ADAVSKIP: ADAPTIVE VISUAL TOKEN SKIPPING ACROSS LAYERS FOR EFFICIENT MLLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal large language models (MLLMs) require substantial computation to process numerous visual tokens across all transformer layers. Most methods for efficient MLLM inference exploit *horizontal redundancy* by compressing visual tokens. Beyond token reduction, recent studies exploit *vertical redundancy* through early exit or fixed-layer skipping. However, we find that the extent and distribution of this redundancy vary across inputs and differ between self-attention and MLP modules. Motivated by these observations, we propose AdaVSkip, which equips each layer with two lightweight routers that independently determine whether visual tokens participate in the computation of the self-attention and MLP modules. These decisions collectively define an input-specific visual computation path. To train these routers, we develop a progressive two-stage training framework that updates only the routers while keeping the backbone frozen. Stage I establishes an initial routing policy by combining supervision from input-specific targets derived from module-wise necessity scores with the downstream language modeling loss. To better account for how routing decisions jointly affect answer quality, we introduce Stage II to further refine the policy. This stage uses reinforcement learning with feedback from answers generated under complete sampled routing paths. It combines an answer correctness reward with a skip-consistency reward that discourages excessive retention of visual token computation. Across three MLLM backbones, AdaVSkip maintains strong task performance with substantially less computation. On LLaVA-NeXT-7B, AdaVSkip reduces language model prefill FLOPs by 53.2% while preserving the original model's average performance. Combining it with visual token compression increases this reduction to 93.1%, while retaining 97.2% of the original performance on average.

1 INTRODUCTION

Multimodal large language models (MLLMs) have achieved remarkable progress across diverse vision-language tasks (Alayrac et al., 2022; Li et al., 2023b; Dai et al., 2023; Liu et al., 2023; 2024a). Recent models increasingly adopt high resolution inputs or fine-grained visual representations, often producing large numbers of visual tokens (Liu et al., 2024b; Li et al., 2024; Chen et al., 2024c; Zhu et al., 2025; Bai et al., 2025a). These visual tokens are processed alongside text tokens by both self-attention and MLP modules at every transformer layer, resulting in substantial inference costs.

To improve computational efficiency, existing methods reduce visual-token counts by learning compact representations before feeding them into the language model (Zhang et al., 2025b; Li et al., 2025; Lu et al., 2026) or by pruning less informative visual tokens during inference (Chen et al., 2024a; Yang et al., 2025; Ye et al., 2025; Zhang et al., 2025a; Dong et al., 2026). These methods primarily exploit *horizontal redundancy* within the visual-token sequence, but the retained tokens may still undergo computation in layers where their contribution is limited. To exploit this *vertical redundancy* across model depth, recent studies allow visual tokens to exit early (Lin et al., 2025; Wu et al., 2024) or freeze visual-token updates in input-invariant layer subsets (Yuan et al., 2025). However, it remains unclear whether fixed or prefix-based skipping patterns can adequately accommodate the varying visual-computation needs of different questions.

To investigate this question, we systematically analyze how skipping visual-token computation in individual self-attention and MLP modules affects the model's output distribution across different

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

inputs. Our analysis yields two key observations. First, vertical redundancy is input-dependent: both the extent of skippable visual-token computation and its distribution across model depth vary substantially across inputs, limiting the effectiveness of a fixed skipping pattern. Second, even within the same layer, self-attention and MLP often differ in their need for visual-token computation, suggesting that a shared layer-wise skipping decision may skip useful computation in one module or retain redundant computation in the other. Together, these observations motivate an input-adaptive, module-wise visual token skipping mechanism.

To this end, we propose **AdaVSkip**, an input-adaptive framework for module-wise visual token skipping. At each transformer layer, AdaVSkip introduces two lightweight routers that independently determine whether to perform visual-token computation in the self-attention and MLP modules, respectively. When a router chooses to skip, visual tokens bypass the module with their hidden states unchanged. They remain in the sequence and can resume computation in subsequent modules. Text tokens continue to be processed in every module regardless of the routing decisions. Collectively, these decisions define an input-specific visual-computation path, allowing AdaVSkip to adapt both the extent and distribution of visual-token skipping across layers and modules.

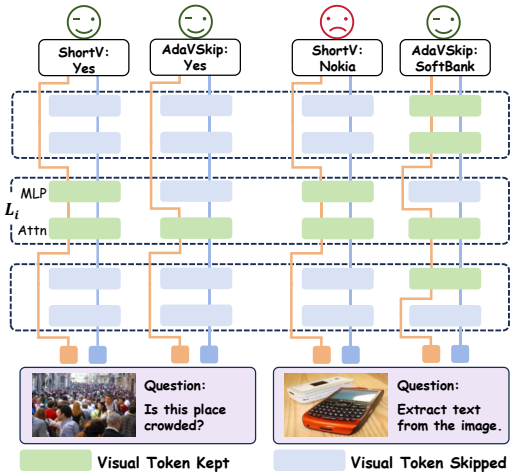


Figure 1: Fixed layer-wise skipping vs. adaptive module-wise visual token skipping.

To train these routers, we develop a progressive two-stage training strategy that updates only the routers while keeping the backbone frozen. In Stage I, we compute module-wise necessity scores by skipping visual-token computation in one module at a time and measuring changes in output distributions using KL divergence. For each module type, we assign input-specific keep targets to the smallest subset of highest-scoring modules whose cumulative necessity reaches a predefined fraction of the total necessity mass. We then train the routers with these targets and the downstream language modeling loss to establish an initial routing policy. However, these scores reflect isolated module interventions and do not directly capture the joint effects of skipping multiple modules. Stage II therefore refines the policy through reinforcement learning, using the correctness of answers generated under complete sampled routing paths as direct feedback. To discourage excessive retention of visual token computation, we introduce a skip-consistency reward that regularizes each input's keep ratios toward their Stage-I levels, encouraging the policy to seek better performance at comparable computational cost.

Extensive experiments across three MLLM backbones and multiple benchmarks demonstrate that AdaVSkip improves performance–efficiency trade-offs over fixed-layer and prefix-based skipping methods, particularly under aggressive skipping. At moderate skipping levels, AdaVSkip reduces FLOPs by approximately 50% while largely preserving average performance. Moreover, AdaVSkip is complementary to visual-token compression. On LLaVA-NeXT-7B, combining it with CDPruner reduces FLOPs by 93.1% while retaining 97.2% of the original performance on average.

Our main contributions are summarized as follows:

- We analyze vertical redundancy in visual-token computation, revealing that its extent and distribution vary across inputs and that self-attention and MLP exhibit distinct patterns.
- We propose **AdaVSkip**, an input-adaptive, module-wise visual token skipping framework with two lightweight routers per layer that independently determine whether visual tokens participate in the computation of the self-attention and MLP modules.
- We develop a progressive two-stage training strategy that first learns an initial routing policy through necessity-guided supervision and then refines it through reinforcement learning with correctness and skip-consistency rewards, while keeping the backbone frozen.
- Across three MLLMs and multiple benchmarks, AdaVSkip better balances performance and efficiency than fixed layer skipping and early exit. With visual token compression, it saves more computation with less performance loss than token compression alone.

2 Related Work

2.1 Multimodal Large Language Models

Recent multimodal large language models (MLLMs) extend the reasoning and instruction-following capabilities of LLMs to visual inputs (Alayrac et al., 2022; Li et al., 2023b; Dai et al., 2023; Chen et al., 2024c; Bai et al., 2025b). A widely adopted design is the projector-based paradigm, which connects a pretrained vision encoder to an LLM through a lightweight projector, mapping images into visual tokens that are processed together with text tokens in the LLM (Liu et al., 2023; 2024a). Building on this paradigm, recent MLLMs improve visual perception by using higher-resolution inputs or introducing more visual tokens for finer-grained visual representation (Liu et al., 2024b; Zhu et al., 2025; Guo et al., 2025; Bai et al., 2025a). However, this trend substantially increases computational costs, making efficient visual-token computation increasingly important for MLLMs.

2.2 Horizontal Redundancy in Visual Tokens

Existing methods for efficient MLLM inference mainly exploit horizontal redundancy by reducing the number of visual tokens processed by the language model. These methods fall into two categories. The first learns compact visual representations before the LLM stage, often through redesigned visual projectors (Zhang et al., 2025b; Li et al., 2025; Lu et al., 2026), but usually requires architectural modifications and training. The second performs training-free token reduction during inference by estimating token importance or redundancy and pruning or merging less informative tokens (Chen et al., 2024a; Zhang et al., 2024; Yang et al., 2025; Ye et al., 2025; Zhang et al., 2025a). Although effective, both approaches reduce the visual-token count while processing the remaining tokens through all LLM layers, leaving vertical redundancy in visual-token computation largely unexploited.

2.3 Vertical Redundancy Across Layers

Vertical redundancy has been explored in LLMs through early-exit and adaptive-depth methods, including SkipDecode (Del Corro et al., 2023), LayerSkip (Elhoushi et al., 2024), Mixture-of-Depths (Raposo et al., 2024), and SkipGPT (Zhao et al., 2025). MLLMs face a similar issue: visual-token computation may be necessary in some layers but contribute little in others. This vertical redundancy remains less explored than horizontal redundancy. VTW (Lin et al., 2025) and DyVTE (Wu et al., 2024) reduce visual-token computation through early exit, with DyVTE supporting input-dependent exit depths. Both methods restrict visual-token computation to a prefix of layers. Challenging this prefix assumption, ShortV (Yuan et al., 2025) shows that important visual-token computation is not confined to early layers and preserves it in selected layers. Rather than using a fixed layer subset or a single exit point, AdaVSkip determines how much visual-token computation to skip and where for each input, with independent decisions for self-attention and MLP at each layer.

