6. Algorithmic Description

In this section, we outline the operations within a single decoding step of an LVLM, containing the operations for compression, attention-aware decompression, and attention score computations, as pseudocode in Algorithm 1. For simplicity, we omit subscripts differentiating visual and textual tokens in algorithm statements, as the same procedure applies to both. Before compression is performed (i.e., when the compressed cache is empty, $T_{cc} = 0$), we have the uncompressed cache $\mathcal{C}_u = \{\mathbf{K}, \mathbf{V} \in \mathbb{R}^{T_{uc} \times HD}\}$, storing the key/value vectors for all T_{uc} tokens from the prefill phase. After computing the query $\boldsymbol{q}$, key $\boldsymbol{k}$, and value $\boldsymbol{v}$ vectors for the current input (line 3), these key/vectors are appended to the uncompressed cache (line 4). If compression had already been performed ($T_{cc} > 0$), tokens are decompressed using partial ranks derived based on the attention score statistics (lines 5-6). These decompressed tokens are then merged with the uncompressed cache (lines 7-10). The attention scores (line 11) and output (line 12) are computed, and the attention score statistics are updated (line 13). At any decoding step, if the uncompressed cache length (T_{uc}) exceeds the compression period T_p, compression is triggered (lines 14-17) to generate the compressed cache and decompression matrices. Once compression is done, the uncompressed cache is emptied and begins collecting new tokens in next decoding steps until it reaches T_p again, when the compression is reapplied. Finally, the cached token statistics $\mathcal{I}$, the output $\mathbf{O}$, the compressed cache $\mathcal{C}_c$, and the updated uncompressed cache $\mathcal{C}_u$ are returned for use in future decoding steps.

Algorithm 1 `Attention Block Operations in Decoding Step-`t

1: **Inputs:** input activation $\mathbf{H} \in \mathbb{R}^{T_q \times HD}$, uncompressed cache for visual ($* = v$) and textual ($* = t$) tokens $\mathcal{C}_{u*} = \{\mathbf{K}_*, \mathbf{V}_* \in \mathbb{R}^{T_{uc*} \times HD}\}$, compressed cache and decompression matrices $\mathcal{C}_{c*} = \{\overline{\mathbf{K}}_* \in \mathbb{R}^{T_{cc*} \times R_{k*}}, \overline{\mathbf{V}}_* \in \mathbb{R}^{T_{cc*} \times R_{v*}}, \mathbf{D}_{k*} \in \mathbb{R}^{R_{k*} \times HD}, \mathbf{D}_v \in \mathbb{R}^{R_{v*} \times HD}\}$, cached token importance scores $\mathcal{I}$, attention block weights $\boldsymbol{\theta} = \{\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o \in \mathbb{R}^{HD \times HD}\}$
2: **Parameters:** T_p (compression period), F (number of decompression groups), R_{k*}, R_{v*} (compression rank for keys and values)
3: Compute new query, key, value vectors: $\mathbf{Q}', \mathbf{K}', \mathbf{V}' \leftarrow \mathbf{H}\mathbf{W}_\dagger$ for $\dagger \in \{q, k, v\}$
4: Append new key, value vectors to corresponding uncompressed cache: $\mathbf{K} \leftarrow \mathbf{K} \oplus \mathbf{K}'$, $\mathbf{V}_* \leftarrow \mathbf{V} \oplus \mathbf{V}'$
5: **if** $T_{cc} > 0$ **then**
6: Determine the tokens $\boldsymbol{m}_f$ for each group $f \in \{1, ..., F\}$ based on $\mathcal{I}$
7: Retrieve cache: $\tilde{\mathbf{K}} \leftarrow \bigoplus_{f=1}^{F} \overline{\mathbf{K}}[\boldsymbol{m}_f, : R_k^{(f)}]\mathbf{D}_k[: R_k^{(f)}] \oplus \mathbf{K}$; $\tilde{\mathbf{V}} \leftarrow \bigoplus_{f=1}^{F} \overline{\mathbf{V}}[\boldsymbol{m}_f, : R_v^{(f)}]\mathbf{D}_v[: R_v^{(f)}] \oplus \mathbf{V}$
8: **else**
9: Retrieve cache: $\tilde{\mathbf{K}} \leftarrow \mathbf{K}$; $\tilde{\mathbf{V}} \leftarrow \mathbf{V}$
10: **end if**
11: Compute attention scores: $\mathbf{A} \leftarrow \texttt{softmax}(\frac{\mathbf{Q}\tilde{\mathbf{K}}^T}{\sqrt{D}})$
12: Compute output: $\mathbf{O} \leftarrow (\mathbf{A}\tilde{\mathbf{V}})\mathbf{W}_o$
13: Update statistics $\mathcal{I}$ based on $\mathbf{A}$ using Eq. 1.
14: **if** $T_{uc} \geq T_p$ **then**
15: Compress cache: $\overline{\mathbf{K}}, \mathbf{D}_k \leftarrow \texttt{svd}(\tilde{\mathbf{K}}, R_k)$; $\overline{\mathbf{V}}, \mathbf{D}_v \leftarrow \texttt{svd}(\tilde{\mathbf{V}}, R_v)$
16: Delete $\mathbf{K}$, $\mathbf{V}$
17: **end if**
18: **return** $\mathbf{O}, \mathcal{C}_u, \mathcal{C}_c, \mathcal{I}$

