Supplementary material for VISion On Request: Enhanced VLLM efficiency with sparse, dynamically selected, vision-language interactions

Adrian Bulat[1,2 *] Alberto Baldrati[1*] Ioannis Maniadis Metaxas[1*]
Yassine Ouali[1] Georgios Tzimiropoulos[1,3]
[1]Samsung AI Cambridge [2]Technical University of Iasi [3]Queen Mary University of London

S1. Identifying promising configurations for adaptive training and inference

Considering a model with L_{SA} self-attention layers, we can create $2^{L_{SA}}$ different configurations by choosing to execute or skip each self-attention layer. This results in a vast configuration space, making it somewhat impractical to evaluate all of them. Moreover, many configurations may lead to catastrophic performance degradation, as they may skip critical layers needed for certain tasks. Hence, to facilitate the training process, we seek to identify a subset of promising configurations that maintain high performance. This subset can then be used for adaptive training and inference.

Figure S1 visualizes the performance of a representative subset of configurations on different datasets, with each row representing a configuration and each column a dataset. The color intensity indicates the relative accuracy achieved by that configuration on the respective dataset. From this visualization, we can identify that: (1) dropping the 1st layer leads to significant performance degradation across all datasets, indicating its critical role; (2) configurations with very few self-attention layers (e.g., 1 or 2) perform poorly on complex tasks, while those with more layers generally yield better results; (3) less vision intensive tasks generally prefer a configuration close to early-exit while more complex tasks benefit from a uniform distribution of self-attention layers.

Based on these observations, for a 0.5B model, we subsequently select the following configurations, where each number denotes the location at which a self-attention layer is executed for the vision tokens: `[1,4]`, `[1,7]`, `[1,4,7]`, `[1,4,16]`, `[1,7,16]`, `[1,10,16]`, `[1,4,7,16]`, `[1,4,7,22]`, `[1,4,10,16]`, `[1,4,7,10,16]`, `[1,4,7,16,22]`, `[1,4,7,10,16,22]`, `[1,4,7,10,16,19,22]`.

*Equal contribution

S2. Per-dataset saving rate

In the main manuscript, for brevity, we report the computational savings aggregated across all datasets. Herein, we provide a more detailed per-dataset analysis. Table S1 summarizes the results. For each variant, the top row indicates the accuracy, while the bottom row shows the FLOPs savings relative to the baseline LLaVA-OV model. The same general pattern holds: easy tasks can be solved with very few self-attention layers, while hard tasks require more layers for optimal performance. Our router is able to correctly identify this trend.

To provide further insight into the routing mechanism's behaviour, we present in Figure S2 the layer configurations it chooses for each test set dataset. Two significant observations can be made from this figure: a) the routing mechanism is largely consistent with regard to the computational budget allocated for each dataset, as the configurations chosen for each dataset tend to have a similar number of layers, and b) despite the fact that the original labels are defined in a per-dataset basis, the routing mechanism indeed operates on a per-sample basis, which makes it adaptive to individual samples' complexity.

S3. Performance across all individual configurations

Our universal model is trained by randomly sampling a viable configuration during training. To evaluate the effectiveness of each configuration, Figure S3 illustrates their performance across all downstream tasks. On the easy partition, most configurations achieve similar performance regardless of their computational cost. In contrast, for challenging tasks, performance improves almost linearly with the computational budget, highlighting once more the importance of additional self-attention layers for fine-grained reasoning.