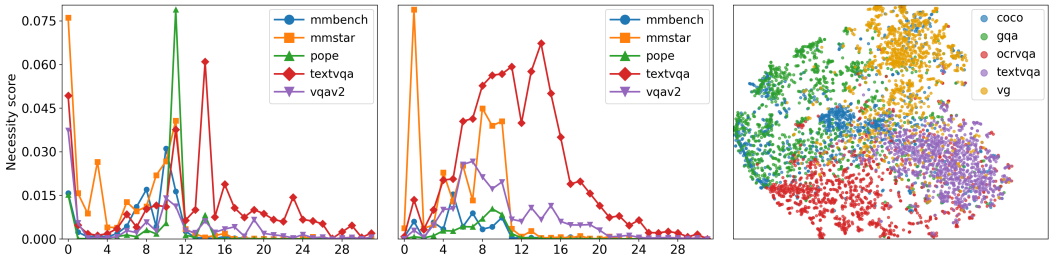
3 Method

3.1 Patterns of Vertical Redundancy

To characterize vertical redundancy, we quantify the necessity of visual-token computation in the self-attention and MLP modules of each transformer layer, yielding $2L$ scores for an L-layer MLLM. For each input x with ground-truth answer $y = (y_1, \ldots, y_T)$, we first run the original model on the input and answer in a single forward pass with teacher forcing, recording its output distributions at all answer-token positions. We then repeat the forward pass, skipping visual-token computation in one module at a time. For layer l and module type $c \in \mathcal{C} = \{\mathrm{attn}, \mathrm{mlp}\}$, we define the necessity score by comparing the resulting distributions with those of the original model:

$$s_l^c(x) = \frac{1}{T} \sum_{t=1}^{T} D_{\mathrm{KL}} \left(P_{\mathrm{full}}(\cdot \mid x, y_{<t}) \,\middle\|\, P_{-(l,c)}(\cdot \mid x, y_{<t}) \right), \tag{1}$$

where P_{full} denotes the original model, and $P_{-(l,c)}$ denotes the intervened model where visual-token computation in the module of type c at layer l is skipped while other computations remain unchanged. A larger $s_l^c(x)$ indicates that skipping visual-token computation in this module causes a larger deviation from the original model, suggesting higher visual-token necessity.



(a) Self-attn necessity across layers. (b) MLP necessity across layers. (c) Training-set t-SNE.

Figure 2: Layer-wise visual-token necessity varies across benchmarks and samples. (a) and (b) report necessity scores for self-attention and MLP, respectively, on representative benchmarks. (c) shows a t-SNE visualization of LLaVA-1.5 SFT samples, each represented by a 64-dimensional vector formed by concatenating its self-attention and MLP necessity scores across all layers.

We examine the resulting necessity scores at both the benchmark and sample levels. Figures 2a and 2b show the necessity distributions across layers for self-attention and MLP on representative benchmarks, revealing two patterns. First, high-necessity regions vary in both location and extent across benchmarks. For example, POPE shows high necessity mainly in a few layers, whereas TextVQA shows elevated necessity across a broader range of layers. Second, necessity scores can differ markedly between self-attention and MLP within the same layer. At layer 0, self-attention generally has high scores across benchmarks, whereas MLP scores are relatively low for most benchmarks. A similar contrast appears at layer 11 on POPE, where self-attention shows a clear peak but the MLP score is much lower. Together, these results reveal distinct patterns of vertical redundancy across benchmarks and between modules within the same layer.

We further examine sample-level variation on the LLaVA-1.5 SFT training set, as shown in Figure 2c. For each sample, we concatenate the necessity scores of the 32 self-attention and 32 MLP modules into a 64-dimensional vector and project it into two dimensions using t-SNE. Samples from the same data source tend to cluster in the visualization, suggesting that visual-token redundancy is associated with task characteristics. Together, the benchmark-level and sample-level analyses motivate input-adaptive visual-token skipping with independent decisions for self-attention and MLP at each layer.

3.2 Adaptive Visual-Token Skipping

We therefore propose **AdaVSkip**, which adaptively determines whether visual tokens participate in each layer's self-attention and MLP modules, while text tokens continue to be processed in both modules. For layer l, let the input hidden states consist of visual tokens $V^l \in \mathbb{R}^{N_v \times d}$ and text tokens $T^l \in \mathbb{R}^{N_t \times d}$, where N_v and N_t denote the numbers of visual and text tokens, respectively, and d is the hidden dimension. At the layer entrance, two lightweight routers, $\mathcal{R}^l_{\text{attn}}$ and $\mathcal{R}^l_{\text{mlp}}$, independently determine visual-token participation in the subsequent self-attention and MLP modules.

Both routers use the same text-side representation as input. Text hidden states encode the user query and can incorporate visual evidence from preceding layers, providing a compact signal for estimating the necessity of visual-token computation. Specifically, we construct the router input as

$$z^l = \left[\frac{1}{N_t - 1} \sum_{i=1}^{N_t - 1} T^l_i \; ; \; T^l_{\text{last}} \right] \in \mathbb{R}^{2d}, \tag{2}$$

where $[\cdot\,;\,\cdot]$ denotes concatenation of the mean-pooled text states (excluding the last token) and the last token's state. Both are computed only over prompt text tokens in training and inference, excluding answer tokens and padding. Given z^l, the two routers produce keep probabilities $p_{(l,\text{attn})}$ and $p_{(l,\text{mlp})}$. At inference, the corresponding binary gates $g_{(l,\text{attn})}$ and $g_{(l,\text{mlp})}$ are set to 1 when their keep probabilities exceed 0.5 and to 0 otherwise. Each gate controls all visual tokens in the corresponding module, with 1 preserving their computation and 0 skipping it. Routing is applied only during prefill, with no routing decisions during autoregressive decoding.

For self-attention, $g_{(l,\text{attn})} = 1$ preserves the original attention computation over visual and text tokens. When $g_{(l,\text{attn})} = 0$, visual tokens do not participate in any computation within the module, while text tokens are updated through text-only attention. This differs from ShortV, which freezes

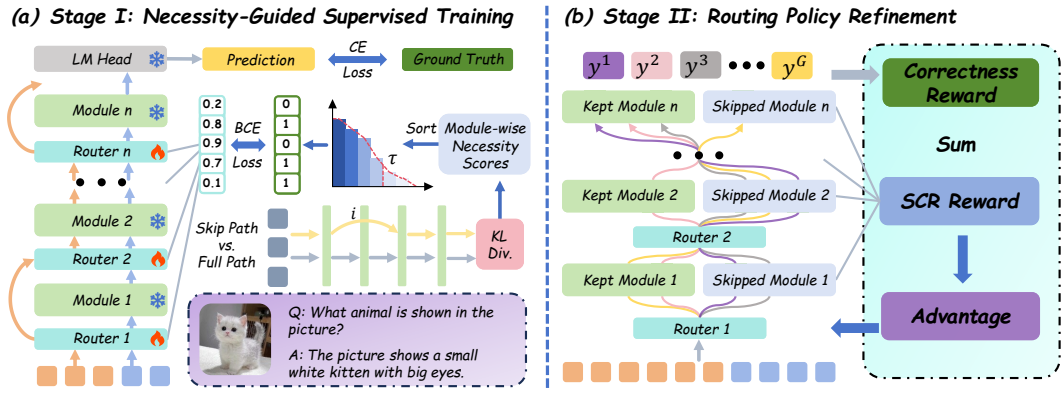


Figure 3: Overview of AdaVSkip's progressive router training framework.

visual-token updates but retains visual keys and values for text attention (Yuan et al., 2025). For MLP, text tokens are always processed, whereas visual tokens participate in its computation only when $g_{(l,\mathrm{mlp})} = 1$. Otherwise, their hidden states remain unchanged.

To train these routers, we adopt the two-stage strategy shown in Figure 3, keeping the backbone frozen. Stage I initializes the policy through necessity-guided supervision and language modeling. Stage II refines it through reinforcement learning using feedback from generated answers.

3.3 Stage I: Necessity-Guided Supervised Training

In Stage I, we construct input-specific routing targets from module-wise visual-token necessity scores. Let $m = (l, c)$ index the module of type $c \in \mathcal{C} = \{\mathrm{attn}, \mathrm{mlp}\}$ at layer l, and let $\mathcal{M}_c$ denote the set of modules of type c across all layers. For each input, we rank the modules in $\mathcal{M}_c$ by their necessity scores in descending order and select the smallest top-ranked subset whose cumulative necessity accounts for at least a fraction τ_c of the total necessity mass within $\mathcal{M}_c$. Selected modules are assigned $q_m = 1$ to preserve visual-token computation, while the remaining modules are assigned $q_m = 0$ to skip it. Even with a fixed τ_c, the number and locations of preserved modules vary with each input's necessity distribution. We can adjust τ_c to control the extent of skipping: larger values favor retaining more modules, while smaller values encourage more skipping.

In addition to necessity-guided routing supervision, we incorporate the downstream language modeling loss to train the routers. During training, the forward pass uses hard gates obtained by thresholding the router probabilities, while a straight-through estimator (STE) approximates gradients through these gates during backpropagation. Let $\mathcal{M} = \bigcup_{c \in \mathcal{C}} \mathcal{M}_c$, and let p_m and g_m denote the keep probability and binary gate for module m, respectively. For an input x with ground-truth answer $y = (y_1, \ldots, y_T)$, the Stage-I training objective is defined as

$$\mathcal{L}_{\mathrm{SFT}} = \lambda_{\mathrm{route}} \sum_{m \in \mathcal{M}} \mathrm{BCE}(p_m, q_m) - \lambda_{\mathrm{lm}} \sum_{t=1}^{T} \log P(y_t \mid y_{<t}, x; \{g_m\}_{m \in \mathcal{M}}) \,. \tag{3}$$

Here, λ_{route} and λ_{lm} control the relative contributions of the two loss terms. The routing loss encourages keep probabilities to match the necessity-derived targets, while the language modeling loss encourages accurate prediction of ground-truth answer tokens under the selected routing path.

To reduce the computational cost of estimating module-wise necessity scores, we also provide an accelerated variant. Appendix C details this variant and compares it with the original necessity estimation procedure, showing comparable downstream performance after router training. All results in the main text use routing targets constructed with the original procedure.

3.4 Stage II: Routing Policy Refinement

Stage-I routing targets are based on necessity scores from isolated module interventions, which do not directly capture the joint effects of skipping multiple modules. Moreover, router updates from the downstream language modeling loss rely on approximate gradients through hard gates. We therefore refine the routing policy through reinforcement learning, using the correctness of answers generated under complete sampled routing paths as direct feedback.

For an input x, a routing trajectory $\mathbf{g} = \{g_m\}_{m \in \mathcal{M}}$ specifies the binary routing decisions across all skippable modules. The policy acts on visual-token computation rather than output tokens: each action determines whether visual tokens participate in the computation of a self-attention or MLP module. For each input, we sample a group of G routing trajectories by drawing actions from the Bernoulli distributions predicted by the routers. The frozen backbone then generates an answer under each sampled trajectory for subsequent correctness evaluation.

