VISIONWEAVE: WEAVING ELASTIC VISUAL REPRESENTATIONS AS A NATIVE CAPABILITY OF MLLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal large language models have become the dominant paradigm for visual understanding, but incur substantial costs by encoding inputs into dense, fixed-size patch tokens. However, visual information is unevenly distributed: some regions require fine-grained detail, while others admit compact representations. Downsampling sacrifices this detail, while existing token pruning and adaptive approaches remain limited in content-adaptive granularity, task generalization, and integration with modern MLLMs and serving infrastructure. Overcoming these limitations calls for foundation models that learn, end to end, where—and at what granularity—to allocate visual representations, a native capability we term *elastic visual representation weaving*. We introduce *VisionWeave*, establishing this capability in frontier-level MLLMs through large-scale training. It combines two components: a gated spatial pooler constructs coarse-grained representations alongside native fine-grained representations within a shared MRoPE coordinate, while a granularity router learns their content-adaptive allocation. Through self-distillation alone, we validate this capability on Qwen3.5-4B and scale to Qwen3.8-27B with over 30K A100 GPU-hours. Based on Qwen3.8-27B, VisionWeave adaptively adjusts token savings to visual content, saving 43.0% tokens on average while retaining 98.9% native performance across eight benchmarks, versus only 88% performance preserved for token pruning baselines with a fixed 50% savings target. Extensive evaluations confirm robust efficiency–quality trade-offs across diverse tasks, resolutions and video frames. When deployed on SGLang serving engine, our method achieves a 2.30× end-to-end throughput gain while reducing mean TTFT by 54.4% and mean TPOT by 60.6%. Together, we believe these results position elastic visual weaving as a promising native capability for next-generation multimodal foundation models.

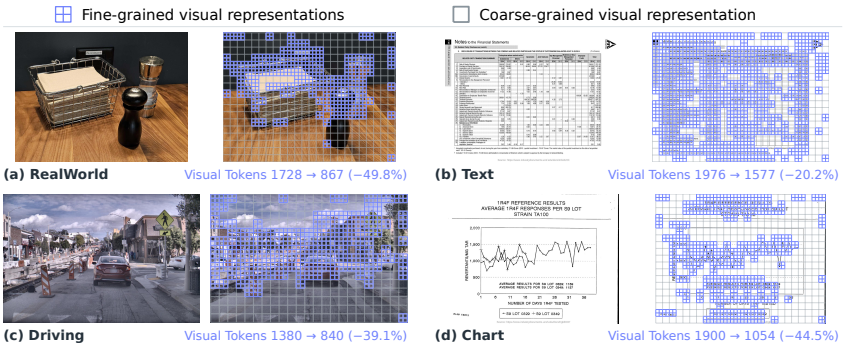

Figure 1: Weaving elastic visual representations. More examples appear in Appendix B.

1 INTRODUCTION

Multimodal large language models (MLLMs) have become the dominant paradigm for visual understanding, supporting diverse tasks such as visual question answering, document analysis, and video understanding within a unified framework (Bai et al., 2025b;a). As these tasks demand finer spatial detail and broader temporal coverage, MLLMs must process higher-resolution images and longer

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.

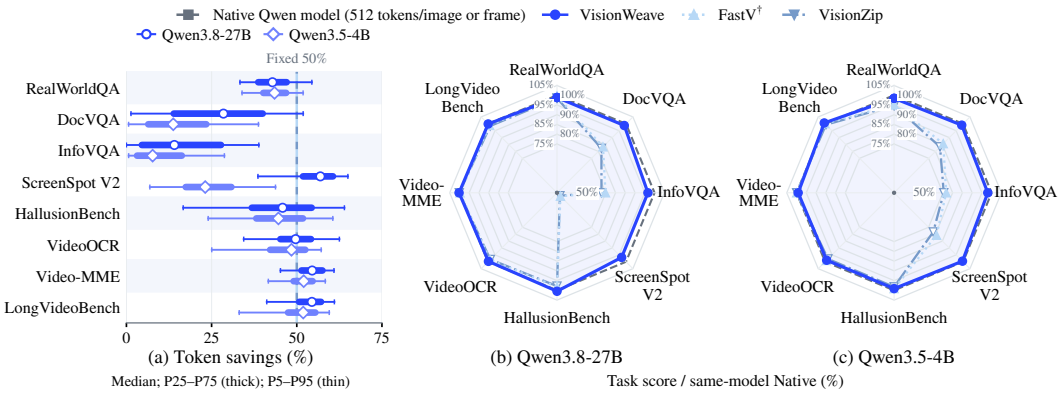

Figure 2: VisionWeave achieves content-adaptive token savings with near-native quality, whereas fix-budget baselines degrade sharply on challenging tasks. Visual case studies on ScreenSpotV2 and DocVQA illustrate this difference (Appendix A). Our VisionWeave still retains higher quality even when baselines are granted its adaptive saving ratios (Appendix E.6).

sequences of video frames. However, their visual inputs are typically partitioned into fixed-size patches and encoded into dense sequences of visual tokens. The resulting growth in sequence length places substantial computational and memory demands, making visual efficiency a central challenge in scaling multimodal capabilities (Shao et al., 2025a; Tao et al., 2024; Shao et al., 2025b).

Visual downsampling offers a straightforward solution but sacrifices fine-grained detail, motivating extensive research into visual token pruning and merging (Chen et al., 2024; Yang et al., 2024; Ye et al., 2024; Guo et al., 2026). However, both heuristic and learned methods introduce a training–inference mismatch and discard task-specific information. Our evaluations of FastV and VisionZip reveal severe degradation on information-dense inputs and grounding tasks. Common design choices in these methods, including attention-weight extraction and in-LLM pruning, also complicate integration with modern serving engines such as SGLang and vLLM, where implementations remain rare and practical efficiency gains insufficiently validated. Preset compression ratios further limit adaptivity to diverse visual inputs. These limitations motivate native, content-adaptive visual processing for general-purpose MLLMs: learning where to retain fine-grained detail and where—and to what extent—to use compact coarse representations.

Recent work has explored content-adaptive visual representations, but realizing this adaptivity as a native capability of general-purpose MLLMs remains challenging. One line of work varies ViT patch sizes using low-level image statistics, such as edge density and entropy (Yu et al., 2025; Choudhury et al., 2025). These methods primarily target standalone vision tasks, and their granularity criteria are not optimized for diverse downstream multimodal objectives. ViCO (Cui et al., 2025) is particularly close to our approach: it partitions an image into tiles and adaptively assigns each a visual token resolution. However, its tile-based design limits applicability to modern MLLMs such as Qwen (Bai et al., 2025a) and Kimi (Team, 2026), which use native-resolution visual processing. Moreover, detail-rich regions and low-information backgrounds are often interleaved (Figure 1), requiring finer-grained spatial allocation than a single resolution per tile can provide.[1]

We envision elastic visual representation weaving as a native capability of foundamental MLLMs that should satisfy three requirements:

1. *Autonomously content-adaptive token savings while preserving quality.*
2. *Consistently favorable trade-off than downsampling across diverse visual task inputs.*
3. *Compatibility with frontier-level MLLM architectures and modern serving infrastructure.*

To this end, we introduce VisionWeave, which enables MLLMs to operate natively on elastic visual representations, adaptively weaving together fine- and coarse-grained representations according to visual content. To our knowledge, this is the first effort to establish elastic visual representation weaving as a native capability of frontier MLLMs through large-scale training, accompanied by

[1] See related work section for further discussion in Appendix C and Figure 14 for a visual comparison.

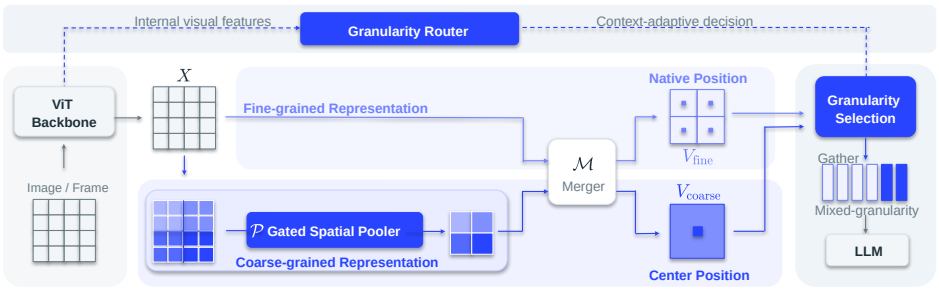

Figure 3: Overall architecture of VisionWeave. The gated spatial pooler complements native fine-grained tokens V_{fine} with coarse-grained representations V_{coarse} within the same MRoPE coordinate, while the granularity router adaptively weaves them into a mixed-granularity sequence for the LLM.

extensive evaluation to assess its viability as an integral capability of foundation models. The method comprises two core components. First, the elastic representation itself: a gated spatial pooler constructs a learned coarse-grained representation that complements the native fine-grained tokens within the same MRoPE frame. Second, learned granularity allocation: a granularity router probes ViT features to determine the appropriate granularity for each spatial block. Through self-distillation alone, we make elastic visual processing a native capability, validating on Qwen3.5-4B and then scaling to frontier Qwen3.8-27B with over 30K A100 GPU-hours.