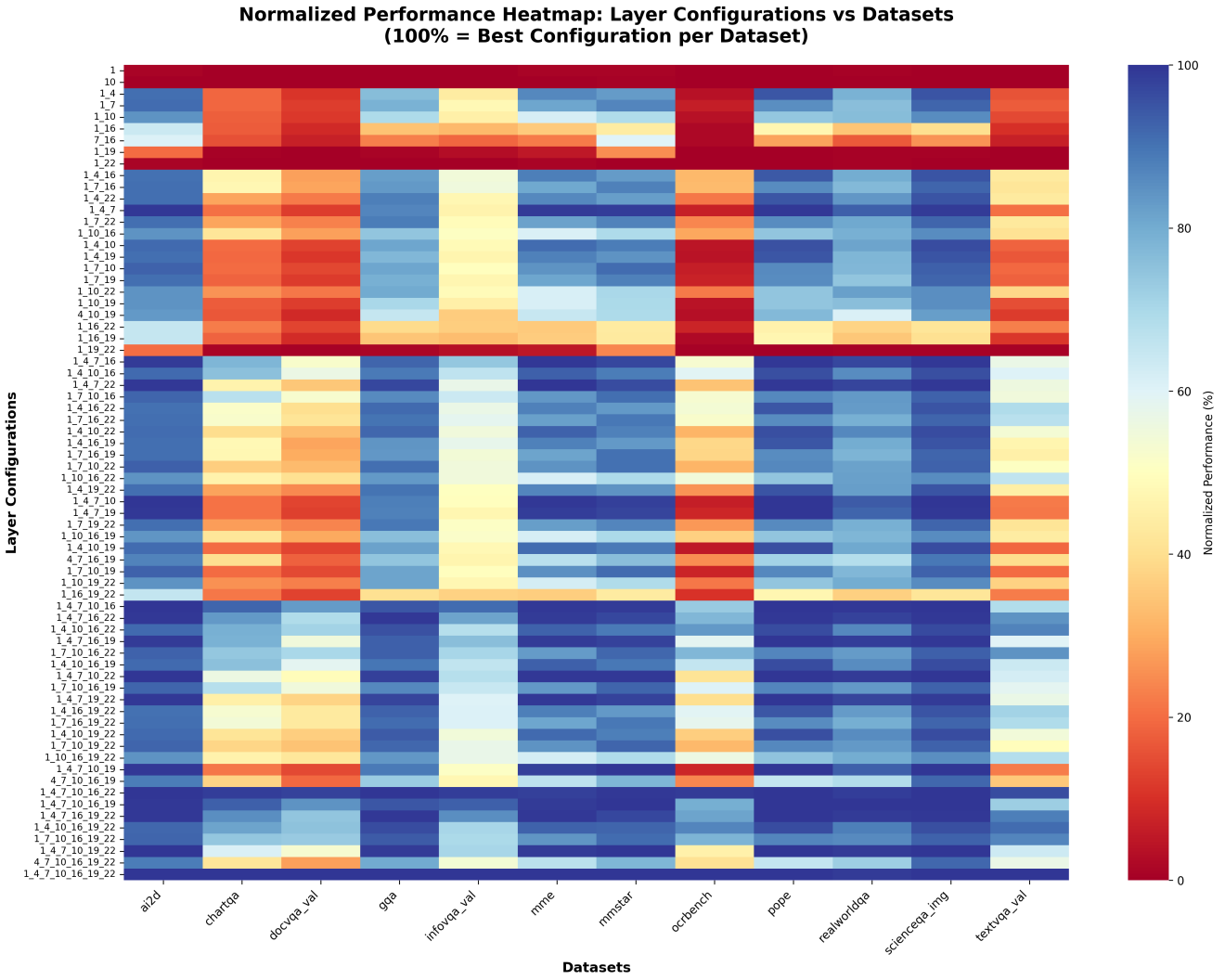

Figure S1. Performance heatmap for different configurations across datasets. Each row represents a configuration, and each column corresponds to a dataset. The color intensity indicates the relative accuracy achieved by that configuration on the respective dataset.

Table S1. Per-dataset saving rates. We compare our method against state-of-the-art approaches using a shared LLaVA-OV (0.5B) backbone. For each method, the top row indicates the accuracy, while the bottom row shows the FLOPs savings relative to the baseline LLaVA-OV model. The metrics used are accuracy for most datasets, except for MME where we report a score (higher is better).

Method	RWQA	SQA	GQA	MME	MMSTAR	POPE	TxtVQA	AI2D	CQA	OCRB	InfoVQA	DocVQA
VISOR	54.6	75.3	61.8	60.5	40.1	87.6	67.8	61.5	65.2	61.8	37.6	68.9
	8.4×	8×	12×	8.3×	10×	12×	6.3×	12×	8×	6.6×	6×	6×
VISOR-TR	55.4	75.4	60.7	59.5	38.5	87.7	67.4	61.9	65.3	60.7	37.4	67.7
	24×	16×	24×	17.7×	17.7×	24×	14.5×	24×	16×	15×	12×	12×

S4. Oracle performance analysis

To assess the potential of our adaptive inference mechanism, we conduct an oracle analysis where we select the optimal configuration for each sample from our predefined set. Figure S4 illustrates the distribution of the selected configurations across all samples such that the overall accuracy is maximized. These results reinforce the conclusion that most samples can be accurately processed using configurations with very few self-attention layers, while hard tasks require more layers for optimal performance.

