EFFIVLM-BENCH: A Comprehensive Benchmark for Evaluating Training-Free Acceleration in Large Vision-Language Models

Zekun Wang[1*], **Minghua Ma**[1*], **Zexin Wang**[1*], **Rongchuan Mu**[1*]
Liping Shan[2], **Ming Liu**[1,2], **Bing Qin**[1,2]
[1]Harbin Institute of Technology, Harbin, China
[2]Pengcheng Laboratory, Shenzhen, China.
[3]Du Xiaoman Science Technology Co., Ltd, Beijing, China
{zkwang,mhma,zxwang,rcmu,mliu,qinb}@ir.hit.edu.cn
Project Page: https://effivlm-bench.github.io/

Abstract

Large Vision-Language Models (LVLMs) have achieved remarkable success, yet their significant computational demands hinder practical deployment. While efforts to improve LVLM efficiency are growing, existing methods lack comprehensive evaluation across diverse backbones, benchmarks, and metrics. In this work, we systematically evaluate mainstream acceleration techniques for LVLMs, categorized into token and parameter compression. We introduce EFFIVLM-BENCH, a unified framework for assessing not only absolute performance but also generalization and loyalty, while exploring Pareto-optimal trade-offs. Our extensive experiments and in-depth analyses offer insights into optimal strategies for accelerating LVLMs. We open-source code and recipes for EFFIVLM-BENCH to foster future research.

1 Introduction

Large vision-language models (LVLMs) (OpenAI, 2024; Team et al., 2024; Lu et al., 2024a; Wang et al., 2024a; Team, 2025) have rapidly advanced, transforming multimodal AI and showing great potential for real-world applications (Anthropic, 2024; Xu et al., 2024a; OpenAI, 2025; Li et al., 2025). However, their remarkable capabilities are often overshadowed by massive computational and memory costs, severely hindering practical deployment. To this end, some studies propose more efficient architectures (Chu et al., 2023; Zhou et al., 2024; Yao et al., 2024; Liu et al., 2024d) or incorporate distillation (Shu et al., 2024; Cai et al., 2024a; Li et al., 2024d) to improve efficiency. But these typically demand full retraining, incurring substantial overhead (An et al., 2024).

As a result, there has been a growing focus on training-free acceleration methods for LVLMs, which are more economical and can be broadly classified into two representative categories: **token compression**, eliminating redundant tokens in inputs (Chen et al., 2024b; Zhang et al., 2024b) or KV cache (Wan et al., 2024; Tu et al., 2024), and **parameter compression**, which reduces parameter size by pruning (Sung et al., 2023) or quantization (Lin et al., 2024; Yu et al., 2025). However, the scope of performance evaluation for these methods remains limited in several key aspects: (1) *Outdated Model Architectures*: Evaluations are ofter stuck on outdated models like LLaVA (Liu et al., 2023) or LLaVA-v1.5 (Liu et al., 2024a), without considering state-of-the-art LVLMs with the dynamic resolution processing mechanism (Li et al., 2024b; Wang et al., 2024a; Chen et al., 2024d). (2) *Limited Benchmarks*: Assessments typically use general VQA tasks, neglecting more challenging benchmarks that require advanced capabilities such as OCR or long-context generation. (3) *Narrow Evaluation Metrics*: The focus is solely on absolute performance, overlooking other critical metrics such as generational quality and loyalty of compression methods. This narrow focus leaves a significant gap in understanding how these techniques generalize across broader scenarios, which severely limits their practical use. Furthermore, there is a lack of systematic exploration into the crucial trade-offs between performance and efficiency (actual inference time), which is essential for real-world LVLM deployment.

To address these limitations, we propose EFFIVLM-BENCH, a unified evaluation framework for systematically assessing training-free acceleration methods of LVLMs. EFFIVLM-BENCH spans a wide range of representative model architectures and tasks, employing comprehensive metrics for performance, generalization, loyalty, and efficiency. With EFFIVLM-BENCH, we conduct a thorough comparison of mainstream token and parameter compression methods. We explore Pareto-optimal performance-efficiency trade-offs and offer a nuanced understanding of their strengths and limita-

[*]Equal Contribution.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 25546–25572

tions. We hope our EFFIVLM-BENCH can provide the research community with insightful perspectives on LVLM acceleration, paving the way for more effective and sustainable deployment.

2 Related Work

Recent advancements in LVLMs (OpenAI, 2024; Team et al., 2024; Li et al., 2024b; Wang et al., 2024a; Deitke et al., 2024; Chen et al., 2024d; Team, 2025) have led to significant improvements in various multimodal tasks. Despite their remarkable capabilities, the computational overhead remains a critical bottleneck of LVLMs for real-world deployment. Some LVLM studies introduce more efficient model architectures (Chu et al., 2023; Zhou et al., 2024; Yao et al., 2024; Chen et al., 2024a; Luo et al., 2024) or knowledge distillation (Wang et al., 2021, 2023a; Li et al., 2024d) to enhance efficiency. However, these methods typically require training from scratch or retraining, which incurs significant computational overhead. Thus, increasing attention has been paid to training-free acceleration methods for LVLMs, the focus of this paper, which can be broadly categorized into token compression and parameter compression.

Token Compression The lengthy input tokens of LVLM impose a substantial computational burden, mainly due to the quadratic scaling of computational costs with the number of input tokens in the attention. Some studies focus on directly **token pruning**, which prune the redundant visual tokens during the forward process (Li et al., 2024c; Wang et al., 2023b; Cha et al., 2024; Huang et al., 2024). As for training-free methods, LLaVA-PruMerge (Shang et al., 2024) first prunes uninformative tokens in visual encoder and then merges the remaining ones using KNN techniques. Yang et al. (2024) also dynamically prunes the tokens in the visual encoder and then merges the pruned tokens to the next layer, while Chen et al. (2024b) and Zhang et al. (2024b) leverage textual information to guide visual token pruning in the initial layers of LLMs. Another line of work mainly focuses on **KV cache compression**. The inference time during LVLM generation is memory-bound, primarily due to the substantial memory consumption of KV caches, particularly when processing high-resolution images. KV cache compression leverages attention sparsity to select fewer key-value pairs, thereby reducing memory overhead and enhancing inference speed. Recent efforts in KV cache compression have evolved from techniques for LLMs (Xiao et al., 2023; Zhang et al., 2023b; Li et al., 2024e; Ge et al., 2024; Cai et al., 2024b; Xu et al., 2024b) to methods tailored for LVLMs. VL-Cache (Tu et al., 2024) introduces a modality-aware strategy that dynamically allocates cache budgets across layers and incorporates a token scoring mechanism tailored to the unique roles of visual and textual tokens. Similarly, LOOK-M (Wan et al., 2024) adopts a *look-once* optimization that minimizes redundant cache entries through eviction and KV pair merging. Both methods show that targeted compression can significantly reduce memory usage and accelerate decoding in multimodal long-context scenarios while maintaining performance.

Parameter Compression Besides reducing the lengthy inputs, compressing the large model parameter size also benefits the efficiency of LVLM. Weight pruning removes redundant parameters to reduce parameter size and computations while preserving accuracy (Sun et al., 2023; Frantar and Alistarh, 2023; Wang et al., 2023a; An et al., 2024; Wang et al., 2024b). Quantization(Frantar et al., 2022; Lin et al., 2024) converts full-precision weights and activations into lower-precision formats (e.g., int8 or int4). Most existing work focuses on compressing the parameters of the LLM backbone in LVLMs, while recent efforts have also begun exploring the compression of the visual encoder (Yu et al., 2025). In this work, we concentrate on reducing the parameters of LLMs, which is a key bottleneck of LVLM efficiency.

3 EFFIVLM-BENCH

In this section, we introduce the details of EFFIVLM-BENCH, including metrics (Section 3.1) and tasks with model architectures (Section 3.2).

3.1 Metrics

Let c denote the compression method, m the target model, and b the target benchmark. $\mathbb{E}(\cdot)$ denotes the mean operation.

Performance For each method c on model m, we define the overall performance $\text{OP}^{m,c}$ as:

$$\text{OP}^{m,c} = \sqrt{\frac{1}{B}\sum_{b=1}^{B}\mathbb{E}\left(\frac{EM_b^{m,c}}{EM_b^m}\right)^2}, \quad (1)$$

where $EM_b^{m,c}$ is the evaluation performance metric (e.g., accuracy) and EM_b^m represents the corre-

sponding metric for the original model. B denotes the number of benchmarks.

Generalization To evaluate the generalization of compression method c, we define the metric OG^c as the coefficient of variation of performance across benchmarks and models:

$$\text{OG}^c = \frac{\sigma_b\left(\mathbb{E}_m\left[\frac{EM_b^{m,c}}{EM_b^m}\right]\right)}{\mathbb{E}_{b,m}\left[\frac{EM_b^{m,c}}{EM_b^m}\right]} \tag{2}$$

where $\sigma_b(\cdot)$ represents standard deviation across benchmarks. A lower OG^c signifies more consistent relative performance (better generalization), while a higher value suggests greater sensitivity to variations in models or benchmarks.

Loyalty Loyalty (OL^c) measures how well a compression method c preserves an original model's predictions (P_b^m) with its compressed version ($P_b^{m,c}$). Ideally, compression should maintain the original model's behavior, avoiding new biases or unexpected performance alterations (Xu et al., 2021). Specifically, we define the loyalty metric as:

$$\text{OL}^c = \mathbb{E}_{b,m}\left[\mathbb{I}(P_b^{m,c}, P_b^m)\right] \tag{3}$$

where $\mathbb{I}(P_1, P_2)$ is an agreement function between predictions. Higher OL^c indicates better loyalty.

Efficiency Actual inference time is our efficiency measure for a compression method c. Unlike FLOPs or parameter counts, it directly reflects real-world latency, which varies significantly with model architecture (e.g., depth vs. width) even for similar theoretical complexities. Specifically, for model m using compression method c on benchmark b, we measure the speedup of inference time per fixed number of samples:

$$\text{OE}^c = \mathbb{E}_{b,m}\left[\frac{T_b^m}{T_b^{m,c}}\right], \tag{4}$$

where $T^{*,c}$ and T^* represent the latency of compressed and original models, respectively. More details of EFFIVLM-BENCH are shown in Appendix A.4.

3.2 Tasks and Models

EFFIVLM-BENCH evaluates across 17 widely-used benchmarks, including DocVQA(Mathew et al., 2020), ChartQA(Masry et al., 2022), TextVQA(Singh et al., 2019), OCRBench(Liu et al., 2024c), AI2D(Kembhavi et al., 2016), GQA(Hudson and Manning, 2019), MMMU(Yue et al., 2024), MME(Fu et al., 2024), RealworldQA(x.ai), MMStar(Chen et al., 2024c), MathVista(Lu et al., 2024b), LLaVA-Wilder(Li et al., 2024a), MMBench(Liu et al., 2024b), MMVet(Yu et al., 2023), ImageDC(Li et al., 2024a), MP-DocVQA(Tito et al., 2022) and MovieChat(Song et al., 2023). These cover diverse tasks—from document understanding and chart interpretation to real-world QA—across single-image, multi-image, and video scenarios (details in Appendix A.1). EFFIVLM-BENCH includes three frontier LVLMs: LLaVA-OneVision(OV)-7B (Li et al., 2024b), Qwen2-VL-7B (Wang et al., 2024a), and InternVL2.5-38B (Chen et al., 2024d), spanning diverse model sizes for comprehensive evaluation. Furthermore, the modular design of EFFIVLM-BENCH allows straightforward extension to support new tasks and emerging LVLMs.

4 Results

4.1 Token Compression

Setup We evaluate token compression effectiveness by examining two mainstream approaches: (1) token pruning, including FastV (Chen et al., 2024b), VisionZip (Yang et al., 2024), and PruMerge+ (Shang et al., 2024), which eliminates redundant visual tokens. (2) KV cache compression, including streamingLLM (Xiao et al., 2023), H2O (Zhang et al., 2023b), SnapKV (Li et al., 2024e), PyramidKV (Cai et al., 2024b), LOOK-M (Wan et al., 2024), and VL-Cache (Tu et al., 2024). Notably, the last two methods are specifically designed for LVLMs. More implementation details are in Appendix A.3 and A.4. We focus on retention budgets up to 40%, as methods generally maintain original performance at higher budgets. Main results of token pruning and KV cache compression are shown in Table 1 and Figure 1. More results of KV cache compression are provided in Appendix A.5 and Table 15.