With this native capability, VisionWeave preserves fine-grained detail where needed and uses compact representations elsewhere (Figure 1), achieving content-adaptive token savings with minimal impact on task quality (Figure 2). At an input budget of 512 tokens per image, VisionWeave on Qwen3.8-27B saves 16.5–55.1% of visual tokens across eight benchmarks, averaging 43.0% savings while preserving 98.9% native performance. In contrast, FastV† and VisionZip retain only 88.2% and 87.5%, respectively, at a fixed 50% savings target. Broader evaluations across input resolutions and video frame budgets demonstrate consistently more favorable efficiency–quality trade-offs than input downsampling. Crucially, these token savings translate into practical serving gains build upon modern SGLang serving engine, achieving around 2$\times$ throughput improvements. Together, content-adaptive savings, cross-task robustness, and practical deployability support elastic visual processing as a native capability of next-general efficient MLLMs.

2 Method

In this work, we realize elastic visual representation weaving through two complementary components. A gated spatial pooler constructs learned coarse-grained representations that complement native fine-grained tokens (§2.2.1). A granularity router probes intermediate ViT features to determine the granularity for each spatial block (§2.2.2). Through self-distillation alone, we make elastic visual processing a native capability of MLLMs (§2.3).

2.1 Preliminaries

For modern MLLMs, the ViT produces patch features $X \in \mathbb{R}^{H\times W\times d}$. A native merger $\mathcal{M}$ concatenates each 2$\times$2 neighborhood and applies an MLP to produce fine-grained visual tokens:

$$V_{\text{fine}} = \mathcal{M}(X) \in \mathbb{R}^{\frac{H}{2}\times\frac{W}{2}\times D}, \tag{1}$$

where d and D are the ViT and LLM hidden sizes, respectively. MRoPE assigns each token coordinates $\pi_{ij} = (\tau, i, j)$, with temporal coordinate τ and spatial indices (i, j). The flattened visual tokens and textual context c condition autoregressive response generation, $p_\theta(y \mid c, V_{\text{fine}})$.

2.2 Architecture

2.2.1 Gated Spatial Pooler

Pooling adjacent patch features is a natural compression primitive: the patches of a block cover contiguous regions whose content is typically correlated—increasingly so wherever the image is

locally homogeneous. Leveraging this correlation, Our *gated spatial pooler* $\mathcal{P}$ maps each 2×2 neighborhood of ViT patch features to a single pooled feature. Passing these features through the native merger yields the coarse-grained representation in the same embedding space.

$$V_{\text{coarse}} = \mathcal{M}\big(\mathcal{P}(X)\big) \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times D}, \qquad \mathcal{P}(X) \in \mathbb{R}^{\frac{H}{2}\times\frac{W}{2}\times d}. \tag{2}$$

Each block (I, J) on the $\frac{H}{4}\times\frac{W}{4}$ grid covers 4×4 patches and admits either four native fine-grained tokens $\{V_{\text{fine},2I+a,2J+b}\}_{a,b\in\{0,1\}}$ or one coarse-grained token $V_{\text{coarse},IJ}$. We use positional interpolation (Peng et al., 2023; Chen et al., 2023) to place each coarse token at the geometric center of its four corresponding fine-grained tokens.

$$\pi_{IJ} = \big(\tau,\ 2I + \tfrac{1}{2},\ 2J + \tfrac{1}{2}\big). \tag{3}$$

We implement the gated spatial pooler $\mathcal{P}$ as a learnable extension of mean pooling, combining content-dependent and channel-wise weighting with feature refinement. For each 2×2 neighborhood, we normalize its four patch features $\{x_k\}_{k=1}^4$ using learnable layer normalization [2] and aggregate them into a shared global context g_k:

$$u_k = \mathrm{LN}(x_k) \in \mathbb{R}^d, \qquad g_k = f_{\text{glb}}\big([u_1; u_2; u_3; u_4]\big) + b_k \ \in \mathbb{R}^d, \tag{4}$$

where $[\cdot;\cdot]$ denotes concatenation and $b_k \in \mathbb{R}^d$ is a learnable position-specific bias (DeepSeek-AI, 2026). The gate then computes channel-wise scores from patch feature u_k and its context g_k:

$$\alpha_k = \underset{k=1,\dots,4}{\mathrm{softmax}}\, f_{\text{gate}}\big([u_k;\ g_k]\big) \ \in (0,1)^d, \tag{5}$$

where the softmax normalizes over the four spatial positions channel-wise, i.e. $\sum_{k=1}^4 \alpha_k = \mathbf{1}_d$. Finally, a value net f_{val} refines each feature, and the pooled output is the gated sum,

$$v_k = u_k + f_{\text{val}}\big([u_k;\ g_k]\big) \ \in \mathbb{R}^d, \qquad \mathcal{P}(x_{1:4}) = \sum\nolimits_{k=1}^4 \alpha_k \odot v_k \ \in \mathbb{R}^d, \tag{6}$$

where f_{glb}, f_{gate}, f_{val} are two-layer MLPs (each mapping into $\mathbb{R}^d$) and $\odot$ is the elementwise product. Beyond its greate expressiveness [3], the pooler offers a favorable starting point: zero-initializing the final layers of f_{gate} and f_{val} reduces $\mathcal{P}$ to mean pooling over normalized patch features.

2.2.2 Granularity Router

With both granularities available, our *granularity router* learns their content-adaptive allocation across spatial blocks [4]. It probes ViT features at multiple depths with a learnable query, combining low-level detail from early layers with semantic information from later layers. We implement this multi-depth probing with one query state $h_{IJ} \in \mathbb{R}^d$ per spatial block, initialized from a shared learnable vector. Each query uses the block-center MRoPE coordinate $\big(4I + \frac{3}{2},\ 4J + \frac{3}{2}\big)$ on the $H\times W$ patch grid, corresponding to the same spatial center as Eq. (3). For a ViT with L layers, we denote the features after layer ℓ by $X^{(\ell)} \in \mathbb{R}^{H\times W\times d}$, with $X^{(0)}$ the patch embeddings and $X^{(L)} = X$. Each query state is updated through cross-attention to ViT features, followed by an MLP, both with residual connections:

$$h' = \mathrm{LN}\Big(h + \mathrm{CrossAttn}\big(h,\ \mathrm{LN}(X^{(\ell)})\big)\Big), \qquad h \leftarrow \mathrm{LN}\big(h' + \mathrm{MLP}(h')\big), \tag{7}$$

where $\mathrm{CrossAttn}$ takes its query from h and keys/values from $\mathrm{LN}(X^{(\ell)})$, with MRoPE applied to queries and keys; the KV-side LN keeps features tapped at different depths on a comparable scale. Each query sequentially probes depths $\ell = 0,\ L//2,\ L$. At each depth, a local block is followed by a global block, yielding six independently parameterized blocks in total. The local block attends to the spatial block's own 4×4 patch features $X_{IJ}^{(\ell)} \in \mathbb{R}^{4\times4\times d}$; a *global* block then opens them to the whole frame:

$$h_{IJ} \leftarrow \mathrm{LocalCrossAttn}^{(\ell)}\big(h_{IJ},\ X_{IJ}^{(\ell)}\big);\ \ h_{IJ} \leftarrow \mathrm{GlobalCrossAttn}^{(\ell)}\big(h_{IJ},\ X^{(\ell)}\big). \tag{8}$$

[2] ViT patch features exhibit large magnitude variation; LayerNorm prevents high-magnitude tokens from dominating the pooling output.

[3] Ablation study in Section 3.6 shows gated pooler consistently outperforms the widely used pixel-unshuffle.

[4] Ablation studies in Appendix E.3 validate the effectiveness of the learned router.

Table 1: Three-stage self-distillation for elastic visual processing.

	Stage 1	Stage 2	Stage 3
Trainable components	Gated Spatial Pooler	Granularity Router	Pooler, LLM
Teacher	Native model	Native model	Native model
Routing	Bypassed (all coarse)	Soft-mixing routing	Hard routing
Loss terms	$\mathcal{L}_{\text{distill}}$	$\mathcal{L}_{\text{distill}} + 0.02\mathcal{L}_{\text{bal}}$	$\mathcal{L}_{\text{distill}}$
Student visual tokens	25%	100%	Content-adaptive

Finally, a linear head maps the normalized query state to two logits, giving the probability of using the coarse-grained representation:

$$p_{IJ} = \text{softmax}\left(W_r \,\text{LN}(h_{IJ})\right)_{\text{coarse}}, \qquad W_r \in \mathbb{R}^{2\times d}. \tag{9}$$

For each spatial block, we use its coarse-grained token if the routing probability exceeds a threshold [5] and retain its four native fine-grained tokens otherwise. The selected tokens are gathered into a mixed-granularity sequence for the LLM backbone within the shared MRoPE frame. Algorithm 1 in Appendix summarizes the complete inference path.

