TreeKV: Smooth Key-Value Cache Compression with Tree Structures

Ziwei He[1], **Jian Yuan**[1], **Haoli Bai**[2], **Jingwen Leng**[1], **Bo Jiang**[1]
[1]Shanghai Jiao Tong University
[2]Huawei Noah's Ark Lab
{ ziwei.he, yuanjian, leng-jw, bjiang}@sjtu.edu.cn

Abstract

Efficient key-value (KV) cache compression is critical for scaling transformer-based Large Language Models (LLMs) in long sequences and resource-limited settings. Existing methods evict tokens based on their positions or importance, but position-based strategies can miss crucial information outside predefined regions, while those relying on global importance scores resulting in strong regional biases, limiting the KV cache's overall context retention and potentially impairing the performance of LLMs on complex tasks. Our wavelet analysis reveals that as tokens approach the end of sequence, their contributions to generation gradually increase and tends to diverge more from neighboring tokens, indicating a smooth transition with increasing complexity and variability from distant to nearby context. Motivated by this observation, we propose TreeKV, an intuitive, training-free method that employs a tree structure for smooth cache compression. TreeKV maintains a fixed cache size, allowing LLMs to deliver high-quality output in long text scenarios and is applicable during both the generation and prefilling stages. TreeKV consistently surpasses all baseline models in language modeling tasks on PG19 and OpenWebText2, allowing LLMs trained with short context window to generalize to longer window with a 16x cache reduction. On the Longbench benchmark, TreeKV achieves the best performance with only 6% of the budget at optimal efficiency.

1 Introduction

Large Language Models (LLMs) exhibit impressive ability to comprehend and produce text at human level, enabling them to perform tasks such as summarization, question answering, and creative writing [Wei *et al.*, 2022; Yuan *et al.*, 2022; Zhang *et al.*, 2024a]. To support efficient token generation, transformer-based LLMs typically store the key-value (KV) pairs of past tokens in memory, referred to as KV cache [Pope *et al.*, 2022]. However, for very long sequences, the KV cache can require a memory that is several times larger than that for storing the model parameters, posing significant challenges in long context scenarios or resource-limited environments [Liu *et al.*, 2024]. This necessitates innovative strategies to optimize the KV cache memory footprint without compromising the performance.

Aiming for training-free methods, recent studies propose to cache the KV pairs of only a subset of past tokens, subject to a pre-defined capacity constraint. Those efficient KV cache methods typically address two different stages, prefilling stage and decoding stage. The prefilling stage is primarily the phase where the model develops a representation of the context it receives. The decoding stage focused on generating text output based on the encoded input. Effective management of both stages ensures that the cache operates not just as a temporary storage solution but as a proactive element enhancing overall system efficiency. For decoding stage optimization, some methods [Xiao *et al.*, 2023; Han *et al.*, 2024] retain only initial and recent tokens, while others [Zhang *et al.*, 2024c; Oren *et al.*, 2024; Liu *et al.*, 2024] select tokens based on their importance scores. Those approaches help manage cache size and allow models to generate sequences longer than the pre-training context length. However, these methods often lack efficient prefilling strategy, leading to high latency due to token-by-token processing. In contrast, there are other methods focusing on optimizing the prefilling stage strategy by intelligently populating the cache based on the input context. For example, [Li *et al.*, 2024] retained only critical tokens from the context, while [Zhang *et al.*, 2024b; Wang *et al.*, 2024] identified key information from KV cache using structural insights.

Despite their relative success, existing cache eviction methods have their limitations. The pure position-based selection strategies may miss out important tokens outside the pre-defined regions. On the other hand, as we will see in Figure 1, the strategies based on importance scores turn out to have very strong regional bias, which will limit the KV cache's ability to maintain a global view and potentially impair LLMs performance on complex, context-rich tasks.

This work aims to overcome the above limitations. To gain a clearer understanding of the characteristics conveyed by the KV cache, we use wavelet transform to examine the frequency representation of information contributed by tokens at different positions during generation. The results reveal that as tokens approach the end of the sequence, the amplitudes of signals corresponding to different frequency components

gradually increase, particularly at higher frequencies. This suggests that the information contributed by a token not only increases, but also becomes more distinct from its neighboring tokens as it approaches the end, indicating a smooth transition with increasing complexity and variability from distant to nearby context. This observation inspired our approach, leading to the design of a structure that is sparse on the left and dense on the right.

Based on the insights, we propose TreeKV, an intuitive, training-free approach that employs a tree structure for smooth cache compression. Unlike other cache eviction strategies, TreeKV optimizes computational efficiency and memory usage by maintaining an abstraction of the input sequence, facilitating structured and smooth transitions in context granularity between short-range and long-range contexts. By strategically removing tokens from the distant past while prioritizing the recent, proposed approach minimizes bias from heavily concentrated regions, enhancing the model's ability to handle tasks requiring comprehensive context. TreeKV distinguishes itself from most cache compression methods by being applicable to both the generation and prefilling phases, facilitating long-form generation and nuanced long context understanding.

