8.3e-6	5.44e-2 ± 1.1e-5	**1.10e-1** ± 2.0e-5	OOM
LA_torch	1.15e-2 ± 2e-4	1.98e-2 ± 2e-5	4.49e-2 ± 6.6e-5	1.41e-1 ± 2.9e-4	2.13e-1 ± 4.2e-4	4.16e-1 ± 9.3e-5	OOM
Recur	1.87e-1 ± 2.1e-3	7.4e-1 ± 7.7e-4	2.98 ± 4.1e-4	1.2e1 ± 6.7e-3	2.1e1 ± 7.6e-2	4.26e1 ± 1.12e-2	OOM

C.3 Memory measurements

Moreover, we measure the GPU memory usage through the peak allocated memory since the beginning of each operator, which is reported in GB (Gigabytes), or simply as "out-of-memory" errors, to reflect memory consumption.

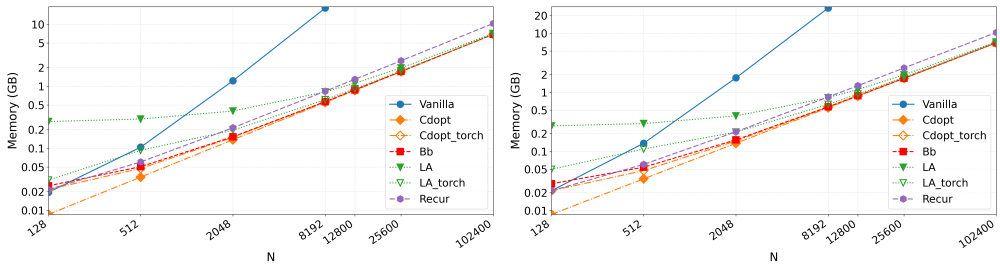
We report the GPU memory usage of each single-layer causal linear attention method across all four settings in Figure 4. Unlike the other evaluated methods, **Vanilla** explicitly constructs a dense attention matrix with $\mathcal{O}(N^2)$ storage. Among the non-dense methods, **Cdotp** exhibits low memory usage, while **Recur** has a larger memory footprint.

D Additional model-level prefill evaluation

D.1 RetNet-1.3B on A800

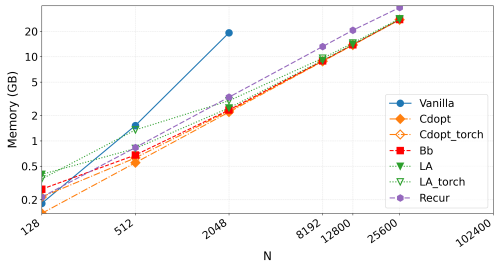
Table 13 details the cumulative prefill latency and performance comparisons for the RetNet-1.3B architecture evaluated under FP32 precision on an A800 GPU. The "Original" method therein denotes the original attention computation method implemented in RetNet-1.3B. The "**LA_ref**" method indicates a production-level reference implementation of RetNet with chunk-wise computation (i.e., a variant of **LA**) [2].

[2] See https://github.com/fla-org/flash-linear-attention/blob/main/fla/layers/multiscale_retention.py.

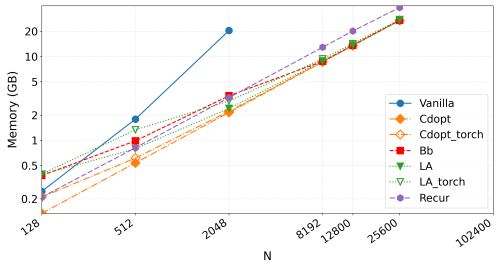


(a) Memory usage under the setting in Table 9.

(b) Memory usage under the setting in Table 10.



(c) Memory usage under the setting in Table 11.



(d) Memory usage under the setting in Table 12.

Figure 4: Measured memory footprint of single-layer causal linear attention on an NVIDIA A800 GPU with FP32 inputs. The plot features natural-log scaling on both axes, and out-of-memory (OOM) points are not displayed.

Table 13: Cumulative prefill time (s) for one pass over all 200 TREC samples from LongBench using RetNet-1.3B in its original FP32 precision on an A800 GPU, with batch sizes of 1 or 2. Entries report the mean and sample standard deviation across three runs after discarding the first of four; each run sums the measured time across input batches. Methods with cumulative times exceeding 1 hour are denoted as $> 1h$.

Method	**FP32 (s)**	
	$B = 1$	$B = 2$
Original	695.48 ± 0.72	817.31 ± 0.30
Vanilla	$> 1h$	$> 1h$
Bb	688.12 ± 0.79	600.67 ± 0.71
Cdotp	397.48 ± 2.81	437.69 ± 0.04
Cdotp_torch	$> 1h$	$> 1h$
LA_ref	366.88 ± 2.47	394.64 ± 0.53
LA	$\mathbf{351.92} \pm 2.80$	$\mathbf{388.9} \pm 0.41$
LA_torch	374.54 ± 1.86	398.47 ± 0.79
Recur	$> 1h$	$> 1h$

D.2 Evaluation on Blackwell

We additionally evaluate TransNormerLLM-7B and RetNet-1.3B on an NVIDIA RTX 5090 GPU with the Blackwell architecture, following the protocol and method naming in Section 6.2.

E Extended single-layer numerical evaluation

E.1 Original single-layer evaluation

The original single-layer evaluation uses FP32 and BF16 inputs with a fixed sequence length $N = 1024$. We report the mean and maximum relative Frobenius norm error against an FP64 reference computed by the **Vanilla** method.

Table 14: Cumulative prefill time (s) for TransNormerLLM-7B on an NVIDIA RTX 5090 GPU, with FP32 and BF16 inputs and batch sizes of 1 or 2. Each run processes all 200 TREC samples and sums prefill times across batches, counting each batch once. Entries report the mean and sample standard deviation over three runs after discarding the first of four. The "Original" method denotes the model's original attention implementation. OOM denotes out-of-memory; cumulative times exceeding one hour are shown as $> 1h$. The lowest measured mean in each column is highlighted in bold.

Method (FP32 inputs) \ Batch size	1	2
	Total time (s)	Total time (s)
Original	491.92 ± 3.14	OOM
Vanilla	OOM	OOM
Bb	1883.95 ± 136.62	OOM
Cdotp	425.39 ± 0.06	OOM
Cdotp_torch	$> 1h$	OOM
LA	**418.84** ± 0.76	OOM
LA_torch	479.20 ± 1.17	OOM
Recur	$> 1h$	OOM

Method (BF16 inputs) \ Batch size	1	2
	Total time (s)	Total time (s)
Original	321.11 ± 1.58	212.26 ± 1.67
Vanilla	673.15 ± 0.05	759.07 ± 0.12
Bb	1303.99 ± 45.98	763.51 ± 2.44
Cdotp	160.57 ± 0.07	173.32 ± 0.03
Cdotp_torch	$> 1h$	2976.97 ± 1.56
LA	**122.47** ± 0.17	**146.95** ± 0.19
LA_torch	314.26 ± 8.02	202.42 ± 1.07
Recur	$> 1h$	3387.84 ± 12.46

Table 15: Cumulative prefill time (s) for RetNet-1.3B on an NVIDIA RTX 5090 GPU, with FP32 inputs and batch sizes of 1 or 2. Each run processes all 200 TREC samples and sums prefill times across batches, counting each batch once. Entries report the mean and sample standard deviation over three runs after discarding the first of four. The "Original" method denotes the model's original attention implementation. OOM denotes out-of-memory; cumulative times exceeding one hour are shown as $> 1h$. The lowest measured mean in each column is highlighted in bold.