2.3 Training

We develop elastic visual weaving through three-stage self-distillation (Table 1).[6] The native forward pass provides output-distribution supervision. Distillation is off-policy [7], with teacher and student conditioned on the same teacher rollout. The student progresses from coarse-grained representations to a differentiable soft-mixing surrogate, and finally to the hard-routed mixed-granularity sequences used at inference.

2.3.1 Stage 1: Coarse-Grained Self-Distillation

We first train the pooler with routing bypassed, using the coarse-grained representation for every block: $V_{\text{student}} = \text{flatten}(V_{\text{coarse}})$. All other components remain frozen. The model's native forward pass serves as the teacher, yielding the distillation objective:

$$\mathcal{L}_{\text{distill}} = \frac{1}{|y|}\sum_{t} \text{KL}\Big(\widehat{p}_{\bar{\theta}}\big(\cdot \mid y_{<t},\, c,\, V_{\text{fine}}\big) \;\Big\|\; \widehat{p}_{\theta}\big(\cdot \mid y_{<t},\, c,\, V_{\text{student}}\big)\Big), \tag{10}$$

where y is a response generated by the teacher and $\bar{\theta}$ denotes its parameters. Both $\widehat{p}_{\bar{\theta}}$ and $\widehat{p}_{\theta}$ are renormalized over the teacher's top-512 vocabulary entries at each response position. Later stages reuse this objective with different constructions of V_{student}.

2.3.2 Stage 2: Soft-Mixing Self-Distillation

We next train only the granularity router using a differentiable soft-mixing surrogate for hard routing. We interpolate feature directions and magnitudes separately to prevent norm differences from biasing the directional mixture. Let $\widehat{V}_{\text{fine}}$ and $\widehat{V}_{\text{coarse}}$ denote the token-wise L2-normalized representations under $\mathcal{N}(v) = v/\max(\|v\|_2, \epsilon)$. We compute

$$\begin{aligned} m_{ij} &= (1 - p_{IJ})\|V_{\text{fine},ij}\|_2 + p_{IJ}\|V_{\text{coarse},IJ}\|_2, \\ V_{\text{mix},ij} &= m_{ij}\,\mathcal{N}\Big((1 - p_{IJ})\widehat{V}_{\text{fine},ij} + p_{IJ}\widehat{V}_{\text{coarse},IJ}\Big). \end{aligned} \tag{11}$$

[5] We set $t = 0.5$ by default, equivalent to selecting the granularity with the higher predicted probability. Section 3.5 further shows how varying this threshold controls the compression level at inference time.

[6] Appendix D provides detailed training settings and analyses of loss and routing dynamics, offering further insight into the learning process.

[7] We try switching to on-policy self-distillation in Stage 3 of Qwen3.5-4B but observe no consistent gains (Appendix E.1). We hypothesize that elastic visual weaving keeps the student's output distribution close to the teacher's, limiting the benefit of on-policy sampling; we therefore use off-policy training for efficiency.

where $I = \lfloor i/2 \rfloor$, $J = \lfloor j/2 \rfloor$, and ϵ ensures numerical stability. Each coarse token is broadcast to its four corresponding native positions. The resulting sequence, $V_{\text{student}} = \text{flatten}(V_{\text{mix}})$, is used in the distillation objective of Eq. (10). We add a per-sample balance loss borrow from MoE training:

$$\mathcal{L}_{\text{bal}} = \frac{F}{\rho} G + \frac{1-F}{1-\rho} (1-G), \tag{12}$$

where F is the fraction of the sample's blocks with $p_{IJ} \geq 0.5$, G is its mean routing probability, ρ is the target coarse-routing fraction. The penalty encourages more coarse routing when $F < \rho$ and less when $F > \rho$. Per-sample statistics prevent larger inputs from dominating the estimate and batch averages from masking input-level imbalance. We set $\rho = 0.8$ and optimize $\mathcal{L}_{\text{distill}} + 0.02\mathcal{L}_{\text{bal}}$.

2.3.3 Stage 3: Mixed-Granularity Self-Distillation

Stage 3 performs self-distillation on the hard-routed mixed-granularity sequences used at inference. [8] The router head includes an offline-calibrated scalar coarse-routing bias, held fixed during training and inference, without auxiliary balancing losses or online adjustments. [9] The student gathers V_{student} from V_{fine} and V_{coarse} using the same token selection and RoPE coordinates as at inference. The native forward pass of a frozen teacher provides supervision through the distillation objective.

2.4 Deploy VisionWeave on Modern Serving Infrastructure

To demonstrate how native elastic visual weaving can be deployed on modern serving infrastructure, we integrate VisionWeave into SGLang (Zheng et al., 2023). This requires addressing two core challenges: supporting block-center MRoPE coordinates and mixed-granularity visual sequences.

Block-center RoPE support. Coarse-grained tokens use the block-center RoPE coordinates defined in Eq. (3). To support these half-integer positions in SGLang's integer-indexed RoPE caches, we double all position coordinates and halve rotary frequencies, preserving rotary phases. We apply this rescaling to cache construction and growth and to position updates during prefill and decode, reusing the existing rotary-embedding and attention kernels.

Mixed-granularity sequence support. We extend SGLang's encoder-disaggregated stage by attaching layout metadata to visual responses. This enables pre-admission embedding slicing and RoPE construction, finalizing sequence layouts before scheduling while preserving dynamic batching, KV-cache management, and CUDA-graph decoding. Serving gains are reported in Section 3.4.

3 Experiments

We organize our evaluation around the three requirements for elastic visual weaving as a native MLLM capability: quality-preserving, content-adaptive token savings (Section 3.2), robust tradeoffs across resolutions and frame budgets (Section 3.3), and practical serving gains on SGLang (Section 3.4). Further analyses are presented in Sections 3.5 and 3.6.

3.1 Experimental Setup

Models and baselines. We evaluate with two frontier open-source MLLMs, the Qwen3.5-4B and Qwen3.8-27B. We compare VisionWeave against two heuristic visual-token reduction methods: VisionZip Yang et al. (2024) and FastV Chen et al. (2024), representing token merging and pruning, respectively. To enable a consistent evaluation within SGLang and support its modern serving features, we use a FastV variant, denoted FastV†, that selects tokens after the vision encoder rather than within the LLM. Unless otherwise stated, VisionWeave uses a default routing threshold of $t = 0.5$.

Tasks. We assess cross-domain robustness on eight benchmarks spanning natural-image (RealWorldQA), document and infographic understanding (DocVQA and InfoVQA), GUI grounding (ScreenSpotV2), visual hallucination (HallusionBench), video details (VideoOCR), and video understanding (Video-MME and LongVideoBench). Evaluation details appear in Appendix E.4.

[8] Appendix E.2 confirm that Stage 3 improves performance across diverse tasks, especially on grounding.

[9] In experiments, we calibrate on a training subset targeting 50% visual-token savings.

Table 2: Content-adaptive token savings with near-native quality.

