# ChronoCache: Adaptive KV State Management for Constant-Memory Edge LLM Inference

Kun Wang, Zimu Zhou, *Member, IEEE,* and Zhenjiang Li, *Senior Member, IEEE*

**Abstract**—Large Language Models (LLMs) are increasingly deployed on edge devices to meet the latency and privacy requirements of real-time applications, yet their memory-intensive key-value (KV) cache poses a major bottleneck for long-context inference. Linear attention mitigates this issue by maintaining a constant-memory KV state, yet suffers from accuracy degradation due to loss of fine-grained context and attention dilution. Decay-based linear attention methods partially alleviate this problem, but either indiscriminately forget important information or incur high computational overhead, and none effectively address the early memory fade-out of crucial tokens appearing early in the sequence. In this paper, we present ChronoCache, a lightweight KV state management framework for linear attention models on edge devices. ChronoCache employs a three-tier hierarchy, in which an adaptive fast-slow decay mechanism assigns varied lifespans to tokens based on importance, while a bounded, non-decaying anchor cache permanently preserves cornerstone tokens. We further incorporate asynchronous CPU execution and a warm-start initialization scheme to ensure low latency and cold-start robustness. Experiments on multiple long-context benchmarks show that ChronoCache consistently outperforms state-of-the-art linear attention models, achieving a superior balance between constant-memory efficiency and long-context inference accuracy.

## 1 Introduction

Large Language Models (LLMs), such as ChatGPT and DeepSeek, have reshaped human-computer interaction through advanced natural language understanding and generation. Although cloud-based APIs offer broad accessibility, deploying LLMs directly on edge devices is increasingly attractive to overcome limitations such as unpredictable latency, network congestion, and data privacy risks [1], [2], [3]. In particular, latency-sensitive applications like healthcare diagnosis and autonomous navigation require real-time inference and strict data confidentiality, which cloud-centric deployments struggle to guarantee. Consequently, edge deployment, empowered by recent lightweight LLM architectures (*e.g.*, Qwen-2.5-3B [4], Phi-4 [5]), emerges as a promising paradigm to provide responsive, privacy-preserving, and self-contained AI services. Indeed, as AIoT applications proliferate [6], [7], [8], [9], [10], [11], bringing LLMs into this ecosystem is a natural next step.

An alternative paradigm is network-assisted inference, where computation is offloaded to cloud, fog, or edge servers [12], [13]. While such approaches can leverage more powerful remote hardware, they introduce network latency, depend on connectivity availability, and raise data privacy concerns when raw inputs leave the device. In this work, we focus on fully local, on-device inference that eliminates network dependency, targeting scenarios with strict privacy requirements, intermittent connectivity, or ultra-low-latency demands. We note that local and network-assisted inference are complementary: a mobile system can seamlessly switch between them based on connectivity and privacy constraints.

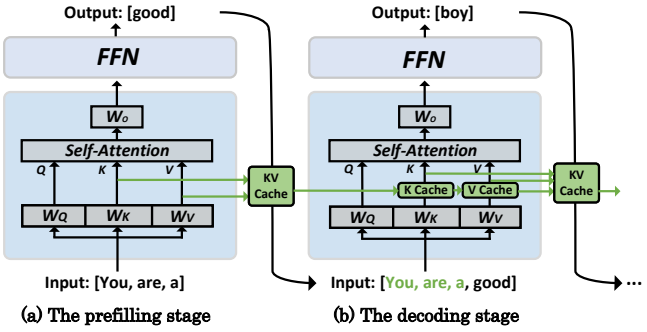



Fig. 1: KV Cache in LLMs token generation.

However, deploying LLMs on edge devices faces a critical memory bottleneck stemming from the *Key-Value (KV) Cache* [14], [15], which stores intermediate attention states to avoid redundant computations during autoregressive token generation (see Figure 1). As the KV cache size grows linearly with input sequence length, it rapidly exhausts the limited memory available on edge devices. For instance, a 3B-parameter model with a 15K-token context window generates a KV cache can reach up 5-6GB, surpassing the model's own size and overwhelming typical edge-device memory, as shown in Figure 2. As applications increasingly rely on long-context inference tasks such as multi-step reasoning (Chain-of-Thought) and retrieval-augmented generation (RAG), this memory issue severely degrades service quality or even leads to inference failures.

Recent studies propose *linear attention* mechanisms [16], [17] to alleviate this memory bottleneck by reformulating traditional softmax-based attention computations. By rearranging matrix multiplications, linear attention reduces


- *K. Wang and Z. Li are with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. E-mail: kwang69-c@my.cityu.edu.hk, zhenjiang.li@cityu.edu.hk.*
- *Z. Zhou is with the Department of Data Science, City University of Hong Kong, Hong Kong, China. E-mail: zimuzhou@cityu.edu.hk.*

computational complexity from quadratic to linear and maintains a *constant memory KV state*, irrespective of sequence length. Despite these efficiency advantages, linear attention inherently compresses historical context into a fixed-size KV state, causing progressive *loss of details* and *attention dilution*, thus significantly degrading accuracy compared to standard softmax attention, particularly over long-context tasks [18].

To mitigate accuracy loss, a promising solution is to introduce a KV state *decay mechanism*, gradually forgetting outdated information to prioritize more recent states within the fixed memory [19], [20], [21]. Existing decay methods employ either a constant [19], [21] or an input-dependent [20] exponential decay factor. Constant decay is computationally efficient but indiscriminately discards information due to their uniform forgetting rates across all tokens. Input-dependent decay adapts decay factors based on token importance, providing fine-grained retention at the expense of significantly higher computational overhead due to additional trainable gating mechanisms. Thus, existing decay strategies cannot effectively balance computational complexity with accuracy, especially for long-context inference on edge devices.

Addressing these limitations demands overcoming two key challenges. First, it is non-trivial to design a sufficiently fine-grained yet computationally lightweight adaptive decay mechanism to balance computational efficiency and inference accuracy. Second, even adaptive decay alone does not fully prevent the eventual forgetting of crucial context, especially foundational information introduced early in the input sequence. This phenomenon, we termed as *early memory fade-out*, severely undermines tasks like multi-round dialogues and long-document reasoning, where cornerstone instructions must persist throughout inference.

To tackle these challenges, we propose ChronoCache, an adaptive, lightweight KV state management framework tailored for constant-memory inference on resource-constrained edge devices. Our insight is to combine input-independent exponential decay, recommended by industrial practices, with an importance-aware, hierarchical KV state organization. Specifically, we introduce a three-tier KV state hierarchy comprising: (1) a *fast-decay channel* to rapidly discard low-importance tokens, (2) a *slow-decay channel* to preserve important context, and (3) a non-decaying *anchor cache* to permanently retain cornerstone tokens. Inspired by cognitive neuroscience principles that differentiate short-term from long-term memory [22], [23], this hierarchical structure allows selective and adaptive retention of information based on token relevance, effectively and efficiently addressing indiscriminate forgetting and early memory fade-out.

ChronoCache realizes this hierarchical KV state management through three components. First, the *Adaptive Fast-Slow Decay* mechanism dynamically assigns each token's KV state into either the fast- or slow-decay channel based on token importance, estimated via attention scores within a sliding observation window. Second, the *Dynamic Anchor Cache* maintains a bounded, non-decaying storage for cornerstone tokens. Unlike decay-based KV states, the anchor cache independently preserves selected key-value vectors without compression, explicitly preventing early memory fade-out. We also continually update anchor cache entries, allowing newly identified critical tokens to replace outdated ones. Third, we propose practical *Runtime Optimization and Initialization* strategies that offload computational overhead of ChronoCache to idle CPU cores, running concurrently with GPU computations. We also introduce a warm-start initialization approach that leverages the initial prompt processing stage to effectively mitigate cold-start degradation. Our main contributions are summarized as follows.

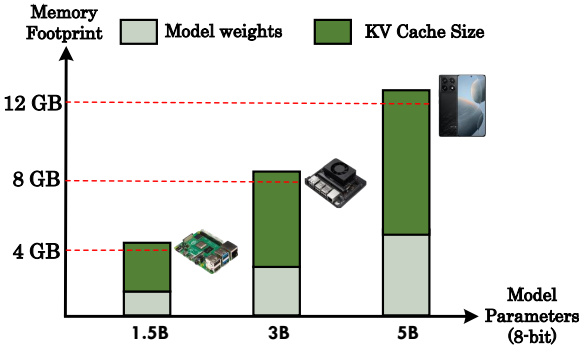



Fig. 2: In common LLM configurations, when context length reach to 15K tokens, the KV Cache size is larger than the model weights size, and total memory footprint beyond RAM of common mobile and edge devices.

- We propose ChronoCache, a novel KV state management framework for linear attention models on edge devices. It employs a three-tier hierarchy (fast-decay, slow-decay, and non-decay anchor cache) to adaptively retain contextual information based on token importance, substantially improving the long-context inference accuracy of constant-memory LLMs.
- ChronoCache is both effective and lightweight. The three-tier design preserves important KV states for long-context reasoning, while our lightweight update mechanisms dynamically manage the constant-size cache. We further optimize ChronoCache for edge deployment through asynchronous execution on idle CPU cores and a warm-start initialization scheme leveraging the prompt prefilling stage, achieving low latency and cold-start robustness without sacrificing throughput.
- Extensive experiments demonstrate that ChronoCache consistently outperforms state-of-the-art input-independent linear attention models [19], [21] on multiple long-context benchmarks [24], offering a superior balance between inference efficiency and accuracy.

## 2 Background

This section provides a review on KV cache (§ 2.1) and linear attention (§ 2.3) to motivate our solution.

### 2.1 Attention and KV Cache

Transformer-based LLMs are built upon the attention mechanism [25], where the inference of each new token is generated by attending to *all previous* tokens. Consider the embedding $X \in \mathbb{R}^{t \times d}$ of an input sequence of $t$ tokens, where $d$ is the dimensionality of each token's embedding

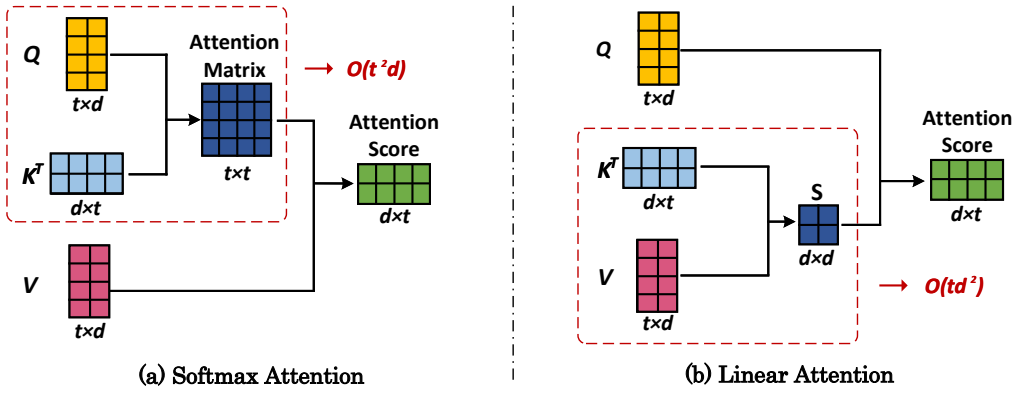



Fig. 3: Different computation order of (a) softmax attention and (b) linear attention.

and also the hidden dimension of the attention layer. The attention works as follows.

$$Q = XW^Q,\ K = XW^K,\ V = XW^V \in \mathbb{R}^{t\times d}$$
$$A = \texttt{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \in \mathbb{R}^{t\times d} \tag{1}$$

$\texttt{softmax}$ is the softmax function; $Q$, $K$, and $V$ are the query, key, and value matrices generated from the embedding of the $t$ input tokens. Assuming naive matrix multiplication, the calculation of $Q$, $K$, and $V$ requires $O(t)$ operations. The computation of $A$ needs $O(t^2)$ operations. Hence, the total computational cost of vanilla attention is $O(t^2)$, which grows *quadratically* with the input sequence length $t$.