Method (FP32 inputs) \ Batch size	1	2
	Total time (s)	Total time (s)
Original	579.70 ± 5.73	OOM
Vanilla	558.84 ± 4.15	482.73 ± 3.01
Bb	899.06 ± 21.01	600.15 ± 14.83
Cdotp	**398.50** ± 2.48	308.44 ± 3.70
Cdotp_torch	3278.71 ± 10.72	2239.88 ± 7.94
LA	407.74 ± 2.79	**297.98** ± 2.59
LA_torch	501.16 ± 6.55	326.59 ± 1.17
Recur	$> 1h$	2974.03 ± 2.59

Table 16: Numerical error (mean/max) with FP32 and BF16 inputs over 64 trials.

Method	FP32		BF16	
	w/o decay	w/ decay	w/o decay	w/ decay
Vanilla	5.10e-7/5.11e-7	2.86e-7/2.87e-7	2.34e-3/2.35e-3	2.85e-3/2.86e-3
Cdotp	1.57e-7/1.58e-7	4.40e-6/4.47e-6	1.66e-3/1.66e-3	9.87e-2/9.95e-2
Cdotp_torch	4.92e-7/4.94e-7	4.40e-6/4.47e-6	1.66e-3/1.66e-3	9.87e-2/9.95e-2
Bb	2.42e-7/2.42e-7	2.48e-7/2.49e-7	6.28e-3/6.30e-3	6.13e-3/6.15e-3
LA	1.47e-3/1.48e-3	1.47e-3/1.48e-3	1.71e-3/1.71e-3	1.79e-3/1.79e-3
LA_torch	3.44e-7/3.44e-7	2.71e-7/2.72e-7	2.35e-3/2.35e-3	2.46e-3/2.46e-3
Recur	3.88e-7/3.89e-7	2.39e-7/2.40e-7	3.17e-3/3.18e-3	4.75e-3/4.76e-3

Table 17: Relative Frobenius norm error for single-layer causal linear attention at $N = 128$, across ranks and input dtypes. Each entry reports mean/max error over 64 trials against the FP64 reference.

Rank	Method	FP32 inputs		BF16 inputs	
		Without decay	With decay	Without decay	With decay
32	**Vanilla**	$1.96 \times 10^{-7}/1.97 \times 10^{-7}$	$1.69 \times 10^{-7}/1.70 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.84 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$1.05 \times 10^{-7}/1.05 \times 10^{-7}$	$1.27 \times 10^{-6}/1.31 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$3.62 \times 10^{-2}/3.72 \times 10^{-2}$
	Cdotp_torch	$1.88 \times 10^{-7}/1.90 \times 10^{-7}$	$1.27 \times 10^{-6}/1.31 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$3.62 \times 10^{-2}/3.72 \times 10^{-2}$
	Bb	$1.39 \times 10^{-7}/1.40 \times 10^{-7}$	$1.48 \times 10^{-7}/1.49 \times 10^{-7}$	$3.24 \times 10^{-3}/3.25 \times 10^{-3}$	$4.18 \times 10^{-3}/4.21 \times 10^{-3}$
	LA	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$2.35 \times 10^{-3}/2.36 \times 10^{-3}$	$2.78 \times 10^{-3}/2.80 \times 10^{-3}$
	LA_torch	$1.96 \times 10^{-7}/1.97 \times 10^{-7}$	$1.69 \times 10^{-7}/1.70 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.34 \times 10^{-3}/2.36 \times 10^{-3}$
	Recur	$1.54 \times 10^{-7}/1.56 \times 10^{-7}$	$1.44 \times 10^{-7}/1.45 \times 10^{-7}$	$2.90 \times 10^{-3}/2.91 \times 10^{-3}$	$4.37 \times 10^{-3}/4.41 \times 10^{-3}$
128	**Vanilla**	$2.63 \times 10^{-7}/2.64 \times 10^{-7}$	$2.44 \times 10^{-7}/2.46 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.84 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$1.09 \times 10^{-7}/1.10 \times 10^{-7}$	$1.28 \times 10^{-6}/1.31 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$3.62 \times 10^{-2}/3.71 \times 10^{-2}$
	Cdotp_torch	$2.25 \times 10^{-7}/2.26 \times 10^{-7}$	$1.28 \times 10^{-6}/1.31 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$3.62 \times 10^{-2}/3.71 \times 10^{-2}$
	Bb	$2.06 \times 10^{-7}/2.07 \times 10^{-7}$	$2.30 \times 10^{-7}/2.31 \times 10^{-7}$	$3.24 \times 10^{-3}/3.25 \times 10^{-3}$	$4.18 \times 10^{-3}/4.21 \times 10^{-3}$
	LA	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.90 \times 10^{-3}/2.91 \times 10^{-3}$
	LA_torch	$2.63 \times 10^{-7}/2.64 \times 10^{-7}$	$2.44 \times 10^{-7}/2.46 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.34 \times 10^{-3}/2.36 \times 10^{-3}$
	Recur	$2.17 \times 10^{-7}/2.18 \times 10^{-7}$	$1.98 \times 10^{-7}/2.00 \times 10^{-7}$	$2.89 \times 10^{-3}/2.90 \times 10^{-3}$	$4.38 \times 10^{-3}/4.40 \times 10^{-3}$
512	**Vanilla**	$4.38 \times 10^{-7}/4.42 \times 10^{-7}$	$4.27 \times 10^{-7}/4.31 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.84 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$2.29 \times 10^{-7}/2.30 \times 10^{-7}$	$1.28 \times 10^{-6}/1.31 \times 10^{-6}$	$1.66 \times 10^{-3}/1.67 \times 10^{-3}$	$3.62 \times 10^{-2}/3.68 \times 10^{-2}$
	Cdotp_torch	$2.06 \times 10^{-7}/2.08 \times 10^{-7}$	$1.28 \times 10^{-6}/1.31 \times 10^{-6}$	$1.66 \times 10^{-3}/1.67 \times 10^{-3}$	$3.62 \times 10^{-2}/3.68 \times 10^{-2}$
	Bb	$3.74 \times 10^{-7}/3.76 \times 10^{-7}$	$4.19 \times 10^{-7}/4.22 \times 10^{-7}$	$3.24 \times 10^{-3}/3.25 \times 10^{-3}$	$4.18 \times 10^{-3}/4.21 \times 10^{-3}$
	LA	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.90 \times 10^{-3}/2.91 \times 10^{-3}$
	LA_torch	$4.38 \times 10^{-7}/4.42 \times 10^{-7}$	$4.27 \times 10^{-7}/4.31 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.34 \times 10^{-3}/2.36 \times 10^{-3}$
	Recur	$3.80 \times 10^{-7}/3.81 \times 10^{-7}$	$3.36 \times 10^{-7}/3.39 \times 10^{-7}$	$2.90 \times 10^{-3}/2.91 \times 10^{-3}$	$4.38 \times 10^{-3}/4.40 \times 10^{-3}$

E.2 Extended sequence-length and rank evaluation

We evaluate sequence lengths $N \in \{128, 1024, 8192\}$ and ranks $r \in \{32, 128, 512\}$ with FP32 and BF16 inputs, with and without exponential decay. Each setting uses 64 trials. The reference output is computed in FP64 using the same dtype-quantized inputs. For decaying masks, the benchmark uses fixed head-specific slopes.

