UNIFIED DAMAGE-BASED KV CACHE COMPRESSION FOR VISION-LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language Models (VLMs) cache the keys and values of context tokens for autoregressive decoding, but the KV cache grows linearly with context length and can be dominated by thousands of visual tokens. Existing eviction methods typically rank tokens by accumulated attention, measuring the sum of attention weights a token receives across queries rather than what is lost when it is removed, while separate heuristic sets per-layer budgets. We present DEKV (Damage-Equalized KV Eviction), a training-free framework that uses a single damage metric to determine both token importance and per-layer budgets. The metric is the exact attention-output perturbation caused by evicting one token, obtained in closed form from a softmax leave-one-out identity. DEKV sums these per-token damages as an additive surrogate for joint eviction, which we bound analytically and validate empirically. Normalizing by each layer's output scale makes damages comparable across layers, and a single global threshold, equivalent to water-filling over the per-layer damage curves, allocates the memory budget without an external allocation policy. DEKV scores only text-query attention rows and keeps the cache at exactly the budget after prefill and after every decoding step. Across six VLMs from the LLaVA, Qwen3-VL, and Phi-3.5 families on four VQA benchmarks, DEKV outperforms the strongest evaluated baseline, TGV-KV, in 22 of 24 settings at a 5% KV budget, retaining 86.6% of full-cache accuracy on average versus 80.0%, with gains of up to 27.5 points on ChartQA and 13.0 points on DocVQA for Phi-3.5-vision, while reducing KV cache memory by 20×.

1 INTRODUCTION

Vision-Language Models (VLMs) (Bai et al., 2025; Liu et al., 2023) extend the reasoning and generation capabilities of Large Language Models (LLMs) to the visual domain, enabling a wide range of multimodal applications. Like LLMs, VLMs use autoregressive generation with a Key-Value (KV) cache that stores the keys and values of context tokens to avoid recomputing prefix attention during decoding. However, the cache grows linearly with context length, and this cost is amplified in multimodal inputs, where high-resolution images and short videos can produce thousands of visual tokens. Consequently, KV cache memory and decode latency can become major bottlenecks.

Two main approaches have emerged for reducing this cost. *Token pruning* removes visual tokens before or during prefill (Chen et al., 2024; Yang et al., 2026; Zhang et al., 2025c), but pruned tokens cannot be recovered in later layers or decoding, and the resulting savings are largely limited to prefill. In contrast, *KV cache eviction* compresses the cache after prefill and can reduce decode-time costs. However, methods developed for language models (Xiao et al., 2024; Zhang et al., 2023; Li et al., 2024b) often degrade on VLMs because visual tokens are highly redundant while text tokens are more diverse. This problem is further amplified by weaker cross-modal attention, which can cause accumulated-attention methods to over-evict text KVs, a modality gap studied in (Liu et al., 2026). VLM-aware methods address this issue by using textual guidance to identify important visual tokens (Huang et al., 2025; Liu et al., 2026).

Despite this progress, current VLM eviction pipelines share two structural weaknesses. First, token importance is still overwhelmingly measured by *accumulated attention*, which is blind to *what a token contributes*. A heavily attended token whose value vector merely duplicates what the query already aggregates perturbs the model's computation negligibly when removed, while a moderately

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

attended token carrying unique content can be irreplaceable. Attention counts how much a token is read, not what is lost if it disappears. Second, the per-layer budget that determines how many KVs each layer may retain is set by mechanisms that are disconnected from the importance criterion. These mechanisms instead rely on fixed depth profiles (Cai et al., 2025), offline calibration (Wang et al., 2025), or attention statistics and entropy (Qin et al., 2025; Liu et al., 2026; Wan et al., 2025a). These proxies are correlated with, but not equal to, the quantity eviction actually destroys, so ranking and budgeting optimize different objectives and can work at cross purposes. Recent output-aware scoring functions (Goel et al., 2025; Gu et al., 2026; Feng et al., 2026) repair the first weakness for uni-modal LLM caches, but are deployed only as drop-in intra-layer *ranking* criteria under externally supplied budgets; no method makes ranking and cross-layer budgeting answer to a single output-perturbation measure of eviction damage in the multi-modal setting.

We introduce DEKV (Damage-Equalized KV eviction), a training-free framework that ties both token ranking and per-layer memory budgets to a single metric: the exact output perturbation caused by a token's eviction. Using a closed-form softmax leave-one-out identity, DEKV measures a token's value-novelty, effectively making highly-attended but redundant tokens cheap to drop. Because these damage scores are normalized as relative output perturbations, DEKV dynamically allocates memory across all layers through a global water-filling approach. This gives more budget to layers with unique content while computing everything efficiently from text-query rows alone. Our key contributions include:

- **Unified Damage Metric (UDM):** A single, closed-form criterion that prioritizes tokens based on output perturbation rather than relying on value-blind attention proxies.
- **Dynamic Water-Filling (DWF):** An adaptive allocation method that dynamically sets per-layer budgets and enforces strict memory limits, requiring a minimal prefill overhead of just $O(HN_tNd)$ per layer.
- **Balanced Distributed Inference:** By equalizing marginal damage rather than skewing retention toward early layers, DEKV creates a balanced memory footprint ideal for deployments sharded across multiple GPUs.
- **State-of-the-Art Extreme Compression:** At an aggressive 5% budget, DEKV beats the top baseline in 22 of 24 settings, retaining 86.6% of full-cache accuracy compared to the baseline's 80.0%. Notably, it achieves massive gains on Phi-3.5-vision (+27.5 points on ChartQA, +13.0 on DocVQA) while shrinking the cache by 20x.

2 Related Work

2.1 Efficient Inference for VLMs

VLMs couple a vision encoder with an LLM decoder (Liu et al., 2023; Bai et al., 2025); dynamic-resolution encoders map images and videos to thousands of tokens, making cache memory the dominant inference cost. *Token pruning* attacks this before the cache exists. For example, FastV (Chen et al., 2024) and VisionZip (Yang et al., 2026) rank visual tokens by attention, CDPruner (Zhang et al., 2025c) maximizes conditional diversity, and subsequent work refines the selection signal (Zhang et al., 2025b; Alvar et al., 2025). Pruning is irreversible, as information needed by later layers or decoding steps is permanently lost, and because decoding rather than prefill dominates end-to-end latency, its practical acceleration is limited. DEKV adopts the eviction paradigm precisely to retain per-layer flexibility. A token dropped from one layer's cache remains available in the layers that keep it, and eviction compresses the decode-time footprint that pruning leaves untouched.

2.2 KV Cache Eviction

2.2.1 Attention-based eviction vs merging

Existing attention-based eviction methods, such as StreamingLLM (Xiao et al., 2024), H2O (Zhang et al., 2023), SnapKV (Li et al., 2024b), and TOVA (Oren et al., 2024), are inherently value-blind. They measure access frequency rather than actual contribution, which penalizes VLMs because redundant visual tokens inappropriately consume the memory budget, often forcing the detrimental

Table 1: Positioning of DEKV. "Output-aware" = uses output perturbation instead of attention alone; "adaptive budget" = per-layer budgets vary with the input; "global alloc." = assigns budgets via a single model-wide selection; "unified" = the same criterion governs both ranking and budgeting; "VLM-aware" = designed for the modality gap. $\sim$ denotes partial support.

Method	Output-aware	Adaptive budget	Global alloc.	Unified ranking+budget	VLM-aware
H2O (Zhang et al., 2023) / SnapKV (Li et al., 2024b)	✗	✗	✗	✗	✗
PyramidKV (Cai et al., 2025) / CAKE (Qin et al., 2025)	✗	✓	✗	✗	✗
Ada-KV (Feng et al., 2025)	✗	$\sim$	$\sim$	$\sim$	✗
CAOTE (Goel et al., 2025) / OBCache (Gu et al., 2026)	✓	✗	✗	✗	✗
LAVa (Shen et al., 2025)	✓	✓	✓	✓	✗
AirCache (Huang et al., 2025)	✗	✗	✗	✗	✓
MEDA (Wan et al., 2025a)	✗	✓	✗	✗	✓
TGV-KV (Liu et al., 2026)	✗	✓	✗	✗	✓
DEKV (ours)	✓	✓	✓	✓	✓

eviction of crucial text KVs due to the modality gap (Liu et al., 2026). DEKV solves this by using exact output perturbation to rank importance, correctly pricing redundant high-attention tokens as cheap and unique moderate-attention tokens as valuable. While some compression methods merge discarded KVs into retained ones (Kim et al., 2024; Wan et al., 2025b; Zhang et al., 2025d; Tian et al., 2025; Wan et al., 2024; Liu et al., 2024b), DEKV remains strictly eviction-only. Unlike sequence-level token pruning, eviction only discards KV pairs from a specific layer's cache. Because DEKV already targets redundant tokens, complex merging offers no accuracy benefits and merely increases computational overhead.

2.2.2 Output-aware importance

Recent methods evaluate the effect of eviction on attention *outputs* rather than relying on attention scores alone. Approaches like DiffKV (Zhang et al., 2025d), CAOTE (Goel et al., 2025), CriticalKV (Feng et al., 2026), OBCache (Gu et al., 2026), and OmniMem (Sun et al., 2026) utilize perturbation-based criteria to estimate token output error. While DEKV shares this foundational approach, it differs significantly in its comprehensive application. Prior methods restrict output-aware scores to intra-layer ranking under externally supplied budgets, often acting as secondary correction terms for host eviction methods in unimodal LLMs. DEKV instead normalizes this metric to dynamically allocate per-layer budgets across the network, creating a self-contained, multimodal policy that requires no host method. Moreover, DEKV computes its metric exclusively from text-query rows. This targeted computation ensures that an exact output-aware criterion remains highly efficient in vision-heavy VLMs, where text queries make up only a small fraction of the full sequence.