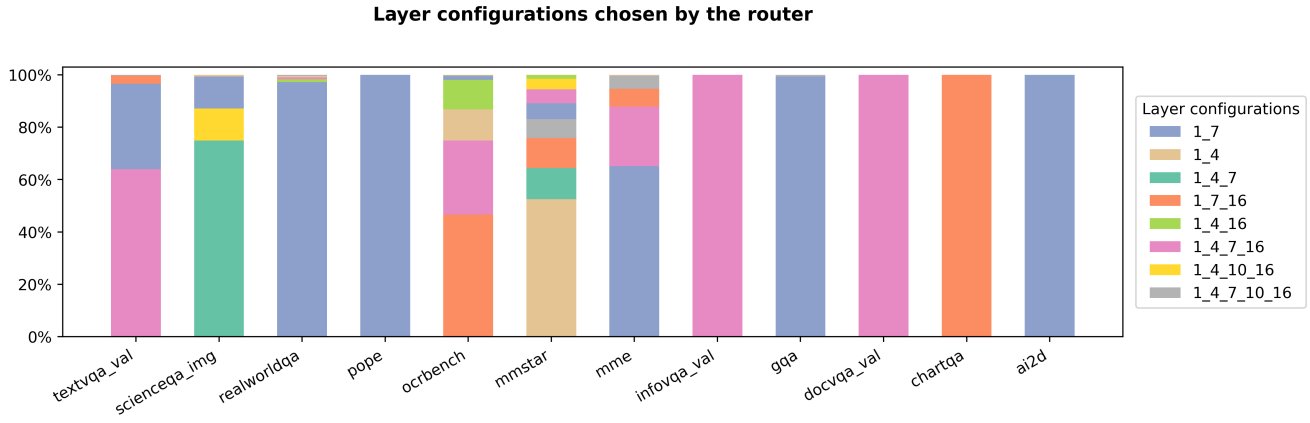

Figure S2. Layer configuration assignments made by the router for each test dataset.

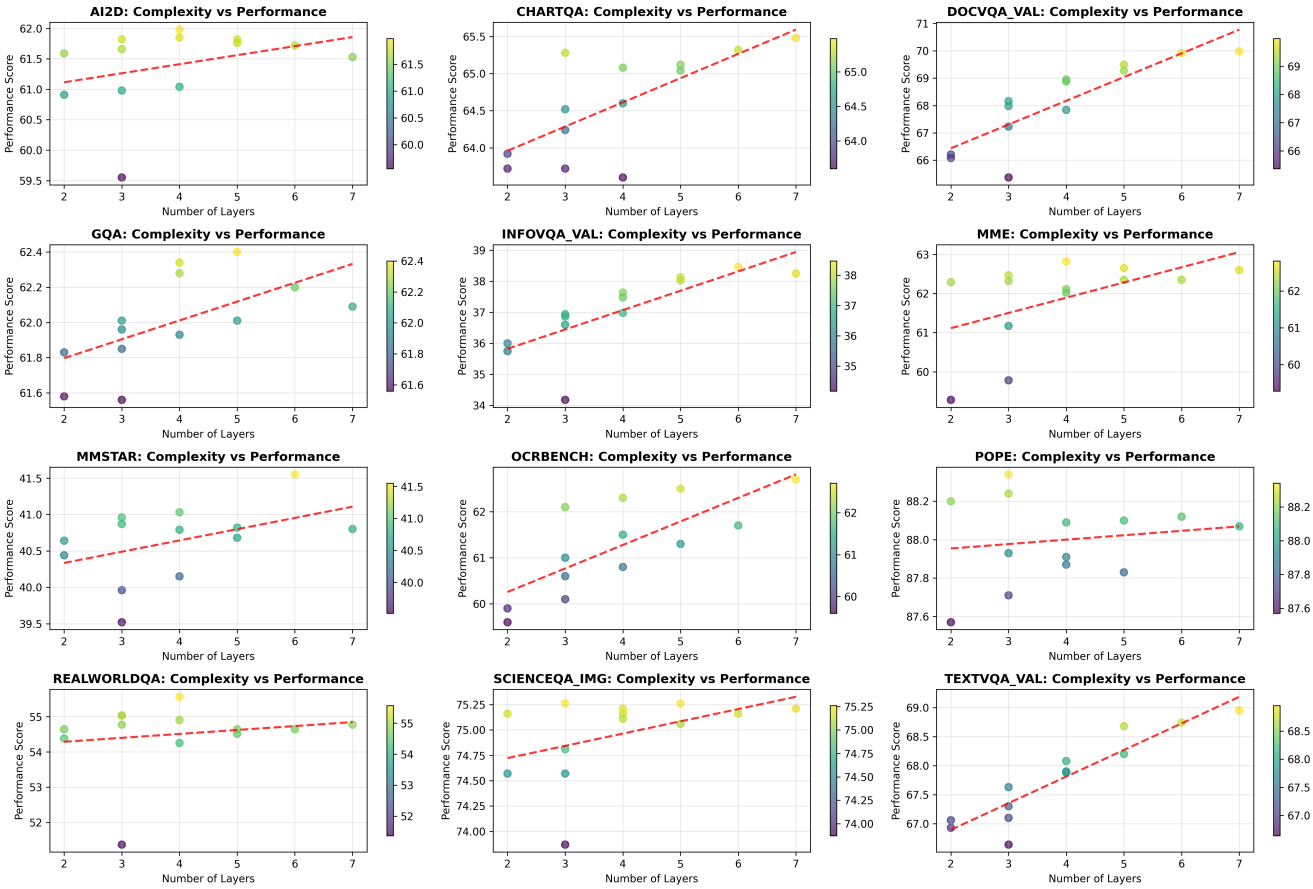

Figure S3. Performance for various VISOR configurations.

To find the optimal configuration per sample, we compare the per-config generations token by token to the tokenized ground-truth, and score it as the number of matches up to the first incorrect match. Then, we select the configuration with the most matches. If multiple ones have the same score, we select the one with the minimum number of SA layers (i.e., the one with the most dropped layers).