For BF16 inputs with exponential decay, the errors of **Cdotp** and **Cdotp_torch** increase with the tested sequence lengths. Varying the rank does not change the main qualitative observations. The practical effect of these deviations may depend on the computation method used during pretraining.

F Numerical evaluation of implementation substitution

We compare alternative attention implementations with each model's native implementation on 200 TREC prompts, using batch size 1 and the models' original input dtypes: FP32 for RetNet-1.3B and BF16 for TransNormerLLM-7B. We report perplexity (PPL), relative Frobenius norm error of the final-layer hidden state at the last input token, and greedy next-token disagreement. The relative error is $\|r_{\text{original}} - r_{\text{method}}\|_F / \|r_{\text{original}}\|_F$, and ΔPPL denotes method PPL minus original PPL.

For RetNet-1.3B with FP32 inputs, the tested substitutions produce small deviations and no greedy next-token disagreements on these prompts. For TransNormerLLM-7B with BF16 inputs, deviations vary considerably across methods.

Table 18: Relative Frobenius norm error for single-layer causal linear attention at $N = 1,024$, across ranks and input dtypes. Each entry reports mean/max error over 64 trials against the FP64 reference.

Rank	Method	FP32 inputs		BF16 inputs	
		Without decay	With decay	Without decay	With decay
32	**Vanilla**	$4.79 \times 10^{-7}/4.80 \times 10^{-7}$	$2.28 \times 10^{-7}/2.29 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.85 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$1.54 \times 10^{-7}/1.55 \times 10^{-7}$	$4.41 \times 10^{-6}/4.53 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$9.87 \times 10^{-2}/9.98 \times 10^{-2}$
	Cdotp_torch	$4.76 \times 10^{-7}/4.78 \times 10^{-7}$	$4.41 \times 10^{-6}/4.53 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$9.87 \times 10^{-2}/9.98 \times 10^{-2}$
	Bb	$1.69 \times 10^{-7}/1.70 \times 10^{-7}$	$1.75 \times 10^{-7}/1.76 \times 10^{-7}$	$6.28 \times 10^{-3}/6.31 \times 10^{-3}$	$6.13 \times 10^{-3}/6.17 \times 10^{-3}$
	LA	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.91 \times 10^{-3}/2.92 \times 10^{-3}$
	LA_torch	$2.95 \times 10^{-7}/2.97 \times 10^{-7}$	$2.07 \times 10^{-7}/2.08 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.46 \times 10^{-3}/2.46 \times 10^{-3}$
	Recur	$3.48 \times 10^{-7}/3.50 \times 10^{-7}$	$1.85 \times 10^{-7}/1.87 \times 10^{-7}$	$3.17 \times 10^{-3}/3.18 \times 10^{-3}$	$4.75 \times 10^{-3}/4.77 \times 10^{-3}$
128	**Vanilla**	$5.10 \times 10^{-7}/5.11 \times 10^{-7}$	$2.88 \times 10^{-7}/2.88 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.85 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$1.57 \times 10^{-7}/1.58 \times 10^{-7}$	$4.40 \times 10^{-6}/4.45 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$9.87 \times 10^{-2}/9.94 \times 10^{-2}$
	Cdotp_torch	$4.92 \times 10^{-7}/4.93 \times 10^{-7}$	$4.40 \times 10^{-6}/4.45 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$9.87 \times 10^{-2}/9.94 \times 10^{-2}$
	Bb	$2.42 \times 10^{-7}/2.42 \times 10^{-7}$	$2.48 \times 10^{-7}/2.48 \times 10^{-7}$	$6.28 \times 10^{-3}/6.30 \times 10^{-3}$	$6.13 \times 10^{-3}/6.15 \times 10^{-3}$
	LA	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.47 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.97 \times 10^{-3}/2.98 \times 10^{-3}$
	LA_torch	$3.44 \times 10^{-7}/3.44 \times 10^{-7}$	$2.71 \times 10^{-7}/2.72 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.46 \times 10^{-3}/2.46 \times 10^{-3}$
	Recur	$3.88 \times 10^{-7}/3.90 \times 10^{-7}$	$2.39 \times 10^{-7}/2.40 \times 10^{-7}$	$3.17 \times 10^{-3}/3.18 \times 10^{-3}$	$4.75 \times 10^{-3}/4.76 \times 10^{-3}$
512	**Vanilla**	$6.19 \times 10^{-7}/6.20 \times 10^{-7}$	$4.54 \times 10^{-7}/4.55 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.85 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$4.93 \times 10^{-7}/4.94 \times 10^{-7}$	$4.40 \times 10^{-6}/4.43 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$9.87 \times 10^{-2}/9.93 \times 10^{-2}$
	Cdotp_torch	$4.84 \times 10^{-7}/4.85 \times 10^{-7}$	$4.40 \times 10^{-6}/4.43 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$9.87 \times 10^{-2}/9.93 \times 10^{-2}$
	Bb	$4.22 \times 10^{-7}/4.23 \times 10^{-7}$	$4.29 \times 10^{-7}/4.31 \times 10^{-7}$	$6.28 \times 10^{-3}/6.29 \times 10^{-3}$	$6.13 \times 10^{-3}/6.14 \times 10^{-3}$
	LA	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.48 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.97 \times 10^{-3}/2.98 \times 10^{-3}$
	LA_torch	$4.91 \times 10^{-7}/4.92 \times 10^{-7}$	$4.43 \times 10^{-7}/4.44 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.46 \times 10^{-3}/2.46 \times 10^{-3}$
	Recur	$5.20 \times 10^{-7}/5.21 \times 10^{-7}$	$3.86 \times 10^{-7}/3.87 \times 10^{-7}$	$3.17 \times 10^{-3}/3.18 \times 10^{-3}$	$4.75 \times 10^{-3}/4.76 \times 10^{-3}$

Table 19: Relative Frobenius norm error for single-layer causal linear attention at $N = 8,192$, across ranks and input dtypes. Each entry reports mean/max error over 64 trials against the FP64 reference.