Our contributions are summarized as follows:

- We analyze the frequency representations of information collected during generation using wavelet decomposition. The results show that as tokens near the end of sequence, all frequency components gradually increase, particularly at higher frequencies. This insight inspired our method, resulting in a structure designed to be sparse on the left and dense on the right.
- We introduce TreeKV, an innovative, training-free method that employs a tree structure to enhance smooth cache compression. Our ablation study further proved the tree structure's significant role in shaping the model's decision making.
- We provide extensive experimental results that validate the effectiveness of TreeKV in both prefilling and generation stages. In langauge modeling task, TreeKV allows LLMs to generalize to sequences of at least 16k, attaining the lowest perplexity among all baseline models. On the Longbench benchmark, TreeKV consistently surpasses other methods across all cache sizes, using only 6% of budget at optimal efficiency.

2 Related Work

In recent years, the field of natural language processing has seen a surge in research addressing the challenges associated with KV cache eviction and memory compression in transformer-based architectures.

StreamingLLM [Xiao *et al.*, 2023] and LM-Infinite [Han *et al.*, 2024] identify that attention scores mainly concentrate on initial and recent tokens in the KV cache, leading their methods to retain only those tokens to reduce cache size. However, this could result in the loss of significant information by discarding tokens between the initial and sliding window. H2O [Zhang *et al.*, 2024c] introduces a cache eviction method that greedily selects tokens based on importance

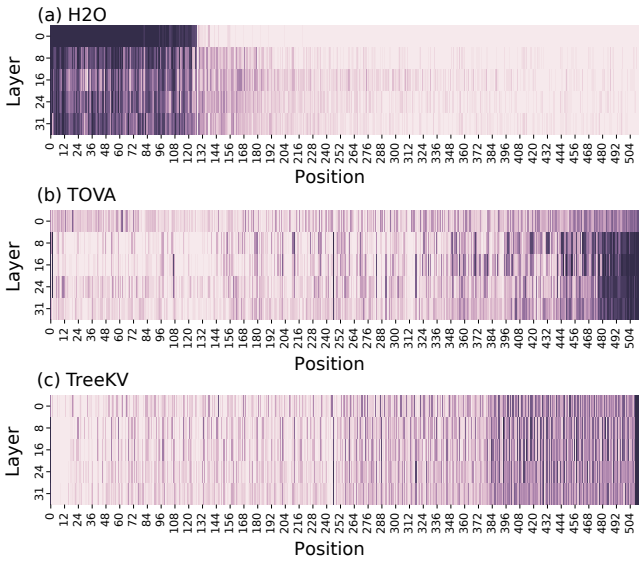

Figure 1: Distribution map of tokens selected by H2O, TOVA and TreeKV, using a cache size of 128 on a 512-length sequence from PG19. We quantify the values of selected token positions as 1 (dark color) and 0 (light color) otherwise, then average across the 32 heads to reduce noise. Note that sinks and recent tokens are not particularly kept.

scores derived from cumulative attention weights during generation. Scissorhands uses a similar strategy but binarizes the scores. TOVA [Oren *et al.*, 2024] selects tokens using the last token's attention scores. These methods, however, often overlook the structure of key information distribution by naively evicting tokens across the entire sequence.

Figure 1 (a) and (b) display the token distribution map for H2O and TOVA over a 512-length sequence, both using a cache size of 128. We quantify the values of selected token positions as 1 and 0 otherwise, averaging across the 32 heads to minimize noise. A notable pattern emerges: H2O and TOVA show significant regional biases due to their disregard for the token eviction scope. which may potentially lead to oversimplified interpretations of sequences and impair the models' ability to grasp nuanced interactions, thereby reducing their effectiveness in tasks requiring holistic understanding.

The aforementioned methods aim to reduce the KV cache produced during long text generation, while other studies concentrate on compressing the KV cache of long context prompts during prefilling. SnapKV [Li *et al.*, 2024] retains important tokens based on attention weights alongside their neighboring tokens for additional details. PyramidKV [Zhang *et al.*, 2024b] and PyramidInfer [Yang *et al.*, 2024] found that attention is widespread in lower layers and progressively concentrates in higher layers. As a result, they adjust the KV cache size across layers and select tokens in a funnel-like manner. Notably, PyramidKV and PyramidInfer are orthognal to our method. Although these eviction policies efficiently reduce the size of KV cache, their myopic view of certain regions neglects the comprehensive contextual importance within the broader narrative.

Another line of research is dedicated to structure-guided long context processing. [Zhao *et al.*, 2022] developed a hierarchy for selecting important tokens across layer, [He *et al.*, 2024] created a multi-scale tree to efficiently capture the long context dependencies. However, these studies are limited to encoder-based models and cannot be directly applied to pre-trained LLMs without additional tuning, restricting their applicability to generative models.

3 Preliminary

In this section, we present our preliminary observations about the input-output dependence in a standard attention layer, which motivates our design of TreeKV.

Previous work has used Fourier transform to analyze the hidden states of language models [Scribano *et al.*, 2023; He *et al.*, 2023], but those analyses ignore the distributions of the frequency components at different locations of the sequence, due to the lack of locality of the Fourier basis functions. Different from previous work, our analysis uses multi-level discrete wavelet decomposition, which is able to capture the local frequency information at different locations. We first introduce the background knowledge including KV caching and multi-level discrete wavelet decomposition, before presenting our observations.