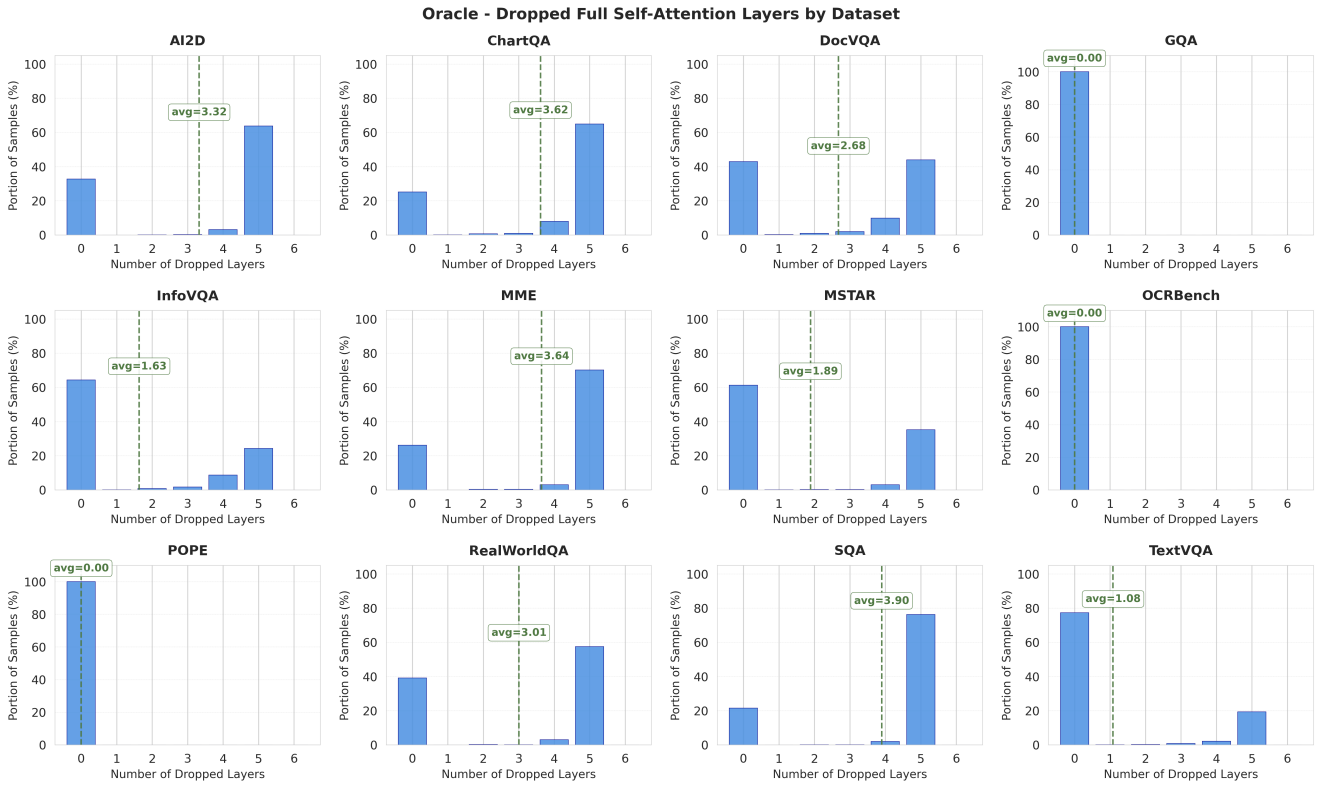

Figure S4. VISOR oracle: smallest amount of layers that maximizes accuracy.

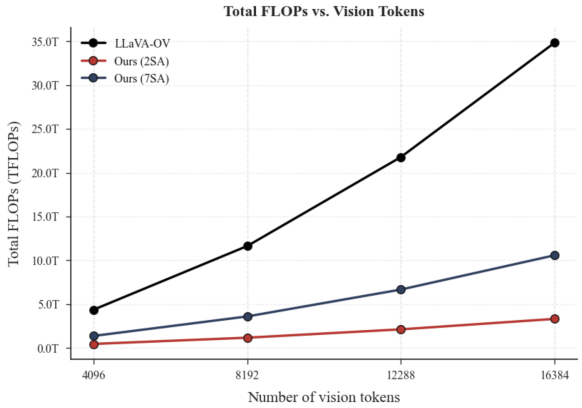

Figure S5. **Efficiency comparison** - number of FLOPS vs. vision sequence length.

S5. Additional discussion on efficiency

In addition to the discussion from the main paper, in Fig. S5 we illustrate the computational cost as a function of the visual sequence length. Our approach significantly reduces the FLOPs compared to the baseline LLaVA-OV model, with longer sequences benefiting more from the reduction. Note that the FLOPs measured here only account for the transformer layers, excluding the vision encoder which is common to all methods.