The standard attention in Equation (1) reveals duplicated computation during the inference of tokens. For instance, the query matrix $Q \in \mathbb{R}^{t\times d}$ used to infer the $(t+1)$-th token is the same as the first $t$ rows of $Q$ used to infer the $(t+2)$-th token. This observation motivates the KV cache (key-value cache, see green part of Figure 1) to accelerate Transformer-based LLMs at the cost of extra memory footprint [26].

Formally, given the embedding of the $(t-1)$-th token $x \in \mathbb{R}^{1\times d}$, KV cache-enabled inference of the $t$-th token operates as follows.

$$Q_t = xW^Q,\ K_t = xW^K,\ V_t = xW^V \in \mathbb{R}^{1\times d}$$
$$K = \texttt{cat}(K^S, K_t),\ V = \texttt{cat}(V^S, V_t) \tag{2}$$
$$A_t = \texttt{softmax}\left(\frac{Q_tK^\top}{\sqrt{d}}\right)V \tag{3}$$

where $K^S$ and $V^S$ cache of key and value matrices of historical tokens, and $\texttt{cat}$ is the concatenation function. From Equation (2), the size of the KV cache grows *linearly* with the length of the generated tokens, *i.e.*, $O(t)$.

**Takeaways.** KV cache avoids duplicated computations in LLM inference, yet its size grows linearly with the sequence length. As shown in Figure 2, the memory footprint of KV cache can be even larger than the model size in long-context scenarios, making KV cache an efficiency bottleneck for LLM inference on resource-constrained edge devices.

## 2.2 Existing Solutions

Several lines of work have been proposed to address the KV cache bottleneck in standard Transformers. Eviction-based methods such as StreamingLLM [27], SnapKV [28], and H2O [29] selectively retain a subset of important KV pairs, while quantization approaches like KIVI [30] compress the cache to lower bit-widths. Architectural modifications including Multi-Query Attention [31] and Grouped-Query Attention [32] reduce the cache size by sharing projections across attention heads. While these techniques effectively reduce per-token storage costs, they do not eliminate the fundamental $O(L)$ memory growth with sequence length.

An alternative direction is sparse attention methods such as BigBird [33] and Sparse Sinkhorn Attention [34], which reduce computational complexity to $O(L)$ by sparsifying the attention matrix. However, these methods still require materializing the full KV cache, resulting in $O(L)$ memory growth that remains prohibitive for long-context inference on edge devices. A more detailed discussion of these and other related approaches is provided in § 9.

## 2.3 Linear Attention for Constant-Memory Inference

### 2.3.1 Linear Attention and KV State

Linear attention [16], [17] decreases the computation of vanilla attention from $O(t^2)$ to $O(t)$, and the size of KV cache from *linear* growth to *constant*. It replaces the softmax function in the vanilla attention via a kernel function $\phi(\cdot)$ as follows:

$$A = \texttt{softmax}\left(\frac{Q_tK^\top}{\sqrt{d}}\right) \approx \phi(Q)\phi(K^\top)V \tag{4}$$

which allows computing $\phi(K^\top)V$ first, thus reducing the time complexity to calculate $A$ to $O(t)$, as shown in Figure 3.

More importantly, such reorder of matrix multiplication results in constant-size KV cache during token inference. Specifically, given the embedding of the $(t-1)$-th token $x \in \mathbb{R}^{1\times d}$, inference of the $t$-th token with KV cache-enabled linear attention becomes:

$$Q_t = xW^Q,\ K_t = xW^K,\ V_t = xW^V \in \mathbb{R}^{1\times d}$$
$$S_t = S_{t-1} + \phi(K_t^\top)V_t \in \mathbb{R}^{d\times d} \tag{5}$$
$$A_t = Q_tS_t \tag{6}$$

where $S_t$ is the accumulate KV state $\phi(K^\top)V$ to infer token $t$. Since $S_t \in \mathbb{R}^{d\times d}$, this token generation process maintains a constant memory footprint regardless of the input sequence length $t$, as shown in Figure 4.

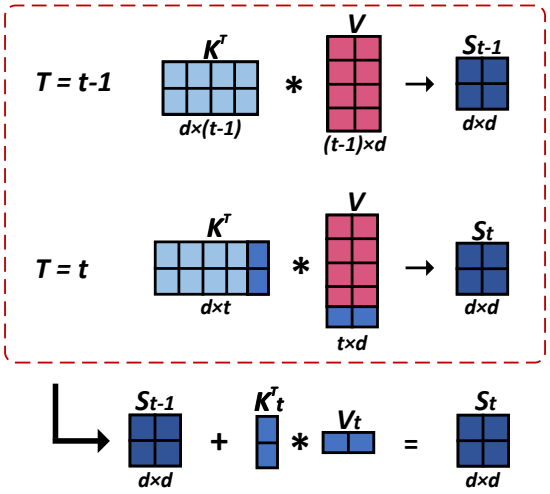



Fig. 4: Accumulative KV state updating mechanism with constant-memory.

**Drawback.** The memory efficiency of linear attention comes at the cost of degraded accuracy [18]. This is because in vanilla attention, the scope is incrementally expanded per generation step, allowing the model to explicitly maintain individual relationships between the current query and every historical token. In contrast, linear attention *compresses* and *aggregates* all historical information into a fixed-size state space, leading to two problems, as shown in Figure 5.

- *Loss of details.* The query cannot distinguish the strength of its association with specific historical tokens.
- *Attention dilution.* The model cannot selectively forget unimportant information or reinforce critical one.

Both problems impair the model accuracy, particularly in long-context tasks.

#### 2.3.2 KV State Decay

A promising strategy to address the above problems in linear attention is to *decay* the KV state [19], [20]. Specifically, the *uniform weight* in Equation (5) is replaced with an *exponential decay* below:

$$S_t = \gamma S_{t-1} + \phi(K_t^\top) V_t \tag{7}$$

where $\gamma$ is a decay factor. The decay mechanism proactively *forgets* ancient information, allowing the model to focus on more recent and relevant information in the fixed-size KV state, thereby improving the accuracy of standard linear attention. The decay can be either constant [19], [21] or input-dependent [20], [35], where the latter offers more flexible information retention, yet at increased computational overhead *e.g.*, by introducing extra trainable gates in LLMs [36]. Figure 6 shows that the computation speed of input-independent methods are usually faster around 40% to 50% than the input-dependent methods, which are more suitable in mobile and edge devices.

**Our Objective.** Inspired by the accuracy gains of KV state decay, we aim to develop an *adaptive* and *lightweight* KV state management mechanism dedicated to *long-context* inference on edge devices, while preserving the constant-memory footprint of linear attention.

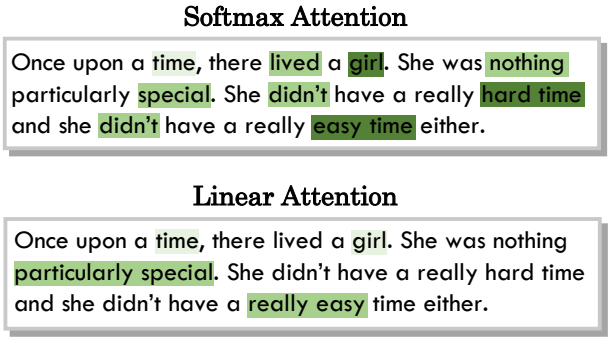



Fig. 5: Examples of attention dilution and loss of details in linear attention.

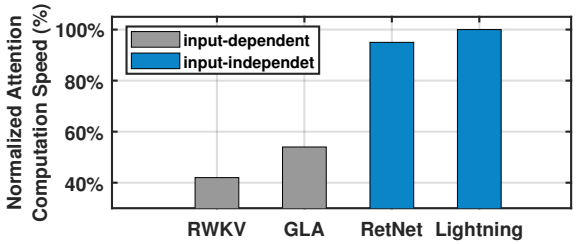



Fig. 6: Normalized attention computation speed between the input-dependent and input-independent methods.

## 3 System Overview

This section presents the design philosophy and architectural components of ChronoCache, our proposed adaptive KV cache decay mechanism for constant-memory LLM inference on edge devices.

### 3.1 Design Philosophy

ChronoCache is designed to improve the accuracy of constant-memory LLM inference by tailoring the accuracy-computation tradeoff in KV state decay. It follows an *input-independent* decay policy for computational efficiency, as recommended by industrial deployments such as MicroSoft [19] and MiniMax [37]. Yet it enhances this approach with an *adaptive*, importance-aware KV decay mechanism to boost accuracy in long-context tasks.

Specifically, ChronoCache partitions the KV states into a *three-tier* hierarchy: (1) a *fast-decay* channel for short-term, low-importance tokens, (2) a *slow-decay* channel for medium-term, more essential tokens, and (3) a *non-decay* channel for cornerstone tokens that must persist across long contexts. This design is motivated by two observations.

- The separation between short-term and long-term memory is a well-established abstraction in neuroscience and cognitive science [22], [23], and has proven effective in model architectures such as LSTM [38].
- The non-decay channel is critical for preventing *early memory fade-out*, a common failure mode in previous decay-based linear attention. In real-world LLM usage, users often provide key instructions at the start of a session. Forgetting these tokens can lead to contradictions or loss of context in generation.

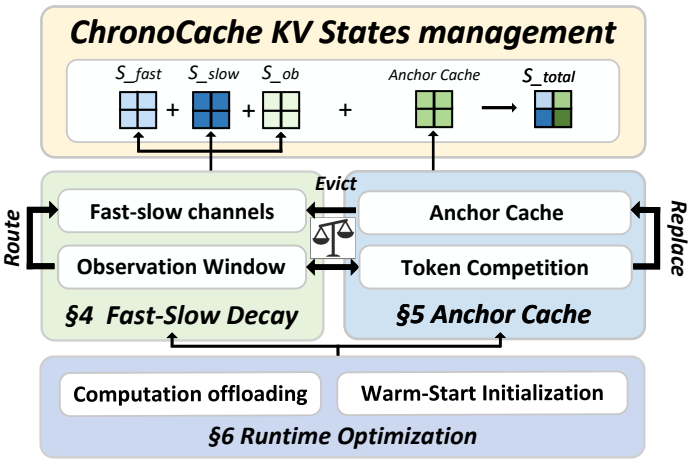



Fig. 7: ChronoCache overview.

Our evaluations show that this tri-tier hierarchy improves accuracy in long-context tasks while avoiding the computational complexity of fully input-dependent decay.

### 3.2 ChronoCache Components

We realize the above philosophy through three key components, illustrated in Figure 7.

- **Adaptive Fast-Slow Decay (§ 4).** We introduce a dual-channel decay mechanism that assigns each new token's KV state to either a fast- or slow-decay channel. This emulates short- and long-term memory in human cognition. The assignment is based on the average attention the token receives within a sliding observation window, which serves as an efficient proxy for its contextual importance over a local generation span (*e.g.*, a sentence or paragraph).
- **Dynamic Anchor Cache (§ 5).** To avoid early memory fade-out, we introduce an anchor cache, *i.e.*, a fixed-size, non-decaying KV store for essential information. In addition, new important tokens will dynamically compete and replace old less important tokens in the anchor cache, enabling continual adaptation to new global instructions or overriding contextual cues.
- **Runtime Optimization and Initialization (§ 6).** For efficient inference, we offload ChronoCache to a background CPU thread, overlapping its latency with the GPU's FFN computation. We also propose an initialization scheme for ChronoCache that leverages the prefilling stage to avoid cold-start degradation.