The correctness reward is 1 if the generated answer exactly matches the ground-truth answer after normalization and 0 otherwise. However, optimizing correctness alone may encourage overly conservative routing that preserves excessive visual-token computation. To discourage this tendency, we introduce a skip-consistency reward (SCR) that penalizes deviations from sample-specific reference keep ratios. Before refinement, we run the Stage-I routers on the Stage-II training set to record these references. For each sampled trajectory, we combine the correctness reward with SCR:

$$R(\mathbf{g}^i) = R_{\text{corr}}(\mathbf{g}^i) - \lambda_{\text{SCR}} \sum_{c \in \mathcal{C}} \left| \rho_c(\mathbf{g}^i, x) - \bar{\rho}_c(x) \right|, \tag{4}$$

where R_{corr} is the answer correctness reward, $\rho_c(\mathbf{g}^i, x)$ denotes the fraction of type-c modules preserving visual-token computation, and $\bar{\rho}_c(x)$ is the corresponding Stage-I reference. The coefficient λ_{SCR} controls the regularization strength. This reward encourages the policy to explore better-performing routing paths at a computational cost comparable to Stage I.

Finally, we adapt GRPO (Shao et al., 2024) to optimize the router policy. For each input, we normalize rewards within the sampled group to obtain a trajectory-level advantage $\hat{A}^i$, which is shared by all routing actions in trajectory $\mathbf{g}^i$. This yields a relative learning signal for the following RL objective:

$$\mathcal{L}_{\text{RL}} = -\mathbb{E}\left[\frac{1}{G|\mathcal{M}|} \sum_{i=1}^{G} \sum_{m \in \mathcal{M}} \min\Big(\eta_m^i \hat{A}^i,\ \text{clip}_\epsilon(\eta_m^i) \hat{A}^i \Big) \right], \quad \eta_m^i = \frac{\pi_\theta(g_m^i \mid x, m)}{\pi_{\theta_{\text{old}}}(g_m^i \mid x, m)}. \tag{5}$$

Here, g_m^i denotes the routing action at module m in trajectory $\mathbf{g}^i$. The policies π_θ and $\pi_{\theta_{\text{old}}}$ denote the current router policy and the policy used to sample the trajectories, respectively.

4 Experiments

Datasets and Models. To validate the effectiveness of AdaVSkip, we conduct experiments on three MLLMs: LLaVA-1.5-7B (Liu et al., 2024a), LLaVA-NeXT-7B (Liu et al., 2024b), and Qwen3-VL-8B-Instruct (Bai et al., 2025a). For LLaVA-1.5-7B and LLaVA-NeXT-7B, we evaluate on VQAv2 (Goyal et al., 2017), GQA (Hudson and Manning, 2019), TextVQA (Singh et al., 2019), POPE (Li et al., 2023c), MME (Fu et al., 2023), MMBench (Liu et al., 2024c), SEED-Bench (Li et al., 2023a), and MMStar (Chen et al., 2024b). For Qwen3-VL-8B-Instruct, we report results on ChartQA (Masry et al., 2022), HallusionBench (Guan et al., 2024), AI2D (Kembhavi et al., 2016), DocVQA (Mathew et al., 2021), InfoVQA (Mathew et al., 2022), MMBench, MME, and SEED-Bench.

Compared Methods. To evaluate AdaVSkip, we consider two dimensions of visual-token redundancy: vertical redundancy across layers and horizontal redundancy among tokens. For vertical redundancy, we compare AdaVSkip with VTW (Lin et al., 2025), DyVTE (Wu et al., 2024), and ShortV (Yuan et al., 2025). VTW and DyVTE reduce layer-wise computation through early exiting, while ShortV freezes visual-token updates at predefined layers. For horizontal redundancy, we compare with visual token pruning methods, including VisionZip (Yang et al., 2025), CDPruner (Zhang et al., 2025a), MMTok (Dong et al., 2026) and ZOO-Prune (Kim et al., 2026).

4.1 Main Results

Tables 1, 2, and 3 report the main results across three MLLM backbones. We compare AdaVSkip with fixed-layer and early-exit methods to evaluate its effectiveness in exploiting vertical redundancy. For AdaVSkip, K denotes the nominal skipping budget: larger values indicate more skipping, while actual skip counts and locations vary across inputs. Horizontal and vertical redundancy may differ in extent, so we evaluate AdaVSkip as a complement to token compression rather than a replacement. Specifically, we compare token compression with and without AdaVSkip at comparable FLOPs. Token budgets denote the actual retention ratios of standalone baselines; combined methods are assigned equivalent budgets based on comparable FLOPs.

Table 1: Performance comparison of layer skipping and visual token compression on LLaVA-1.5-7B. AdaVSkip-S1 denotes the Stage-I-only variant, while AdaVSkip denotes the full method.

Method	FLOPs↓	VQA^{v2}	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar	Avg.↑
Vanilla	100.0%	78.5	61.9	58.2	85.9	1510.7	64.3	66.1	33.6	100.0%
(a) Layer Skipping Methods										
Approximately 16 Skipped Layers										
VTW (AAAI'25)($N = 16$)	56.5%	66.3	55.1	51.8	86.0	1497.0	64.0	66.2	32.8	94.9%
DyVTE (NeurIPS'25)	56.4%	76.6	60.0	56.6	81.6	1491.4	64.7	66.1	–	98.0%
ShortV (ICCV'25)($N = 19$)	56.9%	75.7	60.8	54.4	85.5	1502.2	64.2	65.5	33.0	98.0%
AdaVSkip-S1($K = 16$)	51.8%	77.6	61.1	56.8	86.3	1516.3	64.0	66.2	33.3	99.3%
AdaVSkip($K = 16$)	52.0%	78.3	61.8	57.9	86.4	1512.8	64.9	65.9	33.4	**100.0%**
Approximately 20 Skipped Layers										
VTW (AAAI'25)($N = 20$)	45.6%	59.6	51.3	50.6	82.1	1475.4	63.4	64.9	32.7	91.6%
ShortV (ICCV'25)($N = 23$)	47.8%	75.3	59.4	50.4	85.6	1420.0	61.9	65.1	33.1	95.7%
AdaVSkip-S1($K = 20$)	44.7%	77.0	60.7	55.6	86.3	1465.7	63.9	65.7	32.8	98.2%
AdaVSkip($K = 20$)	45.2%	77.7	61.2	56.7	86.1	1449.9	64.0	66.4	33.8	**99.0%**
Approximately 24 Skipped Layers										
VTW (AAAI'25)($N = 24$)	34.8%	46.1	43.3	44.2	51.5	815.3	41.4	45.4	26.8	66.4%
ShortV (ICCV'25)($N = 27$)	38.7%	68.9	54.7	45.6	72.9	1169.0	58.6	56.8	31.1	85.8%
AdaVSkip-S1($K = 24$)	32.3%	72.2	57.0	52.2	83.1	1362.6	60.6	62.6	33.0	93.5%
AdaVSkip($K = 24$)	32.5%	74.0	57.6	52.6	83.3	1407.9	61.3	63.1	34.4	**95.1%**
(b) Visual Token Compression Methods										
128 Retained Visual Tokens (22.2%)										
CDPruner (NeurIPS'25)	32.0%	76.6	59.9	56.2	87.4	1431.4	63.1	63.2	33.4	97.6%
MMTok (ICLR'26)	32.0%	76.3	59.2	57.0	86.3	1442.2	62.3	63.1	33.7	97.4%
ZOO-Prune (CVPR'26)	32.0%	76.5	59.4	57.8	87.1	1418.0	61.8	62.8	33.2	97.2%
CDPruner + AdaVSkip	30.5%	77.6	60.8	57.1	87.3	1447.9	63.8	65.1	35.1	**99.4%**
64 Retained Visual Tokens (11.1%)										
CDPruner (NeurIPS'25)	22.5%	75.4	58.6	55.3	87.1	1415.1	61.1	62.1	32.4	95.8%
MMTok (ICLR'26)	22.5%	75.2	58.2	56.0	85.7	1394.9	61.2	61.4	33.4	95.7%
ZOO-Prune (CVPR'26)	22.5%	75.0	58.5	55.3	85.8	1369.9	60.2	60.9	31.7	94.5%
CDPruner + AdaVSkip	21.8%	76.4	60.1	56.4	87.5	1424.0	63.0	64.1	34.9	**98.3%**
32 Retained Visual Tokens (5.6%)										
CDPruner (NeurIPS'25)	17.7%	73.6	57.0	53.2	87.4	1373.0	59.6	60.8	30.4	93.1%
MMTok (ICLR'26)	17.7%	73.4	56.6	53.4	85.0	1343.6	59.4	59.7	33.6	93.4%
ZOO-Prune (CVPR'26)	17.7%	72.2	55.8	54.1	84.7	1336.9	58.9	58.2	31.8	92.1%
CDPruner + AdaVSkip	17.4%	75.3	58.5	55.0	87.6	1405.9	60.6	62.6	32.9	**95.9%**

On LLaVA-1.5-7B, AdaVSkip achieves better performance–efficiency trade-offs than existing skipping methods across all evaluated settings. At $K = 16$, it achieves 100.0% average relative performance with only 52.0% of the original FLOPs, maintaining the original model's average performance while outperforming ShortV by 2.0 percentage points at lower computational cost. The performance gap widens as skipping becomes more aggressive. At $K = 20$, AdaVSkip achieves 99.0% average relative performance with 45.2% of the original FLOPs, exceeding ShortV and VTW by 3.3 and 7.4 percentage points, respectively. At $K = 24$, it still retains 95.1% average relative performance with only 32.5% of the original FLOPs, whereas ShortV and VTW drop to 85.8% and 66.4%, respectively. The corresponding margins increase to 9.3 and 28.7 percentage points, indicating that AdaVSkip better preserves performance as the computational budget tightens. Beyond standalone skipping, combining AdaVSkip with CDPruner achieves 99.4%, 98.3%, and 95.9% average relative performance at equivalent visual token budgets of 22.2%, 11.1%, and 5.6%, respectively. Compared with CDPruner alone, these configurations improve average relative performance by 1.8, 2.5, and 2.8 percentage points while requiring fewer FLOPs in each setting.

These benefits also extend to stronger backbones. On LLaVA-NeXT-7B, AdaVSkip achieves 100.4% average relative performance at $K = 16$ with 46.8% of the original FLOPs, exceeding ShortV and VTW by 1.2 and 5.1 percentage points, respectively, at lower computational cost. At the 5.6% equivalent visual token budget, the combination of AdaVSkip and CDPruner achieves 97.2% average relative performance with only 6.9% of the original FLOPs, compared with 94.5% at 7.7% FLOPs for CDPruner alone. On Qwen3-VL-8B-Instruct, AdaVSkip achieves 98.6% average relative performance at $K = 18$ with 51.4% of the original FLOPs, outperforming ShortV and VTW by 1.8 and 16.8 percentage points, respectively. At the 10% equivalent visual token budget, its combination with VisionZip achieves 85.5% average relative performance with 13.5% FLOPs, compared with 71.7% at 13.9% FLOPs for VisionZip alone. Here, aggressive token compression substantially degrades performance on tasks requiring fine-grained visual perception. Jointly exploiting vertical and horizontal redundancy mitigates these losses: the combination outperforms MMTok, the strongest standalone compression baseline at this budget, by 18.4, 13.8, and 7.7 points on ChartQA, DocVQA, and InfoVQA, respectively. We provide results under additional experimental settings in the appendix.