2.2.3 Budget allocation

Several methods adaptively allocate budgets across layers, including PyramidKV (Cai et al., 2025), CAKE (Qin et al., 2025), PrefixKV (Wang et al., 2025), Ada-KV (Feng et al., 2025), MEDA (Wan et al., 2025a), and TGV-KV (Liu et al., 2026). However, they rely on attention statistics or entropy proxies, which creates a disconnect between their budgeting and ranking criteria because these metrics do not accurately measure information loss. While LAVa (Shen et al., 2025) successfully unifies these criteria under a single information-loss metric, it is limited to unimodal LLMs. DEKV extends this unified concept to multimodal architectures through a cross-layer water-filling allocation driven by output perturbation from text-query rows. By addressing the modality gap directly, DEKV ensures per-layer budgets dynamically adapt to each sample, emerging as an output of the allocation process rather than a predefined input.

2.3 Positioning

Table 1 shows that prior VLM methods manage scoring and adaptive budgeting through disconnected mechanisms, relying on attention proxies for budgets and separate scores for ranking. DEKV is the first training-free method to unify both token ranking and cross-layer budget allocation un-

der a single output-perturbation criterion computed solely from text-query rows. As our ablations demonstrate (Sec. 4.3), both this unified damage metric and the water-filling allocation are strictly necessary to achieve accuracy gains at extreme memory budgets.

3 METHODOLOGY

We introduce DEKV, a training-free KV cache eviction framework for vision-language models (VLMs). Unlike earlier pipelines that rely on separate heuristics for token ranking and layer budgets (Liu et al., 2026; Cai et al., 2025; Qin et al., 2025), DEKV determines both using a single, closed-form *damage* functional. As shown in Fig. 1, this shared metric evaluates the exact perturbation a token's eviction causes to the model's attention outputs (Sec. 3.2, Sec. 3.3).

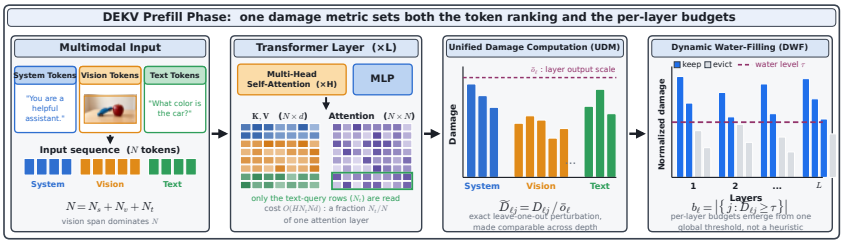

Figure 1: Overview of the DEKV prefill phase. The framework primarily consists of two stages: Unified Damage Computation (UDM) and Dynamic Water-Filling (DWF).

3.1 PRELIMINARIES AND PROBLEM FORMULATION

A multimodal prompt consists of $N = N_v + N_t$ tokens, with N_v vision tokens in a contiguous span $\mathcal{V}$ and text tokens in $\mathcal{T}$. At layer ℓ and head h, prefill generates cached keys $\mathbf{K}^{(\ell,h)}$, values $\mathbf{V}^{(\ell,h)} \in \mathbb{R}^{N\times d}$, and post-softmax attention $\mathbf{A}^{(\ell,h)} \in \mathbb{R}^{N\times N}$. The attention output for query i is:

$$\mathbf{o}_i \;=\; \sum_{k=1}^{N} a_{ik}\,\mathbf{v}_k, \qquad a_{ik} = \left[\mathbf{A}^{(\ell,h)}\right]_{ik}, \tag{1}$$

where layer and head indices are omitted for simplicity. An eviction policy retains b_ℓ slots per layer under a total memory limit $\sum_{\ell=1}^{L} b_\ell \le B$. Instead of pre-assigning b_ℓ, DEKV assigns a binary decision $f_{\ell j} \in \{\text{KEEP}, \text{EVICT}\}$ to each token. Letting $D_{\ell j} \ge 0$ denote the attention output perturbation caused by evicting token j, DEKV solves:

$$\min_{\{f_{\ell j}\}} \sum_{\ell=1}^{L} \sum_{j\,:\,f_{\ell j}=\text{EVICT}} D_{\ell j} \quad \text{s.t.} \quad \sum_{\ell=1}^{L} \left|\{j : f_{\ell j} = \text{KEEP}\}\right| \;\le\; B, \tag{2}$$

By jointly retaining the B globally most damaging tokens, the per-layer budgets b_ℓ naturally emerge as optimization outputs. DEKV executes this by first defining $D_{\ell j}$ (Sec. 3.2) and then solving Eq. 2 (Sec. 3.3).

3.2 THE DAMAGE METRIC

The central quantity in Eq. 2 is the exact output perturbation caused by eviction. Removing key-value pair j rescales the remaining softmax weights in Eq. 1 to $a_{ik}/(1-a_{ij})$ for $k \neq j$. The updated output becomes $\mathbf{o}_i' = (\mathbf{o}_i - a_{ij}\mathbf{v}_j)/(1-a_{ij})$, yielding the exact perturbation:

$$\Delta_{ij} \;\triangleq\; \mathbf{o}_i - \mathbf{o}_i' \;=\; \frac{a_{ij}}{1-a_{ij}}\left(\mathbf{v}_j - \mathbf{o}_i\right). \tag{3}$$

Eq. 3 decomposes eviction damage into an *attention renormalization* factor and a *value-novelty* term $\|\mathbf{v}_j - \mathbf{o}_i\|$. Unlike attention-only criteria (Zhang et al., 2023; Li et al., 2024b; Liu et al., 2026) that waste budgets on redundant high-attention visual tokens, this formulation correctly prices redundant

tokens as cheap and protects unique tokens. While similar criteria exist for unimodal LLMs (Goel et al., 2025; Gu et al., 2026; Feng et al., 2026), we integrate this exact functional into cross-layer multimodal budget allocation (Sec. 3.3).

Since future decoding queries are unavailable during prefill, we evaluate Eq. 3 over the conditioning text queries $i \in \mathcal{T}$. We apply normalized weights w_i to upweight the final q instruction tokens by a factor ρ. Averaging over heads, the eviction damage of token j is:

$$D_j \;=\; \frac{1}{H}\sum_{h=1}^{H}\sum_{i\in\mathcal{T}} w_i\Big(\tfrac{a_{ij}^{(h)}}{1-a_{ij}^{(h)}}\Big)\,\big\|\mathbf{v}_j^{(h)}-\mathbf{o}_i^{(h)}\big\|, \tag{4}$$

To enable scale-invariant comparisons across layers, we divide each token's damage by the layer's mean output norm:

$$\bar{o}_\ell \;=\; \frac{1}{HN_t}\sum_{h=1}^{H}\sum_{i\in\mathcal{T}}\big\|\mathbf{o}_i^{(h)}\big\|, \qquad \widetilde{D}_{\ell j} \;=\; \frac{D_{\ell j}}{\bar{o}_\ell}, \tag{5}$$

This ensures $\widetilde{D}_{\ell j}$ represents a relative perturbation suitable for global allocation (Sec. 3.3).

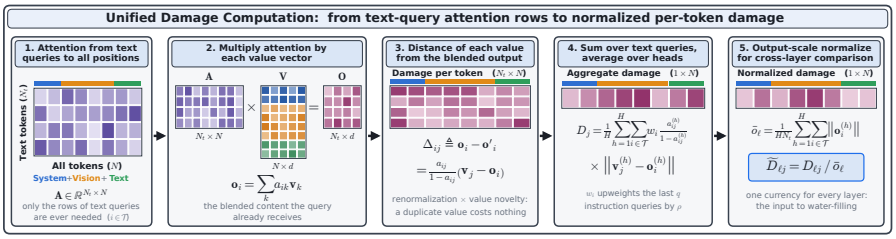

Figure 2: Computation steps for the damage metric. The process measures the distance of each value vector from the blended output and normalizes the aggregate damage for cross-layer comparability.

3.3 Budget Allocation by Damage Water-Filling

To maximize avoided damage $\sum_{(\ell,j)\in\mathcal{K}}\widetilde{D}_{\ell j}$ under the constraint $|\mathcal{K}| \le B$, Eq. 2 optimally retains the B globally most damaging tokens:

$$\mathcal{K}^\star = \big\{(\ell,j) : \widetilde{D}_{\ell j} \ge \tau\big\}, \qquad \tau = \widetilde{D}_{(B)}, \tag{6}$$

where τ is the B-th largest damage across all tokens, naturally yielding the per-layer budgets $b_\ell = |\{j : \widetilde{D}_{\ell j} \ge \tau\}|$.

By applying a single global threshold τ, Eq. 6 adaptively allocates the memory budget. Layers with redundant tokens clear few elements above τ, inherently ceding capacity to layers possessing unique, high-damage content. This extends intra-layer global selection (Feng et al., 2025) to a cross-layer framework governed by an exact output-perturbation metric, eliminating the need for heuristic depth profiles (Cai et al., 2025), calibration sets (Wang et al., 2025), or entropy proxies (Qin et al., 2025; Liu et al., 2026; Wan et al., 2025a). Additionally, to prevent any layer from becoming vision-blind under extreme text-heavy budgets, every layer retains a minimum floor of c_{vis} vision tokens.

3.4 Decode-Phase

During generation (Fig. 3), the new query's attention yields an instantaneous damage $\widetilde{D}_j^{(t)}$ based on the single-query specialization of Eqs. 4 and 5. This is updated via an exponential moving average to strictly enforce the memory budget at every step:

$$\hat{c}_j^{(t)} = \gamma\,\hat{c}_j^{(t-1)} + \widetilde{D}_j^{(t)}, \tag{7}$$

where seeding $\hat{c}^{(0)}$ with prefill damages ensures text-grounded rankings persist instead of being overwritten by a single noisy attention row. By standardizing damage (Eq. 5), DEKV evaluates all L layers in parallel via a single fused pass. This global execution eliminates synchronization bottlenecks while strictly enforcing the exact memory budget at every decoding step.