Next, we elaborate on each component in detail.

## 4 Adaptive Fast-Slow Decay Mechanism

This section presents our adaptive fast-slow decay mechanism for managing the KV state. We begin by analyzing the attention patterns of standard and linear attention (§ 4.1), which motivates a lightweight dual-channel decay design that approximates the behavior of standard attention (§ 4.2). To determine the decay channel for each token, we introduce an adaptive assignment strategy based on token importance, estimated within an observation window (§ 4.3).

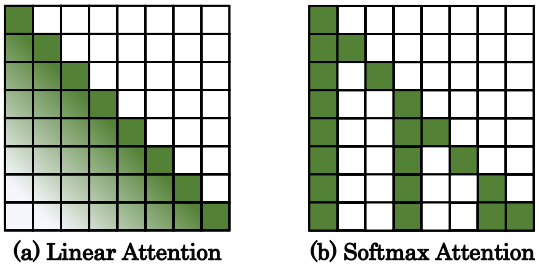



Fig. 8: Illustration of attention patterns.

### 4.1 Attention Pattern Analysis

As discussed in § 2.3, linear attention suffers from detail loss and attention dilution due to its fixed-size KV state storage. These limitations can be partially addressed by decay mechanisms that gradually forget older or less relevant KV states to make room for newer ones. However, there is a fundamental tradeoff between the *complexity* and *effectiveness* of such decay strategies. This raises a key question: *What are the minimal requirements for a KV decay mechanism to effectively approximate the behavior of standard attention?*

To address this question, we compare the attention patterns of linear attention with uniform, input-independent decay (*i.e.*, the simplest form of decay), with those of standard (softmax-based) attention. Here, an attention pattern refers to the distribution of attention scores from a newly generated token toward previously generated tokens. As illustrated in Figure 8(a), linear attention with uniform decay exhibits rapidly diminishing attention to earlier tokens as the sequence length grows. This behavior stems from exponential forgetting based solely on token age, irrespective of token importance. In contrast, standard attention does not decay based on time, as shown in Figure 8(b). Instead, it allocates attention dynamically based on content relevance. A small subset of early (*e.g.*, the first) and intermediate tokens consistently receive strong attention even after many subsequent tokens have been generated.

From these observations, we hypothesize that a minimal yet effective approximation of standard attention requires at least *two* decay channels: one for preserving important tokens more slowly (or not at all), and another for aggressively decaying less important ones. Within each channel, decay can still proceed based on token age, as in uniform decay, but the separation of tokens by importance is critical. As a remark, the use of decay is unavoidable due to the fixed-size constraint of the KV cache. without decaying, new tokens would eventually exhaust memory, making long-context inference infeasible.

### 4.2 Fast-Slow Decay Channels

Building on the analysis in § 4.1, we propose a dual-channel decay scheme that partitions the KV state into two logical channels with distinct decay rates, as illustrated in the upper part of Figure 9. The decay rates for each channel are configured based on typical sequence lengths in common LLM application scenarios, such as document summarization conversational agents, and instruction-following tasks.

- **Slow-decay channel.** This channel is associated with a low decay rate $\gamma_{low}$ to retain important information

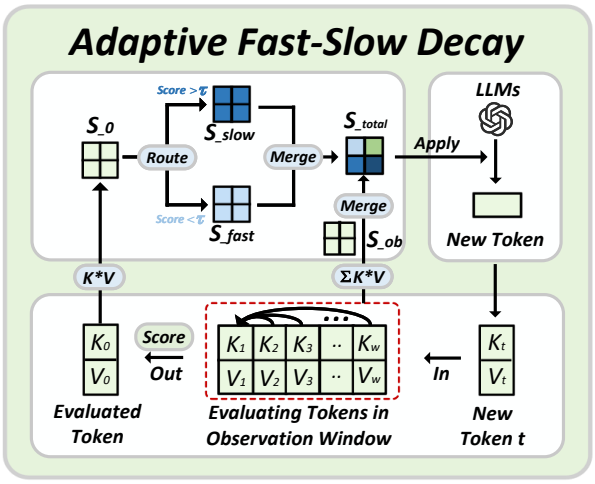



Fig. 9: Adaptive fast-slow decay mechanism.

for long-term memory. We set $\gamma_{low}$ within the range $[0.999, 0.9999]$. For example, with $\gamma_{low} = 0.9999$, the signal strength of a token remains at $0.9999^{10000} \approx 36.8\%$ after 10K generation steps, which is sufficient for preserving dependencies across long contexts.

- **Fast-decay channel.** This channel uses a high decay rate $\gamma_{high}$ to rapidly attenuate unimportant information within a local semantic unit (*e.g.*, a paragraph). We set $\gamma_{high}$ within the range $[0.9, 0.95]$. For instance, with $\gamma_{high} = 0.9$, the retained signal drops to $0.9^{64} \approx 0.13\%$ after 64 steps (a typical paragraph length), effectively making the contribution negligible and freeing up storage for new tokens.

These decay rates are not arbitrary but are grounded in the structural characteristics of natural language text. We define the *effective lifetime* $L_{\text{eff}}$ as the number of steps after which a token's contribution decays to 1% of its original magnitude:

$$L_{\text{eff}} = \frac{\log(0.01)}{\log(\gamma)} = \frac{-4.605}{\log(\gamma)}. \tag{8}$$

For the fast-decay channel ($\gamma_{high} \in [0.9, 0.95]$), $L_{\text{eff}} \approx 44$–$90$ steps, matching the typical length of a paragraph or utterance. For the slow-decay channel ($\gamma_{low} \in [0.999, 0.9999]$), $L_{\text{eff}} \approx$ 4,600–46,000 steps, spanning the range of typical document lengths in LLM applications. Thus, the two channels are calibrated against two complementary timescales inherent in natural language structure. A sensitivity analysis in § 8.4.6 confirms that performance is robust across the recommended ranges.

### 4.3 Token Assessment and Assignment

A key to our dual-decay mechanism is determining the *importance* of each token, to assign its KV state to either the slow-decay or fast-decay channel. Since it is impractical to assess a token's importance across the entire future sequence during generation, we approximate importance using the token's average attention received within a fixed-size sliding window, referred to as the *observation window*. This approximation is motivated by an empirical observation. If a token receives high attention from newly generated tokens within a local observation window, it is likely to remain important in the longer-term context as well, as illustrated in Figure 8(b).

This local-to-global predictability is grounded in the well-documented *attention sparsity* phenomenon: prior studies [29], [28], [27] have consistently shown that a small fraction of tokens ("heavy hitters") receive disproportionately high attention across different query positions. Because these tokens' importance arises from their semantic role (*e.g.*, task instructions, key entities) rather than local context, their high attention persists beyond any finite window. More formally, let $\alpha_t^{(i)}$ denote the attention score that token $t$ receives from query position $i$. The local importance within a window of size $W$ is $\hat{I}_t^{(\text{local})} = \frac{1}{W-1}\sum_{i=t+1}^{t+W-1} \alpha_t^{(i)}$, while the true global importance is $I_t^{(\text{global})} = \frac{1}{T-t}\sum_{i=t+1}^{T} \alpha_t^{(i)}$. Since heavy-hitter tokens exhibit stationary importance profiles with bounded variance, $\hat{I}_t^{(\text{local})}$ converges to $\mathbb{E}[\alpha_t^{(i)}]$ as $W$ increases, making $W = 64$–$128$ sufficient for reliable estimation. Our correlation analysis in § 8.4.4 confirms that local importance scores ($W = 128$) achieve a Spearman rank correlation of $\rho \geq 0.82$ with global importance across sequence lengths up to 16K. Moreover, the dynamic anchor cache (§ 5) provides a self-correcting mechanism: tokens initially underestimated by the local window are promoted if they receive high attention in subsequent windows. The bottom part of Figure 9 depicts this assessment process.

Formally, for a token generated at time step $t$, we initiate an observation window $W_O$ of size $W$, implemented as a FIFO queue. During the next $W - 1$ decoding steps, the token's Key ($K_t$) and Value ($V_t$) vectors are temporarily held outside the decayed KV state. They are preserved in their original form to track how much attention the token receives from subsequent queries. We find that setting $W$ between 64 and 128 captures most of the performance benefits to retain essential long-term context while keeping the lightweight through evaluation results in § 8.4.1.

To quantify token importance, we leverage the core computation of attention: the dot product between a query and a key. Specifically, the importance score $Score_t$ of token $t$ is defined as the average of the raw attention scores it receives from future queries in the observation window $Score_t = \frac{1}{W-1}\sum_{j=t+1}^{t+W-1} Q_j \cdot K_t$.

Once the observation window ends, we compare $Score_t$ against an importance threshold $\tau$, which is set based on empirical analysis. Specifically, we compute the average attention score of the top 10% most attended tokens in a standard Transformer model evaluated over large-scale text corpora. This threshold choice is motivated by prior studies showing that a small subset of tokens, often the top 10%, accounts for the majority of meaningful contributions to subsequent generation. We adopt this empirical mean as the value of $\tau$ to ensure that our importance filtering aligns with well-established attention sparsity patterns. A sensitivity analysis in § 8.4.5 demonstrates that ChronoCache is robust across thresholds from top-5% to top-20%, and a cross-domain validation in § 8.4.8 confirms that the threshold generalizes well across different text corpora.

If $Score_t \geq \tau$, the token is considered important and its KV state $\phi(K_t^\top)V_t$ is assigned to the slow-decay channel. Otherwise, it is assigned to the fast-decay channel.

## 5 Dynamic Anchor Cache

This section presents the dynamic anchor cache, which complements the fast-slow decay mechanism in § 4 by offering a non-decaying storage for the KV vectors of cornerstone tokens. We first define the concept of the anchor cache (§ 5.1), and then describe a dynamic competition mechanism that comparing and replacing tokens in anchor cache based on their relevance (§ 5.2). Finally, we integrate the dynamic anchor cache with the dual decay mechanism (§ 5.3).

### 5.1 Principle of Anchor Cache

As discussed in § 3.1, some tokens, *e.g.*, global instructions or task definitions provided at the beginning of a session, play a central role in long-context LLM inference. These tokens must be preserved indefinitely to maintain coherence and correctness, and thus should not be decayed.

However, such non-decaying KV storage cannot be implemented as a third decay channel with an extremely slow decay rate. Even the slowest decay still requires a *non-zero* decay factor to ensure that fixed-size memory can accommodate new tokens. In contrast, a non-decaying storage must grow unless explicitly bounded, making it fundamentally different from the two decay-based channels in § 4.

To address this, we introduce the *anchor cache*, denoted by $M_{anchor}$, which consists of permanent memory slots that store the Key and Value vectors of cornerstone tokens in their original form. This design is similar to the conventional KV cache used in standard attention, as shown in the upper-right part of Figure 10. Importantly, unlike linear attention's *compressed* and *decaying* KV states, the anchor cache stores $K$ and $V$ vectors *independently* and *without decay*.

Since tokens in $M_{anchor}$ are never evicted based on age, its size would grow unbounded over time, even if only a small number of tokens qualify as cornerstone tokens. To maintain a constant memory footprint compatible with linear attention's goals, we impose a strict upper bound on the anchor cache size, denoted by $M$. We find that setting $M$ between 128 and 256 captures most of the performance benefits to retain essential long-term context while keeping the anchor cache lightweight through evaluation results in § 8.4.2.