	Native Model	Fixed Savings (s=50%)			Content-adaptive Savings	
Qwen3.5-4B	512 token/Image	Downsampling	FastV†	VisionZip	VisionWeave	Savings (%)
RealWorldQA	74.64	72.16 (↓3.32%)	70.98 (↓4.90%)	70.72 (↓5.25%)	**73.46** (↓1.58%)	43.2
DocVQA	91.92	82.71(↓10.02%)	78.69(↓14.39%)	76.57(↓16.70%)	**91.03** (↓0.97%)	16.0
InfoVQA	67.44	52.87(↓21.60%)	51.75(↓23.27%)	50.89(↓24.54%)	**66.11** (↓1.97%)	10.6
ScreenSpotV2	89.86	80.42(↓10.51%)	72.56(↓19.25%)	70.44(↓21.61%)	**89.47** (↓0.43%)	24.1
HallusionBench	69.88	69.09 (↓1.13%)	**69.26** (↓0.89%)	68.64 (↓1.77%)	**69.26** (↓0.89%)	44.0
VideoOCR	64.51	59.74 (↓7.39%)	**63.90** (↓0.95%)	63.18 (↓2.06%)	63.79 (↓1.12%)	47.0
Video-MME	67.78	67.07 (↓1.05%)	67.70 (↓0.12%)	**67.74** (↓0.06%)	67.22 (↓0.83%)	51.4
LongVideoBench	59.99	58.86 (↓1.88%)	60.06 (↑0.12%)	59.61 (↓0.63%)	**60.36** (↑0.62%)	49.8
Average	73.25	67.87 (↓7.11%)	66.86 (↓7.96%)	65.97 (↓9.08%)	**72.59** (↓0.90%)	35.8
Qwen3.8-27B						
RealWorldQA	78.82	74.64 (↓5.30%)	**78.43** (↓0.49%)	77.65 (↓1.48%)	77.91 (↓1.15%)	43.1
DocVQA	92.47	83.19(↓10.04%)	77.05(↓16.68%)	76.01(↓17.80%)	**91.33** (↓1.23%)	27.3
InfoVQA	74.13	58.48(↓21.11%)	55.49(↓25.15%)	53.96(↓27.21%)	**71.61** (↓3.40%)	16.5
ScreenSpotV2	94.42	91.04 (↓3.58%)	49.53(↓47.54%)	49.21(↓47.88%)	**91.43** (↓3.17%)	55.1
HallusionBench	73.25	72.10 (↓1.57%)	71.92 (↓1.82%)	71.66 (↓2.17%)	**73.60** (↑0.48%)	44.9
VideoOCR	70.72	65.59 (↓7.25%)	69.69 (↓1.46%)	69.54 (↓1.67%)	**70.51** (↓0.30%)	49.6
Video-MME	70.89	70.41 (↓0.68%)	70.70 (↓0.27%)	70.96 (↑0.10%)	**71.04** (↑0.21%)	54.2
LongVideoBench	66.49	64.70 (↓2.69%)	65.74 (↓1.13%)	65.45 (↓1.56%)	**66.42** (↓0.11%)	53.0
Average	77.65	72.52 (↓6.53%)	67.32(↓11.82%)	66.81(↓12.46%)	**76.73** (↓1.08%)	43.0

3.2 Content-Adaptive Token Savings with Preserved Quality

With a default routing threshold $t = 0.5$, VisionWeave achieves content-adaptive token savings while maintaining near-native quality (Table 2). Across all eight benchmarks, VisionWeave incurs an average performance loss of only 0.90% on Qwen3.5-4B and 1.08% on Qwen3.8-27B, relative to the corresponding native models. Specifically, on Qwen3.8-27B, VisionWeave saves 16.5% of tokens on InfoVQA and 27.3% on DocVQA, compared with 49.6–54.2% across the three video benchmarks, where performance losses remain at or below 0.30%. Qwen3.5-4B shows a similar pattern, saving 10.6–16.0% on the two document benchmarks and 47.0–51.4% on video.

In contrast, FastV† and VisionZip incur average losses of 11.82% and 12.46% on Qwen3.8-27B at a fixed 50% token-savings target. Although losses are modest on video, they degrade sharply on grounding and information-dense tasks: VisionZip incurs losses of 47.88% on ScreenSpotV2 and 17.80% on DocVQA. VisionWeave, by contrast, loses only 3.17% on ScreenSpotV2 despite more aggressive token savings of 55.1%, while adaptively reducing its savings to 27.3% on DocVQA with only a 1.23% loss. These results highlight the advantage of content-adaptive savings: reducing visual tokens according to input demands while preserving quality. Appendix E.6 further shows that FastV† and VisionZip incur larger losses even when assigned VisionWeave's adaptive savings ratios.

3.3 Robustness Across Input Resolutions and Frame Budgets

We next demonstrate the robustness of VisionWeave's efficiency–quality trade-offs across tasks, spatial resolutions, and frame budgets. Consistently improving upon input downsampling is nontrivial: a method that works well in one setting may fail in another. Figure 4 shows that VisionWeave consistently maintains favorable trade-offs across the evaluated datasets and resolutions. FastV† and VisionZip remain competitive on VideoOCR yet degrade sharply on ScreenSpotV2. Their effectiveness also varies with resolution within the same task. On Qwen3.8-27B DocVQA, for example, VisionZip offers a competitive trade-off at a 256-token input budget but falls behind simple downsampling at higher resolutions. By contrast, VisionWeave consistently offers a more favorable efficiency–quality trade-off than input downsampling across the evaluated tasks and resolutions. The same advantage extends to temporal coverage (Figure 5). With a 256-frame cap, Qwen3.8-27B VisionWeave outperforms the native model capped at 128 frames by 1.20 points on LongVideoBench

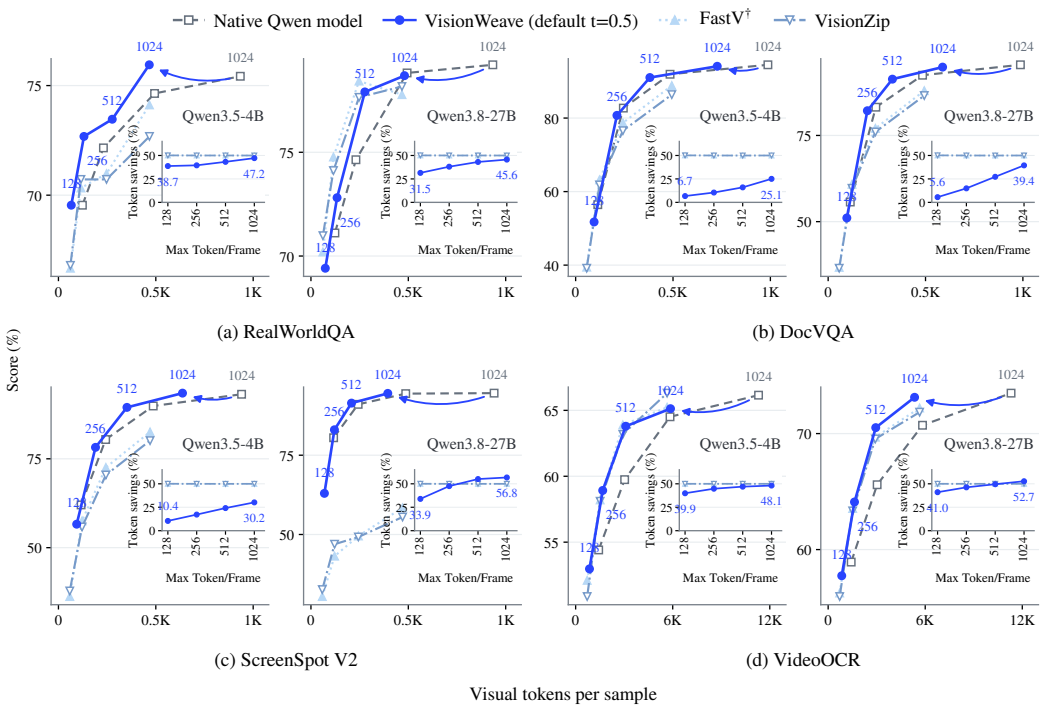

Figure 4: VisionWeave achieves consistently favorable trade-offs across tasks and resolutions.

while using fewer visual tokens. Together, these results support the robust performance of native elastic visual weaving across diverse input conditions. More results appear in Appendix E.5.

3.4 End-to-End Serving Efficiency on SGLang

Table 3: VisionWeave accelerates Qwen3.8-27B serving on SGLang.

Model	Throughput (req/min) ↑	Mean TTFT (s) ↓	P95 TTFT (s) ↓	Mean TPOT (ms) ↓	P95 TPOT (ms) ↓
Native Model	0.90	169.90	281.41	355.64	715.58
VisionWeave	**2.07** (2.30×)	**77.54** (↓54.4%)	**120.59** (↓57.1%)	**140.29** (↓60.6%)	**289.19** (↓59.6%)

To evaluate practical deployability, we benchmark native Qwen3.8-27B and VisionWeave on SGLang using our integration (Section 2.4). Both run on two A100-80GB GPUs (BF16, TP=2), with chunked prefill and CUDA-graph decoding enabled. We use the LongVideoBench with fps2, 256 max frames per video, a spatial budget of 1024 tokens/frame. Client concurrency is four, with a server limit of eight. More details appear in Appendix E.7. Overall, VisionWeave delivers a 2.30× end-to-end throughput speedup (Table 3). Both mean and tail latency improve: P95 time to first token (TTFT) and time per output token (TPOT) decrease by 57.1% and 59.6%, respectively. These results demonstrate that native elastic visual processing translates into higher serving throughput and lower response latency under this long-video workload.