7. Experiment Setup Details

7.1. Datasets

We experiment on five datasets from different image/video tasks and domains. First, we consider three question-answering datasets based on image understanding: (i) A-OKVQA [25], a multiple-choice question dataset requiring a broad base of commonsense and world knowledge containing 1147 test images, (ii) OCR-VQA [23], a visual understanding dataset containing book cover images and related questions requiring optical character recognition capabilities containing 2382 test images, (iii) MMMU [31] a multi-discipline multi-modal reasoning benchmark containing questions from college exams containing 900 test images. Second, we consider two video understanding benchmarks: (i) MSVD-QA [28] and (ii) MSRVTT-QA [28] containing short video clips with multiple question-answer pairs for each video, with 1970 and 2990 test videos, respectively.

7.1.1. Implementation Details

For image QA tasks, we experiment with 7B and 13B variants of the LLaVA1.5 [20], which is an end-to-end trained large multi-modal model that connects a visual encoder backbone (CLIP [24]) with an LLM decoder (Vicuna [6]). LLaVA1.5 converts each input image to 24x24 representation, resulting with 576 tokens. We also consider QwenVL-Chat-7B for experiments with grouped query attention [2]. In video QA tasks, we consider VideoLLaVA-7B, which has a similar architecture with LLaVA, but processes visual input through image and video encoders [34], and pre-aligns before feeding into the LLM decoder. VideoLLaVA converts each frame to 14x14 representation, resulting in 256 tokens per frame. The code is available at https://github.com/gitdisl/AttentionPack.

For comparisons, we consider standard inference with full KV caching and two recent token eviction techniques developed for LLMs: H2O [32], Scissorhands [21], a token skipping technique for LVLMS: FastV [5] and a recent cache compression technique Minicache [19]. For our approach, during compression, we use randomized SVD [12] for key and value compression with $R_{kv} = R_{vv} \in \{32, 64\}$ for visual tokens in image QA, $R_{kv} = R_{vv} =\in \{64, 128\}$ in video QA and do not apply compression for textual

Method	Average Cache Size per Instance (MB)	Dataset						
		Question Answering - F1 (%)				*Summarization - ROUGE-L (%)*		avg.
		qasper	mfqa_en	hotpotqa	2wikimqa	gov_report	multi_news	
Full KV	555.36	34.68	42.66	43.66	35.65	29.27	26.68	35.43
H2O ($T = 1000$)	131.30	31.68	40.35	42.09	34.44	**27.64**	25.03	33.53
AttentionPack ($T = 1500$)	119.39	**33.86**	**41.00**	**42.47**	**34.58**	27.26	**25.67**	**34.14**
AttentionPack ($T = 1000$)	76.71	30.60	39.71	42.10	33.89	25.99	24.82	32.85

Table 6. Text QA and summarization results on LongBench datasets with LLaMA3.1-8B.