**Graceful Handling of High-Density Important Tokens.** In scenarios where the number of important tokens exceeds $M$ (*e.g.*, 500 important tokens with $M = 128$), the three-tier hierarchy provides graceful degradation rather than overflow: the top-$M$ tokens occupy the anchor cache with no decay, while the remaining important tokens are routed to the slow-decay channel, where their contribution persists for thousands of steps (*e.g.*, with $\gamma_{low} = 0.9999$, signal strength remains at 45% after 8K steps). Thus, no information is abruptly lost; it is retained at progressively lower persistence levels, and the dynamic competition mechanism continuously rotates the most currently relevant tokens into the non-decaying anchor cache.

### 5.2 Anchor Cache Update

Since the anchor cache $M_{anchor}$ is non-decaying and fixed in size, its entries must be continuously updated to reflect the most critical tokens throughout generation. While initial cornerstone tokens often appear at the beginning of a session (*e.g.*, task instructions), new important concepts may emerge later, especially in long sequences like multi-round dialogues. We adopt a lightweight, importance-driven update policy that integrates seamlessly with the observation window described in § 4.3.

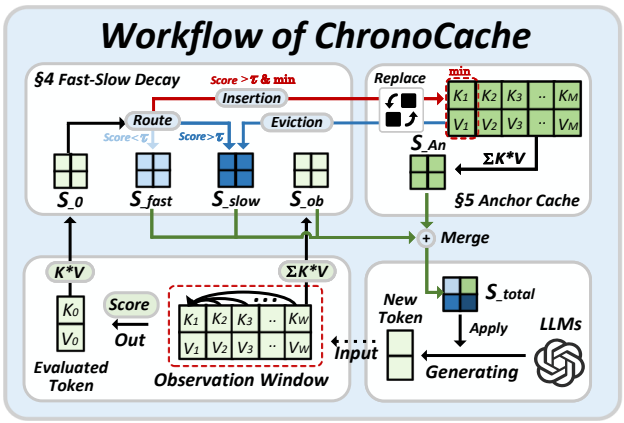



Fig. 10: Complete ChronoCache workflow with anchor cache.

#### 5.2.1 Tracking Token Importance in Anchor Cache

Each entry $i$ in $M_{anchor}$ maintains a running importance score $Score_{anchor(i)}$, which reflects how often it is attended by future tokens. At each decoding step $t$, after generating the query vector $Q_t$, we compute its dot product with every key vector $K_i$ in $M_{anchor}$ and update the corresponding $Score_{anchor(i)}$ via a running average. This ensures that the cache reflects not just initial token importance, but sustained relevance over time.

#### 5.2.2 Dynamic Token Competition in Anchor Cache

When a token $t - W$ reaches the end of its observation window, its importance score $Score_{t-W}$ is finalized as in § 4.3. We then evaluate whether this token should be inserted into $M_{anchor}$ as follows.

- **1) Eligibility Check.** If $Score_{t-W} < \tau$, the token is considered unimportant and is directly assigned to the fast-decay channel. If $Score_{t-W} \geq \tau$, it becomes a candidate for the anchor cache.
- **2) Token Competition and Replacement.** Suppose $i^* = \arg\min_i Score_{anchor}(i)$ is the least important token in $M_{anchor}$. If $Score_{t-W} > Score_{anchor}(i^*)$, we evict entry $i^*$ and insert token $t - W$ to replace it. Otherwise, the token $t - W$ is assigned to the slow-decay channel.
- **3) Score Initialization.** Upon replacement, the new inserted anchor token inherits an initial importance score of $Score_{t-W}$. In addition, the evicted anchor token is not discarded. Instead, its KV state is assigned to the slow-decay channel to ensure a graceful degradation of its contribution.

This dynamic competition policy ensures that $M_{anchor}$ continually adapts to retain tokens with the highest long-term importance, without exceeding memory constraints.

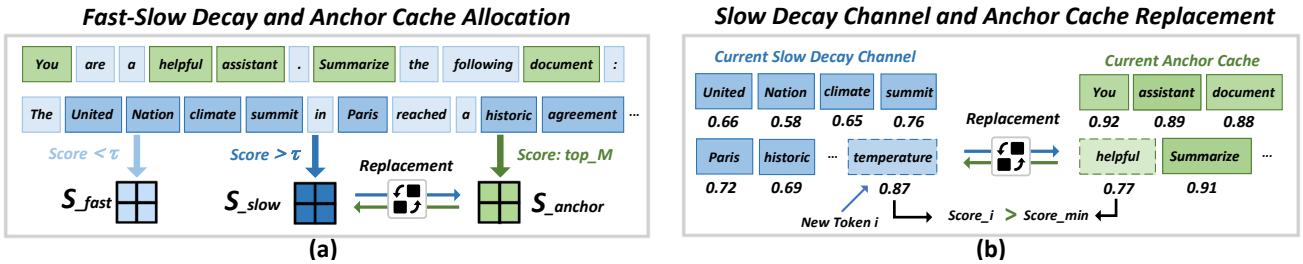



Fig. 11: Case study of (a) Token-level allocation, colored by tier (green = anchor, deepblue = slow-decay, lightblue = fast-decay). (b) Dynamic anchor cache replacement where "`temperature`" (0.87) replace "`helpful`" (0.77).

**Handling Borderline and Temporal Effects.** When multiple tokens have similar scores near the threshold $\tau$, the precise routing decision has minimal impact: borderline tokens are either retained in the slow-decay channel (persisting for thousands of steps) or the fast-decay channel. In either case, the anchor cache's dynamic competition mechanism self-corrects within a few observation windows. Genuinely important tokens that were narrowly missed will accumulate high attention and be promoted. The observation window also naturally encodes temporal locality: at each step, only the most recent $W$ tokens are under evaluation, ensuring that the system continuously reassesses importance based on up-to-date attention patterns. Additionally, the running-average importance score of anchor cache entries implicitly favors recently attended tokens.

### 5.3 Integration with Fast-Slow Decay

Figure 10 illustrates the complete KV state management workflow that integrates the anchor cache with the fast-slow decay mechanism for linear attention computation. Figure 11 further provides a concrete token-level case study on a representative instruction-following prompt, showing how individual tokens are routed to the anchor cache, slow-decay channel, or fast-decay channel, and how the dynamic competition mechanism replaces a low-scored anchor entry with a newly important token.

To verify that this case study reflects the actual runtime behavior, we instrumented an inference run (16K context, 1.3B model) and logged each token's routed tier and every anchor-cache replacement (Table 1). The measured statistics match the design intuition: only $0.9\%$ of tokens enter the non-decaying anchor cache while $89.8\%$ fall to the fast-decay channel, and the anchor cache stabilizes quickly, with $87\%$ of its 23 replacements within the first $40\%$ of decoding steps. Tokens from the initial instruction span are evicted far less than transient mid-sequence tokens ($6\%$ vs. $71\%$), confirming that foundational tokens are preferentially retained, consistent with Figure 11(b).

At each time step $t$, we have a query vector $Q_t$. As shown by the green arrows in Figure 10, the KV state $S_{total}$ is the sum of four components: anchor cache $M_{\text{anchor}}$ containing non-decaying KV pairs, fast-decay state $S_{fast,t}$, slow-decay state $S_{slow,t}$, and the observation window state $S_{ob,t}$ that holds unmerged KV pairs. To aid readability, we recap the notation: $S_{fast}$ is the fast-decay state for transient information, $S_{slow}$ is the slow-decay state for persistent information, $S_{ob}$ is the observation window state for recently arrived tokens pending evaluation, and $M_{anchor}$ is the anchor cache storing the most critical KV pairs in their original, non-decaying form.

TABLE 1: Measured token allocation and anchor replacement statistics from an actual inference run (16K context, 1.3B model), compared against the design intuition of Figure 11.

| Statistic | Intuition | Measured |
|---|---|---|
| Tokens above threshold $\tau$ | ∼top-10% | 10.2% |
| – ever admitted to anchor cache | small, bounded by $M$ | 0.9% |
| – routed to slow-decay channel | moderate | 9.3% |
| Fast-decay share | majority | 89.8% |
| Instruction-token eviction rate† | low | 6% |
| Transient-token eviction rate† | high | 71% |

†Among tokens ever admitted to the anchor cache from each span.

The anchor cache and observation window components are computed as follows:

$$M_{anchor} = \sum_{i=1}^{M} \phi(K_i^\top) V_i \tag{9}$$

$$S_{ob,t} = \sum_{i=t-W}^{t} \phi(K_i^\top) V_i \tag{10}$$

The total KV state used for attention at time $t$ is thus:

$$S_{total,t} = M_{anchor} + S_{slow,t} + S_{fast,t} + S_{ob,t} \tag{11}$$

Finally, the linear attention output is computed as:

$$Output_t = \phi(Q_t) \left( S_{total,t} + \phi(K_t^\top) V_t \right) \tag{12}$$

This integration ensures that the model benefits from both long-term memory (via the anchor cache and slow-decay channel) and efficient context forgetting (via the fast-decay channel), all within a constant memory budget.

## 6 Runtime Optimization and Initialization

This section introduces two system-level techniques to enhance ChronoCache. The first improves runtime efficiency by overlapping the execution of ChronoCache with feed-forward network (FFN) operations, hiding the extra latency introduced by ChronoCache (§ 6.1). The second improves inference effectiveness by warm-starting ChronoCache with a hierarchy-aware initialization during the prefilling stage, which mitigates the cold-start degradation (§ 6.2).

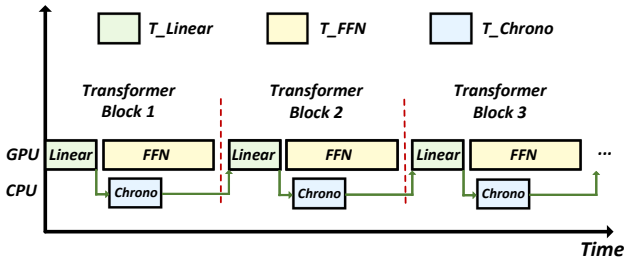



Fig. 12: Parallelizing ChronoCache with FFN computation.

### 6.1 Parallelizing ChronoCache with FFN Computation

In Transformer decoding, latency is typically dominated by two components: the attention computation and the feed-forward network (FFN). In models using linear attention, the FFN accounts for a large portion of per-block latency, particularly when deployed on edge devices, where GPU resources are often saturated by FFN execution. However, CPU cores are frequently underutilized during this phase.

To exploit the idle CPU, we offload the computation of ChronoCache to a background CPU thread, enabling parallel execution with the GPU-resident FFN. This strategy effectively hides the extra latency introduced by ChronoCache behind the FFN latency, as shown in Figure 12.

We identify two components suitable for asynchronous CPU execution.

- **Observation Window Token Importance Assessment**. When $token_{t-W}$ leaves the observation window, its importance is computed by averaging the dot products between its key vector $K_{t-W}$ and the query vectors from the subsequent $W-1$ steps. This operation has a computational complexity of $O(W * d_{model})$, which is lightweight and well-suited for modern CPUs, given that $W$ is typically small.
- **Anchor Cache Importance Maintenance**. At each decoding step $t$, we update the running importance of all $M$ entries in the anchor cache by computing dot products with the current query $Q_t$. This requires $O(M * d_{model})$ operations, which can easily be handled by the CPU due to the modest size of $M$.

Operationally, after the GPU computes the attention and the query $Q_t$ at timestep $t$, it immediately proceeds to the FFN. Meanwhile, the importance-related computations, such as scoring $token_{t-W}$ and updating the anchor cache, are dispatched to a background CPU thread, along with the relevant key and query vectors. Because $T_{FFN}$ significantly exceeds $T_{chrono}$, this parallelization ensures that the KV state updates and token importance assessments incur negligible impact on the end-to-end inference latency.

### 6.2 Warm-Start Initialization of ChronoCache

As ChronoCache maintains a hierarchical cache of KV states, it suffers from a cold-start issue at the beginning of inference, particularly during the initial decoding steps when the cache has not yet accumulated meaningful content. This may degrade the effectiveness of linear attention.