3.1 KV Caching

Before diving into our method, it's essential to grasp how KV caches are maintained during LLM's inference. The following example illustrates the auto-regressive decoding process in the generation phase. As KV cache management methods operate consistently across different batches or heads, we have omitted these two dimensions for simplicity. We use $\mathbf{W}_q \in \mathbb{R}^{d\times d}, \mathbf{W}_k \in \mathbb{R}^{d\times d}, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ to denote the weights of the attention modules for a layer, where d is the hidden dimension of the model. We use $\mathbf{K}$, $\mathbf{V}$ to denote KV cache and superscript t to denote the generation step.

For a new input $\mathbf{x}^{(t)} \in \mathbb{R}^{1\times d}$ at generation step t, the attention module first transforms it into a set of query, key, and value:

$$\mathbf{q}^{(t)} = \mathbf{x}^{(t)}\mathbf{W}_Q, \mathbf{k}^{(t)} = \mathbf{x}^{(t)}\mathbf{W}_K, \mathbf{v}^{(t)} = \mathbf{x}^{(t)}\mathbf{W}_V.$$

The key and value caches grow linearly by appending the new key and value as:

$$\mathbf{K}^{(t)} = \begin{pmatrix}\mathbf{K}^{(t-1)}\\ \mathbf{k}^{(t)}\end{pmatrix}, \quad \mathbf{V}^{(t)} = \begin{pmatrix}\mathbf{V}^{(t-1)}\\ \mathbf{v}^{(t)}\end{pmatrix}.$$

The attention scores $\mathbf{a}^{(t)}$ and the output of the attention $\mathbf{o}^{(t)}$ are computed as follows:

$$\mathbf{a}^{(t)} = \mathrm{SoftMax}\left(\frac{\mathbf{q}^{(t)}{\mathbf{K}^{(t)}}^\top}{\sqrt{d}}\right), \quad \mathbf{o}^{(t)} = \mathbf{a}^{(t)} \cdot \mathbf{V}^{(t)}.$$

KV caching reduces redundant computations in LLMs, but as the cache lengthens, the computational cost of attention calculations rises quadratically. This necessitates innovative strategies to optimize the KV cache memory footprint without compromising LLMs performance.

3.2 Multi-Level Discrete Wavelet Decomposition

The multi-level discrete wavelet decomposition is a signal processing technique that allows for the representation of a signal at multiple resolution levels using wavelets. Unlike sine and cosine functions used in Fourier transforms, wavelets are localized and can represent both frequency and time (or location in our case) information. In single-level discrete wavelet decomposition, a discrete signal $\mathbf{s}[n]$ is filtered by a pair of low-pass filter $\mathbf{g}[n]$ and high-pass filter $\mathbf{h}[n]$, followed by down-sampling:

$$\mathbf{A}_1[n] = \sum_{k=-\infty}^{\infty} \mathbf{s}[k]\mathbf{g}[2n-k],$$
$$\mathbf{D}_1[n] = \sum_{k=-\infty}^{\infty} \mathbf{s}[k]\mathbf{h}[2n-k].$$

The approximation coefficients $\mathbf{A}_1[n]$ represent the low-frequency coefficients of the signal. They capture the main features and general trends of the signal. The detail coefficients $\mathbf{D}_1[n]$ correspond to the high-frequency coefficients of the signal, which capture the finer details. We use the Haar wavelet in the following analysis, for which the low-pass filter $\mathbf{g}[n]$ and high-pass filter $\mathbf{h}[n]$ are

$$\mathbf{g}[i] = \begin{cases}\sqrt{2}/2 & i = 0, 1\\ 0 & \text{otherwise}\end{cases}, \mathbf{h}[i] = \begin{cases}-\sqrt{2}/2 & i = 0\\ \sqrt{2}/2 & i = 1\\ 0 & \text{otherwise}\end{cases}.$$

The single-level reconstruction under the Haar wavelet is

$$R(\mathbf{A}_1, \mathbf{D}_1)[n] = \begin{cases}\frac{\sqrt{2}}{2}\left(\mathbf{A}_1\left[\frac{n+1}{2}\right] + \mathbf{D}_1\left[\frac{n+1}{2}\right]\right) & \text{odd } n\\ \frac{\sqrt{2}}{2}\left(\mathbf{A}_1\left[\frac{n}{2}\right] - \mathbf{D}_1\left[\frac{n}{2}\right]\right) & \text{even } n\end{cases}.$$

In multi-level discrete wavelet decomposition, the approximation coefficients are further decomposed through repeated single-level decomposition. Specifically, at level $l > 1$, the approximation coefficients $\mathbf{A}_l[n]$ and the detail coefficients $\mathbf{D}_l[n]$ are obtained by filtering and down-sampling $\mathbf{A}_{l-1}[n]$. The coefficients of an L-level discrete wavelet decomposition can be organized into a list $[\mathbf{A}_L, \mathbf{D}_L, \mathbf{D}_{L-1}, \cdots, \mathbf{D}_1]$, where $\mathbf{A}_L$ gives the lowest frequency coefficient of the signal in the decomposition, and $\mathbf{D}_L, \cdots, \mathbf{D}_1$ give the coefficients at progressively higher frequencies. Applying the single-level reconstruction from high to low levels recovers the original signal.