3.5 Inference-Time Control of Token Savings

The default threshold $t = 0.5$ already provides content-adaptive token savings without task-specific tuning. We further show that varying this threshold enables flexible inference-time control without retraining (Figure 7). At the full-retention endpoint, labeled $t = 1$ in the figure, every block uses its fine-grained representation, with no token savings. These results suggest that native visual weaving expands the model's efficiency–quality operating range while preserving its fully fine-grained performance. Lowering the threshold then favors coarse-grained representations and increases token savings. On Video-MME, reducing t from 0.5 to 0.25 increases savings from 54.2% to 66.0% on

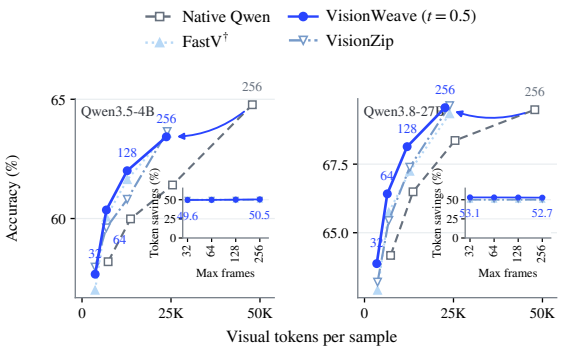

Figure 5: VisionWeave maintains favorable trade-offs across frame budgets on LongVideoBench.

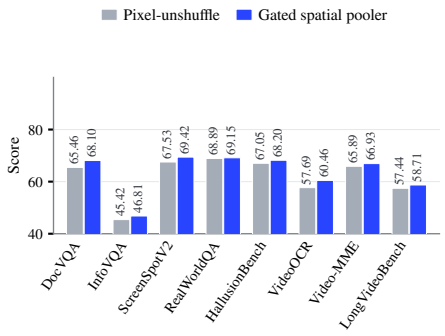

Figure 6: Gated pooling outperforms pixel-unshuffle across eight benchmarks.

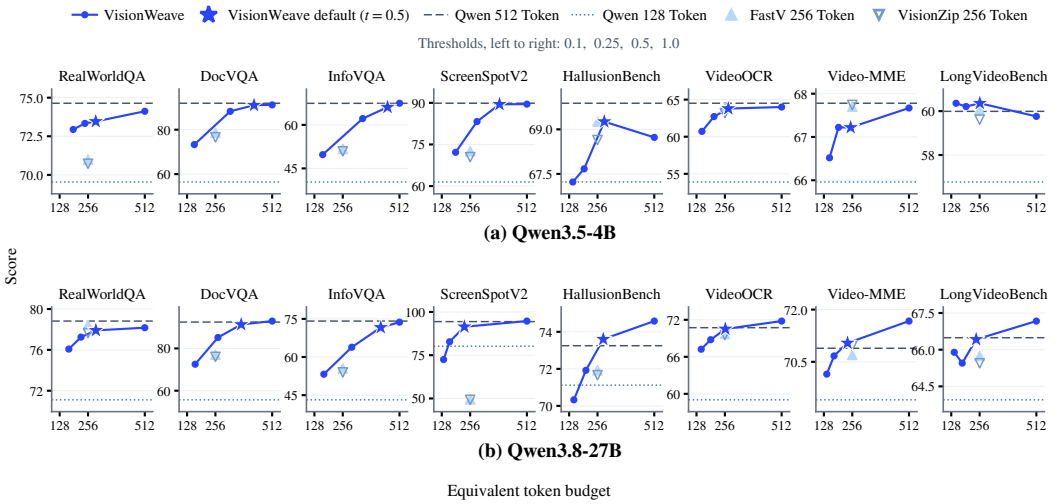

Figure 7: VisionWeave enables flexible inference-time control of token savings while closely matching native-model performance when savings are disabled.

Qwen3.8-27B, with only a 0.37-point performance loss. On Qwen3.5-4B, the same adjustment increases savings from 51.4% to 62.1% while preserving the score of 67.22. These results highlight a key practical advantage of native visual weaving: a single model can accommodate different quality and cost requirements without retraining, while closely matching the native model's performance ceiling when token savings are disabled.

3.6 Ablation Study of Spatial Pooler Design

We compare our gated spatial pooler with a widely used pixel-unshuffle-based projector on Qwen3.5-4B after Stage 1, both at 75% token savings. As shown in Figure 6, our pooler achieves higher scores across all eight benchmarks, including gains of 2.64 points on DocVQA and 2.77 points on VideoOCR. These results support the effectiveness of gated spatial pooling for constructing coarse-grained visual representations.

4 Conclusion

We introduced VisionWeave, establishing elastic visual representation weaving as a native capability of MLLMs through self-distillation alone. It combines content-adaptive token savings with robust efficiency–quality trade-offs across tasks, resolutions, and frame budgets. Our SGLang integration further translates these token savings into higher throughput and lower latency on modern serving infrastructure. These findings establish representational granularity as a learnable degree of freedom, validating content-adaptive computational effort as a feasible architectural principle.

AI USE DISCLOSURE

In this work, we used generative AI tools to assist with dataset reformatting. We did not use these tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, clean datasets, conduct qualitative or thematic data analysis, or interpret results. Assistance with developing or writing mathematical proofs was not applicable to this work. Additionally, we used these tools for language editing to improve grammar, clarity, and readability. We reviewed all AI-assisted work, including the reformatted datasets and manuscript text. We take responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix D details the training data, hyperparameters, and training dynamics, including raw and smoothed loss curves, router entropy, and routing-probability distributions. Evaluation protocols and metrics are provided in Section 3.1 and Appendix E.4. Serving configurations and the inference algorithm are documented in Appendices E.7 and F. We will publicly release our SGLang integration code.

REFERENCES

Xiang An, Yin Xie, Kaicheng Yang, Wenkang Zhang, Xiuwei Zhao, Zheng Cheng, Yirui Wang, Songcen Xu, Changrui Chen, Didi Zhu, Chunsheng Wu, Huajie Tan, Chunyuan Li, Jing Yang, Jie Yu, Xiyao Wang, Bin Qin, Yumeng Wang, Zizhen Yan, Ziyong Feng, Ziwei Liu, Bo Li, and Jiankang Deng. Llava-onevision-1.5: Fully open framework for democratized multimodal training, 2025. URL https://arxiv.org/abs/2509.23661.

Shuai Bai, Yuxuan Cai, Rui-Zhe Chen, Ke qin Chen, Xiong-Hui Chen, Zesen Cheng, Liang-Hao Deng, Wei Ding, Rongyao Fang, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Qidong Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Ming-Sheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xue-Jing Liu, Jiawei Liu, Cheng-Long Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Lingchen Meng, Xin yi Ren, Sibo Song, Yu chen Sun, Jun Tang, Jianhong Tu, Jian-Qiang Wan, Peng Wang, Pengfei Wang, Qiu-Yue Wang, Yuxuan Wang, Tianbao Xie, Yihe Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bo-Wen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Botao Zheng, Humen Zhong, Jingren Zhou, Fanxi Zhou, Jingren Zhou, Yuanzhi Zhu, and Keming Zhu. Qwen3-vl technical report. *ArXiv*, abs/2511.21631, 2025a. URL https://api.semanticscholar.org/CorpusID:283262018.

Shuai Bai, Ke qin Chen, Xue-Jing Liu, Jia-Lin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jian-Qiang Wan, Peng-Fei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *ArXiv*, abs/2502.13923, 2025b. URL https://api.semanticscholar.org/CorpusID:276449796.

Mu Cai, Jianwei Yang, Jianfeng Gao, and Yong Jae Lee. Matryoshka multimodal models, 2024. URL https://arxiv.org/abs/2405.17430.

Junjie Chen, Xuyang Liu, Zichen Wen, Yiyu Wang, Siteng Huang, and Honggang Chen. Variation-aware vision token dropping for faster large vision-language models, 2026. URL https://arxiv.org/abs/2509.01552.

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*, 2024. URL https://api.semanticscholar.org/CorpusID:268358224.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *ArXiv*, abs/2306.15595, 2023. URL https://api.semanticscholar.org/CorpusID:259262376.

Rohan Choudhury, Jungeun Kim, Jinhyung Park, Eunho Yang, László A. Jeni, and Kris M. Kitani. Accelerating vision transformers with adaptive patch sizes. *ArXiv*, abs/2510.18091, 2025. URL https://api.semanticscholar.org/CorpusID:282246197.

Long Cui, Weiyun Wang, Jie Shao, Zichen Wen, Gen Luo, Linfeng Zhang, Yanting Zhang, Yu Qiao, and Wenhai Wang. Vico: A training strategy towards semantic aware dynamic high-resolution. *ArXiv*, abs/2510.12793, 2025. URL https://api.semanticscholar.org/CorpusID:282064724.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. 2026. URL https://api.semanticscholar.org/CorpusID:289623513.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *ArXiv*, abs/2407.11550, 2024. URL https://api.semanticscholar.org/CorpusID:271218006.