To address this, we propose a warm-start initialization that leverages the prefilling stage of LLM inference. During prefilling, the entire input prompt is available and processed in a single forward pass. This enables us to compute importance scores for all prompt tokens, allowing ChronoCache to effectively initialize the fast-decay channel, slow-decay channel, and anchor cache before decoding begins.

To achieve this, we reuse the token importance concept from § 4.3. For each prompt token $t$ (where $1 \leq t \leq N$), we define its importance score as the sum of raw attention it receives from all subsequent tokens in the prompt: $Score_t = \sum_{j=t+1}^{N} Q_j \cdot K_t^\top$, where $Q_j$ and $K_t$ are the query and key vectors for tokens $j$ and $t$, respectively.

Once the importance scores are computed, we initialize ChronoCache as follows:

- **Anchor Cache.** We populate the anchor cache $M_{anchor}$ with the $M$ tokens that have the highest $Score_t$ values across the prompt. This ensures that globally important tokens are retained without decay.
- **Slow-Decay Channel.** Tokens not selected for the anchor cache but whose $Score_t \geq \tau$ are aggregated into the slow-decay state $S_{slow}$.
- **Fast-Decay Channel.** Remaining tokens with $Score_t < \tau$ are assigned to the fast-decay state $S_{fast}$.

## 7 Implementation

We implement ChronoCache by extending RetNet [19] using Python 3.8, CUDA 11.8 [39], and PyTorch 2.4.0 [40]. The core algorithmic modifications are made in the `recurrent_retention` and `parallel_retention` functions of the `MultiScaleRetention` class.

To support adaptive fast-slow decay, we replace the original single KV state with two separate KV buffers: one for the slow-decay channel ($S_{slow}$) and one for the fast-decay channel ($S_{fast}$). The observation window $W_O$ is implemented as a fixed-size FIFO queue that temporarily holds raw Key and Value vectors of newly generated tokens prior to their importance evaluation.

For the anchor cache $M_{anchor}$, we store the most important KV pairs via a tensor list. In each update, the KV pair at the head of the observation window is compared with the least important entry in the anchor cache. If the importance score exceeds the threshold, the two entries are swapped.

We utilize Python's `threading` to offload ChronoCache's computations to a background CPU thread. After the GPU completes the linear attention computation, the relevant metadata is immediately dispatched to a worker thread on the CPU. Meanwhile, the main GPU thread continues to the FFN module without waiting for ChronoCache to finish. Thread safety for shared memory updates is enforced using `threading.Lock`.

During CUDA kernel execution (*e.g.*, the FFN forward pass), PyTorch releases the Python Global Interpreter Lock (GIL), enabling true parallelism between the GPU computation and our CPU worker thread. According to [6], on the NVIDIA Jetson platforms with a unified memory architecture (CPU and GPU share the same physical DRAM), the `.cpu()` call primarily involves virtual address remapping rather than data copy, significantly reducing overhead. Our measurements confirm that $T_{\text{FFN}} \approx 3\text{–}5 \times T_{\text{chrono}}$ on both testbed devices, ensuring that ChronoCache's computation is fully hidden behind the FFN latency. In the unlikely event that FFN completes first, the main thread simply waits for

| Methods | Single Key | Multi Key | QA | CWE & FWE | RULER score (Avg) |
|---|---|---|---|---|---|
| Standard Transformer | **0.565** | **0.338** | **0.421** | **0.287** | **0.403** |
| ChronoCache | 0.301 | 0.209 | 0.237 | 0.188 | 0.234 |
| TransNormer | 0.019 | 0.03 | 0.026 | 0.02 | 0.024 |
| RetNet | 0.021 | 0.025 | 0.027 | 0.017 | 0.023 |
| Linear Transformer | 0.025 | 0.018 | 0.02 | 0.016 | 0.02 |

TABLE 2: Performance on long-range information retention compared with baselines.

| Methods | ARC-c | ARC-e | HellaSwag | LMB | PIQA |
|---|---|---|---|---|---|
| Standard Transformer | **0.392** | 0.701 | **0.582** | **0.469** | **0.721** |
| ChronoCache | 0.371 | **0.715** | 0.577 | 0.467 | 0.72 |
| TransNormer | 0.366 | 0.696 | 0.58 | 0.448 | 0.719 |
| RetNet | 0.368 | 0.706 | 0.578 | 0.453 | 0.72 |
| Linear Transformer | 0.181 | 0.372 | 0.281 | 0.212 | 0.365 |

TABLE 3: Performance on various language modeling benchmarks compared with baselines.

the CPU worker via lock acquisition, with a bounded worst-case overhead of $T_{\text{chrono}}$ ($\sim$0.3–0.8 ms per layer).

## 8 Evaluation

Our evaluation aims to answer the following questions:

- How does ChronoCache compare to state-of-the-art input-independent linear attention and standard softmax attention in terms of language modeling accuracy and long-range recall (§ 8.2)?
- What are the contributions of ChronoCache's core components (§ 8.3)?
- How do key hyperparameters and model scale affect the performance and efficiency of ChronoCache (§ 8.4)?

### 8.1 Experimental Setup

#### 8.1.1 Testbed

We evaluate two representative edge devices, NVIDIA Jetson Xavier NX with 8GB shared memory, used for deploying 340M-parameter models, and NVIDIA Jetson Orin NX with 16GB shared memory, used for deploying 1.3B-parameter models. All models are quantized with 8-bit.

#### 8.1.2 Tasks, Datasets and Models

We evaluate all models across two categories of tasks:

- **Long-Context Recall.** We evaluate long-range information retention using the RULER suite [24], a benchmark designed for testing long-context recall, including Single-Key Needle-in-a-haystack (Single Key), Multi-Keys Needle-in-a-haystack (Multi Keys), Question Answering (QA), Common Words and Frequent Words Extraction (CWE & FWE).
- **Commonsense Language Modeling.** We assess general reasoning and language understanding using standard benchmarks: ARC-Challenge (ARC-c) [41], ARC-Easy (ARC-e) [41], HellaSwag [42], LAMBADA (LMB) [43], and PIQA [44].

#### 8.1.3 Baselines

We compare ChronoCache against the following baselines:

- **Standard Transformer [25]**: The original Transformer with softmax attention. It serves as an upper bound in accuracy but is computationally and memory intensive.
- **Linear Transformer [16]**: A Transformer using naive linear attention without any decay, representing a lower bound in accuracy.
- **TransNormerLLM [21]**: A hardware-optimized model that uses Lightning Attention and the same fixed decay strategy as RetNet. Lightning Attention is also used in MiniMax-01 (456B parameters), one of the only scaled-up commercial linear attention models.
- **RetNet [19]**: A state-of-the-art linear attention model using a recurrent formulation with data-independent decay.

All models are pretrained on the FineWeb-Edu dataset [45] using the flash-linear-attention library [46]. The 340M-parameter models are trained on 20 billion tokens, while the 1.3B-parameter models are trained on 100 billion tokens. Training uses the AdamW optimizer with a cosine learning rate schedule. Batch sizes are set to 50K tokens for the 340M models and 1 million tokens for the 1.3B models. All evaluations are conducted in a zero-shot setting, without task-specific fine-tuning or prompt engineering.

#### 8.1.4 Evaluation Metrics

Our evaluation employs the following metrics:

- **Language Modeling Score.** Accuracy or task-specific scores on language modeling benchmarks.
- **Long-Context Recall Score.** Average scores on the RULER benchmarks.
- **Memory Usage (GB).** Total memory consumption (including the KV cache and model parameters) during decoding with long contexts.
- **Throughput (Tokens/Second).** Average number of tokens generated per second during decoding.

### 8.2 Overall Performance

This section presents the results for the 1.3B model scale.

| Methods | Single Key | Multi Key | QA | CWE & FWE | RAM Usage (GB) | Throughput (t/s) |
|---|---|---|---|---|---|---|
| ChronoCache | 0.301 | 0.209 | 0.237 | 0.188 | 2.61 | 31.2 |
| w/o Fast-Slow Decay | 0.236 | 0.168 | 0.173 | 0.146 | 2.51 | 31.1 |
| w/o Anchor Cache | 0.215 | 0.132 | 0.141 | 0.117 | 2.56 | 31.9 |
| w/o Runtime Opt | 0.265 | 0.184 | 0.199 | 0.163 | 2.61 | 23.1 |

TABLE 4: Performance comparison on long-range information retention, memory usage, and throughput between ChronoCache and ablation variants.

#### 8.2.1 *Long-Range Recall and Common Sense Language Modeling performance*

As shown in Table 2, the superiority of ChronoCache is pronounced in the long-range information retention. Specifically, it shows that RetNet, TransNormer and linear Transformer suffer from severe performance degradation, only achieving near-zero scores due to early memory fade-out. In contrast, ChronoCache achieves an acceptable RULER score of 0.234, which is nearly tenfold higher than the other linear attention baselines. In addition, due to the excellent long-range modeling ability of the standard Transformer, there is still a gap between ChronoCache and the standard Transformer, which motivates future work.

As shown in Table 3, ChronoCache also achieves scores comparable to the standard Transformer, RetNet and TransNormer in most tasks. For instance, ChronoCache scores 0.371 on ARC-c and 0.715 on ARC-e, on par with or slightly better than RetNet (0.368 and 0.706) and TransNormer (0.366 and 0.696). This indicates that our designs in ChronoCache have not compromised the performance of general language modeling, and have even improved general language understanding in some tasks. In addition, the standard Transformer scores higher than all other methods in most tasks, while the linear Transformer results in the lowest scores in all tasks due to its memory dilution problem.

Note that the benchmarks in Table 3 are *not* long-context tasks. Typical input lengths range from tens to a few hundred tokens. In this short-context regime, all tokens are comprehensively evaluated during the prefilling stage via our warm-start initialization (§ 6.2), which computes importance scores using the full prompt context, bypassing the observation window entirely. Moreover, the anchor cache stores fewer tokens than its maximum capacity, introducing no eviction overhead, and the fast-slow decay has minimal effect over the small number of decoding steps. Consequently, ChronoCache naturally reduces to a standard linear attention model with negligible overhead for short inputs.

#### 8.2.2 *Memory Usage and Throughput*

Figure 13 illustrates the memory usage of different methods. Specifically, the standard Transformer's memory usage grows linearly, surpassing 5 GB at a 16k sequence length and becomes prohibitive on edge devices. All linear attention models, including ChronoCache, maintain a constant and low memory footprint around 2.5 GB to 2.6 GB, making them viable for edge deployment. ChronoCache incurs a tiny memory overhead compared to other linear models due to its anchor cache and fast-slow channels, but this is negligible, as we will discussed later in § 8.5.1.

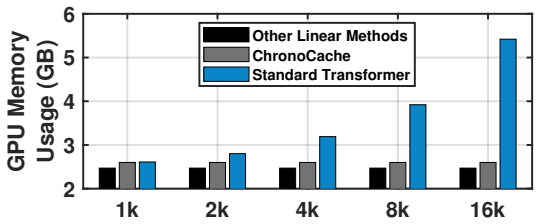



Fig. 13: Memory usage of ChronoCache and other baselines in different sequence length. Since the other linear methods have the same configuration and only have one KV State, we consider their memory overhead as a single entry.

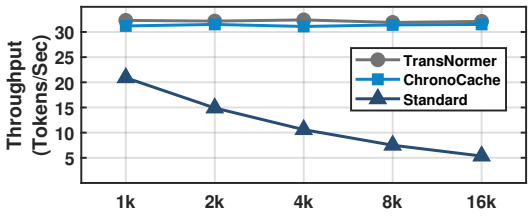



Fig. 14: Throughput of ChronoCache and other baselines in different sequence length. Since our ChronoCache is based upon the RetNet, they have nearly the same results, so we do not consider RetNet here.