Rank	Method	FP32 inputs		BF16 inputs	
		Without decay	With decay	Without decay	With decay
32	**Vanilla**	$1.33 \times 10^{-6}/1.33 \times 10^{-6}$	$2.38 \times 10^{-7}/2.38 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.85 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$3.47 \times 10^{-7}/3.49 \times 10^{-7}$	$5.19 \times 10^{-6}/5.22 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$1.11 \times 10^{-1}/1.12 \times 10^{-1}$
	Cdotp_torch	$1.33 \times 10^{-6}/1.33 \times 10^{-6}$	$5.19 \times 10^{-6}/5.22 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$1.11 \times 10^{-1}/1.12 \times 10^{-1}$
	Bb	$2.78 \times 10^{-7}/2.79 \times 10^{-7}$	$1.81 \times 10^{-7}/1.81 \times 10^{-7}$	$1.57 \times 10^{-2}/1.58 \times 10^{-2}$	$6.52 \times 10^{-3}/6.53 \times 10^{-3}$
	LA	$1.48 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.47 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.92 \times 10^{-3}/2.93 \times 10^{-3}$
	LA_torch	$3.16 \times 10^{-7}/3.17 \times 10^{-7}$	$2.10 \times 10^{-7}/2.11 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.48 \times 10^{-3}/2.49 \times 10^{-3}$
	Recur	$9.41 \times 10^{-7}/9.47 \times 10^{-7}$	$1.93 \times 10^{-7}/1.94 \times 10^{-7}$	$3.21 \times 10^{-3}/3.22 \times 10^{-3}$	$4.93 \times 10^{-3}/4.94 \times 10^{-3}$
128	**Vanilla**	$1.34 \times 10^{-6}/1.34 \times 10^{-6}$	$2.96 \times 10^{-7}/2.96 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.85 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$3.48 \times 10^{-7}/3.49 \times 10^{-7}$	$5.19 \times 10^{-6}/5.22 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$1.11 \times 10^{-1}/1.12 \times 10^{-1}$
	Cdotp_torch	$1.33 \times 10^{-6}/1.34 \times 10^{-6}$	$5.19 \times 10^{-6}/5.22 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$1.11 \times 10^{-1}/1.12 \times 10^{-1}$
	Bb	$3.29 \times 10^{-7}/3.29 \times 10^{-7}$	$2.52 \times 10^{-7}/2.52 \times 10^{-7}$	$1.57 \times 10^{-2}/1.58 \times 10^{-2}$	$6.52 \times 10^{-3}/6.53 \times 10^{-3}$
	LA	$1.48 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.47 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.98 \times 10^{-3}/2.98 \times 10^{-3}$
	LA_torch	$3.62 \times 10^{-7}/3.62 \times 10^{-7}$	$2.74 \times 10^{-7}/2.74 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.48 \times 10^{-3}/2.48 \times 10^{-3}$
	Recur	$9.58 \times 10^{-7}/9.61 \times 10^{-7}$	$2.47 \times 10^{-7}/2.47 \times 10^{-7}$	$3.21 \times 10^{-3}/3.21 \times 10^{-3}$	$4.93 \times 10^{-3}/4.94 \times 10^{-3}$
512	**Vanilla**	$1.38 \times 10^{-6}/1.38 \times 10^{-6}$	$4.59 \times 10^{-7}/4.59 \times 10^{-7}$	$2.34 \times 10^{-3}/2.35 \times 10^{-3}$	$2.85 \times 10^{-3}/2.86 \times 10^{-3}$
	Cdotp	$1.33 \times 10^{-6}/1.33 \times 10^{-6}$	$5.19 \times 10^{-6}/5.20 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$1.11 \times 10^{-1}/1.12 \times 10^{-1}$
	Cdotp_torch	$1.33 \times 10^{-6}/1.33 \times 10^{-6}$	$5.19 \times 10^{-6}/5.20 \times 10^{-6}$	$1.66 \times 10^{-3}/1.66 \times 10^{-3}$	$1.11 \times 10^{-1}/1.12 \times 10^{-1}$
	Bb	$4.80 \times 10^{-7}/4.81 \times 10^{-7}$	$4.32 \times 10^{-7}/4.33 \times 10^{-7}$	$1.57 \times 10^{-2}/1.57 \times 10^{-2}$	$6.52 \times 10^{-3}/6.53 \times 10^{-3}$
	LA	$1.48 \times 10^{-3}/1.48 \times 10^{-3}$	$1.47 \times 10^{-3}/1.47 \times 10^{-3}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.98 \times 10^{-3}/2.98 \times 10^{-3}$
	LA_torch	$5.04 \times 10^{-7}/5.04 \times 10^{-7}$	$4.45 \times 10^{-7}/4.45 \times 10^{-7}$	$2.35 \times 10^{-3}/2.35 \times 10^{-3}$	$2.48 \times 10^{-3}/2.48 \times 10^{-3}$
	Recur	$1.02 \times 10^{-6}/1.02 \times 10^{-6}$	$3.93 \times 10^{-7}/3.93 \times 10^{-7}$	$3.21 \times 10^{-3}/3.21 \times 10^{-3}$	$4.93 \times 10^{-3}/4.94 \times 10^{-3}$

G Detailed complexity analyses

This appendix provides detailed derivations and pseudocode for the complexity bounds summarized in Table 1. Unless otherwise specified, complexity is reported per batch and per head. Because applying a decaying mask introduces additional computational overhead without altering the asymptotic complexity, we focus our analysis on the non-decaying mask $\boldsymbol{M}$.

We begin with the **`vanilla`** method. Computing eq. (1) first requires the matrix product $\boldsymbol{B}\boldsymbol{C}^\intercal$ with time complexity $\mathcal{O}(N^2 r)$. The resulting dense $N \times N$ matrix is then multiplied element-wise with the mask $\boldsymbol{M}$, incurring $\mathcal{O}(N^2)$ time and $\mathcal{O}(N^2)$ space for the intermediate matrix. Finally, multiplication with $\boldsymbol{V}$ takes $\mathcal{O}(N^2 d)$ time. Thus, a direct GPU implementation yields an overall time complexity of $\mathcal{O}(N^2(r+d))$ and a space complexity of $\mathcal{O}(N^2 + N(r+d))$, including the input and output tensors and the materialized $N \times N$ matrix.

Following the algorithm taxonomy in Section 3.2, we then first analyze the **iterative** methods in Section G.1. Although their time complexities are established in the original literature, we provide a

Table 20: Numerical comparison against each model's native attention implementation on 200 TREC prompts with batch size 1. Relative errors are summarized across prompts; next-token disagreement reports the count and percentage of greedy predictions differing from the native implementation.