Table 2: Performance of layer skipping and visual token compression on LLaVA-NeXT-7B.

Method	FLOPs↓	VQA^{v2}	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar	Avg.↑
Vanilla	100.0%	81.8	64.2	61.3	86.5	1519.0	67.4	70.2	35.8	100.0%
(a) Layer Skipping Methods *(∼50% of layers skipped)*										
VTW (AAAI'25)($N = 16$)	51.4%	75.6	55.8	47.3	87.5	1518.2	67.1	70.2	37.6	95.3%
ShortV (ICCV'25)($N = 19$)	51.1%	78.8	63.4	56.8	87.3	1526.0	67.2	70.4	37.4	99.2%
AdaVSkip-S1($K = 16$)	46.3%	81.0	63.6	57.3	87.3	1509.2	67.0	69.4	37.6	99.4%
AdaVSkip($K = 16$)	46.8%	81.3	64.1	59.4	87.3	1535.4	66.4	70.0	38.3	**100.4%**
(b) Visual Token Compression Methods *(5.6% visual tokens retained)*										
VisionZip (CVPR'25)	7.7%	71.4	55.2	55.0	75.8	1327.8	58.6	58.3	31.8	87.1%
CDPruner (NeurIPS'25)	7.7%	76.7	60.8	55.4	86.8	1425.3	64.2	65.5	33.9	94.5%
MMTok (ICLR'26)	7.7%	75.6	60.0	54.2	83.8	1425.1	62.9	64.5	34.0	93.1%
ZOO-Prune (CVPR'26)	7.7%	76.1	59.9	55.4	83.1	1397.9	64.2	64.1	35.0	93.6%
CDPruner + AdaVSkip	6.9%	78.6	62.4	56.2	88.8	1473.9	65.9	67.6	35.3	**97.2%**

Table 3: Performance of layer skipping and visual token compression on Qwen3-VL-8B-Instruct.

Method	FLOPs↓	ChartQA	HallB	AI2D	DocVQA	InfoVQA	MMB	MME	SEED	Avg.↑
Vanilla	100.0%	85.9	62.4	84.1	96.1	83.1	84.5	1746.2	78.7	100.0%
(a) Layer Skipping Methods *(∼50% of layers skipped)*										
VTW (AAAI'25) ($N = 18$)	52.2%	27.7	58.5	81.7	94.3	37.6	79.6	1709.9	75.2	81.8%
ShortV (ICCV'25) ($N = 19$)	51.5%	80.0	59.8	82.8	94.4	78.7	82.6	1702.7	78.0	96.8%
AdaVSkip-S1 ($K = 18$)	51.5%	80.2	63.3	82.4	94.4	79.0	83.7	1720.1	77.5	97.8%
AdaVSkip ($K = 18$)	51.4%	81.2	63.8	83.0	95.2	80.8	84.0	1709.2	78.2	**98.6%**
(b) Visual Token Compression Methods *(10.0% visual tokens retained)*										
VisionZip (CVPR'25)	13.9%	44.8	45.0	72.4	53.8	35.5	75.5	1528.0	69.3	71.7%
CDPruner (NeurIPS'25)	13.9%	33.3	45.6	69.4	43.4	31.9	74.9	1534.7	68.9	67.7%
MMTok (ICLR'26)	13.9%	48.6	54.5	77.0	62.4	41.9	77.9	1636.3	73.3	78.7%
ZOO-Prune (CVPR'26)	13.9%	50.5	48.2	70.7	51.6	33.4	76.1	1516.6	72.7	72.9%
VisionZip + AdaVSkip	13.5%	67.0	56.0	79.2	76.2	49.6	81.4	1628.2	73.9	**85.5%**

4.2 Ablation Study of AdaVSkip Training Components

We conduct an ablation study of AdaVSkip on LLaVA-1.5-7B under $K = 16$ to examine the effects of routing granularity, Stage I supervision, and Stage II refinement, as shown in Table 4. We first compare layer-wise routing with module-wise routing under the full Stage I setting. The module-wise design improves average performance from 98.9% to 99.3% while reducing retained FLOPs from 52.4% to 51.8%, demonstrating the benefit of making separate routing decisions for self-attention and MLP modules. We then progressively add the downstream CE loss and enable adaptive keep counts in module-wise routing, starting from necessity targets that retain the highest-scoring half of the modules of each type for every input. The downstream CE loss improves average performance from 98.2% to 98.5%, while allowing the target keep counts to vary across inputs further increases it to 99.3% and reduces retained FLOPs to 51.8%. For Stage II, RL refinement without SCR achieves 100.0% average performance but increases retained FLOPs to 53.5%. Adding SCR reduces retained FLOPs to 52.0% while maintaining a comparable average performance of 100.0%, confirming its effectiveness in controlling the computation budget during policy refinement.

4.3 Analysis of Adaptive Routing

Table 5 examines two aspects of AdaVSkip's input adaptivity: how many modules to skip and which modules to skip. First, Fixed-Budget AdaVSkip retains router-based module selection but fixes the skip ratio at 50% for every input. With similar average skipping ratios, AdaVSkip achieves 100.2% average relative performance versus 98.6% for this variant, supporting the benefit of input-dependent computation budgets. For example, its average skip ratio is lower on TextVQA (36.1%) than on POPE (62.2%). Second, Count-Matched Prior uses AdaVSkip's skip counts for each input, separately for self-attention and MLP. Within each type, it selects the prescribed number of modules to skip using a fixed necessity ranking shared across all input samples. With both skip counts matched, AdaVSkip achieves 100.2% average relative performance versus 98.7% for Count-Matched Prior, demonstrating the benefit of input-dependent module selection.

4.4 Inference Efficiency Analysis

Table 6 compares inference efficiency and MME performance on LLaVA-NeXT-7B. AdaVSkip alone achieves $2.09\times$ prefill and $1.78\times$ end-to-end speedups while retaining only 43.2% of the

Table 4: Ablation study of AdaVSkip training components on LLaVA-1.5-7B under $K = 16$.

Method	FLOPs↓	VQA^{v2}	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar	Avg.↑
Layer-wise Routing (Full Stage-I)	52.4%	77.7	61.1	56.1	86.0	1500.7	64.0	65.9	33.1	98.9%
Module-wise + Necessity Targets	54.4%	76.8	60.6	54.9	85.4	1448.4	64.8	66.2	33.3	98.2%
+ Downstream CE Loss	53.9%	77.2	59.2	55.6	86.5	1470.8	64.5	65.8	33.9	98.5%
+ Adaptive Keep Counts	51.8%	77.6	61.1	56.8	86.3	1516.3	64.0	66.2	33.3	99.3%
+ RL Refinement w/o SCR	53.5%	78.2	61.6	57.6	86.4	1512.9	64.6	66.0	33.9	100.0%
+ Skip-Consistency Reward	52.0%	78.3	61.8	57.9	86.4	1512.8	64.9	65.9	33.4	100.0%

Table 5: Ablation study of adaptive module routing on LLaVA-1.5-7B under $K = 16$.

Method	**GQA**	**TextVQA**	**POPE**	**MME**	**MMBench**	**Avg.**
Fixed-Budget AdaVSkip	60.6	55.4	86.8	1484.5	64.6	98.6%
Count-Matched Prior	61.5	57.7	85.6	1455.9	63.7	98.7%
AdaVSkip	61.8	57.9	86.4	1512.8	64.9	100.2%
Skipped Modules (%)	45.0	36.1	62.2	57.6	60.2	52.2

Table 6: Comparison of efficiency on MME using LLaVA-NeXT-7B.

Method	**FLOPs** (%)	**Prefill** (ms)	**E2E** (ms)	**MME**
Vanilla	100.0	161.1	194.3	1519.0
AdaVSkip	43.2	77.1 (2.09×)	109.3 (1.78×)	1535.4
VisionZip	7.2	25.0 (6.44×)	63.3 (3.07×)	1327.8
MMTok	7.2	25.0 (6.44×)	73.3 (2.65×)	1425.1
CDPruner + AdaVSkip	6.1	26.4 (6.10×)	64.6 (3.01×)	1473.9

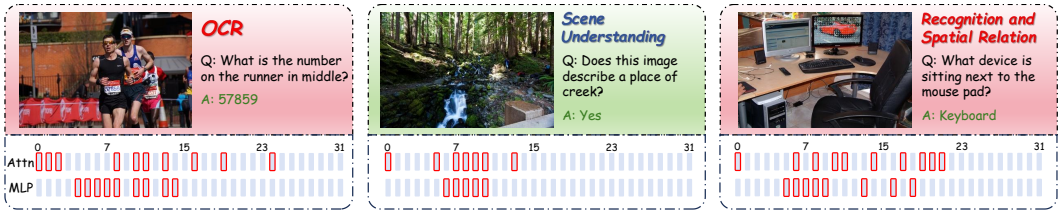


Figure 4: AdaVSkip routing visualization. Red boxes denote preserved visual-token computation.

original FLOPs and improving the MME score from 1519.0 to 1535.4. Combining it with CDPruner further reduces FLOPs to 6.1% and increases these speedups to 6.10× and 3.01×, respectively, while maintaining an MME score of 1473.9, close to that of the original model. Compared with VisionZip and MMTok, the combined method achieves higher MME scores while maintaining competitive inference speeds, thereby offering a better balance between performance and efficiency.

4.5 Visualization and Case Study

To further illustrate AdaVSkip's adaptive behavior, we visualize the module-wise routing decisions for three representative samples under the $K = 24$ budget. As shown in Figure 4, different inputs lead to different amounts and locations of preserved visual-token computation across model depth. For the scene-understanding sample in the middle, the question can be answered with relatively coarse visual information, and AdaVSkip skips a large number of modules while still producing the correct answer. In contrast, the OCR sample requires reading fine-grained text from the image, and the spatial-relation sample requires recognizing the object and its relation to the surrounding scene; AdaVSkip preserves more modules for these two cases. Moreover, although AdaVSkip preserves more modules for both samples, their selected subsets differ, showing that the routing pattern adapts to the specific visual reasoning requirements of each input. These examples show that AdaVSkip adapts not only the amount of visual computation but also how it is allocated across modules.