In terms of throughput, Figure 14 illustrates that ChronoCache is highly competitive. It achieves comparable throughput and is only slightly smaller than the hardware-optimized Lightning Attention. At a 16k context length, ChronoCache keeps 31.2 tokens/sec, nearly matching TransNormer and significantly outperforming the standard Transformer, whose throughput drops to around 5 tokens/sec. This is because our parallel offloading mechanism effectively hides the latency of ChronoCache's overhead. For the standard transformer, due to the limitation of memory bandwidth and the increase in KV cache, the throughput is reduced. These results confirm that ChronoCache achieves a superior balance, delivering decent long-range information retention performance while preserving the constant-memory and high throughput.

### 8.3 Ablation Study

To isolate and verify the contribution of each design in ChronoCache, we conduct a thorough ablation study on the following three variants of the 1.3B model.

- **ChronoCache w/o Fast-Slow Decay:** This variant removes the dual fast-slow decay mechanism and uses a single, fixed decay rate for all non-anchored tokens.

- **ChronoCache w/o Anchor Cache:** This variant disables the anchor cache and the dynamic competition mechanism, relying solely on the fast-slow channel decay.
- **ChronoCache w/o Runtime Optimization:** In this variant, ChronoCache is executed synchronously on the main GPU thread and does not have the warm-start.

Since our system design is not specifically focused on language modeling capabilities, we will omit the performance comparison in language modeling here. Instead, we mainly focus on the ability of different variants for long-range information retention and their memory usage and throughput. Table 4 presents the results of the ablation study.

**Contribution of Fast-Slow Decay.** The variant w/o Fast-Slow Decay shows a notable drop in its RULER score. Its average score across tasks falls from 0.234 to 0.181, a 22.6% reduction in performance. Without the ability to assign different lifecycles to tokens of varying importance, the model struggles to manage the KV state effectively, leading to the premature decay of useful information. This proves that our fast-slow decay is essential for fine-grained context management. For the memory usage, since the fast and slow KV states are not included, the memory usage is accordingly reduced. For throughput, no significant change is observed.

**Contribution of Anchor Cache.** The variant w/o Anchor Cache experiences the most dramatic performance collapse, over 35%, from 0.234 to 0.151, demonstrating a catastrophic failure in long-range recall. This powerfully validates that the non-decaying anchor memory and its dynamic competition mechanism are the cornerstone of ChronoCache, fundamentally solving the early memory fade-out problem that plagues all purely decay-based models. For the memory usage, since the anchor cache is not included, the memory usage is accordingly reduced. For throughput, no significant change is observed.

**Contribution of Runtime Optimization.** The variant w/o Runtime Optimization not only has a lower RULER score compared with the full ChronoCache but also suffers a substantial drop in throughput. Specifically, for the performance, the RULER score decreases from 0.234 to 0.203; the key reason is that without the warm-start initialization, the hierarchical states (including the anchor cache, the fast-slow decay channels) will be empty at the beginning, thus leading to lower-quality output. For the throughput drop, the key reason is that the synchronous execution of ChronoCache introduces significant latency, causing throughput to fall from 31.2 tokens/sec to 23.1 tokens/sec, an approximately 26% decrease. This result confirms the effectiveness of our runtime optimization, proving that offloading the system overhead to a background CPU thread and warm-start of hierarchical states are critical for maintaining high throughput without sacrificing model performance.

## 8.4 Micro-benchmarks

In this section, we conduct further micro-benchmarks to analyze the impact of key hyperparameters and model scale.

### *8.4.1 Impact of Observation Window size $W$*

Figure 15 illustrates the trade-off associated with $W$. A larger $W$ provides a longer observation period, enabling a more accurate assessment of a token's importance and thus improving the RULER score. For example, increasing $W$ from 8 to 128 improves the RULER score from approximately 0.17 to over 0.23. However, this comes at the cost of increased memory overhead. Our results indicate that a $W$ between 64 and 128 strikes an excellent balance between recall performance and memory efficiency.

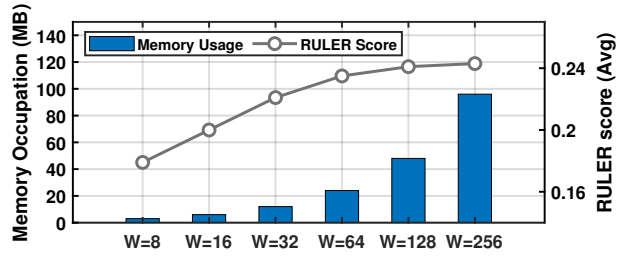



Fig. 15: RULER Score and Memory occupation in different Observation Window Size $W$ configurations.

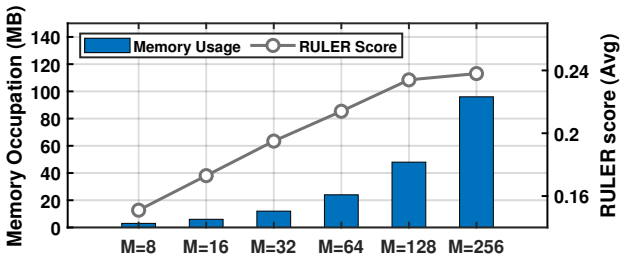



Fig. 16: RULER Score and Memory occupation in different Anchor Cache size $M$ configurations.

### *8.4.2 Impact of Anchor Memory size $M$*

Figure 16 analyzes the effect of the size of anchor cache $M$. The recall performance on RULER improves as $M$ increases, *e.g.*, from 0.15 to 0.23 as $M$ grows from 8 to 128, since more foundational tokens can be permanently stored. However, the gains begin to saturate, while the memory cost grows linearly with $M$. We find that M between 128 and 256 captures most of the performance benefits for typical long-document tasks, making it a robust default choice.

### *8.4.3 Impact of Model size*

To assess the scalability of our approach, we evaluated ChronoCache on a 340M parameter model, with results presented in Table 5. The findings confirm that the advantages of ChronoCache remain pronounced at this smaller scale. In long-context recall tasks, the 340M ChronoCache model achieves a RULER score of 0.147, substantially outperforming both RetNet and TransNormer, which each scored only 0.017. Furthermore, ChronoCache maintains its efficiency, consuming only 0.669 GB of RAM compared to the 2.12 GB required by the standard Transformer at a 16K sequence length. Its throughput remains high at 25.8 tokens per second, far exceeding the standard Transformer's 7.1 tokens per second. These results demonstrate that ChronoCache is a highly scalable architecture, proving its effectiveness and efficiency across different model sizes, which is essential for enabling powerful, long-context LLMs on edge platforms.

| Methods | RULER score (Avg) | RAM (GB) | Throughput (t/s) |
|---|---|---|---|
| Standard | 0.269 | 2.12 | 7.1 |
| ChronoCache | 0.147 | 0.669 | 25.8 |
| TransNormer | 0.017 | 0.633 | 26.3 |
| RetNet | 0.017 | 0.645 | 25.7 |
| Linear | 0.014 | 0.643 | 25.1 |

TABLE 5: Performance comparison on long-range information retention, memory usage, and throughput between ChronoCache and other methods with 340M model parameters. Result of memory usage and throughput are obtained in 16K sequence length with NVIDIA Jetson Xavier NX.

#### 8.4.4 Local vs. Global Importance Correlation

To validate that local observation windows reliably predict global token importance, we computed the Spearman rank correlation between local importance scores (within a $W = 128$ window) and global importance scores (computed over the full sequence during prefilling). Table 6 summarizes the results on the RULER benchmark using the 1.3B model. The consistently high correlation ($\rho \geq 0.82$) confirms that local windows are strong predictors of long-term importance, with only a modest decrease at longer sequences due to additional variance from distant queries.

TABLE 6: Spearman rank correlation ($\rho$) between local importance scores ($W = 128$) and global importance scores.

| Sequence Length | 4K | 8K | 16K |
|---|---|---|---|
| Spearman $\rho$ | 0.87 | 0.84 | 0.82 |

#### 8.4.5 Threshold Sensitivity Analysis

We sweep the importance threshold $\tau$ from the top-5% to top-20% of attention scores to assess robustness. As shown in Table 7, the RULER score remains stable across the entire range at both model scales, with relative variation below 10%. The default top-10% setting performs near-optimally, confirming that $\tau$ does not require precise per-task tuning. The top-10% threshold achieves the best results because it corresponds to the "knee point" of the attention concentration curve [29], [28]: below this level, the anchor cache is under-utilized and important tokens are unnecessarily relegated to decay channels; above it, marginally important tokens crowd the anchor cache and risk displacing critical entries via the competition mechanism.

TABLE 7: Threshold sensitivity: RULER scores under different top-$k$% thresholds.

| Configuration | 5% | 10% | 15% | 20% |
|---|---|---|---|---|
| 340M, 16K context | 0.134 | **0.147** | 0.142 | 0.133 |
| 1.3B, 16K context | 0.218 | **0.234** | 0.228 | 0.219 |

#### 8.4.6 Decay Rate Sensitivity Analysis

We conduct a grid sweep over $\gamma_{\text{low}} \in \{0.999, 0.9995, 0.9999\}$ and $\gamma_{\text{high}} \in \{0.9, 0.92, 0.95\}$ on the RULER benchmark (16K context, 1.3B model). To support claims resting on small margins, we repeat each configuration over 5 random seeds (varying evaluation-time sampling and the RULER haystack construction order) and report mean $\pm$ standard deviation. As shown in Table 8, the RULER score varies by less than 0.015 across the grid, confirming that ChronoCache is not sensitive to the exact decay rate within the recommended ranges. The run-to-run standard deviation is small ($\pm$0.003–0.005), as the anchor cache deterministically preserves the highest-scored tokens. Within this variance, $\gamma_{\text{low}} = 0.9995$ ($L_{\text{eff}} \approx 9{,}210$ steps) and $\gamma_{\text{low}} = 0.9999$ ($L_{\text{eff}} \approx 46{,}052$ steps) are indistinguishable, so we treat the method as robust across this range; the only separation exceeding the variance is that the aggressive $\gamma_{\text{low}} = 0.999$ ($L_{\text{eff}} \approx 4{,}603$ steps) is consistently weaker, decaying too fast for 16K sequences. Overall the differences remain marginal, as the anchor cache already protects the most critical tokens regardless of decay settings.

TABLE 8: RULER score under different decay rate combinations (16K context, 1.3B model). Mean $\pm$ std over 5 seeds.

| $\gamma_{\text{low}}$ \ $\gamma_{\text{high}}$ | 0.9 | 0.92 | 0.95 |
|---|---|---|---|
| 0.999 | $0.224 \pm 0.004$ | $0.227 \pm 0.004$ | $0.221 \pm 0.005$ |
| 0.9995 | $0.231 \pm 0.003$ | $0.234 \pm 0.003$ | $0.226 \pm 0.004$ |
| 0.9999 | $0.230 \pm 0.004$ | $0.233 \pm 0.003$ | $0.225 \pm 0.004$ |

#### 8.4.7 Comparison with Input-Dependent Decay

ChronoCache adopts input-independent decay, which is more efficient on edge devices than input-dependent decay. To quantify the accuracy gap traded against this efficiency, we add a representative input-dependent baseline, Gated Linear Attention (GLA) [20], under the same 1.3B configuration and training budget. As shown in Table 9, GLA attains substantially higher RULER accuracy than plain input-independent RetNet (0.239 vs. 0.023) but at a $\sim$40% throughput cost on the Jetson Orin NX, consistent with the speed gap in Figure 6. ChronoCache resolves this tension, reaching accuracy comparable to GLA (0.234 vs. 0.239) while retaining the full throughput of input-independent decay (31.2 vs. 18.7 t/s), thus combining the recall quality of input-dependent decay with the efficiency of input-independent decay.