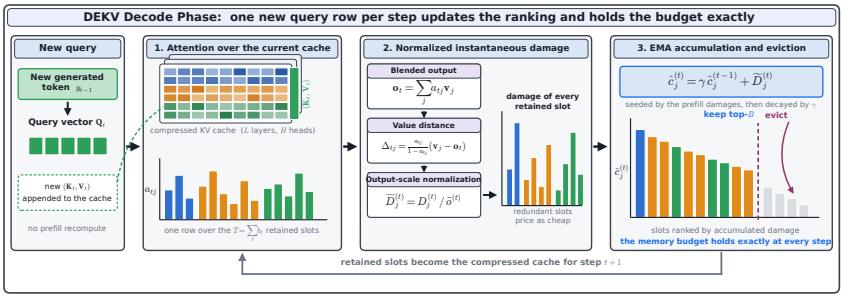

Figure 3: The DEKV decode phase. Instantaneous damage from each new query is combined with an exponentially decaying accumulated damage to continuously update token rankings.

Table 2: Comparison of KV cache compression methods across four VQA benchmarks at 20% and 5% KV cache retention rates. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	ChartQA Rel. Acc.↑		DocVQA ANLS↑		VizWiz Acc.↑		TextVQA Acc.↑	
	20%	5%	20%	5%	20%	5%	20%	5%
LLaVA-1.5-7B (Liu et al., 2024a)								
Vanilla	18.0	18.0	23.9	23.9	54.4	54.4	47.9	47.9
StreamingLLM	14.4	13.4	15.3	13.6	52.5	**48.2**	35.4	32.5
SnapKV	16.6	0.4	20.5	1.7	53.6	5.0	44.6	9.2
H_2O	3.7	3.7	7.4	4.5	10.0	7.6	27.6	21.0
ElasticCache	3.2	3.9	3.7	2.8	6.1	9.3	14.6	9.1
PrefixKV	2.0	0.9	7.5	1.6	3.9	0.5	19.6	4.8
TGV-KV	**18.0**	13.8	21.8	14.3	54.0	31.2	47.4	33.9
DEKV (Ours)	17.8	**15.9**	**22.3**	**17.6**	**54.3**	19.2	**47.7**	**43.4**
Qwen3-VL-4B-Instruct (Bai et al., 2025)								
Vanilla	84.1	84.1	93.6	93.6	68.5	68.5	80.6	80.6
StreamingLLM	70.6	66.9	62.2	47.0	63.8	59.6	63.8	51.7
SnapKV	74.7	19.8	90.9	12.3	64.9	48.4	75.8	27.0
H_2O	71.0	63.4	82.9	53.6	64.2	54.8	74.5	55.7
ElasticCache	74.6	58.8	57.8	33.1	54.7	52.8	57.4	39.8
PrefixKV	78.1	65.4	73.1	43.0	59.2	49.5	67.9	47.6
TGV-KV	82.5	67.2	93.1	87.3	67.8	**63.1**	80.1	70.4
DEKV (Ours)	**83.6**	**78.3**	**93.4**	**89.0**	**68.0**	62.7	**80.4**	**72.6**

4 Experiments

4.1 Experimental Setup

Models We evaluate DEKV without finetuning across three diverse VLM families to ensure the observed gains generalize across various architectures and parameter scales. These include the LLaVA series (LLaVA-1.5-7B (Liu et al., 2024a), LLaVA-NeXT-Mistral-7B (Liu et al., 2024a), and LLaVA-OneVision-Qwen2-0.5B (Li et al., 2024a)), the Qwen3-VL models (Qwen3-VL-4B-Instruct and Qwen3-VL-8B-Instruct (Bai et al., 2025)), and Phi-3.5-vision-instruct (Abdin et al., 2024). This selection comprehensively covers basic, high-resolution, compact, and state-of-the-art setups. The experimental results for a subset of these models are detailed in Appendix A.4.

Table 3: Full performance comparison on text-dominant benchmarks, LLaVA-1.5-7B. We report BLEU-x (B-x), CIDEr (C), METEOR (M), and ROUGE-L (R). CIDEr scores are reported as raw values. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	TextCaps (Sidorov et al., 2020)							COCO-Caption (Lin et al., 2015)						
	B-1	B-2	B-3	B-4	C	M	R	B-1	B-2	B-3	B-4	C	M	R
Retain 10% KV														
Vanilla	0.71	0.53	0.38	0.27	1.00	0.23	0.47	0.73	0.56	0.41	0.29	1.08	0.28	0.55
StreamingLLM	0.61	0.41	0.26	0.17	0.40	0.17	0.39	0.67	0.47	0.32	0.20	0.68	0.20	0.47
SnapKV	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.01
H2O	0.03	0.02	0.01	0.00	0.01	0.04	0.07	0.04	0.02	0.01	0.00	0.01	0.04	0.08
ElasticCache	0.04	0.02	0.01	0.00	0.01	0.03	0.07	0.04	0.02	0.01	0.00	0.01	0.04	0.08
PrefixKV	0.04	0.02	0.01	0.00	0.01	0.04	0.08	0.04	0.02	0.01	0.00	0.01	0.04	0.08
TGV-KV	0.64	0.45	0.30	0.20	0.64	0.19	0.42	0.71	0.54	0.39	0.27	0.95	0.26	0.53
DEKV (Ours)	**0.69**	**0.50**	**0.36**	**0.25**	**0.87**	**0.22**	**0.45**	**0.75**	**0.57**	**0.42**	**0.30**	**1.09**	**0.28**	**0.55**

Datasets To evaluate accuracy retention, we test models on both *vision-dominant* and *text-dominant* tasks. *Vision-dominant* benchmarks (ChartQA (Masry et al., 2022), DocVQA (Mathew et al., 2021), VizWiz (Gurari et al., 2018), and TextVQA (Singh et al., 2019)) require fine-grained visual extraction. In contrast, *text-dominant* tasks (TextCaps (Sidorov et al., 2020) and COCO-Caption-2017 (Lin et al., 2015)) focus primarily on image-conditioned generation quality.

Baselines We evaluate DEKV against LLM-centric eviction baselines (StreamingLLM (Xiao et al., 2024), SnapKV (Li et al., 2024b), and H2O (Zhang et al., 2023)) alongside VLM-aware methods like ElasticCache (Liu et al., 2024b), PrefixKV (Wang et al., 2025), and the leading competitor TGV-KV (Liu et al., 2026). To ensure that any performance differences stem solely from the core eviction policy, all baselines share identical retention budgets, attention sinks, and recency windows where applicable.

Implementation Details We conduct all accuracy evaluations using the LMMs-Eval toolkit (Zhang et al., 2025a) on NVIDIA A100 GPUs. To analyze performance across mild to extreme compression, we test each method at four KV cache retention budgets: $B \in \{50\%, 20\%, 10\%, 5\%\}$. Following standard LMMs-Eval protocols, we report Relaxed Accuracy for ChartQA, ANLS for DocVQA, and exact-match Accuracy for VizWiz and TextVQA. For text generation on TextCaps and COCO-Caption, we assess textual overlap and semantic similarity against human references using CIDEr, METEOR, ROUGE-L, and BLEU-1-4.

4.2 Results

Vision-dominant tasks Table 2 evaluates two VLMs across four vision-dominant tasks at 20% and 5% retention budgets. DEKV outperforms the top baseline, TGV-KV, in 13 of 16 settings, with its average advantage widening from $+0.4$ points at 20% retention to over $+2$ points at 5%. Meanwhile, baselines like H_2O, ElasticCache, and PrefixKV degrade severely at low retention rates, confirming the value of our modality-aware approach. DEKV's only three losses occur on VizWiz at 5% retention (a task uniquely reliant on low-quality images) and by a marginal deficit on ChartQA at 20% for LLaVA-1.5-7B. For more detailed results, refer to Tables 6,7,and 8 in the appendix.

Text-dominant tasks We evaluate caption generation on TextCaps and COCO-Caption-2017 (Table 3). Under a strict 10% retention budget, DEKV is the only method that consistently rivals the vanilla model. Remarkably, DEKV slightly improves on vanilla's COCO-Caption CIDEr score (1.09 vs. 1.08) and remains closest on TextCaps (0.87 vs. 1.00). While TGV-KV tracks behind with respective CIDEr scores of 0.64 and 0.95, the remaining baselines (SnapKV, H2O, ElasticCache, and PrefixKV) collapse entirely to near-zero scores. This highlights how severely non-modality-aware methods degrade under tight compression. For more detailed results, refer to Tables 9 and 10 in the appendix.

Table 4: Latency, throughput and memory for LLaVA-1.5-7B on a single A100. Mean over 5 runs generating 256 tokens. *Full KV* applies no eviction and runs out of memory at 8K context.

Metric	Method	2K context		4K context		8K context	
		10%	5%	10%	5%	10%	5%
TTFT (s)	Full KV	0.427	0.427	0.942	0.942	—	—
	TGV-KV	0.454	0.455	0.984	0.984	3.220	3.218
	DEKV	0.479	0.480	1.008	1.007	3.192	3.184
Decode Lat. (ms/tok)	Full KV	31.28	31.26	31.86	31.62	—	—
	TGV-KV	37.00	42.11	36.84	38.23	36.11	35.95
	DEKV	38.71	42.46	40.21	37.91	42.65	39.53
Decode Thr. (tok/s)	Full KV	31.97	31.99	31.39	31.63	—	—
	TGV-KV	27.03	24.06	27.15	26.16	27.69	27.82
	DEKV	25.83	23.55	24.87	26.38	23.45	25.29
Overall Thr. (tok/s)	Full KV	30.47	30.48	28.24	28.43	—	—
	TGV-KV	25.89	23.15	24.67	23.85	20.60	20.67
	DEKV	24.73	22.64	22.73	23.99	18.20	19.30
End-to-end (s)	Full KV	8.40	8.40	9.07	9.01	—	—
	TGV-KV	9.89	11.19	10.38	10.73	12.43	12.38
	DEKV	10.35	11.31	11.26	10.67	14.07	13.27
Peak Mem. (GiB)	Full KV	25.74	25.74	49.79	49.79	—	—
	TGV-KV	15.70	15.70	19.28	19.28	33.25	33.25
	DEKV	15.70	15.70	19.28	19.28	33.25	33.25

Latency and Memory Overhead Table 4 highlights the practical trade-offs of our approach. DEKV requires no additional peak memory compared to TGV-KV at matched budgets, meaning its accuracy gains come at no extra storage cost. Continuous token evaluation adds computational overhead that scales directly with the retained cache. Although this overhead introduces minor penalties in time-to-first-token (TTFT) and decode latency, DEKV's decode and overall throughput remain highly competitive. At a tight 5% budget, DEKV's end-to-end latency practically matches TGV-KV (ranging from 0.6% faster to 7% slower), making it exceptionally well-suited for heavily memory-constrained deployments where accuracy is critical.