3.3 Observation

In full attention, the output at a given position is generated by a weighted sum of the values at all positions up to the given one, where the weights are attention weights. Thus, at the generation step t, we view the product of the attention scores and the values up to the position t as signals, which can be formulated as

$$\mathbf{s} = {\mathbf{a}^{(t)}}^\top \circ \mathbf{V}^{(t)} = \begin{pmatrix}\mathbf{a}^{(t)}(1)\mathbf{v}^{(t)}(1)\\ \vdots\\ \mathbf{a}^{(t)}(t)\mathbf{v}^{(t)}(t)\end{pmatrix}.$$

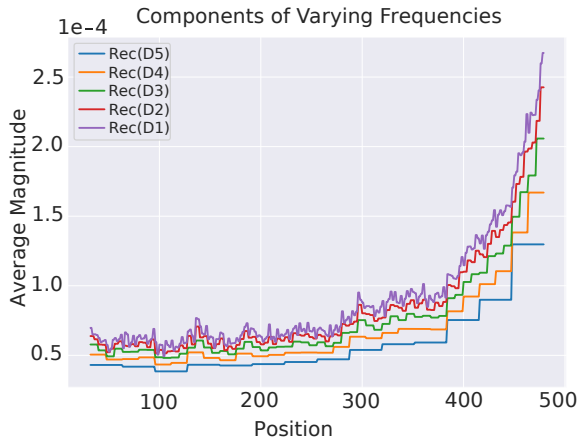

Figure 2: The average magnitude of gathered information's high-frequency components given by wavelet decomposition. We illustrate the top 5 frequency components at the 512th generation step, excluding the first 32 tokens and the last 32 tokens.

$\mathbf{a}^{(t)}$ and $\mathbf{V}^{(t)}$ are defined in Section 3.1, we analyze it using multi-level discrete wavelet decomposition defined in Section 3.2. More precisely, we first apply the multi-level wavelet decomposition along the sequence length dimension to obtain the frequency coefficients at various levels. Next, we reconstruct the signal $\text{Rec}(\mathbf{D}_L)$, which represents frequency components with only the detail coefficients $\mathbf{D}_L$,

$$\text{Rec}(\mathbf{D}_L) = R(R(\cdots R(\mathbf{0}, \mathbf{D}_L)\cdots, \mathbf{0}), \mathbf{0}).$$

Then we average the magnitude of each frequency component across all layers, heads, and samples. Figure 2 shows the average magnitude of each frequency component at different positions for a 5-level wavelet decomposition with the Haar wavelet at the 512th generation step across 2000 samples. Since previous work has highlighted the significance of maintaining sinks and recent tokens [Xiao *et al.*, 2023; Han *et al.*, 2024; Zhang *et al.*, 2024c; Liu *et al.*, 2024; Oren *et al.*, 2024], we exclude the first 32 and the last 32 tokens to concentrate on the middle segment of the sequence in the figure.

We observe that as the positions get closer to the end, all frequency components gradually increase, particularly at higher frequencies. This suggests that the information contributed by a token not only increases, but also tends to diverges more from its neighboring tokens as the position approaches the end of sequence, indicating a smooth transition with increasing complexity and variability from distant to nearby context. This insight inspired our method, resulting in a structure designed to be sparse on the left and dense on the right.

4 TreeKV

This section introduces TreeKV, which organizes keys and values in a tree-like hierarchy to enhance smooth cache compression by leveraging temporal locality. Note that, the structure of TreeKV is inspired by our spectral analysis in Sec 3.3. Unlike most cache compression methods that focus on either the generation or prefilling phases, TreeKV is applicable to both. We will first outline the compression strategy for the decoding stage, then elaborate its application in the prefilling stage.

4.1 Decoding Stage

Overall Idea During decoding, the KV pair of new tokens are added sequentially to the cache. Once the cache is at maximum capacity c, a tree-based approach strategically evicts KV pair of less important token within a specific eviction scope as generation progresses across layer and head. The parameter c indicates the desired number of key-value pairs to retain in the cache, referred to as cache size. This eviction scope cycles through the cache from distant to nearby context, ensuring a smooth distribution of retained tokens that is sparse on the left and dense on the right. Figure 3 provides an illustration of the cache compression process of TreeKV with a maximum cache size set to 4, each subplot demonstrates the eviction process and the state of the cache at the end of step 4, 5, 6 and 17, with steps 1-4 representing the cache filling process. The detailed compression procedure is meticulously outlined in Algorithm 1.

Importance Score Each token is associated with its averaged attention weight $\overline{\mathbf{S}}$ (calculated in line 6, 7, 8 and 10 of Algorithm 1). The averaged attention weight $\overline{\mathbf{S}}$ reflects the importance score of token in the following generation steps, which serves to prioritize which pairs to retain in the cache. We tested various importance criteria in our experiment, including accumulated and normalized attention weights, and found that averaged attention weight delivers the best performance.