S6. Reduced number of tokens vs reduced attention

In the main paper, we demonstrated that intermittent attention (VISOR) is effective for hard tasks and orthogonal to token reduction methods. Here, we conduct a head-to-head comparison of these two paradigms under a similar FLOPs reduction rate of approximately 16×. We use our VISOR-TR variant and compare it against two token reduction methods, M^3 and VisPruner. To ensure a fair comparison, all models were finetuned end-to-end using the same training procedure, with the only difference being how the amount of input vision tokens is reduced. As shown in Table S3, while all methods perform well on easy datasets, our approach maintains a significant performance advantage on harder tasks, highlighting the benefits of preserving a larger visual context over aggressive token reduction.

S7. Routing mechanism generalization

As described in the main manuscript, the internal routing mechanism responsible for deciding the optimal configura-

Table S2. Comparison on various vision-language benchmarks for Qwen2-VL-2B LVLM.

Method	*Easy*									*Hard*				
	RealWorldQA	SQA	GQA	MME	MSTAR	POPE	TextVQA	AI2D	**Avg. (Easy)**	ChartQA	OCRBench	InfoVQA	DocVQA	**Avg. (Hard)**
Qwen2-VL-2B	61.1	78.1	60.2	75.3	43.6	87.7	79.3	70.4	69.5	75.0	77.5	64.1	89.5	76.5
VISOR	60.0	82.1	60.8	75.1	49.2	88.9	76.1	75.0	70.9	78.1	76.0	55.9	87.3	74.3

Table S3. Comparison between token reduction vs intermittent attention (VISOR).

Method	*Easy*									*Hard*					
	RealWorldQA	SQA	GQA	MME	MSTAR	POPE	TextVQA	AI2D	**Avg. (Easy)**	ChartQA	OCRBench	InfoVQA	DocVQA	**Avg. (Hard)**	**Avg. FLOPs Savings**
LLaVA-OV [4]	54.0	67.2	58.3	60.6	40.6	88.4	66.0	56.7	61.5	60.9	58.8	40.0	68.7	57.1	1.0×
+ M^3	53.6	75.1	59.1	63.6	41.5	88.1	65.5	62.2	63.6	62.9	54.0	34.7	58.0	52.4	16.0×
+ VisPruner train.	53.1	69.2	56.7	60.9	36.9	86.7	55.4	56.3	59.4	52.4	44.2	28.2	52.9	44.4	16.0×
VISOR-TR **(Ours)**	55.4	75.4	60.7	59.5	38.5	87.7	67.4	61.9	63.3	65.3	60.7	37.4	67.7	57.8	18×

tion of self-attention layers to be used for each sample is trained using offline, per-dataset labels extracted from our training set. In this subsection, in order to investigate how the router performs on unseen data, we train it while excluding from its train set three datasets (AI2D, DocVQA, and GQA), and evaluate it on vision-language benchmarks including those datasets. In Table S4, we present the outcomes of this experiment (VISOR-TR-excl) and contrast them with the router when trained on the full train set (VISOR-TR). Our results demonstrate that, even though, naturally, the model behavior is changed, this does not lead to a drop of performance in general or in the excluded datasets in particular, thereby indicating that VISOR is robust and can handle samples outside the train set's distribution.

S8. Additional comparison with the state-of-the-art

In the main manuscript, we compared our method against state-of-the-art approaches using a shared LLaVA-OV (0.5B) backbone. Herein, we provide additional results using a larger LLaVA-OV (1.5B) backbone and on a different architecture: QwenVL (2B).

For the 1.5B case, as no official 1.5B variant is openly available, we re-trained it ourselves fully using the same procedure as described in Li et al. [4]. Additionally, we've also re-implemented the baselines under the same unified setting. As the results from Table S5 show, the same conclusions hold and our method continues to outperform existing approaches on the *hard* partition of the dataset, while providing significant efficiency gains.

For QwenVL2 (2B) we report results in Table S2 - the same conclusions hold, with our approach largely matching the full model's performance using significantly fewer FLOPs. We note however, that in this case, as we don't have access to the full QwenVL training data, our method is at disadvantage. In practice this difference results in disproportionate swings for certain datasets (e.g: InfoVQA, MMSTAR).

S9. Additional details regarding Token Packing

To further enhance and validate the efficacy of our approach in the main section, we introduce a light adaptation for token packing, capable of working at non-power-of-two reduction rates.

Specifically, after the vision encoder, we reshape the patch embeddings back to their 2D spatial grid, interpolate the grid by a factor of $1/\sqrt{2}$ along each spatial dimension for a $2\times$ compression ratio, and then apply a space-to-depth transformation (pixel shuffle). This deterministically halves the number of visual tokens with minimal informa-

Table S4. Results for the routing mechanism trained on the full train set (VISOR-TR), contrasted with training excluding samples from AI2D, DocVQA and GQA (VISOR-TR-excl).