Observation 1 **Token compression performance is task-dependent and shows significant sensitivity to benchmark and model.** Most methods are stable at higher budgets but degrade sharply at 1%, especially on tasks requiring fine-grained visual detail (e.g., OCRBench) or long outputs (e.g., LLaVA-Wilder, ImageDC). For token pruning on a 1% budget, pruning tokens within the visual encoder (e.g., VisionZip and PruMerge+) consistently

Models	Budgets	Methods \ Benchmarks	DocVQA	ChartQA	TextVQA	OCRBench	AI2D	GQA	MMMU	MME	RealworldQA	MMStar	MathVista	LLaVA-Wilder	MMBench	MMVet	ImageDC	OP
LLaVA-OneVision-7B	1%	FastV	8	14.00	9.18	27	66.35	35.04	40.89	697.7	36.86	28.97	36.40	49.10	27.69	12.30	20.95	0.48
		VisionZip	**35**	**35.16**	44.48	**194**	**72.11**	53.69	42.56	**1704.2**	53.86	41.30	39.30	**71.20**	**74.10**	28.60	**87.50**	**0.75**
		PruMerge+	27	34.88	**44.66**	121	71.08	**54.18**	**43.44**	1639.2	**58.16**	**42.81**	39.20	63.60	73.09	**34.50**	76.60	0.74
	10%	FastV	48	43.16	52.37	190	72.57	49.63	**45.33**	1669.4	53.86	44.63	40.30	66.90	70.12	31.30	77.75	0.76
		VisionZip	**56**	**49.88**	57.26	**352**	77.97	**58.57**	44.11	**1915.0**	59.87	45.45	**45.00**	**71.80**	**78.86**	36.60	**87.50**	**0.84**
		PruMerge+	37	40.96	55.59	203	74.77	58.54	44.44	1872.7	**61.17**	**47.56**	43.50	65.90	78.47	**37.00**	85.35	0.81
	40%	FastV	**80**	**69.20**	72.48	488	86.23	60.56	46.33	1937.2	62.75	**53.91**	50.50	70.50	**81.22**	47.70	86.35	**0.94**
		VisionZip	72	67.04	68.21	**500**	83.84	61.23	46.11	1956.8	63.01	51.17	52.60	**71.60**	80.43	**48.30**	**87.55**	0.93
		PruMerge+	49	51.40	67.79	382	79.82	**61.78**	45.55	1924.0	**64.70**	53.38	48.00	68.10	80.88	46.50	85.55	0.88
	100%	Original	87	80.00	74.79	595	89.96	61.92	45.44	1974.1	65.88	58.75	58.20	71.40	83.12	55.00	87.25	1.00
Qwen2-VL-7B	1%	FastV	19	12.24	22.45	71	65.22	39.79	44.67	1330.0	41.18	26.83	33.70	51.40	21.91	18.10	24.60	0.51
		VisionZip	**52**	34.76	**61.97**	**187**	**70.23**	48.64	**46.56**	1702.0	**58.56**	**35.77**	39.10	60.80	48.99	**39.00**	**66.15**	**0.70**
		PruMerge+	45	**38.80**	60.16	162	68.04	**51.45**	45.67	**1772.2**	56.86	35.70	**39.50**	**63.00**	**50.67**	34.60	63.10	0.69
	10%	FastV	68	31.04	68.72	272	73.28	50.33	48.00	1750.0	55.82	40.09	40.40	66.10	63.06	42.90	78.70	0.76
		VisionZip	**75**	**57.88**	**72.27**	**318**	**76.06**	55.08	**48.89**	1860.0	60.65	42.56	**44.50**	**67.20**	63.06	**45.10**	79.85	**0.81**
		PruMerge+	61	48.76	68.16	283	70.98	**56.77**	47.67	**1981.2**	**62.09**	**43.17**	43.70	**67.20**	**68.27**	42.40	**81.80**	0.80
	40%	FastV	**92**	67.40	**79.52**	532	**86.53**	58.35	49.22	**2185.0**	**66.93**	50.74	47.10	**71.60**	75.50	59.40	**85.00**	0.92
		VisionZip	91	**74.04**	79.25	**571**	85.10	60.21	**49.44**	2150.0	63.79	51.09	**53.40**	70.50	73.76	**59.50**	84.70	**0.93**
		PruMerge+	85	68.96	76.97	490	75.68	**60.64**	49.00	2169.7	66.27	**51.94**	51.70	70.00	**75.56**	54.90	84.95	0.91
	100%	Original	95	81.56	81.82	813	91.02	62.30	50.77	2327.8	66.53	57.11	58.30	72.70	78.25	67.40	86.35	1.00
InternVL2.5-38B	1%	FastV	11	14.88	10.64	23	67.64	35.49	51.22	1264.1	44.44	29.64	34.60	52.70	22.64	7.10	20.20	0.47
		VisionZip	**15**	**18.56**	**21.21**	**57**	**69.04**	42.34	**53.89**	1612.1	**52.67**	35.25	32.90	**58.80**	45.79	**23.30**	44.35	0.55
		PruMerge+	12	16.44	15.20	36	68.91	**47.43**	53.44	**1655.3**	47.97	**35.34**	**34.90**	55.80	**50.73**	11.50	**58.60**	**0.56**
	10%	FastV	13	16.76	17.41	40	69.55	45.88	54.11	1587.9	49.01	39.39	38.10	61.70	53.13	24.20	62.65	0.58
		VisionZip	**39**	**34.84**	**54.34**	**303**	**78.98**	**58.77**	**56.89**	**2008.6**	**64.18**	**52.17**	**44.60**	**68.20**	**78.30**	**46.60**	**81.45**	**0.76**
		PruMerge+	19	26.28	31.09	62	71.73	56.22	53.66	1880.1	56.99	45.34	44.10	63.90	72.93	35.60	79.30	0.67
	40%	FastV	55	40.60	56.04	268	82.15	56.54	54.66	2084.8	64.31	53.56	50.80	69.50	79.26	44.90	83.25	0.78
		VisionZip	**77**	**77.52**	**78.07**	**609**	**91.77**	**63.49**	**58.56**	**2397.2**	**70.72**	**64.29**	59.50	**72.60**	**85.31**	**60.80**	**85.65**	**0.93**
		PruMerge+	53	58.56	67.85	454	84.03	62.86	56.11	2338.0	69.41	59.94	55.00	71.60	83.52	55.00	85.15	0.84
	100%	Original	94	88.04	82.87	802	95.11	64.48	61.56	2453.9	72.41	69.80	70.20	74.80	86.94	69.40	86.45	1.00

Table 1: Main results of various visual token prune methods on different models and benchmarks. **Bold** denotes the best result under the same setting.

outperforms those pruning in the LLM backbone (e.g., FastV). On LLaVA-OV-7B, FastV's relative performance plummets to 48%, whereas VisionZip retains 75% (Further analysis in Section 5.3.). For KV cache compression, retaining sufficient tokens is crucial for preserving fine-grained information, but sensitivity varies across architectures and budgets. Notably, LLaVA-OV-7B processes thousands of tokens, so even at a 1% budget, it retains more tokens (40.17) than Qwen2-VL-7B (7.62) and InternVL2.5-38B (15.14), leading to a smaller performance drop. These findings underscore the need to tailor compression methods to model and task specifics, and to rigorously assess their cross-benchmark generalization.

Methods	$OG^c \downarrow$		$OL^c \uparrow$	
	1%	40%	1%	40%
StreamingLLM	49.76	12.05	33.26	85.98
H2O	30.93	**2.05**	53.02	**94.57**
SnapKV	31.98	2.34	55.85	93.31
PyramidKV	**29.76**	5.31	**59.52**	91.19
LOOK-M	51.72	3.46	29.67	93.43
VL-Cache	43.25	3.80	55.00	92.04
FastV	58.46	11.87	40.89	80.57
VisionZip	31.32	5.86	59.24	80.52
PruMerge+	34.62	13.47	56.85	83.07

Table 2: Overall performance of generalization and loyalty for different token compression methods, with generalization calculated across all benchmarks and loyalty on a subset(MathVista,LLaVA-Wilder,MMVet). **Bold** indicates the best under the same setting.

Observation 2 **KV cache compression outperforms token pruning in generalization and loyalty.** Table 2 shows H2O and PyramidKV leading in overall performance across metrics at high (40%) and low (10%) budgets, respectively. KV cache compression methods generally exhibit superior generalization and loyalty compared to token pruning. Thus, they should be prioritized when these aspects are critical. Meanwhile, we observe that LOOK-M performs well at higher budgets, but it exhibits a significant drop at lower budgets, which aligns with results in Figure 1, highlighting its high sensitivity under extremely aggressive constraints(detailed analysis is presented in Section 5.4).

Observation 3 **Selecting token pruning or KV cache compression based on task statistics to achieve better performance-efficiency trade-off.** Actual inference time can be broken down into two components: time-to-first-token (TTFT), which reflects the prefill overhead, and decoding time, which measures the latency of subsequent token generation. Figure 2 illustrates the trade-offs between efficiency and performance for token compression methods. KV cache methods yield lim-

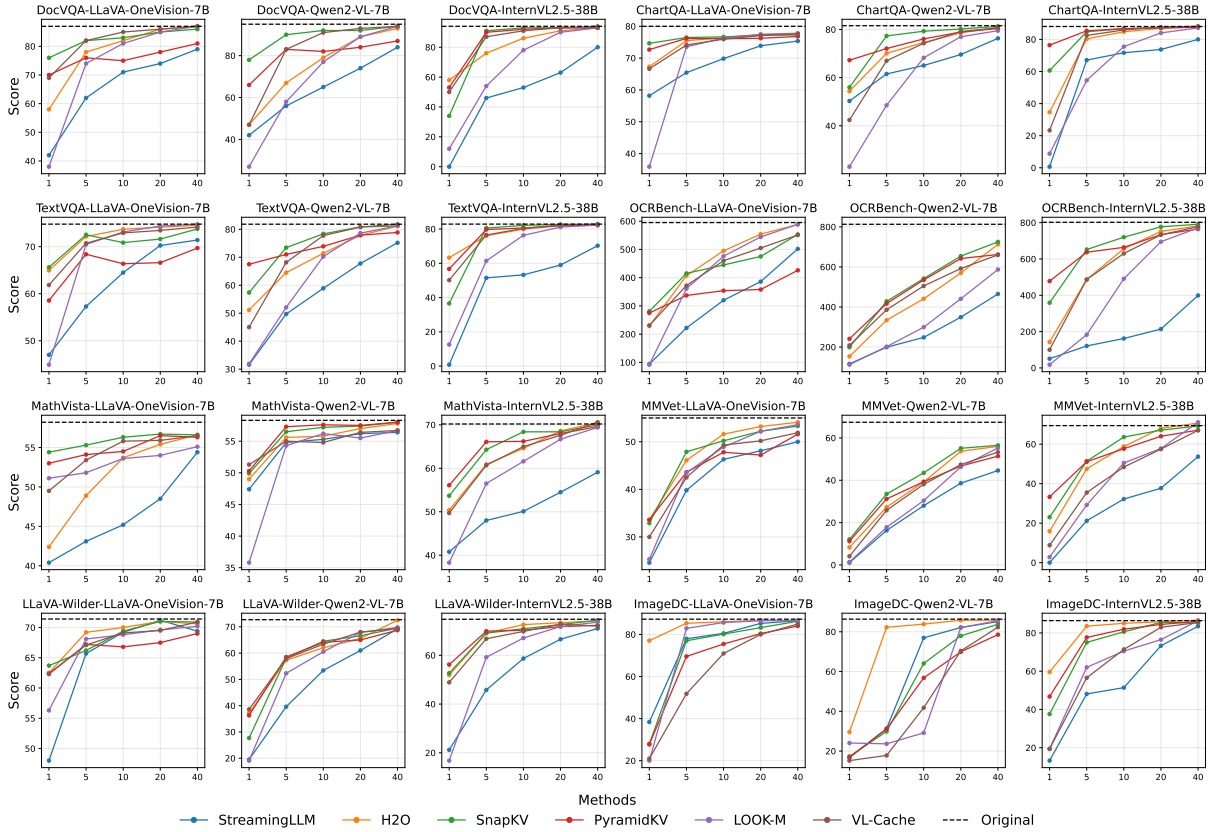


Figure 1: Performance comparison of KV cache compression methods across multiple benchmarks and models.

ited TTFT speedup as they recompute attention weights for token selection during prefill[1]. This computation overhead is budget-independent, keeping their TTFT speedup nearly constant (see vertical lines in left part of Figure 2). In contrast, token pruning removes visual tokens during prefill, drastically reducing TTFT (e.g., 3.2× speedup at 1% budget), making it ideal for short-response tasks like VQA. For instance, at comparable overall performance, H2O achieves only 0.65× TTFT speedup, while VisionZip delivers 2.29× TTFT speedup. For decoding latency (right), KV cache and token pruning methods show similar speedups under the same budget. However, KV cache methods generally outperform token pruning at low budgets for tasks requiring long outputs (e.g., LLaVA-Wilder, ImageDC). Conversely, for tasks involving high-resolution images, the fixed cost of computing attention matrices in KV cache compression leads to significant memory overhead, favoring token pruning in these scenarios.

[1]When using mechanisms like Flash-Attention2 (Dao, 2024) that don't provide full attention matrices for selection, recalculation becomes necessary.

Observation 4 **Consistent performance trends of token compression across single-image, multi-images, and video tasks.** We further evaluate token compression methods on multi-image and video tasks. Table 3 shows the results based on Qwen2-VL-7B. Under a 1% budget, SnapKV achieves the best performance in MP-DocVQA, while LOOK-M suffers a significant drop, consistent with the results in DocVQA. In MovieChat, VL-Cache outperforms other methods. Notably, at higher budgets, some compression methods, such as SnapKV and VL-Cache, can maintain the original model's performance while achieving substantial speedups. These findings indicate that token compression remains consistently effective across single-image, multi-image, and video scenarios. Future work will explore more challenging tasks for further validation.

4.2 Parameter Compression

Setup Pruning and quantization are two mainstream approaches for compressing LVLMs. We evaluate three pruning methods: EcoFLAP (Sung et al., 2023), Wanda (Sun et al., 2023), and SparseGPT (Frantar and Alistarh, 2023); and two

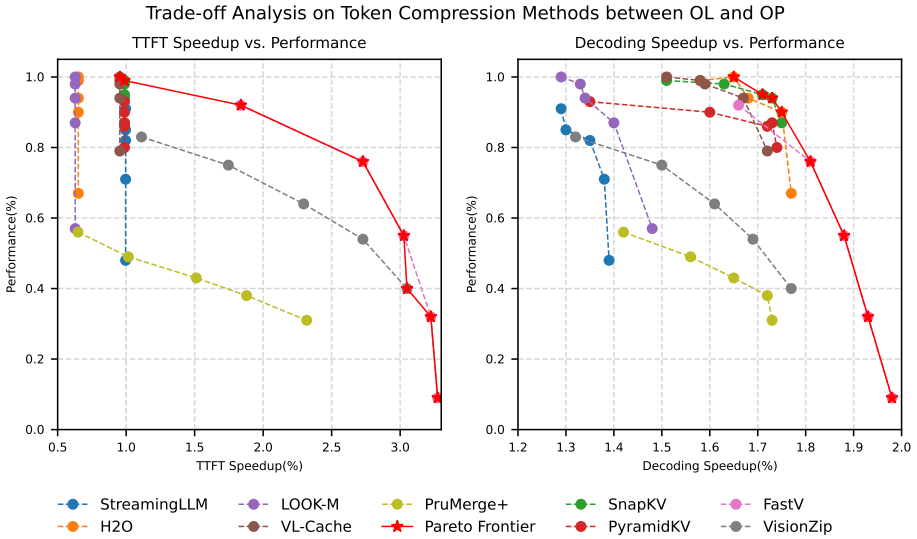


Figure 2: Trade-off Analysis on Token Compression Methods between Efficiency and performance. **Left:** Trade-off between TTFT (time-to-first-token) and Performance across different methods. **Right:** Trade-off between Speedup and performance across different methods. All metrics are expressed as ratios relative to the original model.