Method	PPL	ΔPPL	Relative error (mean / max / median)	Next-token disagreement
RetNet-1.3B (FP32 inputs); Original PPL = 1350.1151				
Vanilla	1350.1150	-9.076×10^{-5}	8.416×10^{-7}/1.253×10^{-6}/8.258×10^{-7}	0 / 200 (0.00%)
Cdotp	1350.1151	-5.255×10^{-5}	9.330×10^{-7}/1.434×10^{-6}/9.168×10^{-7}	0 / 200 (0.00%)
Cdotp_torch	1350.1151	-2.548×10^{-5}	9.280×10^{-7}/1.247×10^{-6}/9.204×10^{-7}	0 / 200 (0.00%)
Bb	1350.1151	-3.025×10^{-5}	9.401×10^{-7}/1.578×10^{-6}/9.234×10^{-7}	0 / 200 (0.00%)
LA	1350.1396	2.448×10^{-2}	2.255×10^{-4}/3.529×10^{-4}/2.193×10^{-4}	0 / 200 (0.00%)
LA_torch	1350.1151	-5.892×10^{-5}	9.179×10^{-7}/1.489×10^{-6}/8.911×10^{-7}	0 / 200 (0.00%)
Recur	1350.1151	-2.548×10^{-5}	9.354×10^{-7}/1.393×10^{-6}/9.222×10^{-7}	0 / 200 (0.00%)
TransNormerLLM-7B (BF16 inputs); Original PPL = 14.0248				
Vanilla	14.0630	3.827×10^{-2}	2.015×10^{-2}/1.186×10^{-1}/1.482×10^{-2}	12 / 200 (6.00%)
Cdotp	16.9204	2.896×10^{0}	6.537×10^{-1}/9.827×10^{-1}/6.677×10^{-1}	184 / 200 (92.00%)
Cdotp_torch	16.9289	2.904×10^{0}	6.536×10^{-1}/9.821×10^{-1}/6.675×10^{-1}	184 / 200 (92.00%)
Bb	14.0383	1.356×10^{-2}	1.674×10^{-1}/5.462×10^{-1}/1.398×10^{-1}	57 / 200 (28.50%)
LA	14.0568	3.202×10^{-2}	2.010×10^{-2}/1.135×10^{-1}/1.542×10^{-2}	10 / 200 (5.00%)
LA_torch	14.0644	3.962×10^{-2}	1.925×10^{-2}/9.935×10^{-2}/1.444×10^{-2}	12 / 200 (6.00%)
Recur	333.2989	3.193×10^{2}	7.403×10^{-1}/1.366×10^{0}/7.274×10^{-1}	153 / 200 (76.50%)

Algorithm 1 `causal-dot-product`.

Require: Matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N \times r}$, $\boldsymbol{V} \in \mathbb{R}^{N \times d}$.
1: Initialize $\boldsymbol{U} = (\boldsymbol{0})_{r \times d} \in \mathbb{R}^{r \times d}$, $\boldsymbol{O} = (\boldsymbol{0})_{N \times d} \in \mathbb{R}^{N \times d}$.
2: **for** $1 \leq i \leq N$ **do**
3: Extract column vectors $\boldsymbol{b}_i = \boldsymbol{B}_{i,:}^\intercal \in \mathbb{R}^{r \times 1}$, $\boldsymbol{c}_i = \boldsymbol{C}_{i,:}^\intercal \in \mathbb{R}^{r \times 1}$, and $\boldsymbol{v}_i = \boldsymbol{V}_{i,:}^\intercal \in \mathbb{R}^{d \times 1}$.
4: Update $\boldsymbol{U} \leftarrow \boldsymbol{U} + \boldsymbol{c}_i \boldsymbol{v}_i^\intercal$.
5: Compute and store $\boldsymbol{O}_{i,:} = \boldsymbol{b}_i^\intercal \boldsymbol{U}$.
6: **end for**
7: Return the output $\boldsymbol{O}$.

more formal analysis to explicitly account for the dimensions r and d, as well as the space complexity. Furthermore, as a rigorous complexity analysis of the **recursive** approach has not been formally established in prior work, we provide an in-depth derivation in Section G.2.

G.1 Iterative methods

We lay out the complexity analysis for iterative methods as follows.

`causal-dot-product`. Using the notation $\boldsymbol{b}_i = \boldsymbol{B}_{i,:}^\intercal$ and $\boldsymbol{c}_i = \boldsymbol{C}_{i,:}^\intercal \in \mathbb{R}^{r \times 1}$, $\boldsymbol{v}_i = \boldsymbol{V}_{i,:}^\intercal \in \mathbb{R}^{d \times 1}$, and $\boldsymbol{O}_i = \boldsymbol{O}_{i,:}^\intercal$ for the i-th token, a single output row is computed as:

$$\boldsymbol{O}_i^\intercal = \sum_{j=1}^{i} \boldsymbol{b}_i^\intercal \boldsymbol{c}_j \boldsymbol{v}_j^\intercal = \boldsymbol{b}_i^\intercal \sum_{j=1}^{i} \boldsymbol{c}_j \boldsymbol{v}_j^\intercal \in \mathbb{R}^{1 \times d} \tag{4}$$

Let $\boldsymbol{U}_i = \sum_{j=1}^{i} \boldsymbol{c}_j \boldsymbol{v}_j^\intercal \in \mathbb{R}^{r \times d}$. Then eq. (4) is realized by the state recurrence in eq. (3).

The full row-based algorithm is described in Algorithm 1. The time complexity for processing each row of $\boldsymbol{O}$ is $\mathcal{O}(rd)$, leading to a total time complexity of $\mathcal{O}(Nrd)$. Including the input and output tensors, the space complexity is $\mathcal{O}(N(r+d)+rd)$.

`block-based` / `Lightning Attention-2`. Transformer-VQ (Lingle, 2024, Theorem 3.6) proposed a block-by-block computation method designed for its own low-rank matrix operations. We observe that this method can be extended to handle a general low-rank matrix product $\boldsymbol{B}\boldsymbol{C}^\intercal$.

Let the block size be B_c. For simplicity, the block slice $[iB_c : (i+1)B_c]$ is abbreviated as $[i]$ (the index i starts from 0). Let $\boldsymbol{M}$ here be a $B_c \times B_c$ lower triangular causal mask. For a final block of $L < B_c$ rows, use the leading $L \times L$ submatrix of $\boldsymbol{M}$ in the formulas and Algorithm 2. Then

Algorithm 2 **`block-based` / `Lightning Attention-2`**.

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

1: Partition $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ row-wise into $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks of B_c rows. Each block is indexed by $[i]$.
2: Initialize $\boldsymbol{U} = (\boldsymbol{0})_{r\times d} \in \mathbb{R}^{r\times d}$, $\boldsymbol{O} = (\boldsymbol{0})_{N\times d} \in \mathbb{R}^{N\times d}$.
3: Construct the $B_c \times B_c$ mask matrix $\boldsymbol{M}$.
4: **for** $0 \le i < T_c$ **do**
5: Compute and store $\boldsymbol{O}_{[i]} = (\boldsymbol{B}_{[i]}\boldsymbol{C}_{[i]}^\intercal \odot \boldsymbol{M})\boldsymbol{V}_{[i]} + \boldsymbol{B}_{[i]}\boldsymbol{U}$.
6: Update $\boldsymbol{U} \leftarrow \boldsymbol{U} + \boldsymbol{C}_{[i]}^\intercal \boldsymbol{V}_{[i]}$.
7: **end for**
8: Return the output $\boldsymbol{O}$.