Layer-wise allocation under extreme compression To explain DEKV's advantage under extreme compression, Figure 4 breaks down per-layer token retention for LLaVA-1.5-7B on TextVQA at a 10% budget. While TGV-KV heavily concentrates retained tokens in the first two layers, DEKV's water-filling approach distributes the same budget much more evenly across all 32 layers. Beyond accuracy, this smooth allocation offers major practical benefits for distributed inference. When sharding layers across multiple accelerators, a device's memory requirement is dictated by its largest hosted KV cache. TGV-KV's skewed profile forces the device hosting the initial layers to over-provision memory, leading to severe underutilization on the remaining devices (Zhao et al., 2025). By equalizing marginal damage instead of relying on fixed attention-mass profiles, DEKV naturally produces a balanced memory footprint, making it highly efficient for multi-GPU deployments (Huang et al., 2019). For more detailed results, refer to Appendix A.5.

4.3 Ablation Study

To isolate the drivers of DEKV's accuracy, we ablate its three core components on LLaVA-1.5-7B across retention budgets from 50% to 5% (Table 5). Evaluating on both ChartQA and TextCaps, we remove each feature individually: *DWF* is replaced by a uniform per-layer budget, *UDM* is swapped for a basic attention-sum score, and the *Vision floor* constraint is replaced with $c_{vis} \in \{0, 2, 4\}$.

UDM causes the largest performance drops on TextCaps across all budgets and on ChartQA at 10% and 5% retention. This occurs because attention alone cannot distinguish irreplaceable tokens from redundant ones that merely receive high attention. While this limitation is harmless under generous memory budgets, it becomes fatal during extreme compression.

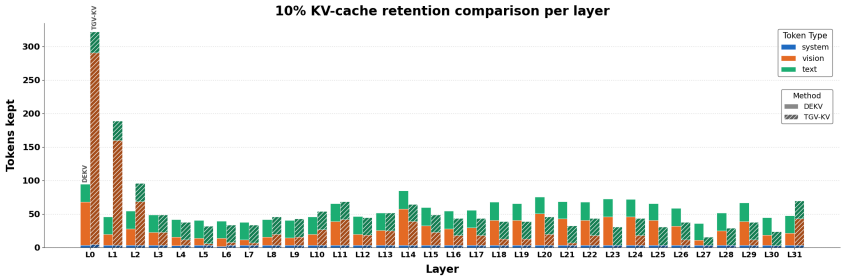

Figure 4: Per-layer KV retention by token segment for LLaVA-1.5-7B on TextVQA at a 10% budget.

Table 5: Component ablation on LLaVA-1.5-7B. ChartQA relaxed accuracy (%) and TextCaps CIDEr at four retention budgets. Full DEKV consists of DWF, UDM, and $c_{\text{vis}} = 8$.

Method	**ChartQA (%)**				**TextCaps (CIDEr)**			
	50%	**20%**	**10%**	**5%**	**50%**	**20%**	**10%**	**5%**
Full DEKV	17.7	17.8	16.7	**15.9**	**1.010**	**0.990**	0.870	0.550
−DWF	17.8	**18.0**	16.5	15.7	1.007	0.974	0.846	0.567
−UDM	**17.9**	17.9	16.3	15.4	0.999	0.960	0.734	0.532
−VF (c_{vis}=0)	17.7	17.8	**16.9**	15.2	1.004	0.982	0.869	0.573
−VF (c_{vis}=2)	17.7	17.8	**16.9**	15.3	1.004	0.982	0.869	**0.576**
−VF (c_{vis}=4)	17.7	17.8	16.7	15.2	1.004	0.986	**0.871**	0.566

On TextCaps, water-filling outperforms the uniform allocation at 50%, 20%, and 10% retention, though the uniform approach slightly edges ahead at the extreme 5% budget. Conversely, on ChartQA, water-filling proves superior at the tightest 10% and 5% budgets, while the uniform baseline performs marginally better at looser settings. Although the damage metric drives the vast majority of DEKV's performance advantage, water-filling still contributes a measurable accuracy boost alongside its critical memory-balancing benefits (Sec. 4.2).

Removing the per-layer vision floor entirely (c_{vis}=0) yields results nearly identical to Full DEKV across most retention budgets. Because the damage criterion naturally protects critical vision tokens, the floor remains completely inactive at 50% and 20% budgets. However, at the extreme 5% budget on the vision-heavy ChartQA task, removing the default c_{vis}=8 floor costs 0.7 accuracy points. Finer sweeps across $c_{\text{vis}} \in \{0, 2, 4\}$ confirm that while the floor contributes negligibly to overall accuracy, it acts as a vital safety margin specifically for extreme compression.

5 Conclusion

We presented DEKV, a training-free KV cache eviction framework that resolves VLM memory bottlenecks without relying on disconnected proxy metrics. By framing compression as budgeted damage minimization, DEKV uses an exact output-perturbation criterion to safeguard unique multi-modal information while aggressively discarding redundant tokens. Its Damage Water-Filling mechanism leverages this unified metric to dynamically distribute the memory budget across all layers, avoiding the skewed profiles of prior methods and seamlessly supporting distributed inference across multiple accelerators. Extensive evaluations across the LLaVA, Qwen3-VL, and Phi model families confirm that DEKV consistently outperforms existing VLM-aware baselines under extreme compression. By preserving near-uncompressed accuracy at drastically reduced cache sizes, DEKV offers a highly scalable and principled solution for long-context multi-modal inference.

AI USE STATEMENT

In this work, we used generative AI tools strictly for text generation and polishing the manuscript to improve clarity and flow. We have not used generative AI tools for data analysis, experimental execution, or the generation of core scientific claims. All AI-assisted text was thoroughly reviewed, verified, and manually edited by the authors to ensure accuracy and adherence to rigorous scientific standards. We take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work focuses on improving the computational efficiency of Vision-Language Models (VLMs) through training-free KV cache compression. By significantly reducing the memory footprint and latency associated with long-context inference, our methodology lowers the hardware barriers to deploying large models, thereby helping to mitigate the environmental impact and energy consumption of AI systems. Because DEKV is applied strictly at inference time without requiring fine-tuning, it does not introduce new training data or fundamentally alter the base model's underlying knowledge representations. We evaluate our method exclusively on standard, publicly available academic benchmarks that do not involve human subjects, sensitive personal information, or the release of new datasets. While any compression technique carries a theoretical risk of disproportionately degrading performance on edge cases or underrepresented contexts, our evaluations demonstrate that DEKV maintains robust, equitable accuracy across diverse vision and text tasks. We foresee no direct harmful applications or privacy risks arising from this efficiency-focused research.

REPRODUCIBILITY STATEMENT

To ensure full reproducibility of our results, we have thoroughly detailed the core DEKV framework in Section 3. The exact prefill procedure is formally summarized in Algorithm 1. All empirical evaluations rely entirely on standard, publicly available benchmarks (e.g., ChartQA, DocVQA, TextCaps) and were executed using the open-source LMMs-Eval toolkit, as outlined in Section 4.1. Comprehensive implementation details, including the evaluated retention budgets (50%, 20%, 10%, 5%) and the vision floor hyperparameters, are specified in Section 4.1 and Section 4.3.

REFERENCES

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

Saeed Ranjbar Alvar, Gursimran Singh, Mohammad Akbari, and Yong Zhang. Divprune: Diversity-based visual token pruning for large multimodal models, 2025. URL https://arxiv.org/abs/2503.02175.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2025. URL https://arxiv.org/abs/2406.02069.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models, 2024. URL https://arxiv.org/abs/2403.06764.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference, 2025. URL https://arxiv.org/abs/2407.11550.

Yuan Feng, Junlin Lv, Haoyu Guo, Yukun Cao, Xike Xie, and S Kevin Zhou. Identify critical KV cache in LLM inference from an output perturbation perspective, 2026. URL https://openreview.net/forum?id=cZ74yWoKYr.

Raghavv Goel, Junyoung Park, Mukul Gagrani, Dalton Jones, Matthew Morse, Harper Langston, Mingu Lee, and Chris Lott. Caote: Kv cache selection for llms via attention output error-based token eviction, 2025. URL https://arxiv.org/abs/2504.14051.

Yuzhe Gu, Xiyu Liang, Jiaojiao Zhao, and Enmao Diao. Obcache: Optimal brain kv cache pruning for efficient long-context llm inference, 2026. URL https://arxiv.org/abs/2510.07651.

Danna Gurari, Qing Li, Abigale J. Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P. Bigham. Vizwiz grand challenge: Answering visual questions from blind people, 2018. URL https://arxiv.org/abs/1802.08218.

Kai Huang, Hao Zou, Bochen Wang, Ye Xi, Zhen Xie, and Hao Wang. Aircache: Activating inter-modal relevancy kv cache compression for efficient large vision-language model inference, 2025. URL https://arxiv.org/abs/2503.23956.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2019. URL https://arxiv.org/abs/1811.06965.

Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. Compressed context memory for online language model interaction, 2024. URL https://arxiv.org/abs/2312.03414.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. Llava-onevision: Easy visual task transfer, 2024a. URL https://arxiv.org/abs/2408.03326.

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, 2024b. URL https://arxiv.org/abs/2404.14469.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL https://arxiv.org/abs/1405.0312.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2024a. URL https://arxiv.org/abs/2310.03744.