Benchmarks	Methods	1%	10%	40%	100%
MP-DocVQA (Split:val) (Metric:ANLS)	StreamingLLM	36.74	51.38	69.04	85.91
	H2O	38.73	63.02	80.67	85.91
	SnapKV	**69.39**	80.73	84.11	85.91
	PyramidKV	63.44	75.93	79.33	85.91
	LOOK-M	18.38	57.09	80.69	85.91
	VL-Cache	57.89	**82.44**	**85.46**	85.91
MovieChat (Split:test) (Metric:Acc)	StreamingLLM	13.11	34.63	37.03	37.73
	H2O	14.23	**37.15**	37.19	37.73
	SnapKV	28.29	36.36	37.53	37.73
	PyramidKV	28.26	**37.15**	36.62	37.73
	LOOK-M	11.95	36.49	37.57	37.73
	VL-Cache	**28.51**	35.99	**37.65**	37.73

Table 3: Results of token compression methods on Qwen2-VL-7B evaluated with multi-image and video benchmarks.

quantization methods: AWQ (Lin et al., 2024) and GPTQ (Frantar et al., 2022). EcoFLAP is specifically designed for LVLMs, while the others are widely-used for general LLM compression. Details are shown in Appendix A.2 and A.3.

Observation 5 **Parameter compression generally preserve performance more effectively than token compression.** Even at higher compression ratios (50% or 2:4 sparsity), the overall performance remains relatively stable. We observe that quantization methods, such as AWQ, tend to preserve higher performance compared to pruning. However, since neither method shortens input token sequences, token compression remains essential for tasks involving very long inputs or high-resolution images. Importantly, the two types of compression are orthogonal that can be effectively combined, Crucially, these two compression types are orthogonal and can be effectively combined and more results are shown in Appendix A.7.

5 Discussion

5.1 Revisiting Layer Adaptive Mechanism

While Cai et al. (2024b) shows that layer-adaptive sparsity benefits KV cache compression in LLM, EFFIVLM-BENCH results reveal that PyramidKV underperforms SnapKV at low sparsity budgets despite using the same token selection metric, due to its layer-adaptive strategy. This suggests that **layer-adaptive sparsity is not always advantageous in LVLM compression, especially under low sparsity**. To further probe this, we analyze VL-Cache, a layer-adaptive KV cache compression method proposed for LVLMs. As shown in Figure 3 and Appendix A.10: At 5% budget, the $0\text{-}th$ layer has nearly $7\times$ of the average budget. This aggressive front-loading reduces budgets for subsequent layers, effectively starving them (less than the average). In addition, visualizations of the tokens selected in the $0\text{-}th$ layer (shown in Figure 8a and Appendix A.11) reveal that most of the tokens chosen are irrelevant. These findings suggest that reducing the budget of early layers while increasing the average allocation for later layers may lead to better performance. Based on the VL-Cache layer adaptive strategy, we propose a hybrid allocation strategy: a portion (40% or 80%) of the total bud-

Settings	Methods	DocVQA	ChartQA	TextVQA	OCRBench	AI2D	GQA	MMMU	MME	RealworldQA	MMStar	MathVista	LLaVA-Wilder	MMBench	MMVet	ImageDC	OP
100%	Original	87	80.00	74.79	595	89.96	61.92	45.44	1974.1	65.88	58.75	58.20	71.40	83.12	55.00	87.25	1.000
20%	EcoFLAP	**87**	**79.32**	74.70	573	**89.86**	61.26	46.56	1951.0	64.84	**60.27**	55.80	70.20	82.17	57.70	85.75	**0.995**
	Wanda	86	78.84	**75.70**	595	81.12	61.71	**46.67**	1939.8	**66.14**	59.11	54.10	68.60	82.17	**60.10**	**86.90**	0.992
	SparseGPT	**87**	77.48	73.92	**598**	80.67	**61.95**	45.78	**1957.2**	**66.14**	58.28	**56.30**	**70.30**	**83.18**	55.40	86.10	0.986
50%	EcoFLAP	**82**	73.24	71.06	502	75.78	59.81	41.56	1732.8	63.40	54.23	**50.20**	53.80	78.69	39.80	57.55	0.877
	Wanda	81	**76.96**	**71.30**	**582**	72.12	60.10	41.67	1726.8	**64.18**	**54.41**	48.50	**65.20**	42.26	47.20	**85.50**	0.900
	SparseGPT	81	72.08	70.20	534	**77.66**	**61.16**	**42.89**	**1741.1**	66.01	53.05	49.30	64.90	**79.82**	**47.30**	85.00	**0.921**
2:4	EcoFLAP	67	62.02	60.70	483	65.19	52.68	32.89	**1309.1**	49.54	45.75	36.00	49.30	71.35	30.20	81.05	0.760
	Wanda	69	**63.52**	59.71	**499**	**67.39**	53.09	34.22	1282.7	57.52	43.09	**38.80**	51.60	**72.81**	30.60	**81.65**	**0.779**
	SparseGPT	**74**	62.68	**65.60**	426	67.10	**56.97**	**34.44**	1296.2	**62.35**	**45.99**	35.60	**52.90**	35.59	**35.70**	81.30	0.772
W4A16†	AWQ	84	75.52	**72.78**	**575**	85.23	61.84	**45.44**	**1978.3**	**67.58**	53.70	52.60	**70.50**	78.92	**54.80**	**87.05**	0.972
	GPTQ	**86**	**75.83**	72.36	571	**85.40**	**61.90**	44.82	1970.2	66.24	**56.25**	**55.80**	70.30	**78.69**	54.30	86.55	**0.975**

Table 4: Main results of various parameter compression methods on LLaVA-OneVision-7B evaluated on different tasks, grouped by *Setting*. † indicates that only the LLM backbone is quantized.

Models	Benchamrks	A-Only	U-40%	U-80%
LLaVA-OneVision-7B	DocVQA	82	83	**84**
	OCRBench	372	398	**410**
	ChartQA	74.12	75.84	**76.24**
	TextVQA	70.73	71.83	**72.36**
Qwen2-VL-7B	DocVQA	83	85	**86**
	OCRBench	386	410	**424**
	ChartQA	67.00	67.84	**68.24**
	TextVQA	68.17	69.93	**70.98**

Table 5: VL-Cache Budget Allocation (5% Total Budget) with Hybrid Strategies: A-Only (adaptive-Only Allocation) vs. U-40% (40% uniform + 60% adaptive) vs. U-80% (80% uniform + 20% adaptive).

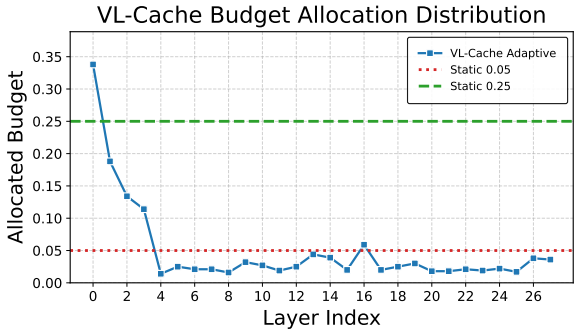


Figure 3: Layer-wise Budget Allocation of VL-Cache on LLaVA-OneVision-7B (5% Total Budget Constraint, OCRBench Example).

get was evenly distributed across all layers, and the remaining (60% or 20%) is adaptively allocated according to the original allocation strategy. As shown in Table 5, evenly allocating 80% of the total budget produced the best results.

5.2 Revisiting Head Adaptive Mechanism

Previous KV cache compression methods diverge on whether heads within a layer should select identical cached tokens. We explore head-adaptive token selection (i.e., allowing heads in the same layer to select different tokens) at a 1% budget. Table 6 shows that head-adaptivity significantly improves performance. This improvement likely stems from different KV or query heads capturing distinct information patterns, enabling head-adaptive selection to preserve critical information under tight budget constraints better. Detailed head attention distributions are in Appendix A.9.

Models	Benchmarks	H2O		SnapKV		VL-Cache	
		w/	w/o	w/	w/o	w/	w/o
LLaVA-OneVision-7B	ChartQA	**67.32**	65.12	**74.64**	73.84	**68.64**	66.64
	TextVQA	**64.96**	62.33	**65.62**	65.60	61.76	**61.85**
	MathVista	**42.40**	41.60	**54.40**	51.20	**52.20**	49.50
	LLaVA-Wilder	**62.50**	60.60	**63.70**	61.40	**62.80**	62.40
Qwen2-VL-7B	ChartQA	**54.36**	45.92	55.96	**56.16**	**57.76**	53.32
	TextVQA	**51.16**	42.85	**57.38**	52.96	**50.69**	47.03
	MathVista	**49.00**	48.50	**49.90**	49.20	**52.90**	52.30
	LLaVA-Wilder	**37.20**	31.60	**27.70**	26.60	**42.30**	39.80

Table 6: Effect of head-adapive strategy for KV cache compression methods.

5.3 Attention Sink Tokens in LVLMs

Xiao et al. (2023) identified *attention sinks* in Transformers: a few initial tokens attract disproportionate attention regardless of semantic relevance, and their removal significantly degrades performance. We explore sink tokens in both textual and visual modalities within LVLMs. For textual sinks, comparing a random baseline against StreamingLLM (which preserves these sink tokens) shows StreamingLLM significantly improves performance at a 1% budget (Table 15), underscoring their importance. Similarly, visual information tends to concentrate on a set of image tokens after passing through the visual encoder Darcet et al. (2024). We find that FastV (using text-guided

Methods	Budgets		
	10%	20%	40%
$\text{FastV}_{\text{origin}}$	31.04	54.20	67.40
FastV_{A1}	30.36	52.40	65.40
FastV_{A2}	**45.56**	**58.60**	**70.04**

Table 7: Comparison of results for $\text{FastV}_{\text{origin}}$, FastV_{A1}, and FastV_{A2} on ChartQA with Qwen2-VL-7B.

Budgets	Methods \ Benchmarks	DocVQA	ChartQA	TextVQA	OCRBench	MathVista	LLaVA-Wilder
1%	$\text{LOOK-M}_{\text{origin}}$	38	34.44	44.74	81	44.30	43.40
	$\text{LOOK-M}_{\text{change}}$	**44**	**56.16**	**48.12**	**117**	**51.80**	**59.30**
5%	$\text{LOOK-M}_{\text{origin}}$	**74**	73.72	70.29	375	53.10	65.50
	$\text{LOOK-M}_{\text{change}}$	**74**	**75.04**	**70.89**	**406**	**53.50**	**69.00**

Table 8: The performance comparison between LOOK-M with origin merge($\text{LOOK-M}_{\text{origin}}$) and modality-specific merge($\text{LOOK-M}_{\text{change}}$) on LLaVA-OV-7B.

metrics for visual token pruning) underperforms VisionZip (relying solely on the visual encoder's attention map). We hypothesize this is because visual sink tokens critically impact performance, yet text-guided metrics often fail to capture them. Two ablations on ChartQA using Qwen2-VL-7B (Table 7) further support this hypothesis: **A1**: Prohibiting FastV from selecting from the top 10% most critical visual tokens slightly degraded performance. **A2**: Conversely, forcing FastV to prioritize these top 10% tokens significantly improved performance. These results indicate that text-based selection can overlook critical visual sink tokens, and that retaining these sinks is vital, particularly under low budgets. The recent attention-sink-free gated attention model of Qiu et al. (2025) also warrants future exploration. More visualizations are available in Appendix A.12.

5.4 How to Merge Evicted Tokens

The merge operation shows promise for *recovering* evicted information (Bolya et al.). Current methods differ: LOOK-M and PruMerge+ merge evicted tokens *into* retained ones, whereas VisionZip *concatenates* them. This poses the question: **What is an effective merge strategy for LVLMs?** Our experiments reveal LOOK-M's performance drops at low budgets. We hypothesize that at a 1% budget, its text-prior mechanism discards visual tokens, which are then merged into remaining text tokens. This cross-modal fusion, we argue, disrupts critical textual features (e.g., sink tokens), degrading performance.. Thus, we modify LOOK-M to merge evicted tokens only within the same modality. Table 8 shows our modification consistently improves performance over the original LOOK-M, underscoring that modality-specific merging is crucial for LVLM token compression. Specific examples are in Appendix A.13.

6 Conclusion

This paper introduces EFFIVLM-BENCH, a comprehensive benchmark for systematically evaluating token and parameter compression methods for LVLMs across diverse tasks, models, and metrics. Our empirical results reveal inherent trade-offs—performance, generalization, loyalty, and efficiency—tied to different compression strategies. Analysis of factors such as attention sinks and layer- or head-adaptive sparsity provides practical insights for optimizing these techniques. We hope that EFFIVLM-BENCH offers a robust foundation for advancing future research in LVLM compression.

Limitations

Our work has several limitations: (1) Although EFFIVLM-BENCH systematically evaluates both token and parameter compression approaches, it focuses primarily on a subset of representative LVLM models and tasks, leaving the performance on other architectures and more specialized domains underexplored. (2) We only consider training-free methods in this study, and the incorporation of training-based compression algorithms could provide deeper insights into the balance between performance gains and resource overhead. (3) Although our analysis delves into several mechanisms under extremely low budgets, there may be additional factors in token and parameter compression methods. In future work, we plan to extend EFFIVLM-BENCH to include more tasks, models, and methods, offering a more comprehensive assessment of LVLM compression.

Acknowledgments

The work is supported by the National Key Research and Development Project (2022YFF0903301), the National Science Foundation of China (U22B2059,62276083). Ming Liu is the corresponding author. Besides, we express gratitude to DuXiaoman Technology for supporting the work.

References

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. Fluctuation-based adaptive structured pruning for large language models. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 10865–10873. AAAI Press.

Anthropic. 2024. Developing a computer use model.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *The Eleventh International Conference on Learning Representations*.