Yuan Feng, Hao-Yu Guo, Junlin Lv, S. Kevin Zhou, and Xike Xie. Taming the fragility of kv cache eviction in llm inference. *ArXiv*, abs/2510.13334, 2025a. URL https://api.semanticscholar.org/CorpusID:282102254.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Criticalkv: Optimizing kv cache eviction from an output perturbation perspective. 2025b. URL https://api.semanticscholar.org/CorpusID:276161406.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents, 2025. URL https://arxiv.org/abs/2410.05243.

Haoyu Guo, Yuan Feng, Junlin Lv, Mingjun Xiao, S Kevin Zhou, and Xike Xie. Videomm: Adaptive macro-micro inference for efficient video mllms, 2026. URL https://arxiv.org/abs/2609.16722.

Yuhang Han, Xuyang Liu, Zihan Zhang, Pengxiang Ding, Junjie Chen, Donglin Wang, Honggang Chen, Qingsen Yan, and Siteng Huang. Filter, correlate, compress: Training-free token reduction for mllm acceleration, 2025. URL https://arxiv.org/abs/2411.17686.

Wenbo Hu, Zi-Yi Dou, Liunian Harold Li, Amita Kamath, Nanyun Peng, and Kai-Wei Chang. Matryoshka query transformer for large vision-language models, 2024. URL https://arxiv.org/abs/2405.19315.

Ahmadreza Jeddi, Negin Baghbanzadeh, Elham Dolatabadi, and Babak Taati. Similarity-aware token pruning: Your vlm but faster. *ArXiv*, abs/2503.11549, 2025. URL https://api.semanticscholar.org/CorpusID:277043961.

Selim Kuzucu, Alessio Tonioni, Vasile Lup, Bernt Schiele, Federico Tombari, and Muhammad Ferjad Naeem. Parcel: Pool-anchored resampling with conditioned elastic queries for efficient vision-language understanding, 2026. URL https://arxiv.org/abs/2605.30126.

Xu-Yang Liu, Xiyan Gui, Yuchao Zhang, and Linfeng Zhang. Mixing importance with diversity: Joint optimization for kv cache compression in large vision-language models. *ArXiv*, abs/2510.20707, 2025. URL https://api.semanticscholar.org/CorpusID:282304428.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *ArXiv*, abs/2309.00071, 2023. URL https://api.semanticscholar.org/CorpusID:261493986.

Kele Shao, Keda Tao, Kejia Zhang, Sicheng Feng, Mu Cai, Yuzhang Shang, Haoxuan You, Can Qin, Yang Sui, and Huan Wang. A survey of token compression for efficient multimodal large language models. *Trans. Mach. Learn. Res.*, 2026, 2025a. URL https://api.semanticscholar.org/CorpusID:280323457.

Kele Shao, Keda Tao, Kejia Zhang, Sicheng Feng, Mu Cai, Yuzhang Shang, Haoxuan You, Can Qin, Yang Sui, and Huan Wang. When tokens talk too much: A survey of multimodal long-context token compression across images, videos, and audios. *ArXiv*, abs/2507.20198, 2025b. URL https://api.semanticscholar.org/CorpusID:290088866.

Keda Tao, Can Qin, Haoxuan You, Yang Sui, and Huan Wang. Dycoke : Dynamic compression of tokens for fast video large language models. *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18992–19001, 2024. URL https://api.semanticscholar.org/CorpusID:274192345.

Kimi Team. Kimi k3: Open frontier intelligence. 2026. URL https://api.semanticscholar.org/CorpusID:290625162.

Jiahui Wang, Zuyan Liu, Yongming Rao, and Jiwen Lu. Sparsemm: Head sparsity emerges from visual concept responses in mllms. *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 23177–23187, 2025. URL https://api.semanticscholar.org/CorpusID:279244559.

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. *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19792–19802, 2024. URL https://api.semanticscholar.org/CorpusID:274514545.

Xubing Ye, Yukang Gan, Yi-Xiao Ge, Xiao-Ping Zhang, and Yansong Tang. Atp-llava: Adaptive token pruning for large vision language models. *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24972–24982, 2024. URL https://api.semanticscholar.org/CorpusID:274436316.

Qiyang Yu, Yu Fang, Tianrui Li, Xuemei Cao, Yan Chen, Jianghao Li, and Fan Min. Dynamic granularity matters: Rethinking vision transformers beyond fixed patch splitting. *ArXiv*, abs/2511.19021, 2025. URL https://api.semanticscholar.org/CorpusID:283244585.

Qizhe Zhang, Meng-Zhen Liu, Li-Chen 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*, abs/2506.10967, 2025a. URL https://api.semanticscholar.org/CorpusID:279318547.

Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. Llava-video: Video instruction tuning with synthetic data, 2025b. URL https://arxiv.org/abs/2410.02713.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chu-Yue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christoforos E. Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems 37*, 2023. URL https://api.semanticscholar.org/CorpusID:266174771.

APPENDIX CONTENTS

A Case Studies **14**

A.1 Case Study in GUI Grounding Tasks 14

A.2 Case Study in DocVQA Tasks 14

B More Visualizations of Elastic Visual Weaving **15**

C Related Work **15**

D Training Details and Dynamics **19**

E Additional Experimental Details **21**

E.1 Ablation of Distillation Policy 21

E.2 Ablation of Stage3: Mixed-Granularity Self-Distillation 21

E.3 Ablation of Learned Granularity Router 22

E.4 Evaluation Details 22

E.5 Additional Resolution Results 22

E.6 Performance at Matched Token Savings 23

E.7 SGLang Serving Configuration 24

F Inference Algorithm **24**

G Limitations and Future Works **25**

A Case Studies

A.1 Case Study in GUI Grounding Tasks

Figure 8 compares visual token allocation on desktop, tablet, and web interfaces using Qwen3.8-27B at a 512-token input budget. VisionWeave interleaves fine- and coarse-grained representations, keeping every region represented within a shared spatial frame. In contrast, FastV† drops tokens covering critical click targets, such as the star button in the web interface, leaving gaps in token coverage that can hinder accurate grounding. This spatially coherent coverage helps explain VisionWeave's substantial advantage over FastV† and VisionZip on grounding tasks.

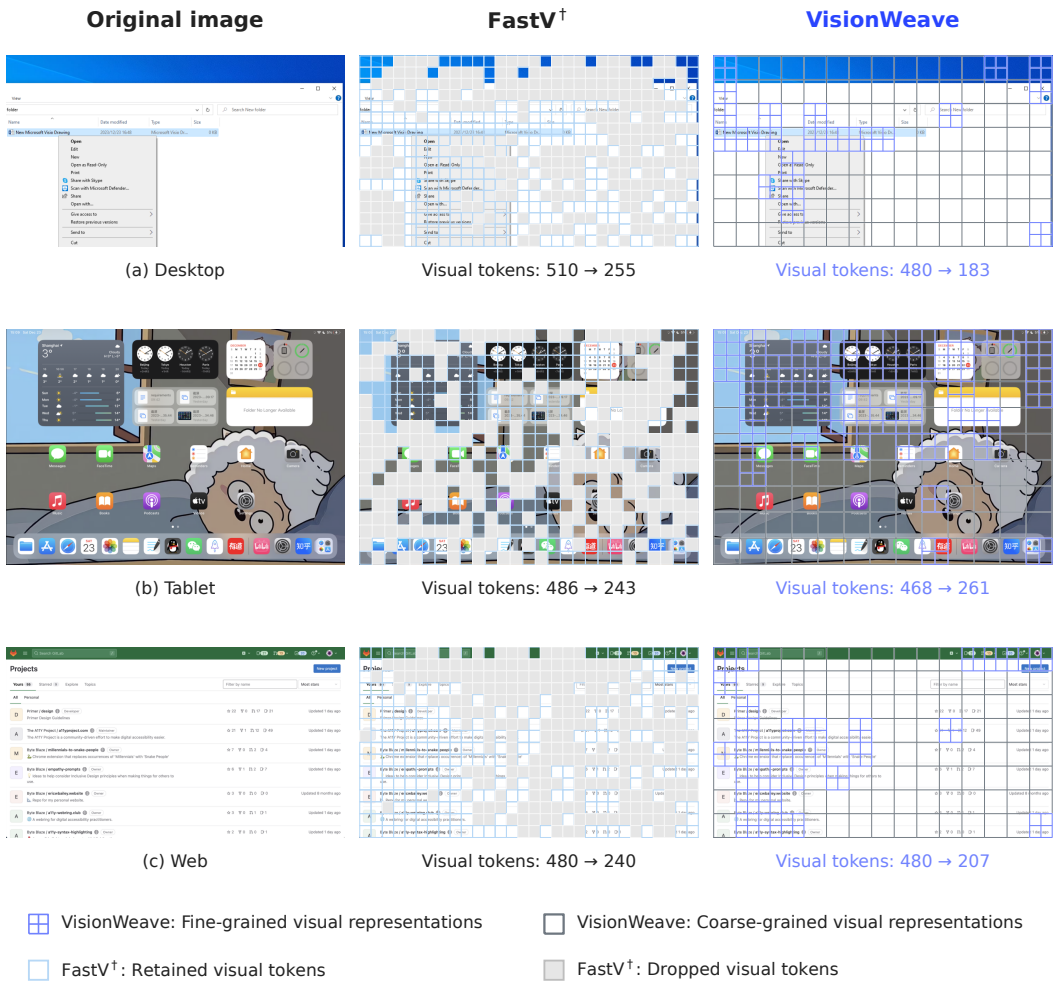

Figure 8: VisionWeave retains full spatial coverage through mixed-granularity representations, whereas FastV† drops visual tokens.