TABLE 9: Accuracy–efficiency trade-off between input-independent and input-dependent decay (1.3B model, 16K context). Throughput measured on Jetson Orin NX.

| Method | Decay type | RULER | Thr. (t/s) |
|---|---|---|---|
| RetNet | input-independent | 0.023 | 31.4 |
| GLA [20] | input-dependent | 0.239 | 18.7 |
| ChronoCache (ours) | input-independent | 0.234 | 31.2 |

#### 8.4.8 Cross-Domain Threshold Validation

To assess whether distribution shift between the training corpus and real-world inference data affects threshold calibration, we compared the attention score distributions across three diverse corpora: FineWeb-Edu (our training corpus), English Wikipedia, and ShareGPT (conversational data). The top-10% attention score threshold deviates by less than $\pm$8% across these domains (Table 10), and the Spearman rank correlation of per-head importance rankings

exceeds $\rho \approx 0.85$, confirming that the threshold generalizes well without domain-specific tuning.

TABLE 10: Cross-domain threshold deviation relative to FineWeb-Edu.

| Corpus | Relative Threshold Deviation |
|---|---|
| FineWeb-Edu (reference) | — |
| Wikipedia | +4.3% |
| ShareGPT (conversational) | −7.1% |

#### 8.4.9 Multilingual Robustness

Since ChronoCache's importance scoring operates on learned embedding representations via the $Q \cdot K$ dot product rather than surface-level linguistic features, it is fundamentally language-agnostic. We emphasize that this is a strictly controlled comparison: ChronoCache and all baselines share the identical backbone, parameter count (1.3B), and English-dominant training corpus (FineWeb-Edu), and ChronoCache adds no trainable parameters. Since language exposure is held constant, the goal is not absolute multilingual proficiency but the language-agnostic robustness of our routing under distribution shift. To verify this, we conducted a preliminary evaluation on multilingual subsets of the CC-100 corpus covering Chinese, German, and French, using the multilingual RULER benchmark at 16K context length (Table 11). ChronoCache achieves an absolute improvement of $+0.190$ over RetNet. Importantly, all linear attention baselines (RetNet, TransNormerLLM, Linear Transformer) exhibit similarly low scores on multilingual inputs as they do on English-only inputs, indicating that the linear attention mechanism itself does not introduce language-specific degradation. This shows their degradation stems from early memory fade-out rather than a lack of multilingual data, so ChronoCache's consistent gain across languages is attributable solely to its three-tier KV state management.

TABLE 11: Multilingual RULER performance (16K context, 1.3B model).

| Method | Multilingual RULER | Abs. Gain |
|---|---|---|
| ChronoCache | **0.210** | +0.190 |
| TransNormerLLM | 0.021 | +0.001 |
| RetNet | 0.020 | — |
| Linear Transformer | 0.016 | −0.004 |

#### 8.4.10 Comparison with Hybrid Architectures

To empirically assess the relationship between ChronoCache and hybrid architectures that interleave linear attention with standard softmax attention, we construct hybrid variants of both ChronoCache and RetNet following a 6:1 mixing ratio: for every 7 consecutive layers, 6 use linear attention and 1 uses full softmax attention with a standard KV cache. All models share the same 1.3B parameter budget and are evaluated on the RULER benchmark at 16K context.

Table 12 reports the long-context recall scores alongside the KV memory footprint (excluding model weights) at 4K and 16K context lengths. Two observations stand out. First, adding softmax attention layers improves recall for both methods: CC-Hybrid achieves higher RULER scores than pure ChronoCache across all subtasks, and RetNet-Hybrid similarly improves over pure RetNet. Notably, ChronoCache's pure-linear configuration still substantially outperforms RetNet-Hybrid on every subtask, demonstrating that effective KV state management can close much of the accuracy gap without resorting to full attention layers. Second, and crucially, hybrid models sacrifice the constant-memory property that is the primary motivation for linear attention on edge devices. While ChronoCache and RetNet maintain exactly the same KV memory at 4K and 16K (0.21 GB and 0.06 GB, respectively), their hybrid counterparts exhibit significant KV memory growth: CC-Hybrid increases from 0.30 GB at 4K to 0.64 GB at 16K ($\times 2.1$), and RetNet-Hybrid from 0.17 GB to 0.51 GB ($\times 3.0$). This growth stems from the softmax attention layers, which require an $O(L)$ KV cache that scales linearly with context length, precisely the bottleneck ChronoCache is designed to eliminate. As context length continues to increase, this gap will widen further.

These results confirm that ChronoCache and hybrid architectures address the efficiency-accuracy tradeoff at different design levels: ChronoCache maximizes recall within the constant-memory constraint, while hybrid models relax that constraint for additional accuracy at the cost of memory scalability. In memory-constrained edge deployments, ChronoCache offers a favorable operating point; when memory permits, ChronoCache can be integrated into hybrid models to further enhance the linear attention layers, as discussed in § 9.

TABLE 12: Comparison with hybrid architectures (6:1 linear-to-full ratio, 1.3B model, 16K context). CC = ChronoCache. KV memory excludes model weights.

| Method | S-Key | M-Key | QA | CWE | KV Mem (GB) | |
|---|---|---|---|---|---|---|
| | | | | | 4K | 16K |
| CC (ours) | 0.301 | 0.209 | 0.237 | 0.188 | 0.21 | 0.21 |
| CC-Hybrid | 0.348 | 0.243 | 0.271 | 0.218 | 0.30 | 0.64 |
| RetNet | 0.021 | 0.025 | 0.027 | 0.017 | 0.06 | 0.06 |
| RetNet-Hybrid | 0.079 | 0.058 | 0.068 | 0.047 | 0.17 | 0.51 |

#### 8.4.11 Parameter Selection Guidelines

Based on the micro-benchmark results above, we summarize practical guidelines for configuring ChronoCache on edge devices:

- **Observation window** $W$: Set to the average paragraph length in the target domain. For general-purpose text, $W = 64$–$128$ works well; for code or structured data with shorter semantic units, $W = 32$–$64$ may suffice.
- **Anchor cache size** $M$: Set to $M = 128$–$256$ for a 16K context. This range captures the typical fraction of structurally critical tokens and scales proportionally with the expected maximum sequence length.
- Both parameters showed consistent effectiveness across our 340M and 1.3B model evaluations, suggesting that these guidelines generalize across model scales within typical edge deployment scenarios.

### 8.5 System Overhead

A critical aspect of ChronoCache is its suitability for resource-constrained edge devices. This section analyzes the

system overheads introduced by our design, focusing on memory and power overhead.

#### 8.5.1 Memory Overhead

ChronoCache is designed to be highly efficient, introducing only a modest and constant overhead. The memory footprint increases slightly due to two components, the dual-channel KV states ($S_low$, $S_high$) with observation window (size W = 64), and the anchor memory (size M = 128) . For our 1.3B model, this amounts to a total fixed overhead of approximately 0.21 GB. This cost is negligible compared to the gigabytes consumed by a standard Transformer's KV Cache in long-context scenarios.

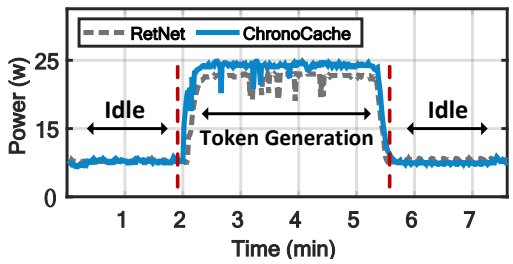



Fig. 17: Power consumption comparison between ChronoCache and RetNet.

#### 8.5.2 Power Consumption

Low power consumption is paramount for mobile and edge deployment. We measured the power draw of ChronoCache on an NVIDIA Jetson Orin NX using the INA3221 power monitor [47] and compared it against the RetNet baseline.

As illustrated in Figure 17, while the baseline RetNet model consumes an average of 22.3 W during inference, ChronoCache draws a slightly higher 25 W. This marginal increase of approximately 2.7 W is primarily attributed to the sustained activity of CPU core executing our background management thread. This minimal power overhead confirms that ChronoCache is an energy-efficient solution suitable for deployment on power-sensitive and battery-operated devices. In addition, the instantaneous drop of power during decoding is caused by the thermal throttling mechanism of the mobile edge devices. Specifically, to prevent overheating, the Dynamic Voltage and Frequency Scaling (DVFS) module of the mobile edge device will actively reduce the processor frequency [7].

## 9 Related Work

### 9.1 Hybrid Architectures

Recognizing the trade-offs between the performance of Softmax attention and the efficiency of its linear alternatives, a new class of hybrid architectures has emerged in the commercial. These models aim to achieve the best of both worlds by combining different attention mechanisms or model components. For instance, in recent commercial LLMs, such as Tencent's Hunyuan-TurboS [48] and Minimax-01/M1 series [37], [49], have reportedly adopted hybrid designs that leverage different model components for different complexities of tasks.

In addition, architectures like Hymba [50] explicitly combine State Space Models (*e.g.*, SSMs from Mamba [51]) with standard Transformer blocks. Others methods like RecurrentGemma [52] from Google employed the Griffin architecture combining linear recurrences with local attention, using a mixture of mechanisms within each layer rather than interleaving different layer types. This approach allows the model to capture long-range dependencies efficiently while retaining the high-fidelity modeling capabilities of Softmax attention.

Our work on ChronoCache is also orthogonal to these hybrid architectural designs. The core innovation of ChronoCache, *i.e.*, its dynamic context management system is not tied to a specific model architecture, but rather enhances the linear attention mechanism itself. Therefore, it can be integrated into the linear attention or SSM components of these hybrid models. By replacing a naive or fixed-decay linear attention module with ChronoCache, hybrid models could significantly amplify their long-range memory retention capabilities without altering their fundamental architectural blueprint, thus further improving their overall performance-efficiency trade-off.

### 9.2 KV Cache Management

In Key-Value (KV) Cache of standard Transformer, the $O(L^2)$ complexity and $O(L)$ memory growth are primary bottlenecks for long-context inference. A substantial body of work has focused on mitigating this issue [28], [30], [27], [29], [32], [53]. One popular approach is eviction, where only a subset of recent or important tokens are kept in the cache. StreamingLLM [27] pioneers this by observing attention sink phenomenon that initial few tokens is crucial for stability, allowing it to maintain a fixed-size sliding window cache. Following this, works like SnapKV [28] and H2O [29] have proposed more sophisticated, data-driven eviction policies to selectively retain heavy-hitter key-value pairs that are deemed important. Other optimization strategies attack the problem from different angles. Quantization, as exemplified by KIVI [30], reduces the memory footprint by compressing the KV cache to lower bit-widths (e.g., 2-bit) with minimal performance loss. Other architectural modifications like Multi-Query Attention (MQA) [31] and Grouped-Query Attention (GQA) [32] reduce the size of the cache by sharing K and V projections across attention heads.

ChronoCache contributes a novel perspective. Based on linear attention, it introduces a dynamic lifecycle management system, instead of simply mixing the KV state uniformly. It differentiates between transient, recent, and foundational information, managing each with a dedicated mechanism (fast-slow decay, observation window, and anchor memory). Crucially, our work is orthogonal to most existing KV Cache optimizations. Techniques like KIVI's quantization or SnapKV's eviction policies could be applied directly to the K and V vectors stored within ChronoCache 's observation window and anchor memory. This synergy could lead to further, compounded reductions in system memory usage, combining the benefits of intelligent lifecycle management with advanced compression and eviction.