Yuxuan Cai, Jiangning Zhang, Haoyang He, Xinwei He, Ao Tong, Zhenye Gan, Chengjie Wang, and Xiang Bai. 2024a. Llava-kd: A framework of distilling multimodal large language models. *CoRR*, abs/2410.16236.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and 1 others. 2024b. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Junbum Cha, Wooyoung Kang, Jonghwan Mun, and Byungseok Roh. 2024. Honeybee: Locality-enhanced projector for multimodal LLM. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 13817–13827. IEEE.

Guiming Hardy Chen, Shunian Chen, Ruifei Zhang, Junying Chen, Xiangbo Wu, Zhiyi Zhang, Zhihong Chen, Jianquan Li, Xiang Wan, and Benyou Wang. 2024a. Allava: Harnessing gpt4v-synthesized data for A lite vision-language model. *CoRR*, abs/2402.11684.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. 2024b. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pages 19–35. Springer.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and 1 others. 2024c. Are we on the right way for evaluating large vision-language models? *arXiv preprint arXiv:2403.20330*.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C. Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server. *ArXiv*, abs/1504.00325.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, and 1 others. 2024d. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*.

Xiangxiang Chu, Limeng Qiao, Xinyang Lin, Shuang Xu, Yang Yang, Yiming Hu, Fei Wei, Xinyu Zhang, Bo Zhang, Xiaolin Wei, and 1 others. 2023. Mobilevlm: A fast, reproducible and strong vision language assistant for mobile devices. *arXiv preprint arXiv:2312.16886*.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. 2024. Vision transformers need registers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Mostafa Dehghani, Basil Mustafa, Josip Djolonga, Jonathan Heek, Matthias Minderer, Mathilde Caron, Andreas Steiner, Joan Puigcerver, Robert Geirhos, Ibrahim M Alabdulmohsin, and 1 others. 2023. Patch n'pack: Navit, a vision transformer for any aspect ratio and resolution. *Advances in Neural Information Processing Systems*, 36:2252–2274.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, Yen-Sung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, and 32 others. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *CoRR*, abs/2409.17146.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, and Rongrong Ji. 2024. Mme: A comprehensive evaluation benchmark for multimodal large language models. *Preprint*, arXiv:2306.13394.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Lianyu Hu, Fanhua Shang, Liang Wan, and Wei Feng. 2024. illava: An image is worth fewer than 1/3 input tokens in large multimodal models. *CoRR*, abs/2412.06263.

Wenxuan Huang, Zijie Zhai, Yunhang Shen, Shaoshen Cao, Fei Zhao, Xiangfeng Xu, Zheyu Ye, and Shaohui Lin. 2024. Dynamic-llava: Efficient multimodal large language models via dynamic vision-language context sparsification. *arXiv preprint arXiv:2412.00876*.

Drew A Hudson and Christopher D Manning. 2019. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. A diagram is worth a dozen images. *Preprint*, arXiv:1603.07396.

Bo Li, Kaichen Zhang, Hao Zhang, Dong Guo, Renrui Zhang, Feng Li, Yuanhan Zhang, Ziwei Liu, and Chunyuan Li. 2024a. Llava-next: Stronger llms supercharge multimodal capabilities in the wild.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and 1 others. 2024b. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*.

Wentong Li, Yuqian Yuan, Jian Liu, Dongqi Tang, Song Wang, Jie Qin, Jianke Zhu, and Lei Zhang. 2024c. Tokenpacker: Efficient visual projector for multimodal llm. *arXiv preprint arXiv:2407.02392*.

Yanwei Li, Yuechen Zhang, Chengyao Wang, Zhisheng Zhong, Yixin Chen, Ruihang Chu, Shaoteng Liu, and Jiaya Jia. 2024d. Mini-gemini: Mining the potential of multi-modality vision language models. *arXiv preprint arXiv:2403.18814*.

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

Yunxin Li, Zhenyu Liu, Zitao Li, Xuanyu Zhang, Zhenran Xu, Xinyu Chen, Haoyuan Shi, Shenyuan Jiang, Xintong Wang, Jifang Wang, Shouzheng Huang, Xinping Zhao, Borui Jiang, Lanqing Hong, Longyue Wang, Zhuotao Tian, Baoxing Huai, Wenhan Luo, Weihua Luo, and 3 others. 2025. Perception, reason, think, and plan: A survey on large multimodal reasoning models. *arXiv preprint arXiv:2505.04921*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024a. Improved baselines with visual instruction tuning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 26286–26296.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, Kai Chen, and Dahua Lin. 2024b. Mmbench: Is your multi-modal model an all-around player? *Preprint*, arXiv:2307.06281.

Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xu-Cheng Yin, Cheng-Lin Liu, Lianwen Jin, and Xiang Bai. 2024c. Ocrbench: on the hidden mystery of ocr in large multimodal models. *Science China Information Sciences*, 67(12).

Zhijian Liu, Ligeng Zhu, Baifeng Shi, Zhuoyang Zhang, Yuming Lou, Shang Yang, Haocheng Xi, Shiyi Cao, Yuxian Gu, Dacheng Li, Xiuyu Li, Yunhao Fang, Yukang Chen, Cheng-Yu Hsieh, De-An Huang, An-Chieh Cheng, Vishwesh Nath, Jinyi Hu, Sifei Liu, and 8 others. 2024d. NVILA: efficient frontier visual language models. *CoRR*, abs/2412.04468.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, Yaofeng Sun, Chengqi Deng, Hanwei Xu, Zhenda Xie, and Chong Ruan. 2024a. Deepseek-vl: Towards real-world vision-language understanding. *CoRR*, abs/2403.05525.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2024b. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations (ICLR)*.

Yaxin Luo, Gen Luo, Jiayi Ji, Yiyi Zhou, Xiaoshuai Sun, Zhiqiang Shen, and Rongrong Ji. 2024. γ-mod: Exploring mixture-of-depth adaptation for multimodal large language models. *CoRR*, abs/2410.13859.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244*.

Minesh Mathew, Dimosthenis Karatzas, R Manmatha, and CV Jawahar. 2020. Docvqa: A dataset for vqa on document images. corr abs/2007.00398 (2020). *arXiv preprint arXiv:2007.00398*.

OpenAI. 2024. Hello gpt-4o.

OpenAI. 2025. Introducing operator.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, and 1 others. 2025. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. 2024. Llava-prumerge: Adaptive token reduction for efficient large multimodal models. *arXiv preprint arXiv:2403.15388*.

Fangxun Shu, Yue Liao, Le Zhuo, Chenning Xu, Guanghao Zhang, Haonan Shi, Long Chen, Tao Zhong, Wanggui He, Siming Fu, Haoyuan Li, Bolin Li, Zhelun Yu, Si Liu, Hongsheng Li, and Hao Jiang. 2024. Llava-mod: Making llava tiny via moe knowledge distillation. *CoRR*, abs/2408.15881.

Amanpreet Singh, Vivek Natarjan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. 2019. Towards vqa models that can read. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8317–8326.

Enxin Song, Wenhao Chai, Guanhong Wang, Yucheng Zhang, Haoyang Zhou, Feiyang Wu, Xun Guo, Tian Ye, Yan Lu, Jenq-Neng Hwang, and 1 others. 2023. Moviechat: From dense token to sparse memory for long video understanding. *arXiv preprint arXiv:2307.16449*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Yi-Lin Sung, Jaehong Yoon, and Mohit Bansal. 2023. Ecoflap: Efficient coarse-to-fine layer-wise pruning for vision-language models. *arXiv preprint arXiv:2310.02998*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *ArXiv preprint*, abs/2403.05530.

Qwen Team. 2025. Qwen2.5-vl.

Rubèn Tito, Dimosthenis Karatzas, and Ernest Valveny. 2022. Hierarchical multimodal transformers for multi-page docvqa. *arXiv preprint arXiv:2212.05935*.

Dezhan Tu, Danylo Vashchilenko, Yuzhe Lu, and Panpan Xu. 2024. Vl-cache: Sparsity and modality-aware kv cache compression for vision-language model inference acceleration. *arXiv preprint arXiv:2410.23317*.

Zhongwei Wan, Ziang Wu, Che Liu, Jinfa Huang, Zhihong Zhu, Peng Jin, Longyue Wang, and Li Yuan. 2024. Look-m: Look-once optimization in kv cache for efficient multimodal long-context inference. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4065–4078.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, and 1 others. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Tiannan Wang, Wangchunshu Zhou, Yan Zeng, and Xinsong Zhang. 2023a. Efficientvlm: Fast and accurate vision-language models via knowledge distillation and modal-adaptive pruning. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13899–13913. Association for Computational Linguistics.

Yuxin Wang, Minghua Ma, Zekun Wang, Jingchang Chen, Huiming Fan, Liping Shan, Qing Yang, Dongliang Xu, Ming Liu, and Bing Qin. 2024b. CFSP: an efficient structured pruning framework for llms with coarse-to-fine activation information. *CoRR*, abs/2409.13199.

Zekun Wang, Jingchang Chen, Wangchunshu Zhou, Ming Liu, and Bing Qin. 2023b. Smarttrim: Adaptive tokens and parameters pruning for efficient vision-language models. *CoRR*, abs/2305.15033.

Zekun Wang, Wenhui Wang, Haichao Zhu, Ming Liu, Bing Qin, and Furu Wei. 2021. Distilled dual-encoder model for vision-language understanding. *CoRR*, abs/2112.08723.

x.ai. Grok-1.5 vision preview.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Ke Xu, Julian J. McAuley, and Furu Wei. 2021. Beyond preserved accuracy: Evaluating loyalty and robustness of BERT compression. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 10653–10659. Association for Computational Linguistics.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024a. Aguvis: Unified pure vision agents for autonomous GUI interaction. *CoRR*, abs/2412.04454.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. 2024b. Think: Thinner key cache by query-driven pruning. *CoRR*, abs/2407.21018.

Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. 2024. Visionzip: Longer is better but not necessary in vision language models. *arXiv preprint arXiv:2412.04467*.

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, and 1 others. 2024. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*.

JiangYong Yu, Sifan Zhou, Dawei Yang, Shuo Wang, Shuoyu Li, Xing Hu, Chen Xu, Zukang Xu, Changyong Shu, and Zhihang Yuan. 2025. Mquant: Unleashing the inference potential of multimodal large language models via full static quantization. *arXiv preprint arXiv:2502.00425*.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. 2023. Mm-vet: Evaluating large multimodal models for integrated capabilities. *Preprint*, arXiv:2308.02490.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, and 3 others. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of CVPR*.

Kaichen Zhang, Bo Li, Peiyuan Zhang, Fanyi Pu, Joshua Adrian Cahyono, Kairui Hu, Shuai Liu, Yuanhan Zhang, Jingkang Yang, Chunyuan Li, and Ziwei Liu. 2024a. Lmms-eval: Reality check on the evaluation of large multimodal models. *Preprint*, arXiv:2407.12772.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and 1 others. 2024b. Sparsevlm: Visual token sparsification for efficient vision-language model inference. *arXiv preprint arXiv:2410.04417*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. 2023a. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710. Curran Associates, Inc.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023b. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Baichuan Zhou, Ying Hu, Xi Weng, Junlong Jia, Jie Luo, Xien Liu, Ji Wu, and Lei Huang. 2024. Tinyllava: A framework of small-scale large multimodal models. *arXiv preprint arXiv:2402.14289*.

A Appendix

A.1 Evaluation Benchmarks

We conduct evaluations in realistic scenarios, using diverse benchmarks that span a wide range of vision-language tasks. These are grouped into three categories:

(1) Chart, Diagram, and Document Understanding This category focuses on textual and diagrammatic comprehension. Benchmarks include DocVQA (Mathew et al., 2020), ChartQA (Masry et al., 2022), TextVQA (Singh et al., 2019), OCR-Bench (Liu et al., 2024c), and AI2D (Kembhavi et al., 2016). These test the ability to accurately detect and recognize text in various forms (printed, handwritten, or embedded in charts/diagrams) and to understand document layouts or diagrammatic structures.

(2) Perception and Multi-discipline Reasoning This category targets visual perception combined with logical or knowledge-intensive reasoning. Benchmarks include GQA (Hudson and Manning, 2019), MMMU (Yue et al., 2024), MME (Fu et al., 2024), MMBench (Liu et al., 2024b), MMVet (Yu et al., 2023), MMStar (Chen et al., 2024c), and MathVista (Lu et al., 2024b).

(3) Real-world Compositional & Interactive QA This category assesses interactive and compositional reasoning in realistic conversational settings. Benchmarks like LLaVA-Wilder (Li et al., 2024a), RealWorldQA (x.ai), and ImageDC (Li et al., 2024a) test a model's ability to handle open-ended questions, multi-turn dialogues, and complex visual-linguistic grounding, as well as its adaptation to diverse topics and coherence over extended interactions.

Across these benchmarks, we evaluate various compression methods on multimodal understanding and reasoning. Input/output lengths differ significantly by task (Table 9). To assess KV cache methods on longer sequences, we select MP-DocVQA (Tito et al., 2022) for multi-images and MovieChat (Song et al., 2023) for video.

A.2 Compression Method

Token Pruning Token pruning compression reduces computational overhead by eliminating redundant tokens during inference. We implement

	Models \ Benchmarks	DocVQA	ChartQA	TextVQA	OCRBench	AI2D	GQA	MMMU	MME	RealworldQA	MMStar	MathVista	LLaVA-Wilder	MMBench	MMVet	ImageDC
Input	LLaVA-OneVision-7B	7313	3882	5147	1233	3497	2693	3613	5768	7778	2226	2935	4559	1903	5006	2704
	Qwen2-VL-7B	2027	406	1010	75	598	383	730	1141	1776	288	465	904	251	1018	363
	InternVL2.5-38B	1835	1367	1744	1212	1389	1800	1588	1506	1834	1478	1346	1337	1387	1400	1492
Output	LLaVA-OneVision-7B	2.72	2.92	2.69	26.04	1.00	1.27	1.05	1.03	1.02	1.00	2.85	252.12	1.20	59.07	414.61
	Qwen2-VL-7B	1.82	2.96	2.96	3.71	1.00	1.23	1.42	1.01	1.01	1.00	2.65	55.54	1.72	24.70	314.22
	InternVL2.5-38B	1.72	3.27	2.86	11.27	1.00	1.23	1.07	1.00	1.01	1.00	78.80	257.73	1.00	145.62	158.66

Table 9: Input and Output token length of different models on various benchmarks. We select 100 samples from each benchmark for calculation.

methods including FastV (Chen et al., 2024b), VisionZip (Yang et al., 2024), and PruMerge+ (Shang et al., 2024), which selectively remove redundant tokens based on relevance scores or token-level importance metrics.