Jizhihui Liu, Ruizi Han, Miao Zhang, Rui Shao, Xuebo Liu, Weili Guan, and Yaowei Wang. Tgv-kv: Text-grounded kv eviction for vision-language models, 2026. URL https://arxiv.org/abs/2606.03075.

Zuyan Liu, Benlin Liu, Jiahui Wang, Yuhao Dong, Guangyi Chen, Yongming Rao, Ranjay Krishna, and Jiwen Lu. Efficient inference of vision instruction-following models with elastic cache, 2024b. URL https://arxiv.org/abs/2407.18121.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. Docvqa: A dataset for vqa on document images, 2021. URL https://arxiv.org/abs/2007.00398.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns, 2024. URL https://arxiv.org/abs/2401.06104.

Xuan Peng, Xuanhua Shi, Haolin Zhang, Yunfei Zhao, and Xuehai Qian. Dawnpiper: A memory-scalable pipeline parallel training framework, 2025. URL https://arxiv.org/abs/2505.05856.

Penghui Qi, Xinyi Wan, Nyamdavaa Amar, and Min Lin. Pipeline parallelism with controllable memory, 2024. URL https://arxiv.org/abs/2405.15362.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences, 2025. URL https://arxiv.org/abs/2503.12491.

Yiqun Shen, Song Yuan, Zhengze Zhang, Xiaoliang Wang, Daxin Jiang, and Nguyen Cam-Tu. Lava: Layer-wise kv cache eviction with dynamic budget allocation, 2025. URL https://arxiv.org/abs/2509.09754.

Oleksii Sidorov, Ronghang Hu, Marcus Rohrbach, and Amanpreet Singh. Textcaps: a dataset for image captioning with reading comprehension, 2020. URL https://arxiv.org/abs/2003.12462.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read, 2019. URL https://arxiv.org/abs/1904.08920.

Guangzhi Sun, Yixuan Li, Yudong Yang, and Chao Zhang. Omnimem: Perturbation-aware memory compression for streaming audio-visual llms, 2026. URL https://arxiv.org/abs/2606.07577.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. Keepkv: Achieving periodic lossless kv cache compression for efficient llm inference, 2025. URL https://arxiv.org/abs/2504.09936.

Zhongwei Wan, Ziang Wu, Che Liu, Jinfa Huang, Zhihong Zhu, Peng Jin, Longyue Wang, and Li Yuan. Look-m: Look-once optimization in kv cache for efficient multimodal long-context inference, 2024. URL https://arxiv.org/abs/2406.18139.

Zhongwei Wan, Hui Shen, Xin Wang, Che Liu, Zheda Mai, and Mi Zhang. Meda: Dynamic kv cache allocation for efficient multimodal long-context inference, 2025a. URL https://arxiv.org/abs/2502.17599.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, Longyue Wang, and Mi Zhang. D2o: Dynamic discriminative operations for efficient long-context inference of large language models, 2025b. URL https://arxiv.org/abs/2406.13035.

Ao Wang, Hui Chen, Jiaxin Li, Jianchao Tan, Kefeng Zhang, Xunliang Cai, Zijia Lin, Jungong Han, and Guiguang Ding. Prefixkv: Adaptive prefix kv cache is what vision instruction-following models need for efficient generation, 2025. URL https://arxiv.org/abs/2412.03409.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. Visionzip: Longer is better but not necessary in vision language models, 2026. URL https://arxiv.org/abs/2412.04467.

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. Lmms-eval: Reality check on the evaluation of large multimodal models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 881–916. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.findings-naacl.51. URL http://dx.doi.org/10.18653/v1/2025.findings-naacl.51.

Qizhe Zhang, Aosong Cheng, Ming Lu, Renrui Zhang, Zhiyong Zhuo, Jiajun Cao, Shaobo Guo, Qi She, and Shanghang Zhang. Beyond text-visual attention: Exploiting visual cues for effective token pruning in vlms, 2025b. URL https://arxiv.org/abs/2412.01818.

Qizhe Zhang, Mengzhen Liu, Lichen Li, Ming Lu, Yuan Zhang, Junwen Pan, Qi She, and Shanghang Zhang. Beyond attention or similarity: Maximizing conditional diversity for token pruning in mllms, 2025c. URL https://arxiv.org/abs/2506.10967.

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C. S. Lui, and Haibo Chen. Diffkv: Differentiated memory management for large language models with parallel kv compaction, 2025d. URL https://arxiv.org/abs/2412.03131.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Bingzhe Zhao, Ke Cheng, Aomufei Yuan, Yuxuan Tian, Ruiguang Zhong, Chengchen Hu, Tong Yang, and Lian Yu. Fairkv: Balancing per-head kv cache for fast multi-gpu inference, 2025. URL https://arxiv.org/abs/2502.15804.

A Appendix

A.1 Analysis of Damage by Token Type

Fig. 5 shows that text tokens tend to carry more serious damage than the bulk of vision tokens, but several text tokens still fall within, or below, the range occupied by vision tokens. So, DEKV, unlike other methods such as TGV-KVLiu et al. (2026), applies no hard rule to shield any token outside the small structurally protected set (system tokens and the final positions).

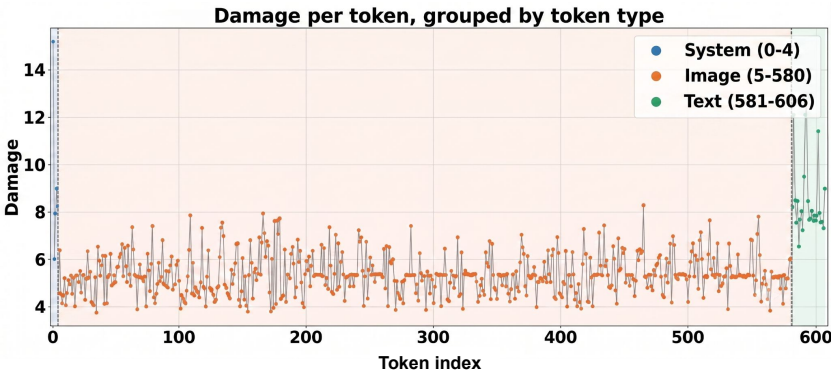

Figure 5: Per-token damage. System tokens carry the highest damage and are structurally protected, while vision and text tokens occupy a substantially overlapping damage range.

A.2 Complexity and Implementation

By restricting computations exclusively to text tokens, DEKV limits its overhead to $O(HN_tNd)$ time and $O(HN_tN)$ transient memory, requiring only a small fraction (N_t/N) of standard attention costs. Algorithm 1 summarizes the complete DEKV prefill procedure.

Algorithm 2 details one decoding step. Each layer keeps its prefill allocation b_ℓ fixed and evicts one slot per step.

Algorithm 1 DEKV Prefill

Require: caches $\{\mathbf{K}_\ell, \mathbf{V}_\ell\}$, text-query attention rows, budget B, floor c_{vis}

for $\ell = 1$ **to** L **do**
  compute $D_{\ell j}$ for all j ▷ Eq. 4
  normalize to $\widetilde{D}_{\ell j}$ ▷ Eq. 5
  set $\widetilde{D}_{\ell j} \leftarrow \infty$ for sinks and the final token
end for
reserve floor slots (c_{vis} per layer), charged against B
$\tau \leftarrow B$-th largest of remaining $\{\widetilde{D}_{\ell j}\}$; $\quad \mathcal{K}^\star \leftarrow \{(\ell, j) : \widetilde{D}_{\ell j} \geq \tau\}$ ▷ Eq. 6
for $\ell = 1$ **to** L **do**
  gather retained slots $\mathcal{K}^\star_\ell$; record b_ℓ; seed decode state $\hat{c}^{(0)}$
end for

Algorithm 2 DEKV Decode (one generation step t)

Require: caches $\{\mathbf{K}_\ell, \mathbf{V}_\ell\}$ after the forward pass on y_{t-1}, with retained slots $\mathcal{K}_\ell$, $|\mathcal{K}_\ell| = b_\ell + 1$; accumulated damage $\hat{c}^{(t-1)}_{\ell j}$ (0 for the new slot); decay γ

for all $\ell = 1, \ldots, L$ **in parallel do** ▷ single fused pass
  compute $\widetilde{D}^{(t)}_{\ell j}$ for all $j \in \mathcal{K}_\ell$ from the new query's row ▷ Eqs. 4–5
  $\hat{c}^{(t)}_{\ell j} \leftarrow \gamma\, \hat{c}^{(t-1)}_{\ell j} + \widetilde{D}^{(t)}_{\ell j}$ ▷ Eq. 7
end for
for $\ell = 1$ **to** L **do**
  $\mathcal{K}_\ell \leftarrow$ top-b_ℓ slots of $\mathcal{K}_\ell$ by $\hat{c}^{(t)}_{\ell j}$ ▷ evicts one slot
end for

A.3 LATENCY AND THROUGHPUT

Fig. 6 shows decode latency and overall throughput for LLaVA-1.5-7B at 2K, 4K and 8K context, generating 256 tokens and averaged over five runs; full KV runs out of memory at 8K. DEKV's decode latency falls as the budget shrinks because its per-step scoring cost scales with the number of retained entries; it is highest at 50% (up to about 1.5× TGV-KV) and approaches TGV-KV at 5%. Overall throughput follows the same trend. At 5%, DEKV is between 0.6% faster and 6.6% slower than TGV-KV, and at 10% it is 4.5–11.7% slower.