the **`block-based`** method can be presented as follows, which happens to be the block version of **`causal-dot-product`**:

$$\begin{aligned}\boldsymbol{O}_{[i]} &= (\boldsymbol{B}_{[i]}\boldsymbol{C}_{[i]}^\intercal \odot \boldsymbol{M})\boldsymbol{V}_{[i]} + \sum_{j=0}^{i-1} \boldsymbol{B}_{[i]}\boldsymbol{C}_{[j]}^\intercal \boldsymbol{V}_{[j]} \\ &= (\boldsymbol{B}_{[i]}\boldsymbol{C}_{[i]}^\intercal \odot \boldsymbol{M})\boldsymbol{V}_{[i]} + \boldsymbol{B}_{[i]} \sum_{j=0}^{i-1} \boldsymbol{C}_{[j]}^\intercal \boldsymbol{V}_{[j]}.\end{aligned}$$

Let $\boldsymbol{U}_{[i]} = \sum_{j=0}^{i} \boldsymbol{C}_{[j]}^\intercal \boldsymbol{V}_{[j]} \in \mathbb{R}^{r\times d}$, with $\boldsymbol{U}_{[-1]} = (\boldsymbol{0})_{r\times d}$. Then, we obtain:

$$\begin{aligned}\boldsymbol{O}_{[i]} &= (\boldsymbol{B}_{[i]}\boldsymbol{C}_{[i]}^\intercal \odot \boldsymbol{M})\boldsymbol{V}_{[i]} + \boldsymbol{B}_{[i]}\boldsymbol{U}_{[i-1]} \\ \boldsymbol{U}_{[i]} &= \boldsymbol{U}_{[i-1]} + \boldsymbol{C}_{[i]}^\intercal \boldsymbol{V}_{[i]}.\end{aligned}$$

The full **`block-based`** algorithm is described in Algorithm 2. The time complexity for processing each block $\boldsymbol{O}_{[i]}$ is $\mathcal{O}(B_c^2 r + B_c^2 d + B_c r d)$; there are $\lceil N/B_c \rceil$ blocks, leading to a total time complexity of $\mathcal{O}(NB_c r + NB_c d + Nrd)$. If we choose $B_c = \mathcal{O}(\min\{r, d\})$, the time and space complexities, including the input and output tensors, are $\mathcal{O}(Nrd)$ and $\mathcal{O}(N(r+d) + rd)$, respectively, matching those of **`causal-dot-product`**.

As discussed in Section 3.2, **`Lightning Attention-2`** (Qin et al., 2024b) separately proposed the same algorithm as in Algorithm 2 and thus enjoyed the same time and space complexity.

G.2 Recursive Method

A key structural property of the triangular mask matrix

$$\boldsymbol{M} = \begin{pmatrix} \tilde{\boldsymbol{M}} & \boldsymbol{0} \\ \boldsymbol{M}_{21} & \tilde{\boldsymbol{M}} \end{pmatrix}$$

is that its two lower triangular sub-matrices on the diagonal are identical, and the bottom left sub-matrix $\boldsymbol{M}_{21}$ is a rank-one matrix and we consider the non-decaying case where $\boldsymbol{M}_{21} = \boldsymbol{1}_{N/2}\boldsymbol{1}_{N/2}^\intercal$. Leveraging this decomposition, (Han et al., 2024) mentioned a scheme to compute their proposed causal sparse attention.

Writing $\tilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{C}^\intercal$, the product $(\tilde{\boldsymbol{A}} \odot \boldsymbol{M})\boldsymbol{V}$ can similarly be rewritten in a partitioned manner:

$$\begin{aligned}(\tilde{\boldsymbol{A}} \odot \boldsymbol{M})\boldsymbol{V} &= \left[\begin{pmatrix} \boldsymbol{B}_{(1)} \\ \boldsymbol{B}_{(2)} \end{pmatrix} (\boldsymbol{C}_{(1)}^\intercal, \boldsymbol{C}_{(2)}^\intercal) \odot \boldsymbol{M} \right] \begin{pmatrix} \boldsymbol{V}_{(1)} \\ \boldsymbol{V}_{(2)} \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{B}_{(1)}\boldsymbol{C}_{(1)}^\intercal \odot \tilde{\boldsymbol{M}} & \boldsymbol{0} \\ \boldsymbol{B}_{(2)}\boldsymbol{C}_{(1)}^\intercal & \boldsymbol{B}_{(2)}\boldsymbol{C}_{(2)}^\intercal \odot \tilde{\boldsymbol{M}} \end{pmatrix} \begin{pmatrix} \boldsymbol{V}_{(1)} \\ \boldsymbol{V}_{(2)} \end{pmatrix}.\end{aligned} \tag{5}$$

The procedure is illustrated in Figure 2. We index the two blocks via the subscripts $(1), (2)$. The block $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(1)}^\intercal$, located entirely below the diagonal, is unmasked attention. We can efficiently compute this output as $\boldsymbol{B}_{(2)}(\boldsymbol{C}_{(1)}^\intercal \boldsymbol{V}_{(1)})$, reducing it to $\mathcal{O}(Nrd)$ via matrix associativity. We can

Algorithm 3 `recursion`.

Require: Matrices $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$.
1: **if** $\text{len}(\boldsymbol{B}) \leq$ termination block size **then**
2: Return $(\boldsymbol{B}\boldsymbol{C}^\intercal \odot \boldsymbol{M})\boldsymbol{V}$
3: **else**
4: Split $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ into equal sized sub-matrices: $\boldsymbol{B}_{(1)}, \boldsymbol{B}_{(2)}; \boldsymbol{C}_{(1)}, \boldsymbol{C}_{(2)}; \boldsymbol{V}_{(1)}, \boldsymbol{V}_{(2)}$.
5: $\boldsymbol{O}_1 \leftarrow \text{Recursion}(\boldsymbol{B}_{(1)}, \boldsymbol{C}_{(1)}, \boldsymbol{V}_{(1)})$
6: $\boldsymbol{O}_2 \leftarrow \text{Recursion}(\boldsymbol{B}_{(2)}, \boldsymbol{C}_{(2)}, \boldsymbol{V}_{(2)})$
7: $\boldsymbol{O}_3 \leftarrow \boldsymbol{B}_{(2)}(\boldsymbol{C}_{(1)}^\intercal \boldsymbol{V}_{(1)})$
8: Return $\begin{bmatrix} \boldsymbol{O}_1 \\ \boldsymbol{O}_2 + \boldsymbol{O}_3 \end{bmatrix}$
9: **end if**

then recursively address the two diagonal blocks $\boldsymbol{B}_{(1)}\boldsymbol{C}_{(1)}^\intercal \odot \tilde{\boldsymbol{M}}$ and $\boldsymbol{B}_{(2)}\boldsymbol{C}_{(2)}^\intercal \odot \tilde{\boldsymbol{M}}$ with half the original size, partitioning them into even smaller blocks, and repeating such a decomposition procedure. The full `recursion` algorithm is described in Algorithm 3.