5 Conclusion

This work studies vertical redundancy in visual-token computation for efficient MLLM inference. Our analysis shows that its extent and distribution vary across inputs and between self-attention and MLP. Motivated by these findings, we propose AdaVSkip, which uses lightweight routers to independently control visual-token computation in self-attention and MLP at each layer. A progressive two-stage training scheme combines necessity-guided supervision with policy refinement, updating only the routers while keeping the backbone frozen. Experiments across three backbones and multiple benchmarks demonstrate better performance–efficiency trade-offs than fixed-layer and early-exit methods. Combining AdaVSkip with visual token compression jointly exploits vertical and horizontal redundancy, achieving better performance at lower computational cost than compression alone.

AI USE STATEMENT

In this work, generative AI tools were used for a limited number of tasks requiring mandatory disclosure. Specifically, they were used to assist with the implementation of the proposed method and with translation. The research ideas, methodological design, experimental setup, and key implementation decisions were developed by the authors, while generative AI tools were used only as supporting tools during implementation. Generative AI tools were not used to generate synthetic datasets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, clean or reformat datasets, support qualitative or thematic data analysis, or interpret experimental results. Additionally, we used generative AI tools for tasks with recommended disclosure. In particular, Codex was used for auxiliary software-engineering tasks, including debugging, code refactoring, and writing supporting experimental scripts. Generative AI tools were also used to assist with manuscript drafting, language editing, readability improvement, and formatting. All AI-assisted code, translations, and written content were manually reviewed and verified by the authors. AI-assisted implementations and code were tested for correctness before being used in the reported experiments. We take responsibility for the final content of this work, including all text, code, claims, results, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the source code in the supplementary materials to facilitate reproduction of our results. Section 3 describes AdaVSkip's module-wise visual-token skipping mechanism, necessity-guided routing target construction, and two-stage router training framework, while Section 4 introduces the evaluated backbones, benchmarks, and comparison methods. Appendix A details the training-data construction, router architecture, training hyperparameters, computational resources, baseline comparison protocol, and FLOPs estimation. Appendix C further documents the necessity-profiling procedures and an optional accelerated variant for preparing routing supervision. Additional analyses, ablations, and complete experimental results are provided in Appendices B and D. Since the backbone parameters remain frozen throughout training, the supplied router checkpoints can be used with the corresponding pretrained backbones to evaluate AdaVSkip without repeating router training.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems*, 2022.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

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

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? In *Advances in Neural Information Processing Systems*, 2024b.

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, et al. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *Science China Information Sciences*, 2024c.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in Neural Information Processing Systems*, 2023.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*, 2023.

Sixun Dong, Juhua Hu, Mian Zhang, Ming Yin, Yanjie Fu, and Qi Qian. Mmtok: Multimodal coverage maximization for efficient inference of vlms. In *International Conference on Learning Representations*, 2026.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Kaituo Feng, Manyuan Zhang, Hongyu Li, Kaixuan Fan, Shuang Chen, Yilei Jiang, Dian Zheng, Peiwen Sun, Yiyuan Zhang, Haoze Sun, et al. Onethinker: All-in-one reasoning model for image and video. *arXiv preprint arXiv:2512.03043*, 2025.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, et al. Mme: A comprehensive evaluation benchmark for multimodal large language models. *arXiv preprint arXiv:2306.13394*, 2023.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, et al. Seed1.5-vl technical report. *arXiv preprint arXiv:2505.07062*, 2025.

Drew A Hudson and Christopher D Manning. 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*, 2019.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *European Conference on Computer Vision*, 2016.

Youngeun Kim, Youjia Zhang, Huiling Liu, Aecheon Jung, Sunwoo Lee, and Sungeun Hong. Zoo-prune: Training-free token pruning via zeroth-order gradient estimation in vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

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.

Bohao Li, Rui Wang, Guangzhi Wang, Yuying Ge, Yixiao Ge, and Ying Shan. Seed-bench: Benchmarking multimodal llms with generative comprehension. *arXiv preprint arXiv:2307.16125*, 2023a.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*, 2023b.

Wentong Li, Yuqian Yuan, Jian Liu, Dongqi Tang, Song Wang, Jie Qin, Jianke Zhu, and Lei Zhang. Tokenpacker: Efficient visual projector for multimodal llm. *International Journal of Computer Vision*, 2025.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, 2023c.

Zhihang Lin, Mingbao Lin, Luxi Lin, and Rongrong Ji. Boosting multimodal large language models with visual tokens withdrawal for rapid inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in Neural Information Processing Systems*, 2023.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024a.

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

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European Conference on Computer Vision*, 2024c.

Dongchen Lu, Zilu Zhang, Leping Huang, Yuyao Sun, Jianliang Zeng, Mao Shu, and Huo Cao. Internvl-x: Advancing and accelerating internvl series with efficient visual token compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

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

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and CV Jawahar. Infographicvqa. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

Qiong Wu, Wenhao Lin, Yiyi Zhou, Weihao Ye, Zhanpeng Zeng, Xiaoshuai Sun, and Rongrong Ji. Accelerating multimodal large language models via dynamic visual-token exit and the empirical findings. *arXiv preprint arXiv:2411.19628*, 2024.

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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Weihao Ye, Qiong Wu, Wenhao Lin, and Yiyi Zhou. Fit and prune: Fast and training-free visual token pruning for multi-modal large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Qianhao Yuan, Qingyu Zhang, Yanjiang Liu, Jiawei Chen, Yaojie Lu, Hongyu Lin, Jia Zheng, Xianpei Han, and Le Sun. Shortv: Efficient multimodal large language models by freezing visual tokens in ineffective layers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Qizhe Zhang, Mengzhen Liu, Lichen Li, Ming Lu, Yuan Zhang, Junwen Pan, Qi She, and Shanghang Zhang. Beyond attention or similarity: Maximizing conditional diversity for token pruning in mllms. *arXiv preprint arXiv:2506.10967*, 2025a.

Shaolei Zhang, Qingkai Fang, Zhe Yang, and Yang Feng. Llava-mini: Efficient image and video large multimodal models with one vision token. In *International Conference on Learning Representations*, 2025b.

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.

Anhao Zhao, Fanghua Ye, Yingqi Fan, Junlong Tong, Zhiwei Fei, Hui Su, and Xiaoyu Shen. Skipgpt: Dynamic layer pruning reinvented with token awareness and module decoupling. *arXiv preprint arXiv:2506.04179*, 2025.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

A Implementation Details and Experimental Protocol

A.1 Training Data

Stage-I training data. For Stage-I router training, we randomly sample 5,500 examples from the LLaVA-1.5 visual instruction-tuning dataset. These samples are used to construct necessity-guided routing targets and train the routers. They cover diverse multimodal tasks, including visual question answering, image captioning, OCR-related understanding, and fine-grained visual reasoning.

Stage-II training data. For each backbone, we construct a Stage-II training set of 3,000 image-QA examples from OneThinker (Feng et al., 2025), excluding video examples. The set comprises 2,000 examples on which the original MLLM answers correctly but the Stage-I routed model answers incorrectly under the target computation budget, together with 1,000 randomly sampled examples to maintain data diversity.

A.2 Router Architecture

At the entrance of each LLM layer, two independent routers operate in parallel on the same layer-input text representation to predict keep probabilities for the self-attention and MLP modules, respectively. Following the main text definition, the router input concatenates the mean-pooled prompt text states, excluding the last token, with the last prompt text token state. Each router consists of an RMSNorm layer, a linear down-projection, a Tanh activation, a second RMSNorm layer, and a linear prediction head. The input and hidden dimensions are 8192 and 256, respectively, for all evaluated backbones.

A.3 Stage-I Training Details

Table 7: Cumulative necessity thresholds used for different skipping budgets.

Backbone	Budget	Self-Attn Threshold τ_{attn}	MLP Threshold τ_{mlp}
LLaVA-1.5-7B	$K = 16$	0.985	0.90
LLaVA-1.5-7B	$K = 20$	0.965	0.83
LLaVA-1.5-7B	$K = 24$	0.935	0.70
LLaVA-NeXT-7B	$K = 16$	0.99	0.925
LLaVA-NeXT-7B	$K = 20$	0.94	0.85
LLaVA-NeXT-7B	$K = 24$	0.80	0.75
Qwen3-VL-8B-Instruct	$K = 18$	0.93	0.93
Qwen3-VL-8B-Instruct	$K = 22$	0.895	0.89

In Stage I, we freeze the MLLM backbone and train only the routers using necessity-guided binary supervision and the downstream language-modeling loss. For each target budget, we select separate cumulative-necessity thresholds for self-attention and MLP modules so that the resulting routing targets match the desired average skipping level on the training set, while allowing the number and locations of skipped modules to vary across samples. Table 7 lists the thresholds for all backbones and budgets. We train for one epoch with AdamW, using a learning rate of 2×10^{-5}, a batch size of 4, and loss weights $\lambda_{\mathrm{route}} = \lambda_{\mathrm{lm}} = 1$. We apply hard routing gates in the forward pass and use a straight-through estimator to propagate gradients from the language-modeling loss to the router.

A.4 Stage-II Training Details

In Stage II, we keep the MLLM backbone frozen and optimize only the routers using a GRPO-style objective. For each training sample, we draw $G = 8$ routing trajectories from the Bernoulli distributions predicted by the routers. Each trajectory receives a reward $R = R_{\mathrm{corr}} + \lambda_{\mathrm{SCR}} R_{\mathrm{SCR}}$, where R_{corr} is 1 if the generated answer exactly matches the ground-truth answer after normalization and 0 otherwise. The skip-consistency reward R_{SCR} penalizes deviations of the self-attention and MLP keep ratios from their sample-specific Stage-I references, which are recorded before refinement and remain fixed throughout training. We normalize rewards within each sampled group to compute group-relative advantages. We use a clipping coefficient of $\epsilon = 0.2$, a reward weight of $\lambda_{\mathrm{SCR}} = 0.8$, and a learning rate of 1×10^{-5} in all experiments.

A.5 Computational Resources

Table 8: GPU configurations and wall-clock times for Stage-I and Stage-II training.

Backbone	Stage-I GPU	Stage-I Time	Stage-II GPU	Stage-II Time
LLaVA-1.5-7B	4× RTX 5090	0.13 h	4× RTX PRO 6000	0.75 h
LLaVA-NeXT-7B	4× RTX 5090	0.35 h	4× RTX PRO 6000	1.45 h
Qwen3-VL-8B-Instruct	4× RTX PRO 6000	0.25 h	4× RTX PRO 6000	1.75 h

Table 8 summarizes the hardware configurations and wall-clock times for router training. Each run uses four NVIDIA GeForce RTX 5090 GPUs or four NVIDIA RTX PRO 6000 GPUs. Stage-I training takes 0.13–0.35 hours across the backbones, whereas Stage-II policy refinement takes 0.75–1.75 hours. The combined wall-clock times for the stages are 0.88, 1.80, and 2.00 hours for LLaVA-1.5-7B, LLaVA-NeXT-7B, and Qwen3-VL-8B-Instruct, respectively.