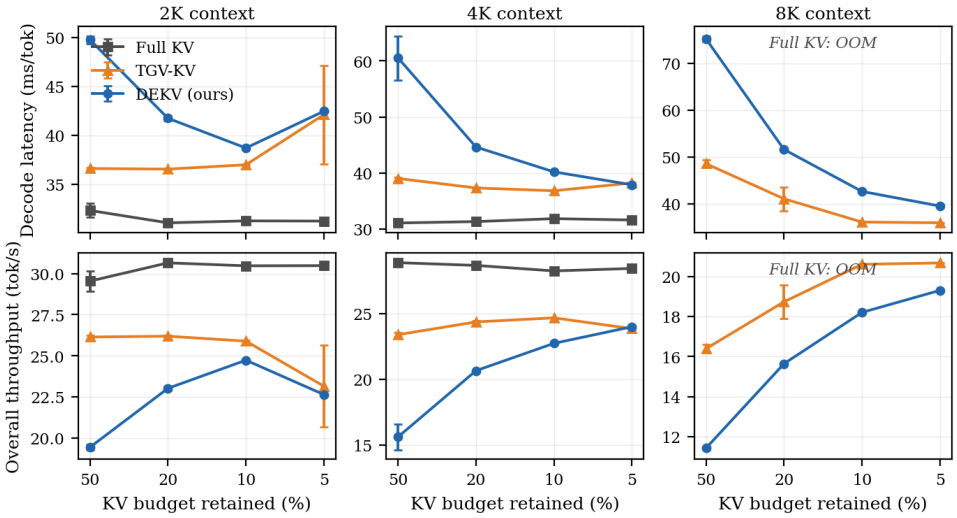

Figure 6: Decode latency and overall throughput vs. KV retention budget, for LLaVA-1.5-7B at 2K, 4K, and 8K context, generating 256 tokens. Full KV applies no eviction and exhausts GPU memory at 8K.

A.4 COMPREHENSIVE EMPIRICAL EVALUATION

In this section, we provide a comprehensive empirical evaluation of the proposed DEKV method against various baseline KV cache compression techniques across multiple model families and strict retention budgets. Specifically, Tables 6 and 7 detail performance on vision-dominant benchmarks (ChartQA, DocVQA, VizWiz, and TextVQA) at varying retention rates (from 50% down to an extreme 5%) across diverse architectures, including several LLaVA variants and Qwen3-VL models. Table 8 repeats this evaluation on Phi-3.5-vision-instruct. This model family features a different vision encoder and tokenization scheme compared to LLaVA and Qwen3-VL. DEKV outperforms TGV-KV in 13 of 16 settings. Furthermore, the performance margin at the 5% budget is markedly larger than on the LLaVA and Qwen3-VL models. Specifically, DEKV achieves gains of $+27.5$ points on ChartQA, $+13.0$ on DocVQA, and $+8.1$ on TextVQA. TGV-KV suffers a sharper degradation on the Phi model. This aligns with its reliance on a hard, text-first retention rule. Complementing this, Tables 9 and 10 offer an in-depth breakdown of generative performance on text-dominant tasks (TextCaps and COCO-Caption) for LLaVA-1.5-7B and Qwen3-VL-8B-Instruct, reporting detailed semantic similarity metrics such as BLEU, CIDEr, METEOR, and ROUGE-L at different compression levels. Collectively, these detailed appendix tables empirically demonstrate DEKV's consistent superiority in preserving full-cache accuracy across varying modalities, task structures, and architectural designs compared to existing state-of-the-art baselines.

Table 6: Comparison of KV cache compression methods on ChartQA and DocVQA across varying KV cache retention rates. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	ChartQA Relaxed Acc.↑				DocVQA ANLS↑			
	50%	20%	10%	5%	50%	20%	10%	5%
LLaVA-1.5-7B Liu et al. (2024a)								
Vanilla	18.0	18.0	18.0	18.0	23.9	23.9	23.9	23.9
StreamingLLM	15.2	14.4	14.3	13.4	17.8	15.3	14.4	13.6
SnapKV	**17.9**	16.6	0.4	0.4	23.0	20.5	1.7	1.7
H_2O	4.6	3.7	3.7	3.7	13.5	7.4	5.9	4.5
ElasticCache	2.6	3.2	3.4	3.9	6.2	3.7	3.6	2.8
PrefixKV	16.0	2.0	1.0	0.9	23.5	7.5	2.9	1.6
TGV-KV	**17.9**	**18.0**	15.5	13.8	23.6	21.8	19.2	14.3
DEKV (Ours)	17.7	17.8	**16.7**	**15.9**	**23.8**	**22.3**	**20.2**	**17.6**
LLaVA-NeXT-Mistral-7B Liu et al. (2024a)								
Vanilla	52.9	52.9	52.9	52.9	63.7	63.7	63.7	63.7
StreamingLLM	39.0	30.0	29.7	30.0	46.3	33.2	29.5	28.1
SnapKV	52.4	49.6	46.2	39.2	63.3	60.8	55.1	44.0
H_2O	50.1	34.5	27.2	27.8	62.9	44.2	35.1	29.4
ElasticCache	44.5	31.1	22.7	18.5	57.9	40.3	27.8	19.9
PrefixKV	52.2	38.0	12.6	10.4	62.6	38.3	18.8	16.3
TGV-KV	52.6	51.8	51.2	49.3	**63.7**	62.8	61.4	58.5
DEKV (Ours)	**52.8**	**52.3**	**51.9**	**50.7**	63.6	**62.9**	**62.4**	**60.6**
LLaVA-OneVision-Qwen2-0.5B Li et al. (2024a)								
Vanilla	59.0	59.0	59.0	59.0	61.9	61.9	61.9	61.9
StreamingLLM	37.7	28.2	26.5	25.8	39.4	28.2	24.4	23.1
SnapKV	53.1	49.6	41.8	30.4	52.2	47.3	39.7	30.0
H_2O	16.8	37.8	33.4	31.0	21.5	41.2	31.0	20.7
ElasticCache	29.8	21.0	13.4	9.8	19.1	12.4	9.7	8.0
PrefixKV	14.0	42.7	24.6	13.7	35.9	25.9	16.5	8.8
TGV-KV	53.5	50.7	47.6	42.5	52.5	48.5	42.8	36.4
DEKV (Ours)	**54.1**	**52.3**	**50.0**	**46.5**	**53.2**	**50.3**	**46.6**	**41.4**
Qwen3-VL-4B-Instruct Bai et al. (2025)								
Vanilla	84.1	84.1	84.1	84.1	93.6	93.6	93.6	93.6
StreamingLLM	79.1	70.6	67.4	66.9	76.0	62.2	51.9	47.0
SnapKV	83.1	74.7	23.5	19.8	93.5	90.9	76.7	12.3
H_2O	82.4	71.0	65.8	63.4	92.7	82.9	67.7	53.6
ElasticCache	80.7	74.6	68.1	58.8	78.6	57.8	42.9	33.1
PrefixKV	83.4	78.1	72.9	65.4	92.2	73.1	52.8	43.0
TGV-KV	83.8	82.5	80.2	67.2	93.5	93.1	91.9	87.3
DEKV (Ours)	**84.0**	**83.6**	**80.9**	**78.3**	**93.6**	**93.4**	**92.6**	**89.0**
Qwen3-VL-8B-Instruct Bai et al. (2025)								
Vanilla	85.3	85.3	85.3	85.3	95.1	95.1	95.1	95.1
StreamingLLM	80.8	73.8	70.8	72.0	81.2	64.8	54.6	49.6
SnapKV	85.0	78.1	42.3	39.3	94.9	91.2	80.1	22.6
H_2O	84.0	78.0	72.7	67.6	94.4	86.5	72.0	58.1
ElasticCache	78.8	68.5	67.0	67.7	87.1	71.4	57.0	45.3
PrefixKV	84.7	78.9	74.2	71.2	93.9	72.7	50.3	45.4
TGV-KV	**85.3**	84.4	82.1	73.1	**95.0**	**94.7**	93.2	88.0
DEKV (Ours)	85.2	**84.9**	**83.0**	**79.5**	94.8	**94.7**	**93.8**	**90.4**