The recursive strategy differs from the iterative computation methods in Section 4.1. Despite its apparent elegance, it incurs a higher asymptotic complexity.

Complexity analysis. As delineated in Lemma 4.1, even equipped with an efficient $\mathcal{O}(N)$ causal linear attention computation method for the base case, the recursive approach is characterized by a computational complexity of $\mathcal{O}(Nrd \log N)$, exhibiting a higher complexity than the above $\mathcal{O}(Nrd)$ causal linear attention computation methods.

We provide the proof of Lemma 4.1.

Proof. As in Algorithm 3, we partition $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{V}$ evenly along the sequence dimension, where each block contains $N/2$ rows. The off-diagonal term in eq. (5) is unmasked and can be evaluated as $\boldsymbol{B}_{(2)}(\boldsymbol{C}_{(1)}^\intercal \boldsymbol{V}_{(1)})$. The intermediate matrix $\boldsymbol{C}_1^\intercal \boldsymbol{V}_1$ has size $r \times d$, and overall the non-recursive time cost at the current level is $\mathcal{O}(Nrd)$. This gives the recurrence

$$T(N) = 2T(N/2) + \mathcal{O}(Nrd).$$

By the master theorem, $T(N) = \mathcal{O}(Nrd \log N)$.

We now analyze the space complexity. The input matrices occupy $\mathcal{O}(N(r+d))$ space. The output, child outputs, cross-block results, and concatenation buffers together occupy $\mathcal{O}(Nd)$ peak space, since the simultaneously live output sizes decrease geometrically along a recursion path. For the remaining auxiliary workspace, the off-diagonal computation stores the fixed-size product $\boldsymbol{S} = \boldsymbol{C}_{(1)}^\intercal \boldsymbol{V}_{(1)} \in \mathbb{R}^{r \times d}$. This summary is formed only after both recursive child calls return, so no ancestor retains such a matrix during a child call. The two child calls execute *sequentially* and reuse the same workspace. Therefore, if $W(N)$ denotes peak auxiliary workspace excluding output-related buffers, then

$$W(N) = \max\{W(N/2), \mathcal{O}(rd)\}.$$

Thus $W(N) = \mathcal{O}(rd)$. The recursion stack uses $\mathcal{O}(\log N)$ space, which is absorbed by the input and output term. Combining the input, output-related, and remaining auxiliary space yields a peak space complexity of $\mathcal{O}(N(r+d) + rd)$. □

H Attention background and factorized formulations

For the reader's convenience, we introduce the standard attention mechanism as a preliminary in Section H.1. Exponentially decaying causal linear attention, the core of this library and benchmark, is defined in Section 2.

H.1 STANDARD ATTENTION MECHANISMS

The dot-product attention (Vaswani et al., 2017) processes three input matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$, wherein N is the number of tokens in the input sequence and d denotes the head dimension. On one hand, the bidirectional attention used in the BERT language model (Devlin et al., 2019) is defined as:

$$\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V}. \tag{6}$$

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

On the other hand, causal attention (or unidirectional attention) is used for autoregressive generative modeling, such as the decoder components of ChatGPT (OpenAI, 2022) and its successor GPT-4 (OpenAI, 2024). In handling long prompts / contexts, causal attention is bottlenecked by the prefilling stage, which computes

$$(\boldsymbol{D'})^{-1}(\boldsymbol{A}\odot\boldsymbol{M})\boldsymbol{V}, \quad \text{where } \boldsymbol{D'} = \mathrm{diag}((\boldsymbol{A}\odot\boldsymbol{M})\boldsymbol{1}_N), \tag{7}$$

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

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

The time complexity for computing both eqs. (6) and (7) is $\mathcal{O}(N^2 d)$. Their space complexity is $\mathcal{O}(N^2 + Nd)$, which is due to the explicit use of the attention matrix $\boldsymbol{A}$.

H.2 (EXPONENTIALLY DECAYING) CAUSAL LINEAR ATTENTION

Researchers have invented various efficient sequence models to mitigate the quadratic computational complexity associated with standard attention mechanisms, including TransNormerLLM (Qin et al., 2024a), RetNet (Sun et al., 2023), RWKV (Peng et al., 2023), and Mamba (Gu & Dao, 2024). In this paper, we focus on attention computations involving a low-rank form with matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N\times r}$, where we formally remove the element-wise exponential function and the row normalization in eq. (6). The resulting formulation,

$$\boldsymbol{A} = \boldsymbol{B}\boldsymbol{C}^\intercal, \tag{8}$$

is used throughout this paper. In TransNormerLLM and RetNet, for example, $\boldsymbol{B} = \boldsymbol{Q}, \boldsymbol{C} = \boldsymbol{K}$.

Exponentially decaying causal linear attention. With eq. (8) at hand, the exponentially decaying causal linear attention computation takes the form in eq. (1), where the lower triangular matrix $\boldsymbol{M}$ is now reloaded, with

$$\boldsymbol{M}_{i,j} := \begin{cases} \lambda^{i-j}, & \text{if } i \geq j, \\ 0, & \text{otherwise.} \end{cases}$$

Here, $\lambda \in (0,1]$ may differ across attention heads but is fixed across token positions within each head. In the case $\lambda = 1$, eq. (1) is simply dubbed *causal linear attention*.

In addition to the aforementioned linear attention adopted in a native linear transformer, all *low-rank approximation methods* can be similarly unified in the form of eq. (8). Low-rank approximation methods, when applied to bidirectional attention mechanisms, involve the utilization of two matrices $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{N\times r}$ of low rank to approximate the attention matrix $\boldsymbol{A}$ through $\tilde{\boldsymbol{A}} = \boldsymbol{B}\boldsymbol{C}^\intercal$, where r indicates the rank of $\tilde{\boldsymbol{A}}$ and conceptually differs from the previous (head) dimension d. Transformer-VQ (Lingle, 2024) uses vector-quantized keys and a factorization through codebook assignments to enable efficient attention computation.

In applying those low-rank techniques for either linear attention or attention approximation, we formally still need to first obtain eq. (1); however, due to the application of a mask to $\tilde{\boldsymbol{A}}$, we cannot directly utilize the low-rankness thereof and reduce the computation complexity by matrix right-multiplication as in bidirectional attention. A naïve implementation of eq. (1) still leads to a time complexity $\mathcal{O}(N^2(r+d))$ and memory complexity $\mathcal{O}(N^2 + N(r+d))$. In Section 3.1, we illustrate the intrinsic linear complexity of eq. (1) through an explicit matrix formulation, which rearranges the computation process of eq. (1) to attain linear complexity.

I Derivation of the non-iterative matrix formulation

This appendix gives the derivation underlying Section 3.1, followed by source and adaptation details for the methods in Section 3.2.