A.6 Baseline Reproducibility and Comparison Protocol

We use the following protocol for baseline comparisons. For vertical compression, we include VTW, DyVTE, and ShortV. We reproduce VTW and ShortV using their official implementations, following the experimental settings provided in their respective repositories. For VTW, N denotes the number of layers in which visual-token computation is skipped, and we set $N = K$ for each corresponding AdaVSkip configuration. ShortV retains visual keys and values in layers where visual-token updates are skipped, resulting in higher computation per skipped layer. We therefore set $N = K + 3$ on LLaVA-1.5 and LLaVA-NeXT and $N = K + 1$ on Qwen3-VL to obtain comparable retained FLOPs. For DyVTE, we use the results reported in the original paper, as it requires training and its trained checkpoints were not publicly available at the time of our experiments. For horizontal compression, we reproduce the baselines using their official implementations and settings. For hybrid compression, we directly combine the trained AdaVSkip routers with CDPruner on LLaVA-1.5 and LLaVA-NeXT and with VisionZip on Qwen3-VL, without additional training. We compare these hybrid configurations with standalone token compression at comparable FLOPs. For example, at the 10% equivalent budget on Qwen3-VL, VisionZip alone retains 10% of the visual tokens, whereas the hybrid retains 20% and applies AdaVSkip with $K = 18$.

A.7 FLOPs Estimation

We estimate transformer prefill FLOPs per sample, counting one multiply-add as two FLOPs and omitting element-wise operations. Let N_v and N_t denote the numbers of visual and text tokens in the original input, respectively, d the hidden dimension, and L the number of transformer layers. Let d_q and d_{kv} denote the query and key/value projection dimensions, respectively, d_{ffn} the MLP intermediate dimension, H the number of query heads, and d_h the head dimension. For attention with n_q query tokens and n_{kv} key/value tokens, and an MLP processing n tokens, the FLOPs are

$$\mathcal{F}_{\text{attn}}(n_q, n_{kv}) = 4n_q d d_q + 4n_{kv} d d_{kv} + 4H n_q n_{kv} d_h, \tag{6}$$

$$\mathcal{F}_{\text{mlp}}(n) = 6 n d d_{\text{ffn}}. \tag{7}$$

The original model's per-layer cost is

$$F_{\text{full}} = \mathcal{F}_{\text{attn}}(N_v + N_t, N_v + N_t) + \mathcal{F}_{\text{mlp}}(N_v + N_t). \tag{8}$$

Let $\widetilde{N}_v^l$ denote the actual number of retained visual tokens at layer l, with $\widetilde{N}_v^l = N_v$ when no pruning is applied. The text-token count N_t remains unchanged. For module type $c \in \mathcal{C} = \{\text{attn}, \text{mlp}\}$, representing self-attention and MLP, respectively, let $g_{(l,c)} \in \{0, 1\}$ denote its binary routing gate at layer l. A gate of 1 preserves visual-token computation, whereas 0 skips it; text tokens are always processed. The number of tokens processed by each module is therefore

$$n_{(l,c)} = N_t + g_{(l,c)} \widetilde{N}_v^l, \tag{9}$$

giving the layer cost

$$F_l = \mathcal{F}_{\text{attn}}\big(n_{(l,\text{attn})}, n_{(l,\text{attn})}\big) + \mathcal{F}_{\text{mlp}}\big(n_{(l,\text{mlp})}\big) + 2F_{\text{router}}. \tag{10}$$

Here,

$$F_{\text{router}} = 4dd_r + 2d_r, \tag{11}$$

which counts the two linear projections of one router with input dimension $2d$, hidden dimension d_r, and scalar output. Each layer contains two routers, one for self-attention and one for MLP.

The remaining FLOPs ratio of AdaVSkip is

$$r = \frac{\sum_{l=0}^{L-1} F_l}{LF_{\text{full}}}, \tag{12}$$

where the denominator uses the original, unpruned visual-token count N_v. We compute this ratio for each sample using its actual token counts and module-wise routing decisions, and report the mean over the evaluation set.

B Additional Analyses and Ablations

B.1 Router Input Ablation

Table 9: Effect of router input on AdaVSkip-S1 using LLaVA-1.5-7B at $K = 16$.

Router Input	GQA	MMB	TextVQA	MME	MMStar	Avg. ↑
Visual Avg	51.6	55.0	48.1	1061.5	32.1	83.5%
Text Avg	61.0	62.7	49.3	1496.3	33.5	95.9%
Last Text Token	61.5	63.9	56.4	1484.7	33.5	98.7%
Text Avg + Visual Avg	52.3	61.1	55.6	1239.1	33.6	91.4%
Visual Avg + Last Text Token	61.2	63.4	57.6	1499.5	33.2	98.9%
Text Avg + Last Text Token	61.1	64.0	56.8	1516.3	33.3	**99.1%**

We study how the choice of router input affects downstream performance. As shown in Table 9, text-based router inputs perform substantially better than visual features alone. We attribute this advantage to text states encoding the user query and incorporating visual evidence from preceding layers, which provides task-specific information for routing. Adding mean-pooled visual features to the mean-pooled text representation reduces average relative performance from 95.9% to 91.4%, while adding them to the last text-token representation yields only a small improvement from 98.7% to 98.9%. Combining the last text-token representation with the mean-pooled text representation achieves the highest average performance of 99.1%, suggesting that the two text representations provide complementary routing signals.

B.2 Actual Visual-Token Skipping Ratios

Tables 10 and 11 report the actual visual-token computation skipping ratios during prefill on LLaVA-1.5-7B, LLaVA-NeXT-7B, and Qwen3-VL-8B-Instruct under different settings. For each module type, we compute the percentage of layers that skip visual-token computation for each sample and average it across all samples in the benchmark. Each entry is reported as self-attention / MLP.

B.3 Additional Training-Component Ablations

Table 12 extends the training-component ablations to $K = 20$ and $K = 24$. At $K = 20$, adding the downstream CE loss to necessity-guided supervision improves average relative performance from 96.8% to 97.6%, showing the benefit of directly optimizing routing decisions for downstream prediction. Introducing adaptive target keep counts further improves performance to 98.2%, supporting the benefit of adapting the amount of preserved computation to each input. RL refinement without SCR increases performance to 99.2%, but also raises retained FLOPs from 44.7% to 48.6%, indicating that the policy improves answer correctness partly by preserving more visual-token computation. Adding SCR reduces retained FLOPs to 45.2%, close to the Stage-I level, while maintaining 99.0% average relative performance. This confirms that SCR helps constrain the routing policy within the target computation budget and achieves a better efficiency-performance trade-off.

At the more aggressive budget of $K = 24$, module-wise routing achieves 93.5% average relative performance, compared with 91.6% for layer-wise routing, while reducing retained FLOPs from

Table 10: Actual visual-token skipping ratios under different budgets on LLaVA-1.5-7B and LLaVA-NeXT-7B.

Method	VQA^{v2}	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar
LLaVA-1.5-7B								
AdaVSkip-S1($K = 16$)	42.9/51.6	42.6/50.5	32.1/40.8	57.4/65.1	49.4/65.2	56.3/62.8	57.3/64.8	57.1/64.0
AdaVSkip($K = 16$)	41.6/50.3	40.1/49.9	31.5/40.7	59.2/65.2	52.4/62.7	55.6/64.8	55.5/65.0	56.8/63.6
AdaVSkip-S1($K = 20$)	52.1/62.5	50.8/60.4	47.2/57.5	58.5/71.9	58.4/70.1	61.9/72.1	60.7/72.0	61.6/72.1
AdaVSkip($K = 20$)	50.6/59.9	49.6/58.9	42.9/57.1	62.6/70.3	64.5/70.4	59.3/69.9	60.8/70.8	61.3/71.7
AdaVSkip-S1($K = 24$)	72.7/76.1	69.7/77.8	69.6/72.7	75.7/83.7	76.0/82.1	77.4/79.9	76.1/82.6	76.0/78.7
AdaVSkip($K = 24$)	71.2/74.8	71.0/76.1	68.8/70.3	81.3/84.0	79.0/82.0	78.2/78.7	77.2/81.2	77.0/77.9
LLaVA-NeXT-7B								
AdaVSkip-S1($K = 16$)	43.7/51.6	43.2/51.3	35.0/44.6	56.1/63.2	55.8/62.4	53.2/70.1	53.1/70.6	52.8/65.8
AdaVSkip($K = 16$)	43.1/51.0	41.5/51.2	35.8/43.1	56.2/62.9	53.8/61.7	56.1/68.1	56.3/67.7	55.3/65.2
AdaVSkip-S1($K = 20$)	52.1/62.3	51.3/60.9	44.8/59.2	59.4/75.0	59.3/77.1	63.1/74.6	62.7/74.9	61.6/72.5
AdaVSkip($K = 20$)	53.7/63.9	53.6/61.6	46.2/57.2	60.2/75.6	59.8/74.0	64.2/73.7	65.6/74.8	63.0/71.8
AdaVSkip-S1($K = 24$)	69.8/74.4	75.5/73.2	62.3/71.3	72.5/81.2	74.2/79.2	72.7/81.3	75.6/81.1	73.0/80.9
AdaVSkip($K = 24$)	71.1/74.2	72.7/73.9	67.0/71.2	75.8/81.2	72.4/81.5	72.6/81.4	74.6/81.2	72.7/80.8

Table 11: Actual visual-token skipping ratios under different budgets on Qwen3-VL-8B-Instruct.

Method	ChartQA	HallB	AI2D	DocVQA	InfoVQA	MMB	MME	SEED
AdaVSkip-S1($K = 18$)	40.7/47.5	45.8/48.7	57.0/56.3	40.0/44.4	42.5/45.3	54.8/55.5	60.2/59.1	57.9/56.5
AdaVSkip($K = 18$)	42.4/46.6	46.9/48.2	57.2/56.2	40.8/44.7	43.9/47.0	55.8/53.9	60.5/58.6	59.2/55.4
AdaVSkip-S1($K = 22$)	62.1/58.6	61.6/56.4	63.5/67.2	62.0/60.2	63.0/57.5	64.2/66.0	69.4/64.8	67.2/67.6
AdaVSkip($K = 22$)	59.1/57.3	61.0/53.5	65.8/66.7	59.6/58.4	60.7/54.5	66.4/65.5	69.1/64.5	69.3/67.3

35.5% to 32.3%. This supports making separate decisions for self-attention and MLP modules: a shared layer-wise decision may skip useful visual-token computation in one module or retain unnecessary computation in the other. Decoupling these decisions allows AdaVSkip to allocate computation more precisely under aggressive skipping budgets. RL refinement with SCR further improves performance from 93.5% to 95.1%, while retained FLOPs increase only slightly from 32.3% to 32.5%, demonstrating that policy refinement remains effective at this budget.