Method	RWQA	SQA	GQA	MME	MMSTAR	POPE	TxtVQA	AI2D	CQA	OCRB	InfoVQA	DocVQA
VISOR-TR	55.4 24×	75.4 16×	60.7 24×	59.5 17.7×	38.5 17.7×	87.7 24×	67.4 14.5×	61.9 24×	65.3 16×	60.7 15×	37.4 12×	67.7 12×
VISOR-TR-excl	55.5 14.5×	75.6 15.5×	61.6 9.8×	59.2 11.4×	38.3 20×	88.3 9.8×	66.9 12×	62.0 14.5×	65.0 16×	61.1 14.1×	37.2 12×	67.6 12×

Table S5. Comparison with state-of-the-art models on various vision-language benchmarks using a LLaVA-OV 1.5B backbone. The metrics used are accuracy for most datasets, except for MME where we report a score (higher is better). MME values are divided by 20.

Method	*Easy*									*Hard*					Avg. FLOPs Savings
	RealWorldQA	SQA	GQA	MME	MMSTAR	POPE	TextVQA	AI2D	**Avg. (Easy)**	ChartQA	OCRBench	InfoVQA	DocVQA	**Avg. (Hard)**	
LLaVA-OV [4]	57.3	75.6	58.6	63.7	40.5	88.1	69.9	60.3	64.3	64.2	60.3	46.7	76.6	62.0	1.0×
Downsample [5]	53.6	76.9	58.4	62.5	39.2	86.1	59.5	60.0	62.0	46.7	48.0	29.1	52.6	44.1	4.0×
VisionZip [10]	52.7	74.8	54.2	64.5	38.3	86.2	54.9	57.2	60.4	37.8	33.8	26.3	44.7	35.7	5.7×
VisionZip† [10]	54.6	76.2	57.2	63.9	38.1	87.0	62.0	58.2	62.2	50.0	44.3	30.8	55.1	45.1	5.7×
SparseVLM [12]	52.0	74.8	50.3	58.4	34.5	82.3	60.6	57.1	58.7	51.3	38.9	34.9	57.9	45.7	4.5×
PyramidDrop [9]	56.7	76.0	55.9	63.7	37.5	87.2	60.3	62.1	59.1	47.9	31.9	34.2	54.5	42.1	4.6×
VisPruner [11]	52.9	76.0	53.4	62.3	38.4	83.3	54.8	57.5	59.8	40.0	34.3	28.4	51.2	38.5	5.7×
HiRED [1]	54.8	76.0	56.3	63.2	39.3	85.9	59.8	58.2	61.7	44.1	42.0	27.6	52.3	41.5	5.0×
VISOR(**Ours**)	57.5	84.7	62.5	68.7	44.9	88.6	68.6	71.7	68.3	68.3	63.7	44.4	75.3	62.9	6.6×

tion loss and no added parameters, complementing the computational savings from our sparse attention design. Note that the interpolation factor can be adjusted upwards to accommodate arbitrary reduction rates.

S9.1. Multi-image performance

To further validate the effectiveness of our approach on multi-image inputs, we finetune VISOR on the LLaVA-OV multi-image dataset and compare the performance of the resulting model with LLaVA on the MUIR, Blink, and MMIU benchmarks. As shown in Table S8, VISOR matches or outperforms LLaVA-OV despite being over 3× faster.

S10. Scaling VISOR with Training Data Size

To showcase the ability of our approach to scale with increased data, we train VISOR using different portions of the original training set. In Table S6, we compare the performance of our method when trained on the full dataset (100%) versus a randomly sampled subset containing half of the data (50%). As observed, reducing the amount of training data generally causes a decline in performance. Notably, this degradation is primarily concentrated on harder, vision-intensive tasks (e.g., DocVQA, InfoVQA). In contrast, the performance on simpler datasets (such as MME, POPE, and RWQA) remains similar. These results highlight that VISOR effectively leverages larger training corpuses to progressively improve its accuracy.

S11. Using VISOR with FastVLM

Vasu et al. [8] introduce FastVLM, an efficient LVLM that uses the FastVITHD [7] vision backbone to encode high-resolution inputs efficiently. The FastVLM vision encoder uses a hybrid convolution–attention architecture and outputs a reduced number of visual tokens. While the standard LLaVA-OV with SigLIP-400M produces 729 vision tokens for each 384 × 384 image patch, FastVLM outputs only 36 tokens for the same input size.

To test whether VISOR remains effective when the vision encoder already produces a small number of vision tokens, we replace the SigLIP-400M encoder in LLaVA-OV with the FastVLM encoder. We first train a LLaVA-OV 0.5B model using the FastVLM vision encoder, following the original three-stage recipe on 7.1M samples. Then, we train VISOR on top of this model to further reduce FLOPs. As shown in Table S7, VISOR improves over LLaVA-OV with FastVLM on both Easy and Hard benchmarks while achieving 3× additional LLM FLOPs savings (60× vs. 20×). Compared to SigLIP-based LLaVA-OV,

Table S6. VISOR LLava-OV 0.5B results when training with different portions of the training data. We report performance when using the full training set (100%) and a reduced subset (50%).