I.1 Matrix formulation

We first introduce a non-iterative matrix formulation of exponentially decaying causal linear attention, which clearly exhibits the intrinsic complexity thereof. Given the input matrices $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$, our objective is to compute the attention output $\boldsymbol{O}$ with linear compute complexity. Here, we mainly describe how to compute $\boldsymbol{O} := (\boldsymbol{B}\boldsymbol{C}^\intercal \odot \boldsymbol{M})\boldsymbol{V}$ in eq. (1). For $1 \le i \le r$, let $\boldsymbol{B}_{:,i}, \boldsymbol{C}_{:,i} \in \mathbb{R}^{N\times 1}$ denote the i-th columns of $\boldsymbol{B}, \boldsymbol{C}$, respectively. Then $\boldsymbol{B}\boldsymbol{C}^\intercal = \sum_{i=1}^{r} \boldsymbol{B}_{:,i}\boldsymbol{C}_{:,i}^\intercal$. We then have:

$$\boldsymbol{O} = \sum_{i=1}^{r} \left(\left(\boldsymbol{B}_{:,i}\boldsymbol{C}_{:,i}^\intercal\right) \odot \boldsymbol{M}\right) \boldsymbol{V} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}_{:,i})\boldsymbol{M}\mathrm{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V},$$

in which the second equation holds due to the known property of the Hadamard product. For a classic causal mask with binary elements in $\{0, 1\}$, we leverage the special structure of the mask matrix $\boldsymbol{M}$ that it is equivalent to a cumsum (cumulative sum) operator; we finally obtain

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

where the cumsum operator is applied to the column vectors of the matrix $\mathrm{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V}$ in parallel. We note in obtaining each summand the time complexity is $\mathcal{O}(Nd)$, and therefore the total computation complexity will remain $\mathcal{O}(Ndr)$.

The algorithm above can be extended to a more general variant with exponentially decaying positional weights (Press et al., 2022; Qin et al., 2022; Peng et al., 2023). In the specific attention variant with exponentially decaying positional embedding, we have a slightly different mask matrix

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

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

We first introduce the so-called discounted cumsum operator (in comparison to the regular cumsum operator above) through a toy example, in which the i-th row of the product $\boldsymbol{O} = \boldsymbol{M}\boldsymbol{V}$ (i.e., $\boldsymbol{B}, \boldsymbol{C}$ is an all-one column vector), and have

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

The matrix $\boldsymbol{M}$ now represents the discounted cumsum operator, which takes $\mathcal{O}(Nd)$ time. Denoting the discounted cumsum operator as λ-cumsum, we combine the pieces above and have

$$\begin{aligned}\boldsymbol{O} &= \left(\left(\boldsymbol{B}\boldsymbol{C}^\intercal\right) \odot \boldsymbol{M}\right) \boldsymbol{V} = \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}_{:,i})\boldsymbol{M}\mathrm{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V} \\ &= \sum_{i=1}^{r} \mathrm{diag}(\boldsymbol{B}_{:,i}) \cdot \lambda\text{-}\mathrm{cumsum}(\mathrm{diag}(\boldsymbol{C}_{:,i})\boldsymbol{V}).\end{aligned}$$

Therefore, the complexity analysis for eq. (2) still applies, and we conclude exponentially decaying causal linear attention maintains an $\mathcal{O}(Nrd)$ time complexity.

I.2 METHOD SOURCES AND ADAPTATIONS

We survey algorithms for causal linear attention from the literature across diverse fields, generalizing some methods from their original settings to causal linear attention. Table 1 presents an overview of the computation algorithms considered in this work; in particular, **`block-based`** and **`recursion`** are not designed for causal linear attention, while we realize that those algorithms can be adapted to computing eq. (1).

We categorize those methods into two groups, **iterative** methods and **recursive** methods; **`causal-dot-product`**, **`block-based`**, and **`Lightning Attention-2`** belong to the **iterative** methods, while **`recursion`** is the only **recursive** method. The iterative methods maintain a prefix summary, updated token by token or block by block. The recursive method instead divides the sequence into subproblems and combines their outputs with cross-block contributions. We give a brief introduction to each algorithm in the following list. Generalization of and improvement over existing implementations are described in Section 5.

`vanilla` refers to the direct computation of eq. (1) using PyTorch. We implemented the computation with both non-decaying causal masks that employ binary values in $\{0, 1\}$ and masks with exponentially decaying positional embedding.

`causal-dot-product`. This algorithm was first introduced by (Vyas et al., 2020) as the `CausalLinearAttention` class in their Fast Transformers library[3] (while this algorithm was not detailed and explained in their paper). This algorithm utilizes the dot product between feature maps $(\boldsymbol{B}, \boldsymbol{C})$ and leverages the triangular structure of the causal masking to compute the attention outputs row by row, achieving linear time complexity. (Choromanski et al., 2021) reinvented the computation method **`causal-dot-product`** for addressing general low-rank formats, as the algorithm description in their appendix is consistent with the code of **`causal-dot-product`** by (Vyas et al., 2020). In particular, (Choromanski et al., 2021, Equation (11)) introduced a prefix sum algorithm for causal linear attention. In each iteration, it returns one row of the output matrix, and we refer to this algorithm as the "row-based" algorithm as well.

`block-based`. In the linear transformer Transformer-VQ (Lingle, 2024), they proposed an algorithm to accelerate causal attention featuring vector-quantized keys through a block-by-block approach. The original design is uniquely tailored to vector-quantized keys and initially applied in a gated activation unit (GAU). Therefore, there is no software / package that directly supports the block-based algorithm in (Lingle, 2024) for computation of eq. (1). A careful analysis of this algorithm in Appendix G.1 reveals that **`block-based`** is indeed a block-wise version of the **`causal-dot-product`** algorithm. They divide a long sequence matrix row-wise into multiple blocks; inside a block, they simply use **`vanilla`** to compute the product $(\boldsymbol{B}\boldsymbol{C}^\intercal \odot \boldsymbol{M})\boldsymbol{V}$, while at the inter-block level, they follow **`causal-dot-product`** and apply a similar linear attention kernel trick. We realized this algorithm can be extended to a more general low-rank factorization format as in eq. (1).

`Lightning Attention-2`. Coinciding with **`block-based`**, (Qin et al., 2024b) similarly leveraged the concept of "divide and conquer" by separately handling the intra-block and the inter-block components in causal linear attention calculation. Moreover, (Qin et al., 2024b) further followed the approach of FlashAttention-1/2 (Dao et al., 2022; Dao, 2024), underscoring IO-awareness and co-design with GPU hardware. The implementation in **`Lightning Attention-2`** is thus substantially optimized; they specifically moved the computation of eq. (1) inside a block from the slow high bandwidth memory (HBM) to fast on-chip static random-access memory (SRAM). More algorithm details can be seen in Section 3.2 of (Qin et al., 2024b).

`recursion`. HyperAttention (Han et al., 2024, Algorithm (4)) proposed a recursive method to compute causal sparse attention. This approach starts with partitioning the matrices into two distinct parts: one is masked, and the other remains unmasked. Following this initial division, the method further recursively partitions the masked part while concurrently applying regular dense matrix computation to the unmasked areas. We have extended the recursive approach to the general low-rank matrix form in eq. (1). We therein similarly divide $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{V}$ recursively to enable more efficient computation of causal linear attention.

[3] See https://github.com/idiap/fast-transformers.