KV Cache Compression We explore several KV cache compression methods that reduce memory usage during inference by selectively retaining key-value states, including StreamingLLM (Xiao et al., 2023), H2O (Zhang et al., 2023a), SnapKV (Li et al., 2024e), PyramidKV (Cai et al., 2024b), LOOK-M (Wan et al., 2024), and VL-Cache (Tu et al., 2024).

Model Pruning Model pruning aims to reduce model size and inference cost while preserving performance. We evaluate EcoFLAP (Sung et al., 2023), Wanda (Sun et al., 2023), and SparseGPT (Frantar and Alistarh, 2023), applying these methods only to the LLM backbone of the LVLM.

Model Quantization Model quantization compresses models by reducing the precision of their weights and activations, thereby lowering memory requirements and accelerating computation. We consider techniques like AWQ (Lin et al., 2024) and GPTQ (Frantar et al., 2022) (applied only to the LVLM's LLM backbone).

A.3 Details of Methods

Token Pruning Let $T_v = \{t_{v,j}\}_{j=1}^{l_v}$ be the set of l_v visual tokens, each $t_{v,j} \in \mathbb{R}^d$. Given a retention budget $b \in (0, 1]$, pruning aims to select $l'_v = \lfloor b \cdot l_v \rfloor$ tokens. This typically involves a scoring function $s : \mathbb{R}^d \to \mathbb{R}$ to assess token importance.

FastV uses a lightweight scoring strategy. For each visual token $t_{v,j}$, its importance $s(t_{v,j})$ is its accumulated attention score among textual tokens. A binary mask $m \in \{0, 1\}^{l_v}$ is defined by

$$m_j = \begin{cases} 1, & \text{if } s(t_{v,j}) \geq \tau, \\ 0, & \text{otherwise,} \end{cases}$$

where the threshold τ is set to ensure the binary mask m satisfies the following constraint:

$$\sum_{j=1}^{l_v} m_j = \lfloor b \cdot l_v \rfloor.$$

The pruned visual token set is given by

$$T'_v = \{\, t_{v,j} \mid m_j = 0 \,\}.$$

VisionZip first selects an initial set of tokens to keep using accumulated attention (similar to FastV), and identifies the discarded set T'_v. To compensate for information loss from T'_v, VisionZip clusters T_d into k groups. The original paper sets $k = \lfloor (b/6.4) \cdot l_v \rfloor$. For each cluster $C_i \subseteq T'_v$, its centroid c_i is computed:

$$c_i = \frac{1}{|C_i|} \sum_{t \in C_i} t.$$

The recycled token set is then defined as

$$R_v = \{\, c_i \mid i = 1, 2, \ldots, k \,\}.$$

Finally, the tokens in R_v are concatenated with the retained tokens.

PruMerge+ also adaptively selects important tokens and merges less important ones to retain critical information. For each visual token $t_{v,j}$, its importance score is computed via the [CLS] token's attention weights in the vision encoder:

$$s(t_{v,j}) = A_j^{[\text{CLS}]},$$

where $A^{[\text{CLS}]} \in \mathbb{R}^{l_v}$ denotes the attention weight from the [CLS] token to visual tokens. For architectures without [CLS] tokens, we compute row-wise averages of the attention matrix:

$$s(t_{v,j}) = \frac{1}{l_v} \sum_{i=1}^{l_v} A_{ij}.$$

The binary mask $m \in \{0,1\}^{l_v}$ is determined through interquartile range (IQR) based outlier detection:

$$m_j = \begin{cases} 1, & \text{if } s(t_{v,j}) \geq \tau, \\ 0, & \text{otherwise,} \end{cases}$$

where threshold τ is adaptively calculated using the IQR method. To ensure budget compliance, we enforce:

$$\sum_{j=1}^{l_v} m_j = \lfloor b \cdot l_v \rfloor.$$

If IQR selection yields fewer than $l'_v = \lfloor b \cdot l_v \rfloor$ tokens (the target budget b), PruMerge+ first supplements this set by uniformly sampling from the remaining highest-scoring unselected candidates. All l'_v tokens in the final retained set then undergo feature merging to consolidate information. Unlike VisionZip, PruMerge+ merges information from pruned tokens into retained ones.

KV Cache Compression In the LVLM prefill stage, let l_v and l_t be the number of visual and text tokens, respectively, with total length $l = l_v + l_t$. Let w be the recent window size and b the compression budget fraction. The attention map is $A \in \mathbb{R}^{l \times l}$, where A_{ij} is the attention weight between query i and key j. To compress the KV caches, we introduce:

$$\mathcal{F} : \mathbb{R}^{l \times l} \rightarrow \{0,1\}^{l},$$

which produces a binary mask $\mathbf{m} \in \{0,1\}^{l}$ indicating which tokens to retain ($\mathbf{m}_j = 1$) or evict ($\mathbf{m}_j = 0$). We consider four variants of $\mathcal{F}$ that rank token importance:

- *Accumulated Attention* $\mathcal{F}_{\text{acc}}$. We sum attention weights along the query tokens dimension:

$$s_j = \sum_{i=1}^{l} A_{ij}.$$

- *Normalized Attention* $\mathcal{F}_{\text{norm}}$. We normalize attention weights for each query i, then sum and average:

$$\tilde{s}_j = \textit{Norm}(\sum_{i=1}^{l} A_{ij})$$

- *Sliding Window Attention* $\mathcal{F}_{\text{sw}}$. We compute accumulated attention scores along the query tokens but only over a recent window:

$$\tilde{s}_j = \sum_{i=l-w+1}^{l} A_{ij}.$$

- *Post-Vision Attention* $\mathcal{F}_{\text{pv}}$. Recognizing that in many LVLMs, the textual tokens following the visual tokens are more critical, this variant computes attention scores using only the queries from the text region. Formally, letting the text region be indexed by $i = l_v+1, \ldots, l$, we define:

$$\tilde{s}_j = \sum_{i=l_v+1}^{l} A_{ij}.$$

These variants of $\mathcal{F}$ offer different strategies for ranking tokens, allowing us to evaluate how attention-based selection influences KV cache compression in LVLMs. In experiments, H2O and LOOK-M use *Accumulated Attention*, SnapKV and PyramidKV use *Sliding Window Attention* and VL-Cache uses *Post-Vision Attention*.

Model Pruning aims to identify a binary mask $S \in \{0,1\}^{m \times n}$ so that the pruned weight matrix $\widetilde{W} = W \odot S$ preserves performance while satisfying a desired sparsity level, i.e., $\|S\|_0 = p \cdot (m \times n)$, with p being the fraction of weights to retain. An importance score s_{ij} is computed for each weight W_{ij}. The binary mask is then determined by selecting the top-p fraction of weights with the highest scores.

Wanda measures the importance of each weight by the product of its magnitude and the norm of its corresponding input activation:

$$s_{ij} = |W_{ij}| \cdot \|X_j\|_2,$$

where X_j is the input activation vector associated with the j-th column of W. This prioritizes weights coupled with strong activations, preserving contributions critical to model performance even if the weights themselves are small.

EcoFLAP prunes LVLMs layer-wise in a coarse-to-fine manner. The coarse phase computes layer i's importance $S(W_i)$ using an expected zeroth-order gradient approximation: $||\nabla_{W_i}\mathcal{L}(W_i,\mathcal{D})||_2 = \mathbb{E}_{d\sim\mathcal{D}}[\mathbb{E}_{z\sim N(0,1)}[|\frac{\mathcal{L}(W_i+\epsilon z,d)-\mathcal{L}(W_i-\epsilon z,d)}{2\epsilon}|]]$. The fine-grained step then uses same score as Wanda locally within each layer to prune weights according to these layer-specific sparsity ratios.

SparseGPT minimizes a layer's output reconstruction error using a second-order approximation. Its importance metric is derived from an approximate diagonal of the Hessian:

$$s_{ij}^{\text{SparseGPT}} = [\frac{||W||^2}{\text{diag}(XX^T+\lambda I)^{-1}}]_{ij}$$

Model Quantization Quantization aims to approximate a full-precision weight matrix $W \in \mathbb{R}^{m\times n}$ with a low-bit representation while minimizing performance degradation. In our experiments, we consider two notable approaches of *Post-Training Quantization*: AWQ and GPTQ.

AWQ refines the PTQ process by incorporating input activation statistics to better preserve the layer output. Given the weight matrix W and its corresponding input activations X, AWQ selects quantization parameters by solving:

$$\min_{s,z} \left\| WX - \widehat{W}X \right\|_F^2$$

with $\widehat{W}_{ij} = s \cdot \left(\text{round}\left(\frac{W_{ij}}{s}\right) - z\right)$.

GPTQ enhances PTQ by leveraging second-order information to minimize the output error induced by quantization. Starting from the uniform quantization of W, GPTQ aims to adjust the quantization parameters so as to minimize the reconstruction error using a Taylor expansion involving an approximate Hessian H (often estimated via the diagonal of $X^\top X$). This Hessian-guided correction allows GPTQ to achieve high quantization fidelity in one shot, even under aggressive bit-width reductions.

A.4 Implementation Details

Token Pruning To ensure fair comparison of token pruning methods, we standardize average token retention rates across layers. We evaluate rates of 1%, 5%, 10%, 20%, and 40%. Higher rates are omitted, as 40% retention typically preserves performance comparable to the original model. Additionally, because the metrics of these methods can be architecture-dependent, we adapt each method consistently across all evaluated LVLMs to maintain fair comparisons.

KV Cache Compression For fair comparison of KV cache compression methods, we standardize budget allocation. We test budgets of 1%, 5%, 10%, 20%, and 40% of the original cache size. Higher budgets are omitted as 40% typically yields performance comparable to the uncompressed model. During prefill, 10% of the currently allocated budget forms a *recent window* to retain the most recent tokens; the remainder is managed by each method's specific mechanism. In the decoding, to maintain uniformity and fairness across all methods, we do not apply any additional KV cache compression.

Model Pruning In our experiments, we focus on pruning the LLM component. We utilize a consistent set of 128 samples from the COCO-Caption(Chen et al., 2015) as our validation set. We apply unstructured pruning at 20% and 50% sparsity levels, as well as semi-structured pruning under 2:4 setting. We do not apply structured pruning because these result in significant performance drops without recovery training (Wang et al., 2024b).

Model Quantization We focus exclusively on quantizing the LLM component, aligning with the existing LVLM quantization work. We utilize the same validation dataset as in our pruning experiments. For the LLM's weights, we apply a W4A16g128 quantization scheme, where weights are quantized to 4 bits and activations are kept in FP16, and a group size of 128 is used during quantization. This scheme offers a balanced trade-off between model performance and efficiency.

Evaluation We use lmms-eval (Zhang et al., 2024a) framework to perform evaluation. Given the diversity of benchmarks and LVLMs, we select specific LVLMs and benchmarks tailored to each method type, which are detailed in Appendix A.1. The batch size is set to 1. We use 8 $\times$ NVIDIA A800 GPUs. For the efficiency experiments, we run 128 samples with an 8k input token length and 100 token output length on a single NVIDIA A800 GPU.

A.5 Detailed Results of KV Cache Compression

Table 15 presents the detailed results of various KV cache compression methods across different mod-

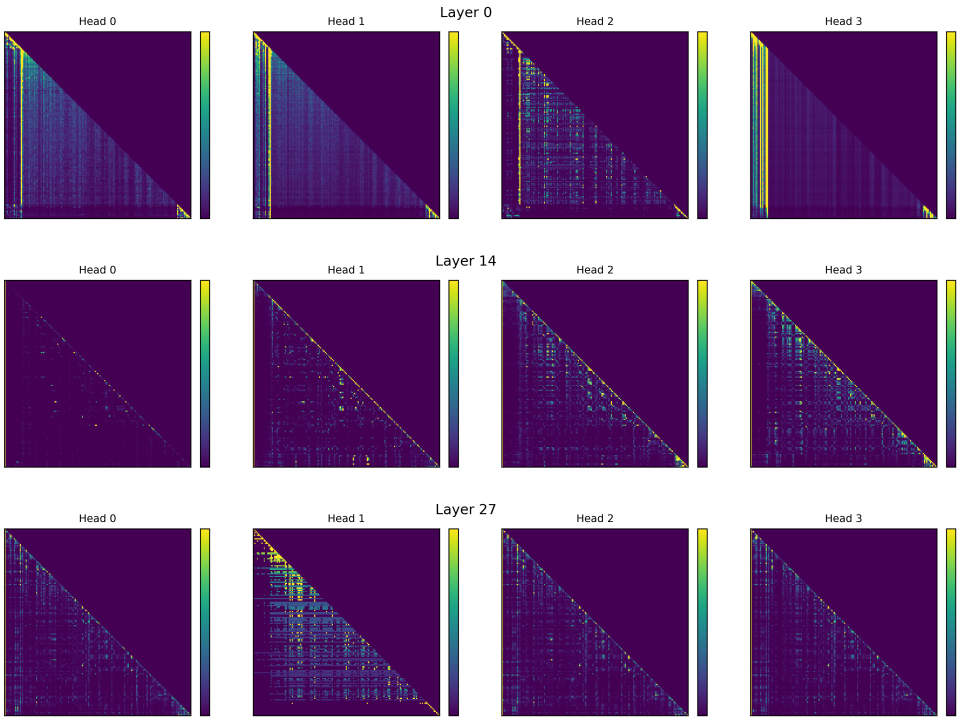


Figure 4: Visualizations of attention weight heatmaps from layers 0, 14, and 27 of Qwen2-VL-7B. Different heads within the same layer exhibit notable pattern variations.

els and benchmarks under varying cache budgets.