Method	Training data %	RWQA	SQA	GQA	MME	MMSTAR	POPE	TxtVQA	AI2D	CQA	OCRB	InfoVQA	DocVQA
VISOR	100%	54.6	75.3	61.8	60.5	40.1	87.6	67.8	61.5	65.2	61.8	37.6	68.9
VISOR	50%	56.9	72.3	60.1	60.2	38.6	88.3	65.1	57.3	62.2	58.3	33.8	62.7

Table S7. Comparison between LLaVA-OV 0.5B and VISOR with FastVLM vision encoder. *Avg. FLOPs Savings* refers only to LLM FLOPs and does not include the cost of vision encoding.

Method	Vision Encoder	*Easy*									*Hard*					Avg. FLOPs Savings
		RealWorldQA	SQA	GQA	MME	MSTAR	POPE	TextVQA	AI2D	**Avg. (Easy)**	ChartQA	OCRBench	InfoVQA	DocVQA	**Avg. (Hard)**	
LLaVA-OV	SigLIP-400M	54.0	67.2	58.3	60.6	40.6	88.4	66.0	56.7	61.5	60.9	58.8	40.0	68.7	57.1	1×
LLaVA-OV	FastVLM	52.5	67.4	56.3	58.4	35.8	87.7	61.3	51.5	58.9	57.9	47.6	32.5	51.1	47.3	20×
VISOR	FastVLM	52.3	73.7	56.3	58.8	37.7	86.7	61.9	55.8	60.4	63.0	49.0	33.4	53.1	49.6	60×

Table S8. Multi-image performance comparison.

Method	MUIR	Blink	MMIU
LLaVA	26.8	40.4	34.2
VISOR	28.1	39.0	34.5

the FastVLM version achieves comparable results on Easy benchmarks, but the gap is larger on Hard ones, confirming that aggressive token reduction mainly hurts performance on fine-grained benchmarks.

S12. Re-implementation of baselines

In this section, we detail our re-implementation of the baselines we compare with, along with the hyperparameters used. Since LLaVA-OV uses a SigLIP-400M vision encoder that does not have a CLS token, for all methods that rely on the CLS token's attention scores to select important tokens, we instead use the average attention each token receives from all other tokens in the sequence, as proposed by Yang et al. [10].

VisionZip. VisionZip [10] is a token reduction method that selects the most important tokens, named *dominant tokens*, using the visual encoder's attention scores. To avoid losing information, the remaining tokens are merged into *contextual tokens* based on semantic similarity. In our experiments, we adapt the official VisionZip LLaVA-Next [6] code to LLaVA-OV. Unlike LLaVA-Next, the LLaVA-OV image processing AnyRes strategy applies bilinear interpolation when the number of tokens exceeds a threshold. To prevent errors from interpolating removed tokens and to stay close to the original design, we remove this step. Yang et al. [10] also introduce VisionZip†, a trained version that fine-tunes the cross-modality projector. For a fair comparison, we train VisionZip† on the LLaVA-OV Single-Image 3.2M dataset [4]. In all VisionZip and VisionZip† experiments, we set the number of retained tokens per patch to 128, split into 104 *dominant tokens* and 24 *contextual tokens*.

VisPruner. VisPruner [11] is a training-free token pruning method that retains visual tokens based on visual attention scores. It first selects the *important tokens*, i.e., those with the highest scores, and then removes duplicates by keeping only *diverse tokens* based on their similarity. As with VisionZip, we adapt the official VisPruner LLaVA-Next code to the LLaVA-OV backbone. In our experiments, we retain 128 tokens per patch, split into 96 *important tokens* and 32 *diverse tokens*.

HiRed. High-Resolution Early Dropping (HiRed) [1] is a plug-and-play token-reduction method designed to work under a fixed budget. It targets high-resolution LVLMs (i.e., LLaVA-Next) and drops tokens before they reach the LLM. The key idea is to evaluate the visual content of each image patch using the attention scores of the full image, then assign a budget to each patch accordingly. Within each patch, the most informative tokens are kept and passed to the LLM. In our experiments, we adapt the official code to LLaVA-OV, following the same approach used for VisionZip and VisPruner. In our experiments, we use the same hyperparameters as the original implementation, setting a token budget of 20%.

M^3. Matryoshka Multimodal Models (M^3) [2] represent visual content as a nested set of tokens capturing information at different levels of detail, from coarse to fine. The visual tokens from the encoder are grouped into sev-

eral coarse-to-fine levels, where the coarser tokens $X_{S_{i-1}}$ are obtained from the finer tokens X_{S_i} using average pooling. M^3 does not add any extra parameters. For a fair comparison, we train M^3 on the LLaVA-OV Single-Image 3.2M dataset [4], updating both the vision encoder and the LLM weights. In our experiments, we define a set of scales $\{X_{S_i}\}_{i=1}^{M}$ that reduce the number of visual tokens by factors of 1, 4, 8, and 16. For a fair comparison with our method, we report the results at an $8\times$ reduction.