Table 7: Comparison of KV cache compression methods on VizWiz and TextVQA across varying KV cache retention rates. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	VizWiz$^{Acc.\uparrow}$				TextVQA$^{Acc.\uparrow}$			
	50%	**20%**	**10%**	**5%**	**50%**	**20%**	**10%**	**5%**
LLaVA-1.5-7B Liu et al. (2024a)								
Vanilla	54.4	54.4	54.4	54.4	47.9	47.9	47.9	47.9
StreamingLLM	53.0	52.5	52.1	**48.2**	40.5	35.4	33.8	32.5
SnapKV	54.1	53.6	6.5	5.0	47.3	44.6	9.2	9.2
H_2O	13.6	10.0	8.9	7.6	34.3	27.6	24.7	21.0
ElasticCache	4.7	6.1	6.5	9.3	20.8	14.6	13.1	9.1
PrefixKV	22.5	3.9	1.4	0.5	47.5	19.6	12.4	4.8
TGV-KV	**54.4**	54.0	52.3	31.2	47.8	47.4	44.5	33.9
DEKV (Ours)	**54.4**	**54.3**	**53.6**	19.2	**48.0**	**47.7**	**46.6**	**43.4**
LLaVA-NeXT-Mistral-7B Liu et al. (2024a)								
Vanilla	63.7	63.7	63.7	63.7	65.7	65.7	65.7	65.7
StreamingLLM	62.4	60.8	60.5	60.5	58.0	47.4	44.6	43.1
SnapKV	63.5	63.0	62.5	61.5	65.3	63.0	59.8	54.9
H_2O	52.0	26.5	24.2	24.0	65.1	60.4	53.6	48.7
ElasticCache	26.3	24.2	20.4	18.9	61.7	47.3	35.5	31.7
PrefixKV	63.5	62.8	52.5	19.2	65.1	58.7	39.9	34.5
TGV-KV	**63.7**	**63.6**	**63.6**	63.2	**65.7**	65.5	65.1	64.0
DEKV (Ours)	63.6	63.5	63.5	**63.4**	**65.7**	**65.7**	**65.5**	**65.0**
LLaVA-OneVision-Qwen2-0.5B Li et al. (2024a)								
Vanilla	47.4	47.4	47.4	47.4	64.8	64.8	64.8	64.8
StreamingLLM	43.8	42.1	41.7	34.3	51.5	39.9	36.3	41.4
SnapKV	45.5	44.2	43.0	42.0	60.4	54.3	47.5	40.1
H_2O	19.5	34.0	31.3	30.0	19.2	56.0	51.7	43.2
ElasticCache	29.4	29.7	32.8	21.4	41.9	35.2	27.9	31.3
PrefixKV	23.3	28.8	29.2	30.9	40.7	48.4	31.4	23.6
TGV-KV	45.9	44.7	43.7	43.1	61.5	57.0	52.0	46.0
DEKV (Ours)	**46.3**	**45.2**	**44.3**	**43.7**	**62.6**	**59.2**	**55.5**	**50.6**
Qwen3-VL-4B-Instruct Bai et al. (2025)								
Vanilla	68.5	68.5	68.5	68.5	80.6	80.6	80.6	80.6
StreamingLLM	66.5	63.8	61.5	59.6	72.3	63.8	56.5	51.7
SnapKV	68.0	64.9	62.1	48.4	80.3	75.8	58.1	27.0
H_2O	67.5	64.2	62.6	54.8	80.2	74.5	65.2	55.7
ElasticCache	55.0	54.7	54.0	52.8	67.7	57.4	47.0	39.8
PrefixKV	67.2	59.2	55.5	49.5	79.9	67.9	53.0	47.6
TGV-KV	68.4	67.8	66.2	**63.1**	80.6	80.1	78.1	70.4
DEKV (Ours)	**68.6**	**68.0**	**66.8**	62.7	**80.9**	**80.4**	**79.3**	**72.6**
Qwen3-VL-8B-Instruct Bai et al. (2025)								
Vanilla	70.0	70.0	70.0	70.0	82.1	82.1	82.1	82.1
StreamingLLM	69.2	66.5	63.5	61.3	76.9	68.7	61.2	54.0
SnapKV	69.8	67.4	64.9	53.8	81.9	78.6	64.0	40.2
H_2O	69.7	67.6	62.0	56.6	81.6	77.5	66.7	58.5
ElasticCache	48.7	51.7	53.4	53.6	70.0	64.7	57.2	48.1
PrefixKV	68.8	59.7	57.1	55.7	81.4	69.3	54.0	49.1
TGV-KV	69.8	69.2	67.6	64.4	82.1	81.8	79.4	72.0
DEKV (Ours)	**70.0**	**69.8**	**69.2**	**65.4**	**82.3**	**82.0**	**80.9**	**74.8**

Table 8: Comparison of KV cache compression methods on *Phi-3.5-vision-instruct* Abdin et al. (2024) across four VQA benchmarks at varying KV cache retention rates. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	ChartQA$^{\text{Relaxed Acc.}\uparrow}$				DocVQA$^{\text{ANLS}\uparrow}$			
	50%	**20%**	**10%**	**5%**	**50%**	**20%**	**10%**	**5%**
Vanilla	72.1	72.1	72.1	72.1	70.3	70.3	70.3	70.3
TGV-KV	**71.7**	69.8	66.9	36.5	69.6	66.5	60.6	41.3
DEKV (Ours)	**71.7**	**70.8**	**69.8**	**64.0**	**69.9**	**67.4**	**63.7**	**54.3**

Methods	VizWiz$^{\text{Acc.}\uparrow}$				TextVQA$^{\text{Acc.}\uparrow}$			
	50%	**20%**	**10%**	**5%**	**50%**	**20%**	**10%**	**5%**
Vanilla	57.7	57.7	57.7	57.7	61.1	61.1	61.1	61.1
TGV-KV	**57.6**	56.8	55.2	52.5	**61.1**	59.5	55.6	42.8
DEKV (Ours)	**57.6**	**56.9**	**55.8**	**53.8**	61.0	**59.8**	**57.3**	**50.9**

Table 9: Full performance comparison on text-dominant benchmarks, LLaVA-1.5-7B Liu et al. (2024a). We report BLEU-x (B-x), CIDEr (C), METEOR (M), and ROUGE-L (R). CIDEr scores are reported as raw values. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	**TextCaps Sidorov et al. (2020)**							**COCO-Caption Lin et al. (2015)**						
	B-1	**B-2**	**B-3**	**B-4**	**C**	**M**	**R**	**B-1**	**B-2**	**B-3**	**B-4**	**C**	**M**	**R**
Retain 50% KV														
Vanilla	0.71	0.53	0.38	0.27	1.00	0.23	0.47	0.73	0.56	0.41	0.29	1.08	0.28	0.55
StreamingLLM	0.68	0.49	0.34	0.24	0.78	0.21	0.44	0.73	0.56	0.40	0.29	1.04	0.27	0.54
SnapKV	**0.71**	0.52	0.37	0.26	0.97	**0.23**	**0.47**	**0.74**	**0.57**	**0.42**	**0.30**	**1.09**	**0.28**	**0.55**
H2O	0.08	0.05	0.03	0.02	0.14	0.08	0.17	0.07	0.05	0.03	0.02	0.15	0.09	0.17
ElasticCache	0.04	0.02	0.01	0.01	0.03	0.05	0.10	0.04	0.03	0.01	0.01	0.03	0.05	0.10
PrefixKV	0.70	0.52	**0.38**	**0.27**	0.94	0.22	**0.47**	0.15	0.11	0.08	0.05	0.63	0.14	0.34
TGV-KV	**0.71**	**0.53**	**0.38**	**0.27**	1.00	**0.23**	**0.47**	**0.74**	**0.57**	**0.42**	**0.30**	**1.09**	**0.28**	**0.55**
DEKV (Ours)	**0.71**	**0.53**	**0.38**	**0.27**	**1.01**	**0.23**	**0.47**	0.73	**0.57**	0.41	**0.30**	**1.09**	**0.28**	**0.55**
Retain 20% KV														
Vanilla	0.71	0.53	0.38	0.27	1.00	0.23	0.47	0.73	0.56	0.41	0.29	1.08	0.28	0.55
StreamingLLM	0.64	0.44	0.30	0.19	0.53	0.19	0.41	0.71	0.53	0.38	0.26	0.93	0.25	0.52
SnapKV	0.68	0.49	0.34	0.24	0.82	0.22	0.45	0.73	0.56	0.41	0.29	1.06	0.27	**0.55**
H2O	0.04	0.02	0.01	0.00	0.02	0.04	0.09	0.04	0.02	0.01	0.00	0.02	0.04	0.10
ElasticCache	0.04	0.02	0.01	0.00	0.01	0.04	0.07	0.04	0.02	0.01	0.00	0.01	0.04	0.09
PrefixKV	0.06	0.04	0.02	0.01	0.12	0.06	0.14	0.05	0.03	0.02	0.01	0.06	0.06	0.15
TGV-KV	0.70	0.51	0.36	0.25	0.88	0.22	0.45	**0.74**	**0.57**	0.41	0.29	1.07	**0.28**	**0.55**
DEKV (Ours)	**0.71**	**0.53**	**0.38**	**0.27**	**0.99**	**0.23**	**0.47**	**0.74**	**0.57**	**0.42**	**0.30**	**1.10**	**0.28**	**0.55**
Retain 10% KV														
Vanilla	0.71	0.53	0.38	0.27	1.00	0.23	0.47	0.73	0.56	0.41	0.29	1.08	0.28	0.55
StreamingLLM	0.61	0.41	0.26	0.17	0.40	0.17	0.39	0.67	0.47	0.32	0.20	0.68	0.20	0.47
SnapKV	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.01
H2O	0.03	0.02	0.01	0.00	0.01	0.04	0.07	0.04	0.02	0.01	0.00	0.01	0.04	0.08
ElasticCache	0.04	0.02	0.01	0.00	0.01	0.03	0.07	0.04	0.02	0.01	0.00	0.01	0.04	0.08
PrefixKV	0.04	0.02	0.01	0.00	0.01	0.04	0.08	0.04	0.02	0.01	0.00	0.01	0.04	0.08
TGV-KV	0.64	0.45	0.30	0.20	0.64	0.19	0.42	0.71	0.54	0.39	0.27	0.95	0.26	0.53
DEKV (Ours)	**0.69**	**0.50**	**0.36**	**0.25**	**0.87**	**0.22**	**0.45**	**0.75**	**0.57**	**0.42**	**0.30**	**1.09**	**0.28**	**0.55**

Table 10: Full performance comparison on text-dominant benchmarks, Qwen3-VL-8B-Instruct Bai et al. (2025). We report BLEU-x (B-x), CIDEr (C), METEOR (M), and ROUGE-L (R). CIDEr scores are reported as raw values. Higher values indicate better performance. **Bold** indicates the best result among compressed methods (excluding Vanilla) per column. **DEKV (Ours)** is our method.