A.2 Case Study in DocVQA Tasks

Figure 9 compares token allocation on two DocVQA examples using Qwen3.8-27B at a 512-token input budget. At the default threshold $t = 0.5$, VisionWeave saves 45.2% of tokens on the form while retaining all native fine-grained tokens for the dense table. FastV† instead removes 50% of tokens in both cases, including tokens covering text. Even in example (a), FastV† drops tokens covering informative text, leaving some characters only partially represented. In contrast, VisionWeave uses coarse-grained representations for larger text while maintaining full spatial coverage. This also helps explain the robustness of VisionWeave across diverse inputs.



Figure 9: VisionWeave adapts visual granularity to document content rather than imposing fixed token savings.

B More Visualizations of Elastic Visual Weaving

Figures 10–13 show additional examples of elastic visual representations on Qwen3.5-4B and Qwen3.8-27B, illustrating how fine- and coarse-grained representations are interleaved according to visual content.

C Related Work

Visual token pruning and merging. Visual token pruning and merging shorten visual sequences through heuristic or learned policies (Jeddi et al., 2025; Ye et al., 2024; Chen et al., 2026; Han et al., 2025; Zhang et al., 2025a), as exemplified by FastV and VisionZip (Chen et al., 2024; Yang et al., 2024). Direct pruning removes tokens deemed unimportant by the selection policy and can substantially alter the visual context available to a pretrained MLLM. Our evaluations of FastV† and VisionZip show modest performance losses on video tasks but substantial degradation on grounding and information-dense inputs. Most methods also rely on preset compression ratios that cannot accommodate the varying information density of visual inputs. In addition, despite extensive research, visual token-pruning methods rarely provide implementations in modern serving engines such as SGLang or vLLM, leaving their practical efficiency gains insufficiently validated. Common algorithmic choices further hinder deployment: some methods require explicit attention weights, which standard FlashAttention kernels do not expose, while others prune tokens within the LLM, complicating engine integration and support for dynamic batching and chunked prefill. Consequently, the growing body of token-pruning research has yet to translate into widespread adoption in practical

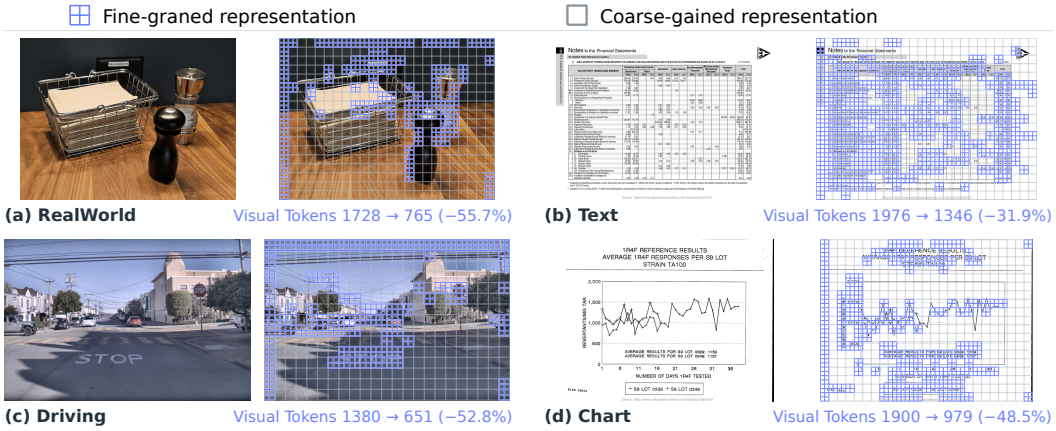

Figure 10: Elastic visual representations on Qwen3.5-4B at a 2048-token input budget.

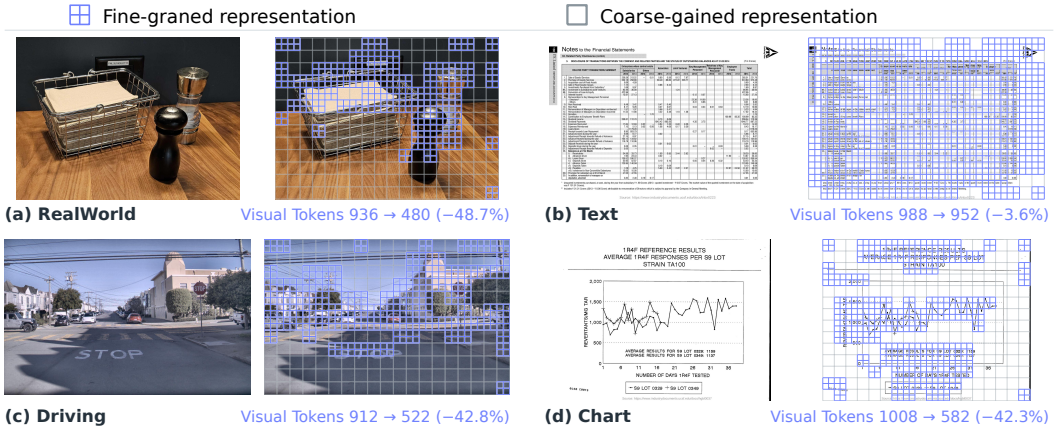

Figure 11: Elastic visual representations on Qwen3.8-27B at a 1024-token input budget.

LLM serving. In contrast, VisionWeave improves visual efficiency through content-adaptive selection of fine- and coarse-grained representations, using the same representation construction and routing during final-stage training and inference. Extensive evaluations on SGLang engine across tasks, input resolutions, and video frame budgets demonstrate consistently favorable efficiency–quality trade-offs, supporting elastic visual weaving as a robust native capability of MLLMs.

Content-adaptive visual representations. At the vision-encoder level, prior work varies patch sizes using low-level image statistics, such as edge density and entropy (Yu et al., 2025; Choudhury et al., 2025). These approaches primarily target standalone vision tasks rather than learning granularity for diverse downstream multimodal objectives. At the MLLM level, Matryoshka Multimodal Models (Cai et al., 2024), Matryoshka Query Transformer (Hu et al., 2024) and PARCEL (Kuzucu et al., 2026) support multiple token budgets, but require the budget to be specified externally. ViCO (Cui et al., 2025) goes further by adaptively assigning a visual token resolution to each image tile. However, its tile-based design limits direct applicability to frontier MLLMs such as Qwen and Kimi, which use native-resolution visual processing. Moreover, interleaved detail-rich regions and low-information backgrounds call for finer spatial allocation than a single resolution per tile provides (Figure 14). VisionWeave instead learns block-wise allocation of fine- and coarse-grained representations within the native-resolution pipeline. Our contribution is to establish this content-adaptive visual processing as a native capability of frontier MLLMs, combining



Figure 12: Additional Qwen3.5-4B visualizations at a 2048-token input budget.

autonomous token savings, robust efficiency–quality trade-offs, and compatibility with modern serving infrastructure.