PyramidDrop. PyramidDrop [9] is a progressive token pruning method that gradually reduces the number of visual tokens as the LLM depth increases. Specifically, the LLM layers are split into stages, and at the beginning of each stage, the number of visual tokens is reduced based on a predefined set of reduction rates, which are defined on a per-stage basis. At each pruning step, the input features are split into visual and text features, and then from the text features, only the features corresponding to the position of the last token of the user's query or instruction are kept, resulting in $N_v \times d$ visual features and a single text feature vector. Then, the next attention layer to be executed is applied over these features, with the text features as the query, resulting in image-text attention weights that are interpreted as a per-visual token importance score. These scores, together with a per-stage pre-defined drop-rate, are used to keep the top-k scoring visual tokens, with k as the target visual token to keep. This procedure is applied progressively throughout the LLM's depth at the beginning of each stage. In our implementation, for LLaVA-OV 0.5B with a 24-layer LLM, we used 4 stages defined as $(1, 2-6, 7-12, 13-24)$ with drop-rates of $(1.0, 0.3, 0.2, 0.1)$ for easy datasets, and $(1-4, 5-10, 11-16, 17-24)$ with drop-rates of $(1.0, 0.5, 0.25, 0.125)$ for hard datasets with average FLOPs saving of $4.2\times$. As for LLaVA-OV 1.5B with a 28-layer LLM we used 5 stages defined as $(1, 2, 3-6, 5-10, 11-28)$ with drop-rates of $(1.0, 0.5, 0.3, 0.2, 0.1)$ for easy datasets, and $(1-2, 3-8, 8-12, 13-18, 19-28)$ with drop-rates of $(1.0, 0.75, 0.5, 0.25, 0.125)$ for hard datasets with an average FLOPs saving of $4.6\times$.

SparseVLM. Similar to PyramidDrop, SparseVLM [12] leverages the self-attention maps in the VLLM to identify the text tokens that are most relevant to the image, and uses them as *raters* to score the importance of visual tokens. SparseVLM then adaptively determines how many visual tokens to prune at each layer based on the rank of the text-to-vision attention matrix, and further reduces information loss with a *token recycling* step that aggregates the most informative pruned tokens into a smaller set of reconstructed tokens. It is a plug-and-play method that does not require additional parameters or fine-tuning.

S13. Additional details on Centered Kernel Alignment (CKA)

In Figure 3 in the main manuscript, we showed how vision features evolve across layers within the LLM transformer of the LVLM by computing the Centered Kernel Alignment (CKA) [3]. More specifically, to compute it, let $X \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^{n\times d}$ represent the vision features extracted from two different layers, where n is the number of tokens and d is the feature dimension. The CKA computation begins by forming the Gram matrices $K = XX^T$ and $L = YY^T$. These matrices are then centered using the centering matrix $H = I_n - \frac{1}{n}1_n$, where I_n is the identity matrix and 1_n is an $n \times n$ matrix of ones. The centered Gram matrices are given by $\tilde{K} = HKH$ and $\tilde{L} = HLH$. The CKA similarity between $\tilde{K}$ and $\tilde{L}$ can then be computed as:

$$\text{CKA}(\tilde{K}, \tilde{L}) = \frac{\langle \tilde{K}, \tilde{L} \rangle_F}{\|\tilde{K}\|_F \|\tilde{L}\|_F}, \tag{S1}$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product, and $\|\cdot\|_F$ represents the Frobenius norm.

References

[1] Kazi Hasan Ibn Arif, JinYi Yoon, Dimitrios S Nikolopoulos, Hans Vandierendonck, Deepu John, and Bo Ji. Hired: Attention-guided token dropping for efficient inference of high-resolution vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1773–1781, 2025. 6, 7

[2] Mu Cai, Jianwei Yang, Jianfeng Gao, and Yong Jae Lee. Matryoshka multimodal models. In *ICLR*, 2025. 7

[3] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012. 8

[4] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024. 5, 6, 7, 8

[5] Chenfei Liao, Wensong Wang, Zichen Wen, Xu Zheng, Yiyu Wang, Haocong He, Yuanhuiyi Lyu, Lutao Jiang, Xin Zou, Yuqian Fu, et al. Are we using the right benchmark: An evaluation framework for visual token compression methods. *arXiv preprint arXiv:2510.07143*, 2025. 6

[6] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, 2024. 7

[7] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. Fastvit: A fast hybrid vision transformer using structural reparameterization. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5785–5795, 2023. 6

[8] Pavan Kumar Anasosalu Vasu, Fartash Faghri, Chun-Liang Li, Cem Koc, Nate True, Albert Antony, Gokula Santhanam, James Gabriel, Peter Grasch, Oncel Tuzel, et al. Fastvlm:

Efficient vision encoding for vision language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19769–19780, 2025. 6

[9] Long Xing, Qidong Huang, Xiaoyi Dong, Jiajie Lu, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, Jiaqi Wang, Feng Wu, et al. Pyramiddrop: Accelerating your large vision-language models via pyramid visual redundancy reduction. *CVPR*, 2025. 6, 8

[10] 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. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19792–19802, 2025. 6, 7

[11] 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. *ICCV*, 2025. 6, 7

[12] Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, et al. Sparsevlm: Visual token sparsification for efficient vision-language model inference. *arXiv preprint arXiv:2410.04417*, 2024. 6, 8