Algorithm 1 ChunkKV

Input: $\mathbf{Q} \in \mathbb{R}^{T_q \times d}$, $\mathbf{K} \in \mathbb{R}^{T_k \times d}$, $\mathbf{V} \in \mathbb{R}^{T_v \times d}$, observe window size w, chunk size c, compressed KV cache max length L_{max}
Output: Compressed KV cache $\mathbf{K}'$, $\mathbf{V}'$
Observe Window Calculation:
$\mathbf{A} \leftarrow \mathbf{Q}_{T_q-w:T_q}\mathbf{K}^T$ {Attention scores for the observe window}
$C \leftarrow \lceil \frac{T_k}{c} \rceil$ {Calculate the number of chunks}
Chunk Attention Score Calculation:
for $i = 1$ to C **do**
  $\mathbf{A}_i \leftarrow \sum_{j=(i-1)c+1}^{ic} \mathbf{A}_{:,j}$ {Sum of observation scores for each chunk}
end for
Top-K Chunk Selection:
$k \leftarrow \lfloor \frac{L_{\text{max}}}{c} \rfloor$
Top_K_Indices $\leftarrow$ indices of Top-k chunks based on $\mathbf{A}_i$
Compression:
$\mathbf{K}', \mathbf{V}' \leftarrow$ index_select($\mathbf{K}, \mathbf{V}$, *Top_K_Indices*)
Concatenation:
$\mathbf{K}' \leftarrow \text{concat}(\mathbf{K}'_{0:L_{\text{max}}-w}, \mathbf{K}_{T_k-w:T_k})$
$\mathbf{V}' \leftarrow \text{concat}(\mathbf{V}'_{0:L_{\text{max}}-w}, \mathbf{V}_{T_v-w:T_v})$
$\mathbf{K}'$, $\mathbf{V}'$

is the observe window, $\mathbf{K}$ is the Key matrix and the window size w is usually set to $\{4, 8, 16, 32\}$. Next, the number of chunks C is calculated as $C = \lceil \frac{T_k}{c} \rceil$, where T_k is the length of the Key matrix and c is the chunk size. The observation scores for each chunk are then computed as $\mathbf{A}_i = \sum_{j=(i-1)c+1}^{ic} \mathbf{A}_{:,j}$ for $i = 1, 2, \ldots, C$. Referring to previous works (Zhang et al., 2023; Li et al., 2024; Yang et al., 2024b; Cai et al., 2024), we still use the top-k algorithm as ChunkKV's sampling policy. For the top-k chunk selection, the top-k chunks are selected based on their observation scores, where $k = \lfloor \frac{L_{\text{max}}}{c} \rfloor$, and L_{max} is the maximum length of the compressed KV cache. The size of the last chunk will equal $\min(c, L_{\text{max}} - (k-1) \times c)$. The indices of the top-k chunks will keep the original sequence order. In the compression step, the key and value matrices are only retained based on the selected indices, resulting in the compressed KV cache. Finally, the observe window of the original KV cache will be concatenated to the compressed KV cache by replacing the last w tokens to keep important information. The compressed KV cache is then used for subsequent attention computations.

3.3. Layer-Wise Index Reuse

Furthermore, we investigated the preserved KV cache indices by ChunkKV and found that they exhibit higher similarity compared to previous methods. Figure 2 shows the layer-wise similarity heatmaps of SnapKV and ChunkKV. Each cell represents the similarity between the preserved

Algorithm 2 Layer-wise Index Reuse for ChunkKV

Input: Number of layers in LLMs N_{layers}, number of reuse layers N_{reuse}
Initialize: Dictionary to store indices $\mathcal{I}_{\text{reuse}} = \{\}$
for $l = 0$ to $(N_{\text{layers}} - 1)$ **do**
  if $l \mod N_{\text{reuse}} == 0$ **then**
    $\mathbf{K}'_l, \mathbf{V}'_l, \mathcal{I}_l \leftarrow \text{ChunkKV}(\mathbf{K}_l, \mathbf{V}_l)$
    $\mathcal{I}_{\text{reuse}}[l] \leftarrow \mathcal{I}_l$
  else
    $\mathcal{I}_l \leftarrow \mathcal{I}_{\text{reuse}}[\lfloor \frac{l}{N_{\text{reuse}}} \rfloor \times N_{\text{reuse}}]$
  end if
  $\mathbf{K}'_l \leftarrow \text{index\_select}(\mathbf{K}_l, \mathcal{I}_l)$
  $\mathbf{V}'_l \leftarrow \text{index\_select}(\mathbf{V}_l, \mathcal{I}_l)$
end for

KV cache indices of two layers, with deeper colors indicating higher similarity. The results demonstrate that the KV cache indices preserved by ChunkKV are more similar to those in neighboring layers. As shown in Table 2, ChunkKV consistently achieves a higher average Jaccard similarity between adjacent layers compared to SnapKV in different model architectures, indicating that the retained token index in ChunkKV is more similar to each other. For a more detailed visualization, please refer to Appendix B.1.2.

Table 2: Retained KV Cache Indices Similarity of Adjacent Layers for Different Models.

Method	H2O	SnapKV	ChunkKV
LLaMA-3-8B	25.31%	27.95%	**57.74%**
Qwen2-7B	14.91%	16.50%	**44.26%**
Mistral-7B	15.15%	15.78%	**52.16%**

Based on the above findings, we propose a training-free *layer-wise index reuse* method to further reduce the additional cost of the KV cache compression time, which reuses compressed token indices across multiple layers. This procedure is formally described in Algorithm 2. The ChunkKV compression process returns the compressed KV cache and their respective token indices, denoted as $\mathcal{I}_l$. For layer-wise index reuse, we define a grouping of layers such that all N_{reuse} layers share the same token indices for ChunkKV. Specifically, for a group of layers $\{l, l+1, \ldots, l+N_{\text{reuse}}-1\}$, we perform ChunkKV on the first layer l to obtain the token indices $\mathcal{I}_l$ and reuse $\mathcal{I}_l$ for the subsequent layers $l+1, l+2, \ldots, l+N_{\text{reuse}}-1$. The notation $\mathbf{K}_l[\mathcal{I}_l]$ and $\mathbf{V}_l[\mathcal{I}_l]$ indicates the selection of key and value caches based on the indices in $\mathcal{I}_l$. The efficiency analysis for layer-wise index reuse is provided in Appendix B.1.1.

Theoretical Understanding. We provide a theoretical understanding from the in-context learning (ICL) (Fang & Xie, 2022) to interpret why maintaining KV cache according to a