A.6 More Results of Token Compression

We additionally evaluate two recent token compression approaches: iLLaVA (Hu et al., 2024) and SparseVLM (Zhang et al., 2024b), which incorporate *progressive sparsification strategy* that gradually sparsifies tokens across multiple layers, rather than enforcing a target sparsity ratio at a single layer. We follow the original implementation details and the results against existing methods in our benchmark are shown in Table 10.

We find that: (i) iLLaVA demonstrates competitive performance at higher token budgets (40%) thanks to its iterative pruning strategy within the visual encoder. This strategy progressively prunes tokens across layers while preserving critical ones in earlier stages. However, at lower token budgets (1%), performance declines significantly similar as the phenomenon of FastV, likely due to the visual sink token: relying on text tokens to guide pruning within LLM may inadvertently remove essential visual sink tokens, leading to performance degradation (details are in Section 5.3). (ii) Regarding SparseVLM, compared to other token pruning methods that also exclusively prune visual tokens in the LLM component (e.g., FastV), it introduces an iterative pruning strategy and a more fine-grained, text-driven selection criterion to guide the pruning process. Consequently, SparseVLM achieves consistently strong performance across various compression budgets and tasks.

A.7 More Results of Parameter Compression

Table 11 shows the results of parameter compression methods on Qwen2-VL-7B across various benchmarks. To investigate the efficacy of combining token-level and parameter-level compression, we select AWQ and SnapKV as representative methods for token and parameter compression, respectively. The results on Qwen2-VL-7B are shown in Table 12. Combining AWQ with SnapKV boosts inference speed by 1.65× over AWQ alone, while maintaining comparable overall performance, particularly on multimodal reasoning tasks (MMMU, MathVista). This highlights the efficiency gains achievable with hybrid compression. However, for tasks demanding fine-grained visual perception (e.g., TextVQA), integrating token compression like SnapKV results in a marginal performance drop. This underscores that acceleration strategies must be tailored to specific target scenario characteristics.

Budgets	Methods	TextVQA	AI2D	MMMU	MME	MathVista
1%	VisionZip	**44.48**	**72.11**	42.56	**1704.2**	39.30
	iLLaVA	22.86	68.78	**43.11**	1101.0	37.50
	SparseVLM	42.29	71.07	43.00	1662.5	**39.40**
10%	VisionZip	57.26	77.97	44.11	**1915.0**	**45.00**
	iLLaVA	64.88	73.47	44.66	1773.9	39.30
	SparseVLM	**65.72**	**79.27**	**45.22**	1841.8	44.60
40%	VisionZip	68.21	83.84	46.11	1956.8	52.60
	iLLaVA	**73.26**	87.62	**47.44**	**1995.3**	**56.60**
	SparseVLM	72.80	**88.66**	45.78	1993.8	54.10
100%	Original	74.79	89.96	45.44	1974.1	58.20

Table 10: Results of iLLaVA and SparseVLM on LLaVA-OneVision-7B model. **Bold** denotes the best result under the same setting.

Settings	Methods	ChartQA	TextVQA	OCRBench	AI2D	GQA	MMMU	MME	RealworldQA	MMStar	MathVista	LLaVA-Wilder	ImageDC	OP
100%	Original	81.56	81.82	813	91.02	62.3	50.77	2327.78	66.53	57.11	58.3	72.7	86.35	1.000
20%	EcoFLAP	81.36	81.82	805	91.13	62.30	50.00	2,288.0	66.67	56.74	57.10	52.70	87.33	0.964
	Wanda	75.52	80.64	754	89.67	60.86	46.11	2,090.8	65.88	53.02	52.40	50.20	84.90	0.918
	SparseGPT	71.68	77.45	722	82.32	57.84	39.56	1,871.9	59.61	45.99	40.70	44.50	81.00	0.842
50%	EcoFLAP	81.08	81.79	800	91.16	62.32	50.44	2,312.6	66.54	56.69	58.90	53.50	87.13	0.970
	Wanda	77.72	80.61	784	88.63	61.20	45.00	2,129.1	64.97	52.74	53.50	51.20	85.55	0.924
	SparseGPT	71.96	78.08	727	82.29	58.66	38.78	1,967.3	57.91	45.63	39.30	44.50	80.30	0.845
2:4	EcoFLAP	81.16	82.08	809	91.03	62.32	50.78	2,317.2	67.45	56.85	59.30	54.80	88.37	0.977
	Wanda	73.96	80.09	790	88.44	61.54	44.89	2,106.9	65.10	53.42	53.10	52.40	86.00	0.924
	SparseGPT	61.04	76.60	721	81.19	58.24	36.89	1,675.00	59.22	42.31	40.40	48.70	85.25	0.832

Table 11: Results of various parameter compression methods on Qwen2-VL-7B.

Methods	Speedup	AI2D	MMMU	MME	TextVQA	MathVista
AWQ	1.0x	90.80	57.20	2320.6	50.26	66.53
AWQ+SnapKV	1.65x	90.74	57.40	2235.9	48.78	66.80

Table 12: Results on Qwen2-VL-7B on combining token and parameter compression.

A.8 Architectural Impact on Compression Effectiveness

We analyze how architectural differences in LVLM backbones (LLaVA-OneVision, Qwen2-VL, InternVL-2.5) affect compression effectiveness. Two primary aspects are considered:

- **High-Resolution Image Processing.** LLaVA-OneVision and InternVL-2.5 use *anyres*-like strategies, splitting high-resolution images into many patches, averaging 4180 visual tokens per input on our benchmark. In contrast, Qwen2-VL employs NaViT, a visual encoder with native dynamic-resolution support that merges visual tokens, reducing their average count to 762 (see Dehghani et al. (2023) for NaViT details). This difference in initial visual token count directly impacts *token compression*. For example, LLaVA-OneVision-7B consistently outperforms similarly sized Qwen2-VL-7B in various token compression budgets (Table 9). This suggests that a larger pool of initial visual tokens offers greater flexibility for compression, as its richer representation can tolerate more aggressive reduction without substantial performance loss.

- **Parameter Size.** To isolate the impact of model scale, we evaluate Qwen2-VL-2B against Qwen2-VL-7B using various parameter compression methods. The results are shown in Table 13.

Settings	Method	Size	AI2D	MathVista	MME	MMMU	TextVQA	OP
100%	Original	2B	83.83	46.20	1872.0	40.85	79.70	1.0000
		7B	91.02	58.30	2327.7	50.77	81.82	1.0000
20%	ECOFLAP	2B	80.99	44.00	1883.4	39.44	79.19	0.9769
		7B	91.13	57.10	2288.0	50.00	81.82	0.9900
	Wanda	2B	81.99	45.30	1884.5	39.89	79.31	0.9874
		7B	91.16	58.90	2312.6	50.44	81.79	1.0000
	SparseGPT	2B	82.35	46.30	1852.9	40.00	79.44	0.9901
		7B	91.03	59.30	2317.2	50.78	82.08	1.0030
50%	ECOFLAP	2B	71.99	33.50	1585.2	34.44	76.39	0.8497
		7B	89.67	52.40	2090.8	46.11	80.64	0.9356
	Wanda	2B	72.83	34.90	1721.8	36.44	76.26	0.8812
		7B	88.63	53.50	2129.1	45.00	80.61	0.9350
	SparseGPT	2B	72.15	35.70	1535.1	33.56	75.75	0.8472
		7B	88.44	53.10	2106.9	44.89	80.09	0.9309
2:4	ECOFLAP	2B	44.79	28.20	1008.7	27.11	66.12	0.6445
		7B	82.32	40.70	1871.9	39.56	77.45	0.8313
	Wanda	2B	47.83	27.70	1207.2	26.89	67.01	0.6695
		7B	82.29	39.30	1967.3	38.78	78.08	0.8336
	SparseGPT	2B	49.92	26.20	1027.8	29.22	66.91	0.6624
		7B	81.19	40.40	1675.0	36.89	76.60	0.8000
W4A16	AWQ	2B	82.90	46.50	1835.4	39.82	79.34	0.9893
		7B	90.80	58.30	2232.5	48.78	81.31	0.9826

Table 13: Results of various parameter compression methods on different model sizes of Qwen2-VL.

A.9 Visualization Head Attention Distribution

To illustrate how head-adaptive mechanisms can capture diverse attention patterns, we analyzed attention distributions across heads in several randomly selected layers. Qwen2-VL-7B was chosen for this visualization because its relatively fewer image tokens enhance attention map readability. Indeed, Figure 4 shows that different heads within the same layer exhibit distinct attention patterns.

A.10 Visualizations on VL-Cache Budget Allocation

We evaluated VL-Cache on eight benchmarks across three LVLMs, all at a 5% total budget. The models consistently exhibited similar, highly skewed layer-wise budget allocations (Figures 5 and 6). True to its design, VL-Cache's layer-adaptive mechanism heavily favored early "dense" layers, particularly the first two. Allocation minima were observed at layer 4 for LLaVA-OV-7B and Qwen2-VL-7B, and at layer 5 for InternVL2.5-38B, with subsequent layers typically receiving far below-average budgets. This consistent front-loading suggests performance could be improved by reallocating budget from these over-resourced early layers to the under-resourced later ones. Effective layer-adaptive strategies therefore demand a more nuanced resource balance, rather than just aggressive front-loading.

A.11 Visualization of VL-Cache Token Selection

We visualize visual tokens selected by VL-Cache at layers 0, 1, and 2 in LLaVA-OV-7B (5% budget, across 6 benchmarks). Figure 8 reveals that for the first two layers (0 and 1), selected token distributions are strikingly similar: they predominantly cluster towards the end of the visual token sequence, potentially limiting their utility in deriving final answers. Layer 2, in contrast, displays a more uniform distribution of selected tokens across the entire sequence. Although VL-Cache's strategy allocates a higher token density to these early layers, these observed patterns suggest such aggres-

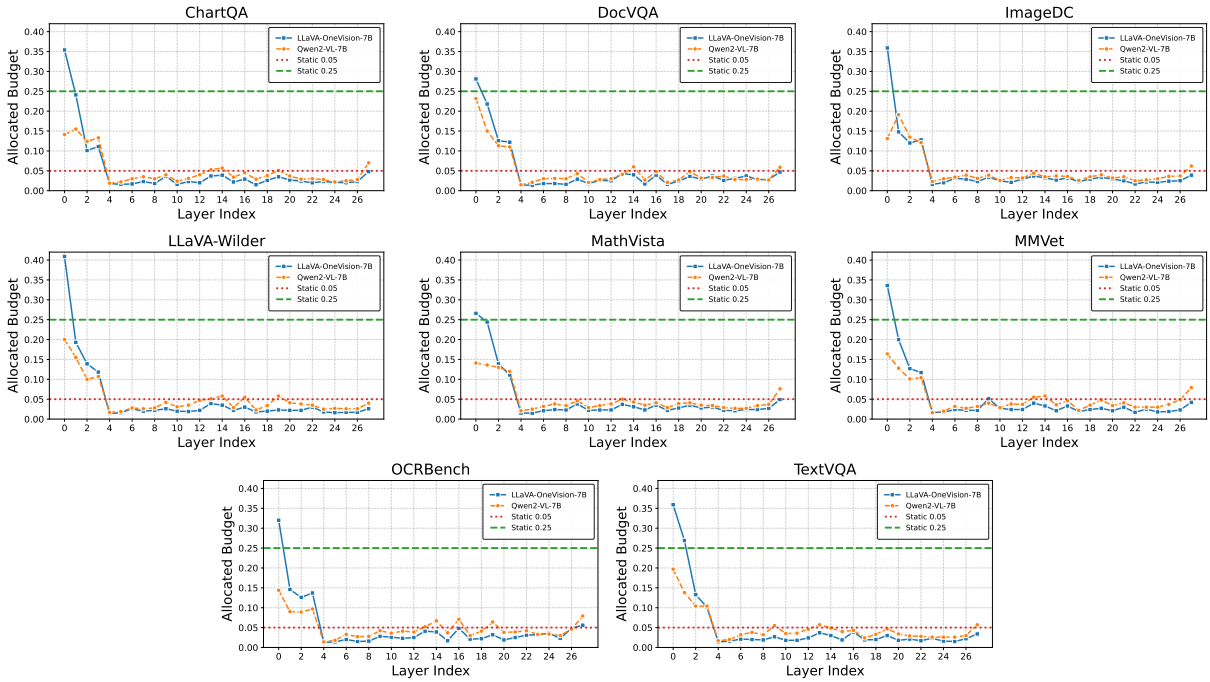


Figure 5: Budgets distribution of LLaVA-OV-7B and Qwen2-VL-7B across 8 benchmarks under 5% budget. Each subplot shows a random sample from the respective benchmark.

sive, front-loaded budgeting may not be optimal.

A.12 Visualization of Visual Sink Tokens

To further illustrate the visual token sink phenomenon, we visualize attention heatmaps for 15 randomly selected ChartQA samples using LLaVA-OV-7B and Qwen2-VL-7B. Figure 10 clearly shows a visual sink emerging after the visual encoder's forward pass: image information consistently converges to a limited subset of visual tokens. Furthermore, Figures 7 and 9 compare visual token selection by VisionZip (image-guided) against FastV (text-guided). VisionZip tends to retain these critical sink tokens, whereas FastV often fails to capture many of them, explaining observed performance differences.

A.13 Case of Different Merge Strategies

Table 14 contrasts LOOK-M's outputs under two token merge strategies: cross-modal versus modality-specific. Modality-specific merging yields an accurate image caption; in contrast, cross-modal merging results in an image-irrelevant output.