B.4 Routing Visualization Across Budgets

Figure 5 provides additional examples of AdaVSkip routing decisions at $K = 16$, $K = 20$, and $K = 24$, with larger K corresponding to more aggressive visual-token computation skipping. Within each budget, both the number and locations of modules that preserve visual-token computation vary across inputs. The OCR examples at $K = 16$ and $K = 20$ preserve visual-token computation across more modules than the other examples under the same budget, consistent with the need to extract fine-grained textual information from the image. For instance, reading the Twitter account name at $K = 16$ retains more modules, including self-attention modules in deeper layers, than identifying hair color or checking object presence. Self-attention and MLP also exhibit different keep/skip patterns within the same sample, illustrating how AdaVSkip allocates visual-token computation separately to the two module types. Together, these examples show that AdaVSkip adapts both the amount of visual-token computation and its allocation across layers and modules to the input.

C Accelerated Necessity Profiling

As discussed in the main text, full module-wise profiling incurs substantial offline cost because each sample requires evaluating $2L$ interventions for an L-layer model. We provide an accelerated variant that combines prefix activation reuse with visual token compression. Below, we describe these two components and evaluate their effects on necessity estimation and subsequent router training.

Table 12: Ablation study of AdaVSkip training components on LLaVA-1.5-7B.

Method	FLOPs↓	VQA^{v2}	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar	Avg.↑
				$K = 20$						
Layer-wise Routing (Full Stage-I)	45.4%	77.0	60.2	54.6	84.8	1457.6	63.4	64.2	33.6	97.5%
Module-wise + Necessity Targets	45.3%	76.3	60.1	51.8	84.8	1427.0	63.7	65.6	33.5	96.8%
+ Downstream CE Loss	45.0%	76.2	59.7	53.6	85.6	1475.7	63.6	66.1	33.3	97.6%
+ Adaptive Keep Counts	44.7%	77.0	60.7	55.6	86.3	1465.7	63.9	65.7	32.8	98.2%
+ RL Refinement w/o SCR	48.6%	78.0	60.7	57.4	86.3	1470.2	63.9	66.1	33.7	99.2%
+ Skip-Consistency Reward	45.2%	77.7	61.2	56.7	86.1	1449.9	64.0	66.4	33.8	99.0%
				$K = 24$						
Layer-wise Routing (Full Stage-I)	35.5%	72.8	56.1	51.4	80.5	1350.0	60.9	61.6	30.4	91.6%
Module-wise Routing (Full Stage-I)	32.3%	72.2	57.0	52.2	83.1	1362.6	60.6	62.6	33.0	93.5%
+ RL Refinement w/ SCR	32.5%	74.0	57.6	52.6	83.3	1407.9	61.3	63.1	34.4	95.1%

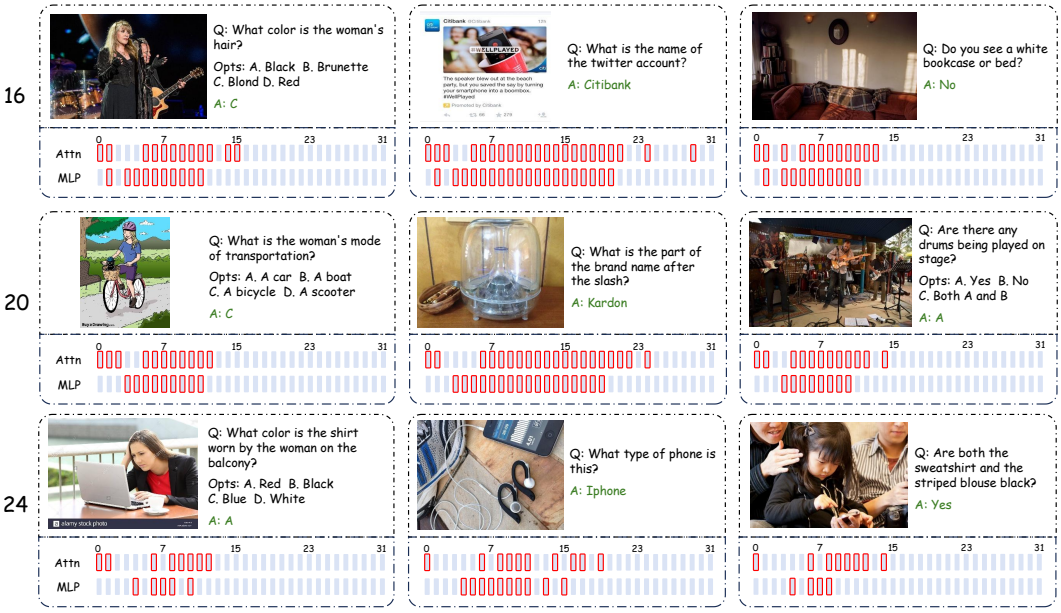


Figure 5: Visualization of AdaVSkip routing decisions under different skipping budgets. Red boxes indicate modules where visual-token computation is preserved.

C.1 Acceleration Strategy

Prefix activation reuse. Each profiling pass skips visual-token computation in a single module, leaving all preceding computations unchanged. We therefore cache the input hidden states of the self-attention and MLP modules during the reference forward pass. When evaluating a module, we resume computation from its cached input, skip visual-token computation in that module, and recompute all subsequent modules to obtain the output distribution. Under deterministic inference, this reuse is equivalent to recomputing the shared prefix and leaves the necessity scores unchanged. Prefix activation reuse is therefore not specific to the accelerated variant and can also be applied to the full profiling procedure described in the main text.

Visual token compression. To further reduce the cost of necessity estimation, we apply visual token compression during profiling, retaining approximately 20% of the original visual tokens. The reference and intervened passes use the same retained tokens and teacher-forced answer sequence, ensuring that necessity scores are computed under a consistent compression setting. This reduces the computation required for repeated profiling passes. Token compression is applied only during offline necessity estimation, leaving the subsequent router training and evaluation settings unchanged.

Table 13: Profiling time and speedup.

Model	Base (h)	Full (h)	Accel. (h)
LLaVA-1.5	1.26	0.66 (1.91×)	0.25 (5.04×)
LLaVA-NeXT	3.97	2.05 (1.94×)	0.68 (5.84×)

Table 14: Necessity profile agreement.

Model	Overlap (%)	Retention (%)
LLaVA-1.5	91.78	99.08
LLaVA-NeXT	91.50	99.05

Table 15: Effect of profiling on AdaVSkip-S1 ($K = 16$). Avg. is the mean of the eight per-benchmark performance ratios relative to the corresponding backbone baseline.

Profiling	FLOPs (%)↓	VQAv2	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar	Avg. (%)↑
LLaVA-1.5										
Full	51.8	77.6	61.1	56.8	86.3	1516.3	64.0	66.2	33.3	99.3
Accelerated	51.5	78.0	61.4	57.7	86.5	1500.9	64.1	66.0	32.5	99.3
LLaVA-NeXT										
Full	46.3	81.0	63.6	57.3	87.3	1509.2	67.0	69.4	37.6	99.4
Accelerated	46.7	80.1	63.8	58.1	87.7	1491.3	66.3	69.5	37.3	99.2

C.2 Experimental Validation

Profiling efficiency. We evaluate three profiling configurations using the same batch size for LLaVA-1.5-7B and LLaVA-NeXT-7B. Base uses all visual tokens without prefix activation reuse. Full retains all visual tokens and reuses prefix activations across interventions to avoid redundant computation. All main text experiments use necessity profiles obtained with this configuration. Accel. further applies visual token compression, retaining approximately 20% of the original visual tokens during profiling. As shown in Table 13, prefix reuse reduces profiling time from 1.26 to 0.66 h on LLaVA-1.5 and from 3.97 to 2.05 h on LLaVA-NeXT, corresponding to $1.91\times$ and $1.94\times$ speedups over Base, respectively. With token compression, profiling time drops to 0.25 and 0.68 h, achieving overall speedups of $5.04\times$ and $5.84\times$ over Base, respectively. These optimizations reduce the offline cost of preparing necessity-guided routing supervision.

Necessity-profile agreement. We examine the agreement between necessity signals obtained from full and token-compressed profiling. For the same samples, we compare profiles computed using all visual tokens (Full) and approximately 20% of them (Accel.), with prefix reuse enabled in both configurations. For each sample, overlap is the fraction of modules retained by full profiling that are also retained by compressed profiling. Importance retention is the sum of full-profile importance scores over the modules retained by compressed profiling, divided by the sum over all modules. As shown in Table 14, retained-module overlap reaches 91.78% on LLaVA-1.5 and 91.50% on LLaVA-NeXT. Importance retention reaches 99.08% and 99.05%, respectively. These results indicate strong agreement between the necessity signals obtained from full and token-compressed profiling.

Downstream performance after router training. We further evaluate whether accelerated profiling provides effective supervision for learning the routing policy. We train AdaVSkip-S1 at $K = 16$ using routing targets derived from full or accelerated profiling, keeping all other training and evaluation settings unchanged. Evaluating Stage-I-only models isolates the effect of profiling on supervised router training from subsequent reinforcement learning. As shown in Table 15, accelerated profiling achieves 99.3% average relative performance on LLaVA-1.5, compared with 99.3% for full profiling, at FLOPs ratios of 51.5% and 51.8%, respectively. On LLaVA-NeXT, the corresponding average relative performance is 99.2% versus 99.4%, at FLOPs ratios of 46.7% and 46.3%. These results suggest that the necessity signals obtained through accelerated profiling provide similarly effective routing supervision, yielding comparable downstream performance at similar computational costs.

D Complete Results

D.1 Complete Results on LLaVA-NeXT-7B

Table 16 reports the complete results on LLaVA-NeXT-7B. AdaVSkip achieves better performance–efficiency trade-offs than ShortV and VTW across all three evaluated skipping budgets. At $K = 16$, it

Table 16: Performance of layer skipping and visual token compression on LLaVA-NeXT-7B.