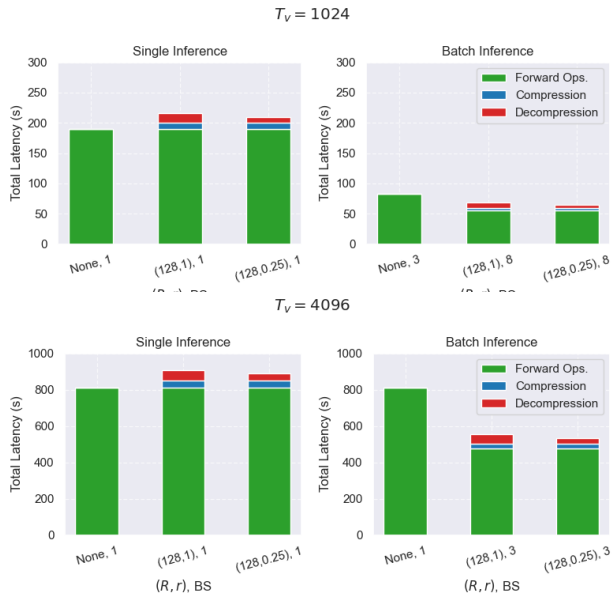

Figure 9. Total decoding latency for 100 queries across various context lengths (4/16 frames resulting in 1024/4096 visual tokens).

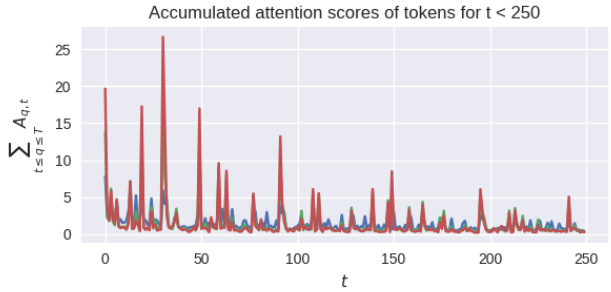

Figure 10. Accumulated attention scores of the first 250 tokens at the timestep $T = 500$ from a random conversational text sample from MTBench [33] (for the first three layers, which are represented with different colors).

tokens, unless stated otherwise. We perform attention-aware decompression over value cache with $F = 2, r_1 = 0.25, r_2 = 0.75, R_{vv} = R_{vv}^{(1)} = 4R_{vv}^{(1)}$. In initial experiments, $\alpha \in [0.05, 0.75]$ gave robust and stable results and we set $\alpha = 0.25$. We use half-precision except for the attention computation, which is at full-precision.

8. Additional Results and Analysis

8.1. Latency at Various Context Lengths

For latency analysis at various context lengths, we repeat the setting in Figure 5 for MSVD-QA using VideoLLaVA-7B with number of frames in $\{4, 16\}$ resulting in $\{1024, 4096\}$ visual tokens. We observe latency gains in batch inference are valid across various context lengths as shown in Figure 9.

8.2. Analysis of Textual Token Compression

Given the vision-language datasets considered have most of the context from visual inputs, e.g., the number of visual tokens in image QA is 576, and textual tokens is less than 100; treating visual tokens with a rank $R << min(T, HD)$ for high compression offers better cache size efficiency. In initial experiments, we observed compressing the visual and textual tokens together resulted in very poor performance (around 20% loss in A-OKVQA). In contrast, focusing on high compression ratio of visual tokens, though simple, shows big gains. Still, to analyze the text domain effectiveness, we conducted a small experiment with LLaMA3.1-8B [9] on LongBench [10] datasets and compared with the representative cache eviction technique [32]. Cache length, avg. cache size (MB) and F1 scores on question-answering, ROUGE-L on summarization tasks are reported. We observe that value matrices requires higher rank compared to key matrices in these datasets, which motivates us to only compress key matrices ($R_k = 64$). Our approach is effective, with around 40% cache reduction under same cache length, and +0.6% performance under same cache size. H2O is effective by evicting tokens with low accumulated attention scores as shown in Figure 10. But, AttentionPack enables longer context windows thanks to compression along hidden dimension and yields better performance under same constraints.

8.3. Impact of Multi-Head Compression

We carry out an ablation study on OCR-VQA to validate the analysis in Figure 2. For OCR-VQA, we find that achieving similar performance (52.25%) requires 1.48x larger cache (3.44x comp. ratio) when heads are compressed separately with $R_{kv} = 24$. Under the same compression ratio (5x), multi-head approach outperforms by 1.8%.