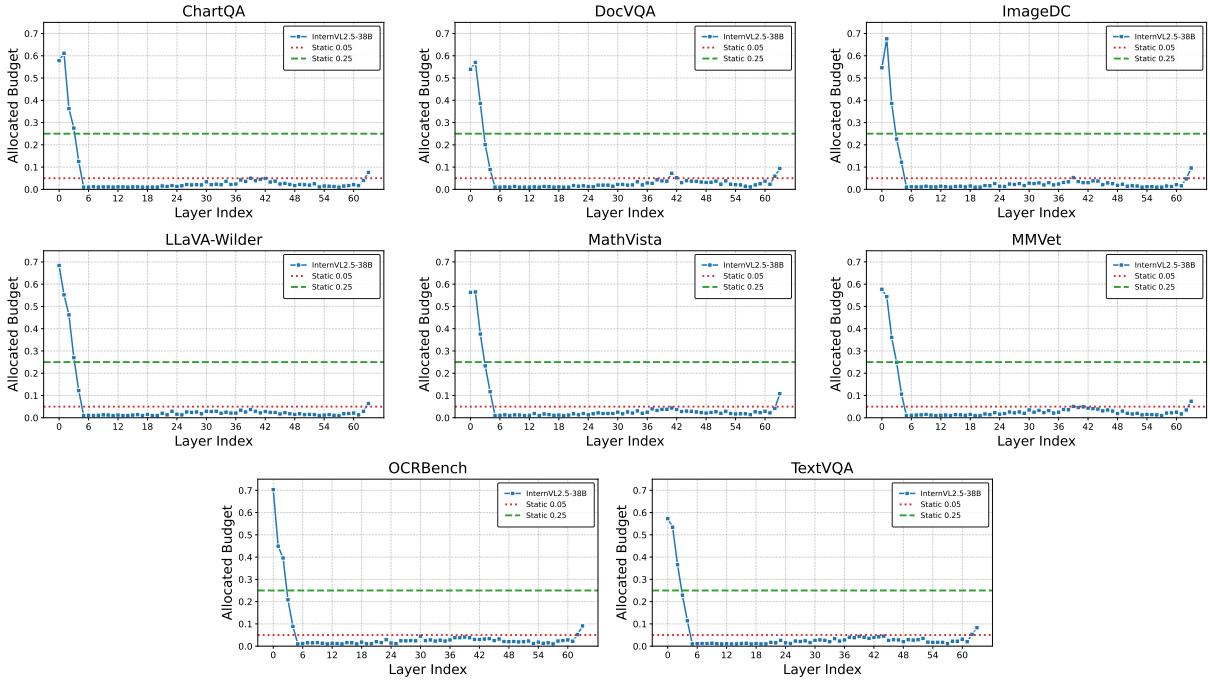


Figure 6: Budgets distribution of InternVL2.5-38B across 8 benchmarks under 5% budget. Each subplot shows a random sample from the respective benchmark.

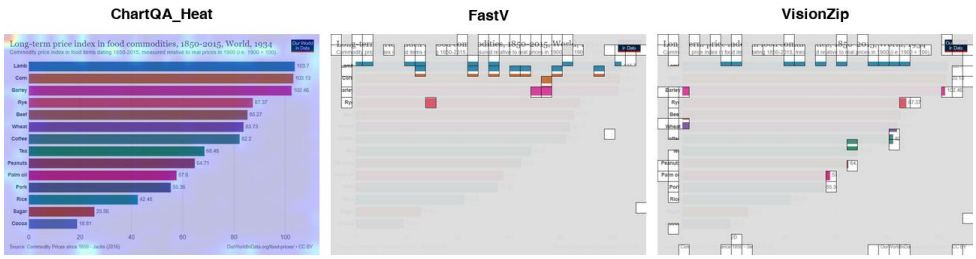
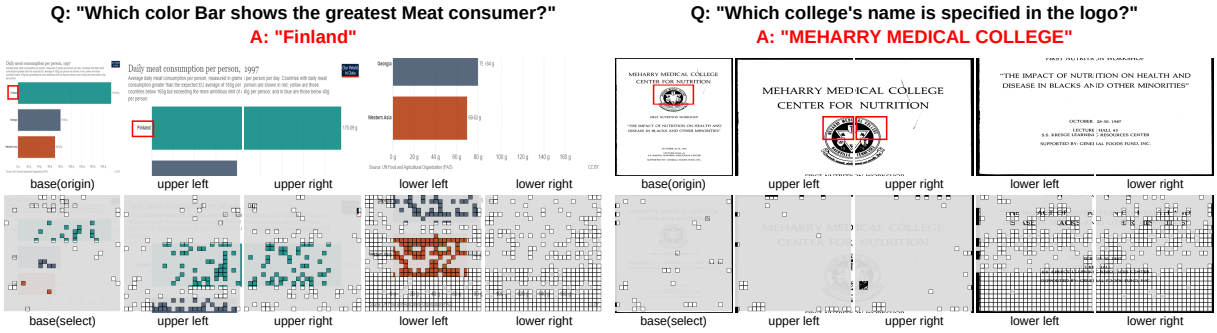
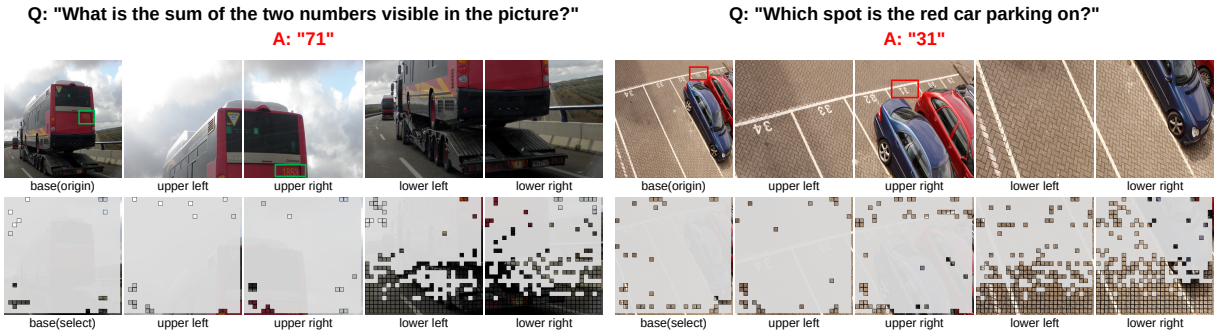


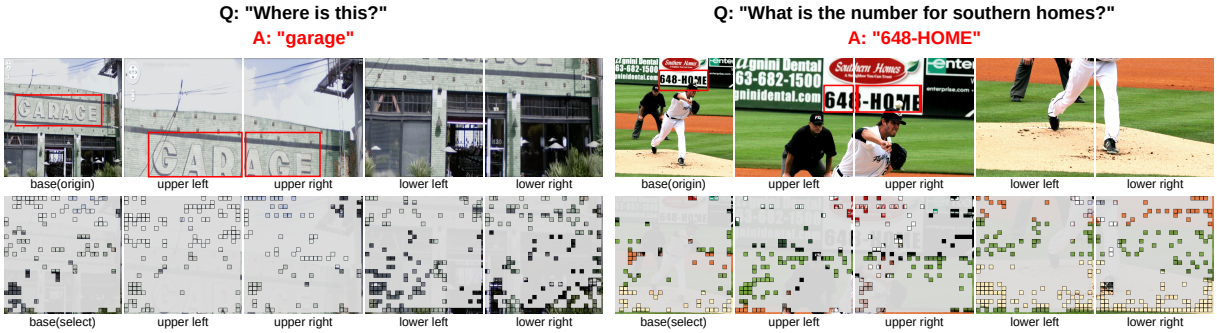
Figure 7: Visualization of token selection strategies on ChartQA with Qwen2-VL-7B under 10% budget. Left to right: attention heatmap, tokens retained by FastV and VisionZip. VisionZip selects more critical tokens.



(a) Visualization of Layer 0 visual token selection: representative cases from ChartQA (left) and DocVQA (right)

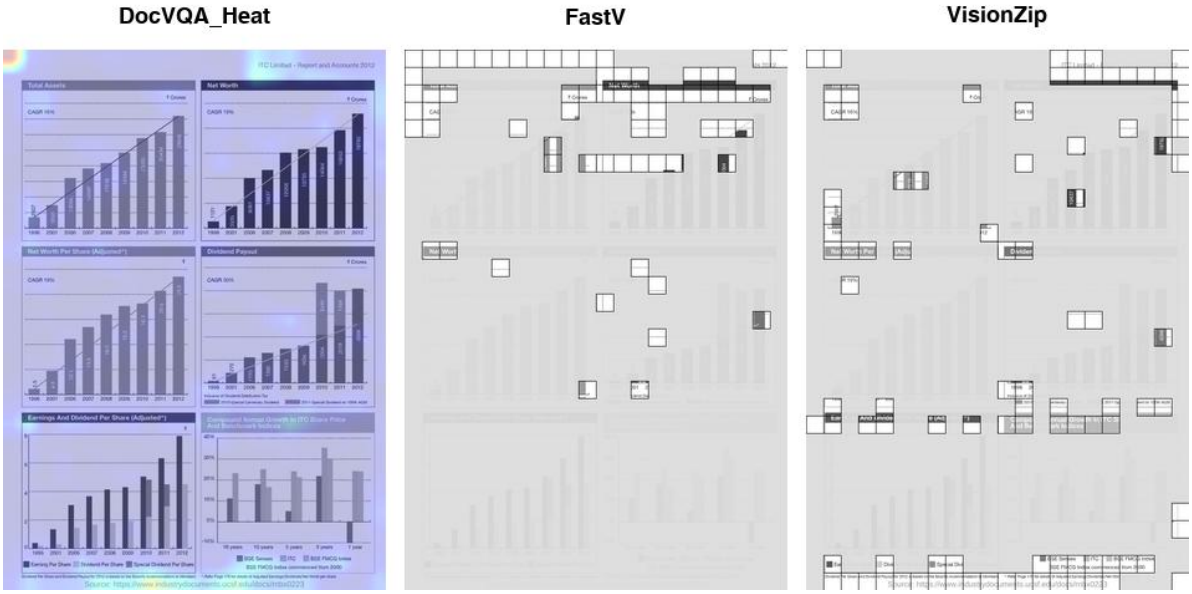


(b) Visualization of Layer 1 visual token selection: representative cases from MathVista (left) and MMVet (right)

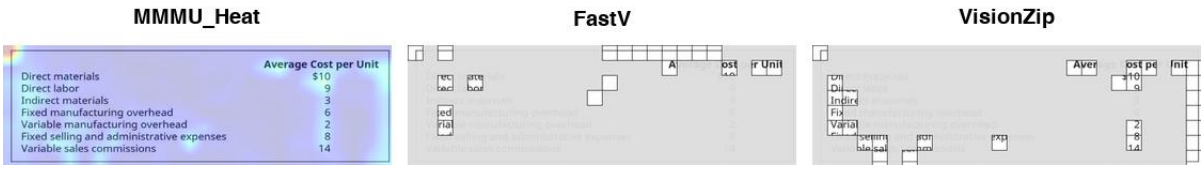


(c) Visualization of Layer 2 visual token selection: representative cases from OCRBench (left) and TextVQA (right)

Figure 8: Visualization of VL-Cache selected visual tokens in layers 0–2 on LLaVA-OneVision-7B. Highlighted regions with bounding boxes are critical to the answer.



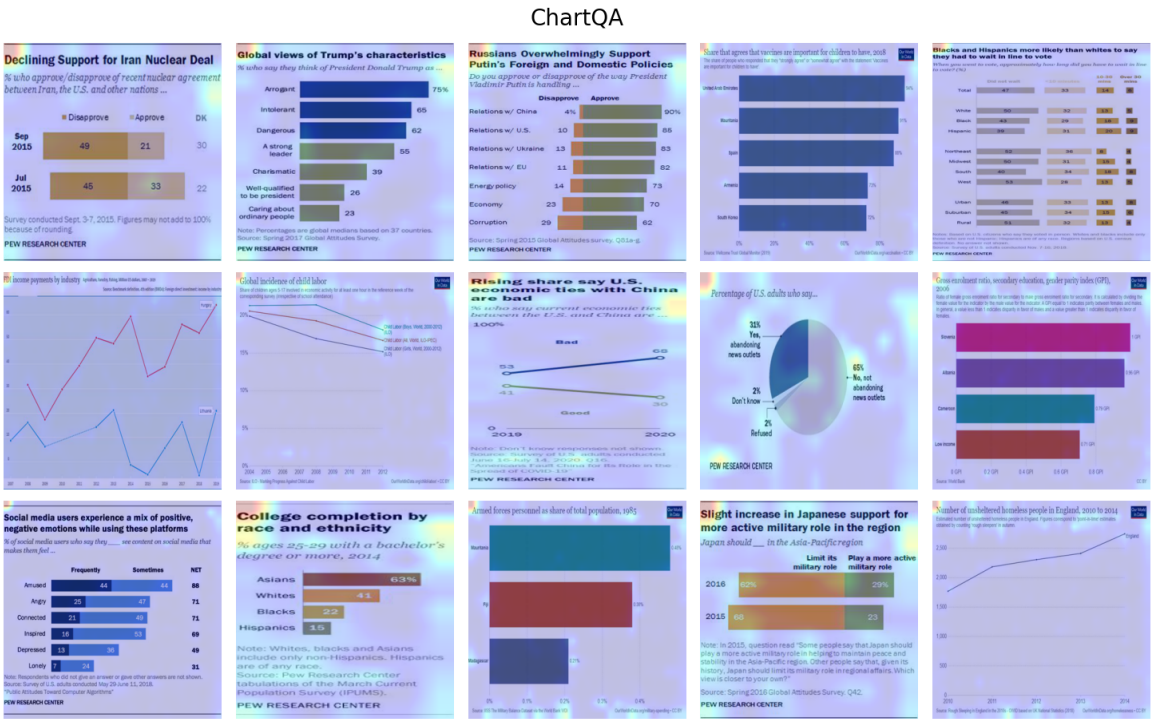
(a) Visualization results of randomly selected cases from DocVQA



(b) Visualization results of randomly selected cases from MMMU

Figure 9: Visualization of token selection strategies on DocVQA and MMMU with Qwen2-VL-7B under 10% budget. Left to right: attention heatmap, tokens retained by FastV and VisionZip. VisionZip selects more critical tokens than FastV.