Eviction Scope When the maximum cache size c is reached, TreeKV evicts an old KV pair within an "eviction scope" to maintain the fixed size. The scaler idx defines this eviction scope, idx cycles through the cache from distant to nearby contexts, building a tree structure that is sparse on the left and dense on the right. Specifically, the eviction scope is $\{\text{idx}, \text{idx}+1\}$, indicating that we will remove either the idx-th or the $(\text{idx}+1)$-th token in the cache with the lower importance score. For example, in Figure 3, when the eviction process starts at step 5 with $\text{idx} = 1$, TreeKV evicts the first token in the scope $(1, 2)$ with the lower importance score, resulting in the removal of "The". The cache then holds: "sunset", "slowly", "over" and "the". The scaler idx cycles through 1, 2, 3, and 4, shifting the eviction scope from left to right to remove older tokens first while prioritizing the recent ones. At step 17, upon the arrival of the token "evening," the idx cycles back to 1, token "vibrant" from the ("subset", "vibrant") is evicted. In the figure, we keep the complete eviction process to make the tree structure visible. Note that our algorithm not just construct trees layer by layer but also facilitates the merging of tokens across different tree levels, maintaining a coherent representation of the input sequence.

Position Encoding Following StreamingLLM [Xiao *et al.*, 2023], the positional encodings are re-assigned after KV eviction. For example, if the current cache contains tokens $\{0, 1, 2, 3, 7, 8, 9\}$, when decoding the 10th token, the as-

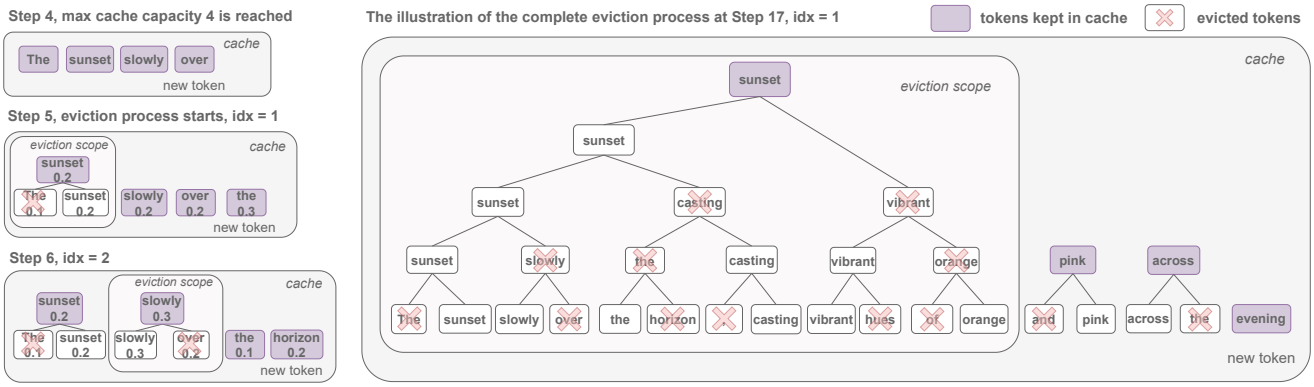

Figure 3: Applying TreeKV cache compression method on a 17-length sequence with cache size 4. Each subplot demonstrates the eviction process and state of the cache at the end of step 4, 5, 6 and 17, respectively, while steps 1-4, showing the cache filling process, are omitted. Variable idx and selection scope are explained in Section 4.1.

Algorithm 1 Compression by TreeKV

Input: Inputs $x^{(1)}, \cdots, x^{(T)} \in \mathbb{R}^{1\times d}$, Cache size c.

Initiate $\mathbf{S}, \mathbf{C}, \mathbf{K}^{(0)}, \mathbf{V}^{(0)}$ as empty tensors, $\mathrm{idx} = 1$.
for i from 1 to T **do**
  $\mathbf{q}^{(i)} = \mathbf{x}^{(i)}\mathbf{W}_Q, \mathbf{k}^{(i)} = \mathbf{x}^{(i)}\mathbf{W}_K, \mathbf{v}^{(i)} = \mathbf{x}^{(i)}\mathbf{W}_V$.
  $\mathbf{K}^{(i)} = \mathbf{K}^{(i-1)} \cup \{\mathbf{k}^{(i)}\}$.
  $\mathbf{V}^{(i)} = \mathbf{V}^{(i-1)} \cup \{\mathbf{v}^{(i)}\}$.
  $\mathbf{a}^{(i)} = \mathrm{SoftMax}\left(\frac{\mathbf{q}^{(i)}\mathbf{K}^{(i)\top}}{\sqrt{d}}\right)$.
  $\mathbf{C} = (\mathbf{C} \cup \{0\}) + \mathbf{1}$.
  $\mathbf{S} = (\mathbf{S} \cup \{0\}) + \mathbf{a}^{(i)}$.
  if $\mathrm{length}(\mathbf{K}^{(i)}) > c$ **then**
    $\overline{\mathbf{S}} = \mathbf{S}/\mathbf{C}$. ▷ Importance Scores
    if $\overline{\mathbf{S}}_{\mathrm{idx}} > \overline{\mathbf{S}}_{\mathrm{idx}+1}$ **then**
      Evict $(\mathrm{idx}+1)$-th elements in $\mathbf{K}^{(i)}$, $\mathbf{V}^{(i)}$, $\mathbf{C}$, $\mathbf{S}$.
    else
      Evict idx-th elements instead.
    end if
    $\mathrm{idx} = (\mathrm{idx}+1) \mod c + 1$.
  end if
end for

signed positions are $\{0, 1, 2, 3, 4, 5, 6, 7\}$ instead of the original positions $\{0, 1, 2, 3, 7, 8, 9, 10\}$.