Methods	**TextCaps Sidorov et al. (2020)**							**COCO-Caption Lin et al. (2015)**						
	B-1	**B-2**	**B-3**	**B-4**	**C**	**M**	**R**	**B-1**	**B-2**	**B-3**	**B-4**	**C**	**M**	**R**
Retain 50% KV														
Vanilla	0.47	0.30	0.20	0.14	0.34	0.25	0.39	0.48	0.31	0.19	0.12	0.27	0.25	0.42
StreamingLLM	0.46	0.29	0.19	0.12	0.34	0.24	0.38	**0.48**	**0.31**	0.19	0.11	0.29	0.25	**0.42**
SnapKV	**0.47**	0.29	**0.20**	0.13	0.34	0.24	0.38	0.47	0.29	0.18	0.11	0.29	0.24	0.40
H2O	**0.47**	0.30	**0.20**	**0.14**	0.32	**0.25**	**0.39**	0.47	0.30	0.18	**0.12**	**0.32**	**0.25**	0.41
ElasticCache	0.44	0.28	0.19	0.12	0.27	0.24	0.37	0.45	0.28	0.17	0.10	0.21	0.24	0.40
PrefixKV	0.46	0.30	**0.20**	0.13	**0.38**	0.25	0.38	**0.48**	0.30	**0.19**	**0.12**	0.28	**0.25**	**0.42**
TGV-KV	**0.47**	0.30	**0.20**	**0.14**	0.33	**0.25**	**0.39**	**0.48**	**0.31**	**0.19**	**0.12**	0.27	**0.25**	**0.42**
DEKV (Ours)	**0.47**	**0.31**	**0.20**	**0.14**	0.34	**0.25**	**0.39**	**0.48**	**0.31**	**0.19**	**0.12**	0.27	**0.25**	**0.42**
Retain 20% KV														
Vanilla	0.47	0.30	0.20	0.14	0.34	0.25	0.39	0.48	0.31	0.19	0.12	0.27	0.25	0.42
StreamingLLM	0.44	0.27	0.17	0.11	0.32	0.21	0.36	0.45	0.28	0.17	0.10	0.29	0.22	0.40
SnapKV	0.45	0.29	0.18	0.11	0.34	0.23	0.36	0.42	0.27	0.16	0.10	0.24	0.21	0.36
H2O	0.44	0.28	0.19	**0.14**	0.34	0.23	**0.39**	0.40	0.25	0.16	0.09	0.25	0.23	0.39
ElasticCache	0.42	0.26	0.16	0.11	0.27	0.21	0.35	0.42	0.26	0.16	0.09	0.23	0.22	0.38
PrefixKV	0.42	0.26	0.16	0.10	0.28	0.22	0.35	0.46	0.29	0.18	0.11	0.30	0.23	0.41
TGV-KV	0.46	0.29	0.19	0.13	0.35	0.24	0.38	0.47	0.30	0.19	0.11	0.29	**0.25**	**0.42**
DEKV (Ours)	**0.47**	**0.30**	**0.20**	**0.14**	**0.36**	**0.25**	**0.39**	**0.48**	**0.31**	**0.20**	**0.12**	**0.31**	**0.25**	**0.42**
Retain 10% KV														
Vanilla	0.47	0.30	0.20	0.14	0.34	0.25	0.39	0.48	0.31	0.19	0.12	0.27	0.25	0.42
StreamingLLM	0.40	0.23	0.14	0.08	0.25	0.18	0.33	0.39	0.23	0.13	0.07	0.18	0.18	0.35
SnapKV	0.40	0.24	0.14	0.09	0.27	0.17	0.31	0.42	0.24	0.13	0.07	0.16	0.17	0.36
H2O	0.40	0.23	0.13	0.10	0.29	0.19	0.36	0.46	0.25	0.16	0.10	0.26	0.20	0.38
ElasticCache	0.39	0.23	0.13	0.08	0.20	0.18	0.32	0.38	0.22	0.13	0.07	0.16	0.19	0.36
PrefixKV	0.35	0.20	0.11	0.06	0.14	0.16	0.30	0.40	0.24	0.14	0.08	0.18	0.19	0.36
TGV-KV	0.41	0.25	0.16	0.10	0.31	0.21	0.36	0.44	0.27	0.17	0.10	0.27	0.22	**0.40**
DEKV (Ours)	**0.45**	**0.29**	**0.19**	**0.12**	**0.36**	**0.23**	**0.38**	**0.45**	**0.29**	**0.18**	**0.11**	**0.31**	**0.23**	**0.40**

A.5 Per-Device KV Cache under Multi-GPU Inference

When a model is split across GPUs, each layer's KV cache stays on the device that hosts it, so an eviction policy also decides how memory is spread across devices. We split the decoder evenly across four A100 GPUs (8 layers per GPU for LLaVA-1.5-7B, 9 for Qwen3-VL-4B) and run TGV-KV and DEKV on 200 random samples each from ChartQA, TextVQA and TextCaps at 20%, 10% and 5% budgets. We report the mean KV cache left on each GPU after pruning. Both methods keep the same total cache, so only its distribution differs. On LLaVA-1.5-7B (Figure 7), TGV-KV piles the cache onto GPU0, which holds about 2.5× more than GPU3 at every budget. DEKV spreads it evenly (6.6–8.9 MiB per GPU at 10%), cutting the largest device's cache by 28–35%. On Qwen3-VL-4B (Figure 8), TGV-KV peaks on GPU2; DEKV lowers this peak, making it nearly flat at 5%.

This matters because the device with the highest memory usage determines how much memory every device must reserve. A similar issue arises in pipeline-parallel training, where uneven memory usage across stages can create bottlenecks (Qi et al., 2024) and waste up to 40% of available GPU memory (Peng et al., 2025). By allocating memory based on marginal damage, DEKV reduces this imbalance and is therefore well suited for multi-GPU deployment.

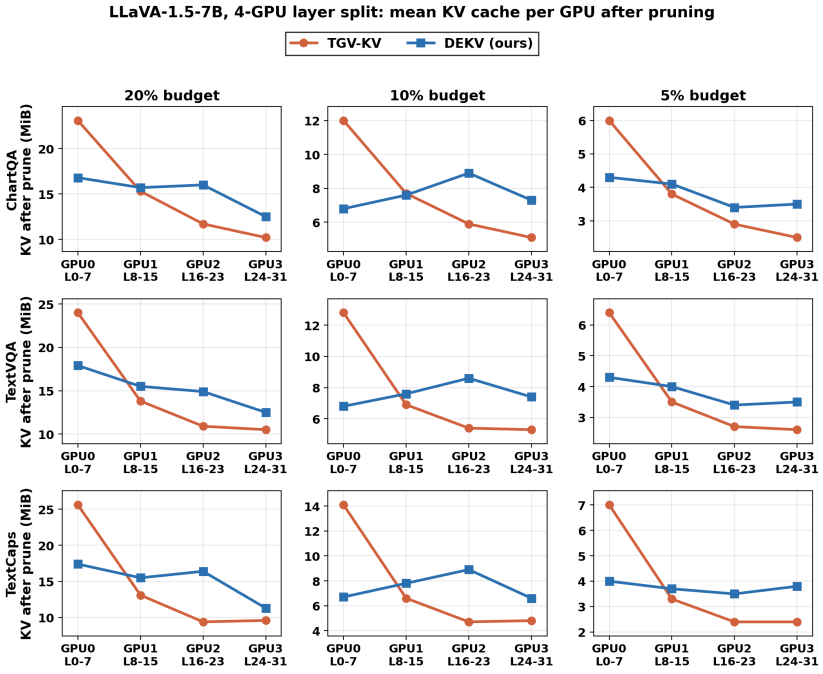

Figure 7: Per-GPU KV cache after pruning for the LLaVA-1.5-7B split across four GPUs. TGV-KV concentrates the cache on GPU0, while DEKV spreads it evenly.

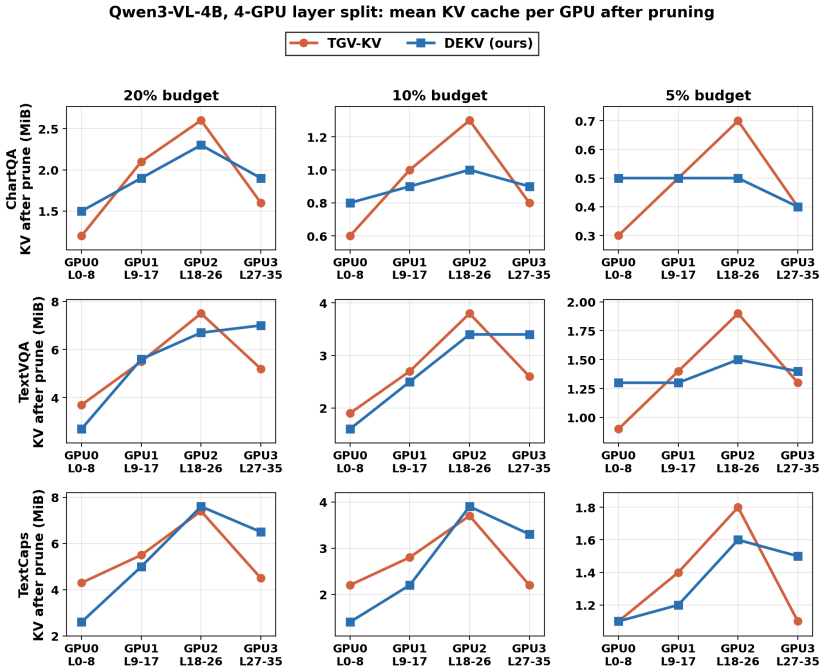

Figure 8: Per-GPU KV cache after pruning for Qwen3-VL-4B split across four GPUs. DEKV lowers TGV-KV's peak on GPU2 and is nearly flat at 5%.

A.6 HYPERPARAMETERS AND EVALUATION SETTINGS

We apply a single, unified configuration for all DEKV experiments (Table 11). To determine which tokens to retain, DEKV calculates a "damage" score, assigning a $10\times$ weight ($\rho = 10$) to the final four queries ($q = 4$) since they directly precede and best predict the upcoming text. During the initial prefill, we protect a minimal set of essential items: four sink tokens ($s = 4$), the final prompt token, and a baseline of eight vision tokens ($c_{\text{vis}} = 8$, analyzed in Sec. 4.3). During decoding, each layer strictly maintains its memory budget (b_ℓ) by evicting one outdated token per step (Algorithm 2) to ensure the total cache stays within budget B. Finally, a damage decay factor ($\gamma = 0.95$) ensures the eviction strategy smoothly transitions from relying on the initial prompt to dynamically adapting to newly generated queries.

Table 11: DEKV hyperparameters and evaluation settings. The same values are used for all models, benchmarks, and budgets unless noted.

SYMBOL	MEANING	VALUE
Prefill		
r	KV retention ratio	50 / 20 / 10 / 5%
s	leading sink tokens, never evicted	4
q	final text queries that are up-weighted	4
ρ	up-weighting factor	10
w_i	query weights in Eq. 4; $Z = N_t - q + \rho q$, so $\sum_i w_i = 1$	ρ/Z for the last q text queries, $1/Z$ for the rest
c_{vis}	per-layer vision floor (counts toward B)	8
Decode		
γ	decode decay (Eq. 7)	0.95