(a) Heatmap visualization results of ChartQA on LLaVA-OneVision-7B



(b) Heatmap visualization results of ChartQA on Qwen2-VL-7B

Figure 10: Heatmap visualizations of 15 randomly selected examples from the ChartQA benchmark on LLaVA-OneVision-7B and Qwen2-VL-7B.

Visual input example for image caption	
	
User	Please give a brief description of this picture.
Qwen2-VL-7B	The image shows a classic Honda motorcycle parked on a gravel surface. The motorcycle has a black and chrome finish, with a quilted seat and a small windshield.
LOOK-M$_{\text{origin}}$	The image shows a close-up of a person's hand holding a small, round object. The object appears to be a small, metallic ball or a similar spherical item.
LOOK-M$_{\text{change}}$	The image shows a motorcycle parked outdoors. The motorcycle has a classic design with a prominent front wheel and a seat that appears to be made of leather.

Table 14: Comparative analysis of response quality in Qwen2-VL-7B original model vs. LOOK-M with origin merge and modality-specific merge strategies under extreme compression ratios

Benchmarks	Models	Methods	1%	5%	10%	20%	40%	100%
DocVQA (Split:test) (Metric:ANLS)	LLaVA-OneVision-7B	Random	31	55	69	79	85	87
		StreamingLLM	42	62	71	74	79	87
		H2O	58	78	82	**86**	**87**	87
		SnapKV	**76**	**82**	83	85	86	87
		PyramidKV	70	76	75	78	81	87
		LOOK-M	38	74	81	85	**87**	87
		VL-Cache	69	**82**	**85**	**86**	**87**	87
	Qwen2-VL-7B	Random	13	60	78	89	**94**	95
		StreamingLLM	42	56	65	74	84	95
		H2O	47	67	79	89	93	95
		SnapKV	**78**	**90**	**92**	92	**94**	95
		PyramidKV	66	83	82	84	87	95
		LOOK-M	27	58	77	89	**94**	95
		VL-Cache	52	83	91	**93**	**94**	95
	InternVL2.5-38B	StreamingLLM	0	46	53	63	80	94
		H2O	58	76	86	91	93	94
		SnapKV	34	**91**	**93**	**93**	**94**	94
		PyramidKV	53	90	92	**93**	93	94
		LOOK-M	12	54	78	90	93	94
		VL-Cache	**64**	87	91	**93**	93	94
ChartQA (Split:overall) (Metric:Acc)	LLaVA-OneVision-7B	Random	45.04	59.76	64.56	69.52	72.72	80.00
		StreamingLLM	58.20	65.44	69.84	73.88	75.36	80.00
		H2O	67.32	75.48	76.44	**77.48**	**77.84**	80.00
		SnapKV	**74.64**	**76.48**	76.68	77.20	77.28	80.00
		PyramidKV	72.68	76.20	76.08	76.24	76.80	80.00
		LOOK-M	34.44	73.72	**76.72**	77.40	77.72	80.00
		VL-Cache	66.64	74.12	75.96	77.08	77.56	80.00
	Qwen2-VL-7B	Random	13.00	45.96	66.08	72.32	79.68	81.56
		StreamingLLM	50.28	61.52	65.04	69.60	76.32	81.56
		H2O	54.36	70.12	74.56	78.60	80.88	81.56
		SnapKV	55.96	**77.32**	**79.28**	**80.20**	**81.32**	81.56
		PyramidKV	**67.24**	72.08	76.08	78.96	80.40	81.56
		LOOK-M	23.00	48.52	68.24	76.96	79.48	81.56
		VL-Cache	53.32	67.00	74.28	79.16	80.68	81.56
	InternVL2.5-38B	StreamingLLM	0.68	67.08	71.68	73.72	80.00	88.04
		H2O	34.64	80.32	84.76	86.88	87.84	88.04
		SnapKV	60.60	84.80	86.64	87.28	87.92	88.04
		PyramidKV	**76.40**	**85.36**	**86.68**	87.24	87.52	88.04
		LOOK-M	8.80	54.48	75.48	84.04	87.16	88.04
		VL-Cache	58.60	82.48	85.88	**87.64**	**88.16**	88.04
TextVQA (Split:val) (Metric:Acc)	LLaVA-OneVision-7B	Random	39.63	57.15	64.29	68.96	71.81	74.79
		StreamingLLM	47.01	57.29	64.49	70.25	71.41	74.79
		H2O	64.96	72.16	**73.71**	74.14	74.54	74.79
		SnapKV	**65.62**	**72.54**	70.86	71.62	73.77	74.79
		PyramidKV	58.56	68.39	66.37	66.60	69.70	74.79
		LOOK-M	44.74	70.29	73.07	**74.39**	**74.76**	74.79
		VL-Cache	61.85	70.73	72.92	73.46	74.16	74.79
	Qwen2-VL-7B	Random	5.21	48.04	61.58	72.78	79.78	81.82
		StreamingLLM	31.60	49.70	58.89	67.77	75.18	81.82
		H2O	51.16	64.50	71.40	77.85	81.13	81.82
		SnapKV	57.38	**73.45**	**78.37**	**80.83**	81.19	81.82
		PyramidKV	**67.50**	71.01	73.90	77.89	78.87	81.82
		LOOK-M	31.89	52.07	70.24	78.62	81.24	81.82
		VL-Cache	47.03	68.17	77.74	80.81	**81.79**	81.82
	InternVL2.5-38B	StreamingLLM	0.83	51.49	53.24	58.97	70.28	82.87
		H2O	63.29	76.08	79.93	81.96	82.52	82.87
		SnapKV	36.56	**80.58**	**82.15**	**82.69**	**82.77**	82.87
		PyramidKV	56.73	79.70	80.64	81.91	82.16	82.87
		LOOK-M	12.56	61.42	76.33	81.13	82.34	82.87
		VL-Cache	**64.40**	76.54	80.35	82.29	82.71	82.87

Benchmarks	Models	Methods	1%	5%	10%	20%	40%	100%
OCRBench (Split:test) (Metric:Acc)	LLaVA-OneVision-7B	Random	67	335	455	530	564	595
		StreamingLLM	94	222	320	386	502	595
		H2O	229	407	**495**	**554**	588	595
		SnapKV	**281**	**415**	445	475	553	595
		PyramidKV	275	337	354	358	426	595
		LOOK-M	81	375	463	544	**589**	595
		VL-Cache	231	372	460	505	551	595
	Qwen2-VL-7B	Random	109	192	336	491	658	813
		StreamingLLM	116	199	248	349	465	813
		H2O	153	334	441	570	713	813
		SnapKV	199	**428**	**542**	**654**	**725**	813
		PyramidKV	**241**	417	535	642	662	813
		LOOK-M	112	202	299	441	587	813
		VL-Cache	216	386	505	592	660	813
	InternVL2.5-38B	StreamingLLM	51	121	162	214	399	802
		H2O	143	487	656	752	782	802
		SnapKV	359	**652**	**720**	**777**	**791**	802
		PyramidKV	**478**	637	664	734	765	802
		LOOK-M	18	183	491	695	771	802
		VL-Cache	193	487	630	733	778	802
MathVista (Split:testmini) (format: COT) (Metric: GPT)	LLaVA-OneVision-7B	Random	41.10	44.20	47.80	53.50	55.90	58.20
		StreamingLLM	40.40	43.10	45.20	48.50	54.40	58.20
		H2O	42.40	48.90	53.70	55.40	**56.60**	58.20
		SnapKV	**54.40**	**55.30**	**56.30**	**56.70**	**56.60**	58.20
		PyramidKV	53.00	54.10	54.50	56.50	56.30	58.20
		LOOK-M	44.30	53.10	53.60	54.00	55.10	58.20
		VL-Cache	49.50	53.40	55.80	55.90	56.50	58.20
	Qwen2-VL-7B	Random	45.00	49.60	54.00	56.30	57.20	58.30
		StreamingLLM	47.40	54.80	55.30	56.20	56.40	58.30
		H2O	49.00	55.60	55.80	57.00	57.80	58.30
		SnapKV	49.90	56.50	57.20	57.40	**58.00**	58.30
		PyramidKV	**50.30**	**57.30**	**57.60**	**57.50**	**58.00**	58.30
		LOOK-M	35.80	54.30	56.20	55.50	56.70	58.30
		VL-Cache	52.30	55.00	54.80	56.40	56.70	58.30
	InternVL2.5-38B	StreamingLLM	40.80	48.00	50.10	54.50	59.10	70.20
		H2O	50.40	60.90	64.60	**68.60**	70.50	70.20
		SnapKV	53.70	64.30	**68.40**	68.40	70.00	70.20
		PyramidKV	**56.10**	**66.10**	66.20	68.00	69.60	70.20
		LOOK-M	38.30	56.50	61.60	66.70	69.40	70.20
		VL-Cache	53.90	60.70	65.00	67.60	**70.60**	70.20
MMVet (Split:test) (Metric: GPT)	LLaVA-OneVision-7B	Random	19.60	34.10	40.80	46.50	47.70	55.00
		StreamingLLM	24.60	39.80	46.30	48.10	50.00	55.00
		H2O	**33.60**	46.10	**51.60**	**53.20**	**54.10**	55.00
		SnapKV	32.90	**47.90**	50.20	52.20	53.30	55.00
		PyramidKV	**33.60**	43.60	47.80	47.20	51.60	55.00
		LOOK-M	20.60	45.20	49.10	52.20	53.60	55.00
		VL-Cache	30.00	42.50	49.30	50.20	51.90	55.00
	Qwen2-VL-7B	Random	4.10	10.90	22.30	33.90	52.20	65.40
		StreamingLLM	1.10	16.30	28.00	38.60	44.60	65.40
		H2O	8.30	27.30	39.20	53.70	56.00	65.40
		SnapKV	**12.10**	**33.50**	**43.50**	**55.10**	**56.50**	65.40
		PyramidKV	11.20	31.20	39.30	46.70	51.40	65.40
		LOOK-M	1.40	17.80	30.40	46.30	55.30	65.40
		VL-Cache	7.00	25.80	38.10	47.40	53.30	65.40
	InternVL2.5-38B	StreamingLLM	0.00	21.20	32.20	37.70	53.70	69.40
		H2O	15.90	47.50	58.80	**68.00**	70.70	69.40
		SnapKV	23.00	**51.40**	**63.60**	67.10	69.20	69.40
		PyramidKV	**33.30**	51.10	57.70	64.00	67.10	69.40
		LOOK-M	2.80	29.20	50.50	57.80	**71.10**	69.40
		VL-Cache	16.50	35.50	48.40	57.50	67.00	69.40

Benchmarks	Models	Methods	1%	5%	10%	20%	40%	100%
LLaVA-Wilder (Split:test) (Metric: GPT)	LLaVA-OneVision-7B	Random	34.10	51.60	58.20	63.60	67.70	71.40
		StreamingLLM	48.00	65.70	69.20	**71.20**	69.40	71.40
		H2O	62.50	**69.20**	**70.00**	71.00	70.60	71.40
		SnapKV	**63.70**	66.20	69.40	71.00	**70.90**	71.40
		PyramidKV	62.30	67.20	66.80	67.50	69.00	71.40
		LOOK-M	43.40	65.50	65.70	66.80	68.40	71.40
		VL-Cache	62.40	67.10	69.10	69.50	**70.90**	71.40
	Qwen2-VL-7B	Random	12.50	21.00	31.50	42.30	59.90	72.70
		StreamingLLM	19.50	39.60	53.40	61.00	69.20	72.70
		H2O	37.20	57.30	62.00	65.60	**72.50**	72.70
		SnapKV	27.70	58.10	**64.50**	66.60	69.50	72.70
		PyramidKV	36.30	**58.50**	64.10	65.00	68.70	72.70
		LOOK-M	19.10	52.30	60.50	67.80	69.80	72.70
		VL-Cache	**39.80**	57.90	63.40	**68.00**	69.30	72.70
	InternVL2.5-38B	StreamingLLM	21.20	45.80	58.70	66.60	71.00	74.80
		H2O	51.90	69.10	**72.60**	**73.40**	74.10	74.80
		SnapKV	52.70	69.10	70.90	72.60	**74.40**	74.80
		PyramidKV	**56.20**	**69.90**	70.30	71.80	72.30	74.80
		LOOK-M	16.80	59.20	67.10	71.80	73.60	74.80
		VL-Cache	55.20	66.80	69.80	72.80	72.00	74.80
ImageDC (Split:DC100_EN) (Metric: GPT)	LLaVA-OneVision-7B	Random	27.35	54.35	73.80	83.35	86.65	87.25
		StreamingLLM	38.40	77.95	80.50	85.25	86.50	87.25
		H2O	**77.05**	**85.30**	**86.00**	**86.75**	86.53	87.25
		SnapKV	28.00	77.00	80.15	83.25	86.25	87.25
		PyramidKV	27.75	69.55	75.45	80.40	84.00	87.25
		LOOK-M	24.55	83.95	85.99	86.50	**86.70**	87.25
		VL-Cache	21.00	51.80	70.95	80.00	84.95	87.25
	Qwen2-VL-7B	Random	15.25	38.30	56.60	69.50	82.10	86.35
		StreamingLLM	16.70	31.35	76.95	82.15	85.10	86.35
		H2O	**29.55**	**82.25**	**83.85**	**85.80**	**85.80**	86.35
		SnapKV	17.30	29.85	64.05	77.95	83.75	86.35
		PyramidKV	16.90	31.05	56.80	69.90	78.55	86.35
		LOOK-M	24.05	23.65	29.15	82.10	85.55	86.35
		VL-Cache	15.05	17.80	41.80	70.35	82.30	86.35
	InternVL2.5-38B	StreamingLLM	13.25	48.10	51.40	73.30	83.50	86.45
		H2O	**59.60**	**83.55**	**85.05**	**85.85**	**86.65**	86.45
		SnapKV	37.60	75.05	80.60	85.50	85.25	86.45
		PyramidKV	46.75	77.65	81.95	84.45	86.10	86.45
		LOOK-M	19.35	62.00	70.50	76.50	86.40	86.45
		VL-Cache	24.90	56.55	71.40	82.95	85.50	86.45

Table 15: Main results of various KV cache compression methods on different models and tasks.**Bold** denotes the best result under the same setting.