Conclusion of Decoding Stage Figure 1 (c) shows that TreeKV provides a more balanced distributed token map compared to the previous two methods, with a smooth transition from distant coarse-grain to recent fine-grain tokens. Moreover, TreeKV updates the hierarchical structure based on a continuous scoring mechanism, allowing it to adapt to changes in data dynamically. This adaptability ensures that the cache remains relevant to the current context, enabling the system to consistently provide maximum benefit through efficient storage.

4.2 Prefilling Stage

Overall Idea We previously discussed how TreeKV handles eviction during decoding, and this approach also applies to prefilling stage. In this stage, the same principles of importance scoring, eviction scoping and position encoding are employed, but focusing on blocks instead of tokens. To enable the model to operate efficiently, all blocks are processed simultaneously, with the importance scores and eviction scopes computed concurrently. This pre-computation allows the model to swiftly select the most important block within each eviction scope.

Block-level Eviction Most context compression techniques rely on token-level selection [Zhang *et al.*, 2024c; Wang *et al.*, 2024], overlooking the fact that either critical or irrelevant information is often spatially clustered [Li *et al.*, 2024]. Selecting individual tokens can compromise contextual integrity and computational speed. Therefore, we adopt a block-level eviction policy for prompt compression tasks, applying our algorithm on blocks instead of tokens. TreeKV first divides the prompt into multiple blocks of size b, with each block representing one token in Fig 3. Following [Li *et al.*, 2024], we use the last block of the prompt as the observation window to *query* the entire sequence. We obtain the importance score of each block by calculating the averaged importance scores for the b tokens within that block.

Conclusion of Prefilling Stage In summary, the prefilling stage of TreeKV mirrors the strategies employed during the decoding stage by focusing on blocks rather than tokens. Simultaneous block computing allows TreeKV to efficiently managing KV pairs while maintaining integrity and contextual relevance.

5 Experiments

Our evaluation of TreeKV demonstrates its superiority over existing methods in both prefilling and decoding phases. We first assess its performance on the language modeling task with PG19 [Rae *et al.*, 2019] and OpenWebText2 [Gao *et al.*, 2020] for long text decoding. Additionally, we demonstrate that TreeKV can reliably handle texts exceeding 10 million tokens, achieving the lowest perplexity among all baseline models. Finally, we assess TreeKV's prompt compression

	PG19		
Context Length	4k	8k	16k
Budget Ratio	25.0%	12.5%	6.3%
Full Attn	6.84	$> 10^3$	OOM
StreamingLLM	7.19	7.19	7.19
H2O	7.06	7.08	7.25
TOVA	**7.00**	7.06	7.15
TreeKV (ours)	7.02	**6.88**	**6.91**

Table 1: Sliding window perplexity of different context window extension methods using Llama-2-7B model on PG19. The cache size of all the efficient methods is set to 1024.

	OpenWebText2		
Context Length	4k	8k	16k
Budget Ratio	25.0%	12.5%	6.3%
Full Attn	5.44	$> 10^3$	OOM
StreamingLLM	5.78	5.62	5.31
H2O	**5.60**	5.48	5.25
TOVA	5.62	5.50	5.24
TreeKV (ours)	**5.60**	**5.45**	**5.18**

Table 2: Sliding window perplexity of different context window extension methods using Llama-2-7B model on OpenWebText2. The cache size is set to 1024 for all the experiments.

ability during prefilling using LlaMa-3.2-1B-Instruct on the Longbench [Bai *et al.*, 2023] benchmark.

5.1 Setup

For language modeling task, we use Llama-2-7B [Touvron *et al.*, 2023] pre-trained with 4K context length as base model considering its popularity and outstanding performance. We evaluate perplexity using a sliding window approach with a stride of 2048 for PG19 and 1024 for OpenWebText2 respectively. For language understanding tasks on Longbench, we truncate the inputs to 32k in the same manner as SnapKV [Li *et al.*, 2024]. We employ Llama-3.2-1B-Instruct as our base models. All the experiments utilize bf16 precision on Nvidia RTX4090 GPUs.

5.2 Language Modeling

The language modeling task assesses LLMs' ability to predict the next token from the preceding context. We report perplexity on two datasets: PG19 test set [Rae *et al.*, 2019] and OpenWebText2. These two datasets are both commonly used datasets for evaluating long-range language models. The PG19 test set consists of 100 full-length books, each averaging 113k tokens. The OpenWebText2 dataset is derived from the Pile dataset [Gao *et al.*, 2020], from which we randomly selected 100 samples from the test set, averaging 18k tokens each, for evaluation.