Method	FLOPs↓	VQA^{v2}	GQA	TextVQA	POPE	MME	MMB	SEED	MMStar	Avg.↑
Vanilla	100.0%	81.8	64.2	61.3	86.5	1519.0	67.4	70.2	35.8	100.0%
(a) Layer Skipping Methods										
Approximately 16 Skipped Layers										
VTW (AAAI'25)($N = 16$)	51.4%	75.6	55.8	47.3	87.5	1518.2	67.1	70.2	37.6	95.3%
ShortV (ICCV'25)($N = 19$)	51.1%	78.8	63.4	56.8	87.3	1526.0	67.2	70.4	37.4	99.2%
AdaVSkip-S1($K = 16$)	46.3%	81.0	63.6	57.3	87.3	1509.2	67.0	69.4	37.6	99.4%
AdaVSkip($K = 16$)	46.8%	81.3	64.1	59.4	87.3	1535.4	66.4	70.0	38.3	**100.4%**
Approximately 20 Skipped Layers										
VTW (AAAI'25)($N = 20$)	39.3%	56.6	50.9	44.6	84.1	1473.1	66.2	68.2	36.5	89.1%
ShortV (ICCV'25)($N = 23$)	40.7%	68.9	60.8	53.3	86.6	1505.5	65.8	68.4	36.3	95.2%
AdaVSkip-S1($K = 20$)	37.9%	78.7	61.7	53.0	87.1	1511.5	65.9	69.1	36.5	97.1%
AdaVSkip($K = 20$)	37.6%	80.1	63.0	55.1	87.2	1503.6	65.8	69.1	37.0	**98.1%**
Approximately 24 Skipped Layers										
VTW (AAAI'25)($N = 24$)	27.1%	46.1	41.3	40.5	50.5	811.0	42.9	48.5	28.6	63.9%
ShortV (ICCV'25)($N = 27$)	30.4%	69.3	55.4	44.8	68.7	1156.0	55.9	60.3	33.8	82.9%
AdaVSkip-S1($K = 24$)	27.2%	72.5	57.1	48.2	83.1	1259.8	57.0	62.4	34.3	88.1%
AdaVSkip($K = 24$)	26.9%	74.8	57.7	47.4	85.8	1374.9	59.9	65.4	35.3	**91.1%**
(b) Visual Token Compression Methods										
22.2% Retained Visual Tokens										
VisionZip (CVPR'25)	22.7%	79.1	61.2	59.9	86.0	1493.4	65.8	66.7	36.3	97.7%
CDPruner (NeurIPS'25)	22.7%	79.9	62.6	58.4	87.3	1474.5	66.3	68.6	35.3	97.9%
MMTok (ICLR'26)	22.7%	79.3	62.2	58.9	86.7	1505.9	65.2	67.7	35.3	97.6%
ZOO-Prune (CVPR'26)	22.7%	79.6	62.2	58.0	86.7	1490.1	65.2	67.9	37.5	98.2%
CDPruner + AdaVSkip	20.9%	81.1	64.0	58.9	88.5	1496.8	67.4	69.9	37.5	**100.0%**
11.1% Retained Visual Tokens										
VisionZip (CVPR'25)	12.6%	76.2	58.9	58.8	82.5	1397.1	63.3	63.4	34.8	93.7%
CDPruner (NeurIPS'25)	12.6%	78.4	61.6	57.4	87.2	1453.0	65.5	67.1	35.1	96.6%
MMTok (ICLR'26)	12.6%	77.6	60.9	56.9	85.7	1473.4	64.3	66.2	34.1	95.4%
ZOO-Prune (CVPR'26)	12.6%	78.1	61.0	57.3	85.5	1427.2	64.9	66.5	35.7	95.9%
CDPruner + AdaVSkip	11.5%	80.2	63.3	57.5	88.2	1498.6	66.7	68.5	36.7	**98.8%**
5.6% Retained Visual Tokens										
VisionZip (CVPR'25)	7.7%	71.4	55.2	55.0	75.8	1327.8	58.6	58.3	31.8	87.1%
CDPruner (NeurIPS'25)	7.7%	76.7	60.8	55.4	86.8	1425.3	64.2	65.5	33.9	94.5%
MMTok (ICLR'26)	7.7%	75.6	60.0	54.2	83.8	1425.1	62.9	64.5	34.0	93.1%
ZOO-Prune (CVPR'26)	7.7%	76.1	59.9	55.4	83.1	1397.9	64.2	64.1	35.0	93.6%
CDPruner + AdaVSkip	6.9%	78.6	62.4	56.2	88.8	1473.9	65.9	67.6	35.3	**97.2%**

achieves 100.4% average relative performance with only 46.8% of the original FLOPs, outperforming both baselines at lower computational cost. Its advantage widens under aggressive skipping: at $K = 24$, AdaVSkip maintains 91.1% average relative performance with 26.9% of the original FLOPs, compared with 82.9% for ShortV and 63.9% for VTW. When combined with CDPruner, AdaVSkip achieves 100.0%, 98.8%, and 97.2% average relative performance at equivalent visual token budgets of 22.2%, 11.1%, and 5.6%, respectively. The corresponding FLOPs ratios are only 20.9%, 11.5%, and 6.9% of the original model. Across these settings, the combination exceeds the strongest standalone token compression baseline by 1.8, 2.2, and 2.7 percentage points, respectively, while requiring fewer FLOPs. These results demonstrate that jointly exploiting vertical and horizontal redundancy yields better performance–efficiency trade-offs than token compression alone.

D.2 Complete Results on Qwen3-VL-8B-Instruct

Table 17 reports the complete results on Qwen3-VL-8B-Instruct. For visual-token skipping, AdaVSkip achieves 98.6% average relative performance with 51.4% of the original FLOPs at $K = 18$, outperforming ShortV and VTW by 1.8 and 16.8 percentage points, respectively. Under the more aggressive $K = 22$ budget, it maintains 89.3% average relative performance with 41.3% of the original FLOPs, compared with 83.0% for ShortV and 66.3% for VTW at comparable computational cost. The benefits of combining AdaVSkip with visual token compression become more pronounced as the equivalent token budget decreases. VisionZip + AdaVSkip achieves 97.8%, 94.4%, and 85.5% average relative performance at equivalent visual token budgets of 40%, 20%, and 10%, exceeding the strongest standalone compression baseline at each budget by 1.7, 6.5, and 6.8 percentage points, respectively. At the 10% equivalent budget, VisionZip alone retains 10% of the original visual tokens, whereas the combined configuration retains 20% and applies AdaVSkip with $K = 18$. The combination achieves 85.5% average relative performance with 13.5% of the original FLOPs, compared with 71.7% at 13.9% FLOPs for VisionZip alone. These results show that jointly exploiting vertical and

Table 17: Performance of layer skipping and visual token compression on Qwen3-VL-8B-Instruct.

Method	FLOPs↓	ChartQA	HallB	AI2D	DocVQA	InfoVQA	MMB	MME	SEED	Avg.↑
Vanilla	100.0%	85.9	62.4	84.1	96.1	83.1	84.5	1746.2	78.7	100.0%
(a) Layer Skipping Methods										
Approximately 18 Skipped Layers										
VTW (AAAI'25)($N = 18$)	52.2%	27.7	58.5	81.7	94.3	37.6	79.6	1709.9	75.2	81.8%
ShortV (ICCV'25)($N = 19$)	51.5%	80.0	59.8	82.8	94.4	78.7	82.6	1702.7	78.0	96.8%
AdaVSkip-S1($K = 18$)	51.5%	80.2	63.3	82.4	94.4	79.0	83.7	1720.1	77.5	97.8%
AdaVSkip($K = 18$)	51.4%	81.2	63.8	83.0	95.2	80.8	84.0	1709.2	78.2	**98.6%**
Approximately 22 Skipped Layers										
VTW (AAAI'25)($N = 22$)	41.6%	21.9	48.8	70.0	69.1	28.2	65.2	1399.6	62.9	66.3%
ShortV (ICCV'25)($N = 23$)	41.3%	57.4	53.9	76.0	73.0	52.4	77.6	1585.8	77.5	83.0%
AdaVSkip-S1($K = 22$)	40.2%	66.8	53.6	79.1	80.7	67.5	79.6	1622.0	75.8	88.3%
AdaVSkip($K = 22$)	41.3%	66.2	56.0	78.3	83.7	69.2	79.1	1648.5	75.8	**89.3%**
(b) Visual Token Compression Methods										
40% Retained Visual Tokens										
VisionZip (CVPR'25)	42.1%	81.4	59.9	82.4	93.2	70.9	84.5	1746.3	77.2	96.1%
CDPruner (NeurIPS'25)	42.1%	71.9	58.8	80.9	79.7	58.5	83.5	1740.4	75.7	90.3%
MMTok (ICLR'26)	42.1%	80.0	60.1	83.4	90.0	66.1	83.9	1734.4	77.6	94.9%
ZOO-Prune (CVPR'26)	42.1%	82.2	59.7	81.7	91.1	69.5	83.8	1715.5	77.5	95.4%
VisionZip + AdaVSkip	41.8%	81.6	61.0	82.7	94.7	79.3	84.5	1715.1	78.2	**97.8%**
20% Retained Visual Tokens										
VisionZip (CVPR'25)	23.2%	71.3	55.3	80.2	80.2	53.0	81.7	1663.9	74.4	87.6%
CDPruner (NeurIPS'25)	23.2%	53.1	53.2	75.1	62.1	42.6	79.8	1695.7	73.0	79.6%
MMTok (ICLR'26)	23.2%	67.5	58.7	80.4	78.6	52.7	81.8	1697.0	75.5	87.9%
ZOO-Prune (CVPR'26)	23.2%	68.8	53.8	76.1	74.5	47.3	81.1	1650.5	75.5	84.7%
VisionZip + AdaVSkip	22.8%	77.7	58.8	82.1	91.8	69.0	83.7	1699.4	76.9	**94.4%**
10% Retained Visual Tokens										
VisionZip (CVPR'25)	13.9%	44.8	45.0	72.4	53.8	35.5	75.5	1528.0	69.3	71.7%
CDPruner (NeurIPS'25)	13.9%	33.3	45.6	69.4	43.4	31.9	74.9	1534.7	68.9	67.7%
MMTok (ICLR'26)	13.9%	48.6	54.5	77.0	62.4	41.9	77.9	1636.3	73.3	78.7%
ZOO-Prune (CVPR'26)	13.9%	50.5	48.2	70.7	51.6	33.4	76.1	1516.6	72.7	72.9%
VisionZip + AdaVSkip	13.5%	67.0	56.0	79.2	76.2	49.6	81.4	1628.2	73.9	**85.5%**

horizontal redundancy better preserves performance at comparable computational cost than token compression alone, particularly under aggressive compression settings.