### 9.3 Sparse Attention and Collaborative Inference

Beyond linear attention and KV cache management, two additional lines of work address the efficiency challenges of long-context LLM inference from complementary angles.

**Sparse Attention Methods.** BigBird [33] introduces a sparse attention mechanism that combines random attention, sliding-window local attention, and global tokens to approximate full attention with $O(L)$ computational complexity. Sparse Sinkhorn Attention [34] takes a different approach by learning to re-order and block-sparsify the attention matrix via the Sinkhorn operator, attending only to the most relevant blocks. While both methods effectively reduce *computational* cost, they still require materializing the complete KV cache, whose memory grows as $O(L)$. In contrast, ChronoCache achieves *constant-memory* inference by operating on a compressed KV state and managing information retention through adaptive decay and anchor caching. The two paradigms are orthogonal: the sparsity patterns identified by methods like BigBird could serve as additional signals for ChronoCache 's token importance scoring, a promising direction for future work.

**Collaborative and Split Inference.** An alternative paradigm for enabling LLM inference on resource-constrained devices is to distribute computation across network-connected devices. SplitLLM [13] partitions an LLM across multiple devices and optimizes model placement and workload distribution to maximize end-to-end throughput. Similarly, Vainio *et al.* [12] explore pipelined deep learning inference across fog and edge nodes. These approaches address *how computation is distributed*, whereas ChronoCache addresses *how memory is managed* within a single device. The two directions are complementary: in a split inference pipeline, ChronoCache could be applied to each participating device's linear attention layers to reduce per-device memory consumption, allowing either longer contexts or larger model shards. Conversely, ChronoCache enables fully local, network-independent inference for privacy-sensitive or offline scenarios where collaborative approaches are infeasible.

## 10 Conclusion

In this paper, we present ChronoCache, a lightweight and adaptive KV cache management framework for long-context inference of constant-memory linear attention models on edge devices. By combining an adaptive fast-slow decay mechanism with a bounded, non-decaying anchor cache, ChronoCache retains context by token importance while permanently preserving cornerstone information, mitigating attention dilution and early memory fade-out at minimal latency overhead. Extensive evaluations on multiple long-context benchmarks show that ChronoCache consistently surpasses state-of-the-art linear attention methods, striking an optimal balance between memory efficiency and inference accuracy.

## Acknowledgement

This work is supported by the General Research Fund (GRF) grants (CityU 11205624, CityU 11202623 and CityU 11206425) and the Collaborative Research Fund (CRF) grant (C6086-25G) from Research Grants Council of Hong Kong. Zimu Zhou and Zhenjiang Li are corresponding authors.

## References

[1] O. Friha, M. A. Ferrag, B. Kantarci, B. Cakmak, A. Ozgun, and N. Ghoualmi-Zine, "Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness," *IEEE Open Journal of the Communications Society*, 2024.

[2] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile edge intelligence for large language models: A contemporary survey," *IEEE Communications Surveys & Tutorials*, 2025.

[3] F. Wang, Z. Zhang, X. Zhang, Z. Wu, T. Mo, Q. Lu, W. Wang, R. Li, J. Xu, X. Tang *et al.*, "A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness," *arXiv preprint arXiv:2411.03350*, 2024.

[4] Q. Team, "Qwen2.5: A party of foundation models," September 2024. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5/

[5] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*, 2024.

[6] K. Wang, J. Cao, Z. Zhou, and Z. Li, "Swapnet: Efficient swapping for dnn inference on edge ai devices beyond the memory budget," *IEEE Transactions on Mobile Computing*, vol. 23, no. 9, pp. 8935–8950, 2024.

[7] C. Lin, K. Wang, Z. Li, and Y. Pu, "A workload-aware dvfs robust to concurrent tasks for mobile devices," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–16.

[8] K. Wang, Z. Zhou, and Z. Li, "Latte: Layer algorithm-aware training time estimation for heterogeneous federated learning," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 1470–1484.

[9] J. Cao, J. Chen, C. Lin, Y. Liu, K. Wang, and Z. Li, "Practical gaze tracking on any surface with your phone," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 14 689–14 707, 2024.

[10] K. Wang, Z. Zhou, and Z. Li, "Towards accurate training time estimation for on-device heterogeneous federated learning," *IEEE Transactions on Mobile Computing*, 2025.

[11] J. Cao, K. Wang, Y. Liu, and Z. Li, "Neuropath: Practically adopting motor imagery decoding through eeg signals," in *Proceedings of the 2026 ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems*, 2026, pp. 661–674.

[12] A. Vainio, A. Mudvari, D. Kiedanski, S. Tarkoma, and L. Tassiulas, "Fog computing for deep learning with pipelines," in *2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC)*. IEEE, 2023, pp. 64–72.

[13] A. Mudvari, Y. Jiang, and L. Tassiulas, "Splitllm: Collaborative inference of llms for model placement and throughput optimization," *arXiv preprint arXiv:2410.10759*, 2024.

[14] W. Yin, M. Xu, Y. Li, and X. Liu, "Llm as a system service on mobile devices," *arXiv preprint arXiv:2403.11805*, 2024.

[15] Z. Lu, X. Li, D. Cai, R. Yi, F. Liu, X. Zhang, N. D. Lane, and M. Xu, "Small language models: Survey, measurements, and insights," *arXiv preprint arXiv:2409.15790*, 2024.

[16] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International conference on machine learning*. PMLR, 2020, pp. 5156–5165.

[17] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[18] D. Han, Y. Pu, Z. Xia, Y. Han, X. Pan, X. Li, J. Lu, S. Song, and G. Huang, "Bridging the divide: Reconsidering softmax and linear attention," in *NeurIPS*, 2024.

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

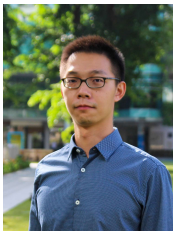
[20] S. Yang, B. Wang, Y. Shen, R. Panda, and Y. Kim, "Gated linear attention transformers with hardware-efficient training," *arXiv preprint arXiv:2312.06635*, 2023.

[21] Z. Qin, W. Sun, D. Li, X. Shen, W. Sun, and Y. Zhong, "Various lengths, constant speed: Efficient language modeling with lightning attention," *arXiv preprint arXiv:2405.17381*, 2024.

[22] E. R. Kandel, "The biology of memory: a forty-year perspective," *Journal of Neuroscience*, vol. 29, no. 41, pp. 12 748–12 756, 2009.
[23] S. Sridhar, A. Khamaj, and M. K. Asthana, "Cognitive neuroscience perspective on memory: overview and summary," *Frontiers in human neuroscience*, vol. 17, p. 1217093, 2023.
[24] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg, "Ruler: What's the real context size of your long-context language models?" *arXiv preprint arXiv:2404.06654*, 2024.
[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[26] H. Li, Y. Li, A. Tian, T. Tang, Z. Xu, X. Chen, N. Hu, W. Dong, Q. Li, and L. Chen, "A survey on large language model acceleration based on kv cache management," *arXiv preprint arXiv:2412.19442*, 2024.
[27] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.
[28] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, "Snapkv: Llm knows what you are looking for before generation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 22 947–22 970, 2024.
[29] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 661–34 710, 2023.
[30] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "Kivi: A tuning-free asymmetric 2bit quantization for kv cache," *arXiv preprint arXiv:2402.02750*, 2024.
[31] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *arXiv preprint arXiv:1911.02150*, 2019.
[32] J. Ainslie, J. Lee-Thorp, M. De Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.
[33] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.
[34] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan, "Sparse sinkhorn attention," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9438–9447.
[35] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, M. Grella *et al.*, "Rwkv: Reinventing rnns for the transformer era," *arXiv preprint arXiv:2305.13048*, 2023.
[36] Y. Mao, Z. Qin, J. Zhou, H. Deng, X. Shen, B. Fan, J. Zhang, Y. Zhong, and Y. Dai, "Autoregressive image generation with linear complexity: A spatial-aware decay perspective," *arXiv preprint arXiv:2507.01652*, 2025.
[37] A. Li, B. Gong, B. Yang, B. Shan, C. Liu, C. Zhu, C. Zhang, C. Guo, D. Chen, D. Li *et al.*, "Minimax-01: Scaling foundation models with lightning attention," *arXiv preprint arXiv:2501.08313*, 2025.
[38] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.
[39] "CUDA Toolkit Documentation," https://docs.nvidia.com/cuda/cuda-runtime-api/.
[40] "PyTorch," https://github.com/pytorch/pytorch.
[41] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.
[42] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.
[43] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández, "The lambada dataset: Word prediction requiring a broad discourse context," *arXiv preprint arXiv:1606.06031*, 2016.
[44] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.
[45] G. Penedo, H. Kydlíček, A. Lozhkov, M. Mitchell, C. A. Raffel, L. Von Werra, T. Wolf *et al.*, "The fineweb datasets: Decanting the web for the finest text data at scale," *Advances in Neural Information Processing Systems*, vol. 37, pp. 30 811–30 849, 2024.
[46] "Flash linear attention," https://github.com/fla-org/flash-linear-attention.
[47] Texas Instruments Inc., "INA3221 power monitor," https://www.ti.com/product/INA3221, 2016.
[48] T. H. Team, A. Liu, B. Zhou, C. Xu, C. Zhou, C. Zhang, C. Xu, C. Wang, D. Wu, D. Wu *et al.*, "Hunyuan-turbos: Advancing large language models through mamba-transformer synergy and adaptive chain-of-thought," *arXiv preprint arXiv:2505.15431*, 2025.
[49] A. Chen, A. Li, B. Gong, B. Jiang, B. Fei, B. Yang, B. Shan, C. Yu, C. Wang, C. Zhu *et al.*, "Minimax-m1: Scaling test-time compute efficiently with lightning attention," *arXiv preprint arXiv:2506.13585*, 2025.
[50] X. Dong, Y. Fu, S. Diao, W. Byeon, Z. Chen, A. S. Mahabaleshwarkar, S.-Y. Liu, M. Van Keirsbilck, M.-H. Chen, Y. Suhara *et al.*, "Hymba: A hybrid-head architecture for small language models," *arXiv preprint arXiv:2411.13676*, 2024.
[51] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.
[52] A. Botev, S. De, S. L. Smith, A. Fernando, G.-C. Muraru, R. Haroun, L. Berrada, R. Pascanu, P. G. Sessa, R. Dadashi *et al.*, "Recurrentgemma: Moving past transformers for efficient open language models," *arXiv preprint arXiv:2404.07839*, 2024.
[53] Y. Liu, H. Li, Y. Cheng, S. Ray, Y. Huang, Q. Zhang, K. Du, J. Yao, S. Lu, G. Ananthanarayanan *et al.*, "Cachegen: Kv cache compression and streaming for fast large language model serving," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 38–56.



**Kun Wang** received the B.E. degree from Xidian University, Xi'an, China, in 2020, and the Ph.D. degree from the City University of Hong Kong, Hong Kong, China, in 2024. He is currently a Postdoctoral Fellow at the Department of Computer Science, City University of Hong Kong. His research interests lie in designing and building efficient on-device AI systems, including on-device inference, on-device training and on-device large language models.



**Zimu Zhou** received the B.E. from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2011, and the Ph.D. from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, in 2015. He is currently an Assistant Professor at the School of Data Science, City University of Hong Kong. His research focuses on mobile and ubiquitous computing.



**Zhenjiang Li** received the B.E. degree from Xi'an Jiaotong University, China, in 2007, and the M.Phil. and Ph.D. degrees from the Hong Kong University of Science and Technology, Hong Kong, China, in 2009 and 2012, respectively. He is currently a Professor with the Department of Computer Science, City University of Hong Kong. His research interests include Internet of Things, edge AI systems and smart sensing.