We compare TreeKV with 4 baseline methods: efficient decoding policies like StreamingLLM [Xiao *et al.*, 2023], H2O [Zhang *et al.*, 2024c], and TOVA [Oren *et al.*, 2024], as well as a full attention method that caches all keys and values. We assessed these five methods using sequences of context lengths 4k, 8k, and 16k, while keeping a cache size of only 1k. The 1k cache kept from each method consists of three components: 4 initial tokens, 508 recent tokens, and 512 method-selected tokens.

Figure 4: We create a 10M length sequence by concatenating the first 50 books from the PG19 test set and applying 4 efficient decoding methods: TOVA, H2O, StreamingLLM, and TreeKV on it. We then plot the negative log-likelihoods against sequence length.

TreeKV surpasses all baselines when context length exceeds the pre-trained limit of the LLM and also performs competitively within that limit. The results are shown in Table 1 and 2. Although, TreeKV is 0.02 behind TOVA for 4k-length contexts in PG19 but soon turn the tide with longer contexts. On OpenWebText2, we achieve the best perplexity across all context lengths. Our method shows the most significant improvement across all the longest length, surpassing the second-best TOVA by 3.6% and 1.1% on the PG19 and OpenWebText2 datasets respectively, with 16k context length and up to 16x reduction in KV cache.

Following [Xiao *et al.*, 2023; Han *et al.*, 2024; Zhang *et al.*, 2024c], we also thoroughly examine TreeKV to test if a LLM pre-trained with a 4k context window can effectively perform language modeling task on exceptionally extended text. We concatenate all the 100 books in PG19 test set to create a 10M length example and evaluate its generation capability using Llama-2-7B with StreamingLLM, H2O, TOVA, and TreeKV. Figure 4 shows the NLL curves for input lengths from 0.1M to 10M. The curves reveal that TOVA and H2O suffer performance degradation with longer sequences, leading to significantly worse NLL than StreamingLLM and TreeKV. In contrast, TreeKV consistently outperforms all other baseline methods including StreamingLLM, demonstrating superior capability with longer inputs.

5.3 Long Context Understanding

Unlike most KV cache compression methods that focus on either the generation or prefill stages, TreeKV is applicable to both. We conduct experiments on long context understanding tasks using the Longbench [Bai *et al.*, 2023] benchmark. Longbench is a multi-task benchmark that includes a wide range of long-context tasks to assess model capabilities in handling extended textual input. Longbench consists

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Cache Size = Full																	
FullKV	20.78	24.05	42.00	36.74	29.30	21.29	28.77	22.06	25.41	64.00	80.05	39.58	4.50	4.09	39.11	43.21	32.86
Cache Size = 2048																	
H2O	19.34	21.59	40.81	35.43	27.57	18.71	23.18	21.33	24.92	45.50	80.12	37.69	3.50	4.09	37.54	42.28	30.23
SnapKV	**20.82**	22.23	**42.34**	**38.12**	**29.17**	**21.16**	**23.96**	**22.15**	**24.97**	**63.00**	80.90	**38.40**	3.50	**4.18**	30.60	38.47	31.50
TreeKV	20.40	**23.20**	41.89	35.62	27.49	18.93	23.34	21.19	24.70	61.00	**81.00**	37.75	**4.50**	3.82	**39.11**	**43.28**	**31.70**
Cache Size = 8192																	
H2O	21.19	23.85	43.19	37.26	29.22	20.60	27.61	**22.18**	**25.53**	63.50	80.90	**39.68**	**3.50**	**4.50**	38.60	43.29	32.79
SnapKV	21.26	24.03	**43.40**	37.22	**28.82**	**22.20**	**28.18**	22.09	**25.53**	**64.00**	80.85	38.57	**3.50**	**4.50**	30.69	37.72	32.04
TreeKV	**21.48**	**24.47**	42.19	**37.62**	28.75	21.69	27.76	21.64	**25.53**	**64.00**	**81.01**	39.13	**3.50**	4.09	**38.65**	**43.29**	**32.80**

Table 3: Performance comparison of H2O, SnapKV and TreeKV on Longbench using LlaMa-3.2-1B as base models. Results are reported with cache sizes of 2048 and 8192. SnapKV results were obtained using their officially released code, while H2O was implemented by the authors. The best scores are highlighted in bold. Our model outperforms H2O and SnapKV on all the cache sizes. The results of Full attention models are shown at the top of the table.

of 16 tasks across 6 categories: single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. The average length across all the sub-datasets is around 11k.

We compare our method against three baseline approaches: H2O [Zhang *et al.*, 2024c], a method that also suitable for both prefilling and generation stages; SnapKV [Li *et al.*, 2024], a state-of-the-art method for long context understanding; and a full attention method that caches all keys and values.

Table 3 summarizes the performance metrics for all tasks and cache sizes in the Longbench benchmark. Overall, our model achieved an average improvement of 0.74 over H2O and 0.48 over SnapKV, demonstrating its superior capacity to retain and recall relevant information from extended text. TreeKV underperforms SnapKV in Multi-Doc QA and Summarization tasks at cache size 2k. Since TreeKV algorithmically prioritizes recent tokens when the cache size is limited, we hypothesize that limiting the tree height to balance early and recent token retention could mitigate this issue, which will explore in future. TreeKV did not surpass the full attention model, which leaves space for us to improve on with minimal resource requirements.

5.4 Ablation Study

An important question is: what is the key component of our approach — the tree structure or the attention weight-based token selection mechanism? To investigate this, we modified our method to consistently select the leftmost token within the eviction scope instead of the one with the highest attention weight. This change allowed us to isolate the effect of the tree structure itself, facilitating a focused examination of how token hierarchy affects model performance, independent of the variability introduced by weight differentiation.

In Figure 5, we present the log mean perplexity of the first book from PG19, which is 65k tokens long. We compare three methods: 1) H2O, which greedily selects tokens based on their cumulative attention weights. 2) TreeKV, which employs the average attention weights to guide evictions through tree structure; and 3) TreeKVSelectLeftToken, a variant of TreeKV that prioritizes the leftmost tokens without using attention scores. The results show that the consistent token selection strategy results in minor variations in perplexity, indicating the tree structure's significant role in shaping the model's decision-making. In conclusion, our investigation confirms that the tree structure is a crucial component of our approach.

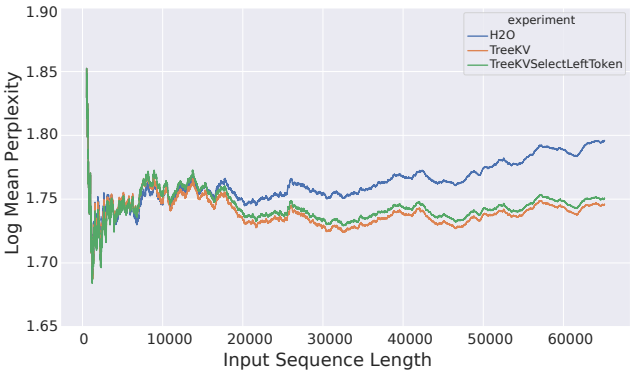

Figure 5: We plot the log mean perplexity of the first book from PG19, which has a length of 65k. In the TreeKVSelectLeftToken experiment, we consistently select the left token within the eviction scope instead of the one with the highest attention weight.

6 Conclusion

In this paper, we first explore the frequency representations of information collected during generation using multi-level wavelet decomposition, leading to the development of TreeKV, a training-free method that utilizes a tree structure for smooth cache compression. Our evaluation shows that TreeKV outperforms existing methods in both prefilling and generation setups.

Contribution Statement

Ziwei He and Jian Yuan have equal contribution, Bo Jiang is the corresponding author.

References

[Bai *et al.*, 2023] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

[Gao *et al.*, 2020] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[Han *et al.*, 2024] Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008, 2024.

[He *et al.*, 2023] Ziwei He, Meng Yang, Minwei Feng, Jingcheng Yin, Xinbing Wang, Jingwen Leng, and Zhouhan Lin. Fourier transformer: Fast long range modeling by removing sequence redundancy with fft operator. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

[He *et al.*, 2024] Ziwei He, Jian Yuan, Le Zhou, Jingwen Leng, and Bo Jiang. Fovea transformer: Efficient long-context modeling with structured fine-to-coarse attention. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 12261–12265. IEEE, 2024.

[Li *et al.*, 2024] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

[Liu *et al.*, 2024] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

[Oren *et al.*, 2024] Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

[Pope *et al.*, 2022] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. corr, abs/2211.05102 (2022), 2022.

[Rae *et al.*, 2019] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[Scribano *et al.*, 2023] Carmelo Scribano, Giorgia Franchini, Marco Prato, and Marko Bertogna. Dct-former: Efficient self-attention with discrete cosine transform. *Journal of Scientific Computing*, 94(3):67, 2023.

[Touvron *et al.*, 2023] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[Wang *et al.*, 2024] Shengnan Wang, Youhui Bai, Lin Zhang, Pingyi Zhou, Shixiong Zhao, Gong Zhang, Sen Wang, Renhai Chen, Hua Xu, and Hongwei Sun. Xl3m: A training-free framework for llm length extension based on segment-wise inference. *arXiv preprint arXiv:2405.17755*, 2024.

[Wei *et al.*, 2022] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[Xiao *et al.*, 2023] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[Yang *et al.*, 2024] Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*, 2024.

[Yuan *et al.*, 2022] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. Wordcraft: story writing with large language models. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*, pages 841–852, 2022.

[Zhang *et al.*, 2024a] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.

[Zhang *et al.*, 2024b] Yichi Zhang, Bofei Gao, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, Wen Xiao, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

[Zhang *et al.*, 2024c] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[Zhao *et al.*, 2022] Jing Zhao, Yifan Wang, Junwei Bao, Youzheng Wu, and Xiaodong He. Fine-and coarse-granularity hybrid self-attention for efficient bert. *arXiv preprint arXiv:2203.09055*, 2022.