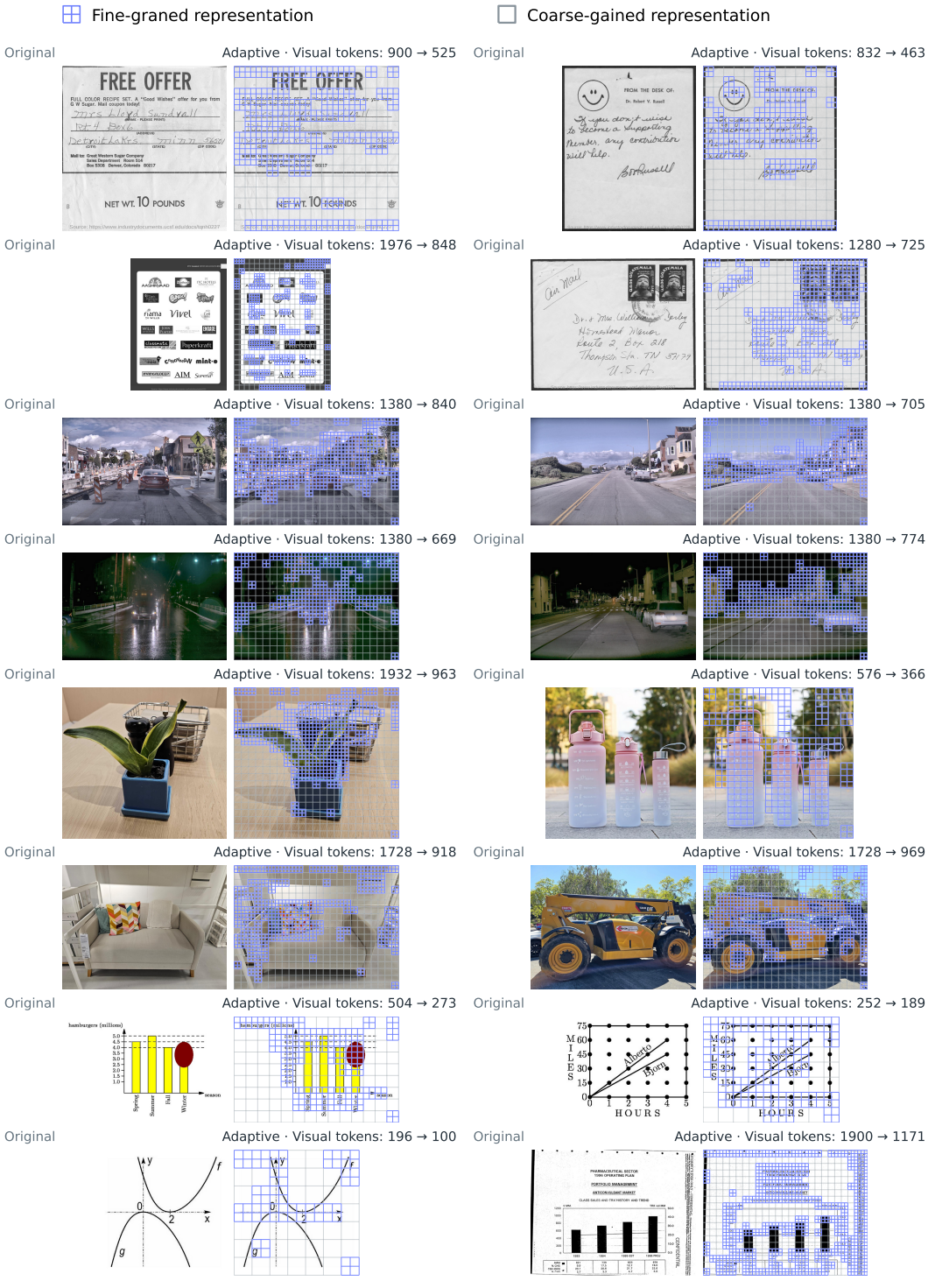

Figure 13: Qwen3.8-27B visualizations corresponding to Figure 12, with the same 2048-token input budget.

KV-cache compression. KV-cache compression addresses a related but distinct aspect of inference efficiency. Visual token efficiency shortens the sequence processed by the LLM, reducing both attention and feed-forward computation during prefill, as well as KV-cache memory, thereby en-

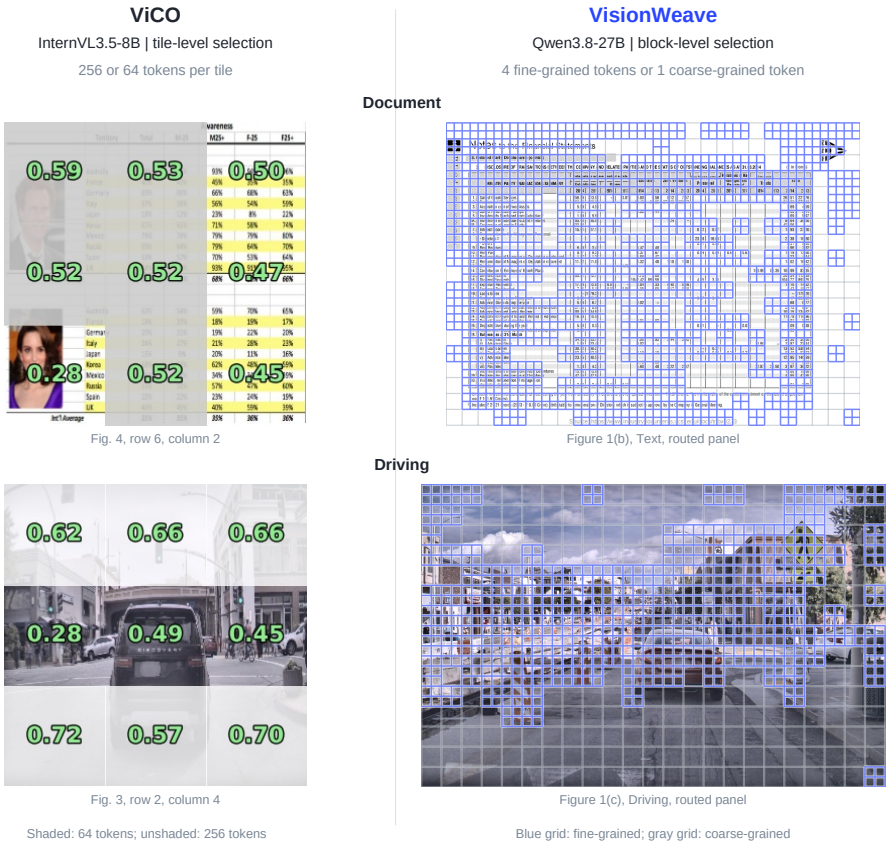

Figure 14: VisionWeave allocates visual granularity at a finer spatial scale than tile-level routing. ViCO panels are cropped from Cui et al. (2025)

abling faster time to first token Feng et al. (2024; 2025b;a); Liu et al. (2025); Wang et al. (2025). In contrast, KV-cache compression compacts the stored key–value states without shortening the input token sequence. When applied after prefill, it primarily reduces memory usage and decoding bandwidth rather than prompt-processing computation. The distinction is therefore between processing fewer tokens and retaining more compact states for those tokens. Learned KV compression, exemplified by DeepSeek-V4 (DeepSeek-AI, 2026), extends this direction by integrating compression into attention itself. Combining such mechanisms with elastic visual representation weaving is a promising direction: fewer visual tokens would reduce computation throughout the LLM, while learned KV compression could further lower the cache memory footprint of dense- or sparse-attention layers.

D Training Details and Dynamics

Architecture. We detail the large-scale Qwen3.8-27B training setup here. The granularity router samples ViT features at depths $[0, 13, 27]$, with one local–global cross-attention pair per depth. Its six layers use 16 attention heads, an MLP expansion ratio of 2, and key/value layer normalization. The router scoring head is kept in FP32.

Training settings. All stages use BF16 training with Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$), zero weight decay, cosine learning-rate decay, and activation recomputation. Table 4 summarizes the stage-specific settings. Distillation is off-policy at temperature 1, using teacher-to-student KL divergence over the teacher's top-512 . Stage 1 weights each sample's mean token loss by the square root of its supervised response length. We observed training instability with this weighting in Stage 2

and therefore use a global mean over supervised response tokens in Stages 2 and 3. Stage 3 updates the pooler and LLM, while keeping the ViT, native merger, and router frozen.

Table 4: Training hyperparameters for Qwen3.8-27B.

Setting	Stage 1	Stage 2	Stage 3
Peak learning rate	1×10^{-4}	1×10^{-5}	3×10^{-6}
Minimum learning rate	5×10^{-6}	0	0
Warmup fraction	5%	5%	2%
Global batch size	32	64	512
Training steps	5,257	2,500	1,000
Gradient clipping	1.0	1.0	0.5
Student maximum sequence length	16,384	24,576	24,576
Packed sequence length	16,384	24,576	Disabled

Training Data. All three stages of Qwen3.8-27B training use the same data mixture, covering image understanding, video understanding, and GUI grounding. The mixture contains approximately 0.78 million samples. Counts refer to individual conversations before sequence packing.

Table 5: Training-data composition for Qwen3.8-27B.

Source	Domain	Approx. training samples
LLaVA-OneVision (An et al., 2025)	Image understanding	246K
LLaVA-Video-178K (Zhang et al., 2025b)	Video understanding	281K
UGround (Gou et al., 2025)	GUI grounding	250K
Total		**777K**

The default input budgets are 4,096 native visual tokens per image and 512 per video frame, with up to 64 sampled frames. Global batch sizes count packed sequences in Stages 1 and 2 and individual samples in Stage 3.

Training dynamics. Figure 15 shows the raw training losses and exponential moving averages (smoothing factor 0.02, initialized at the first step) across the three self-distillation stages. Stage 1 shows a rapid initial decrease followed by slower improvement. Stage 2 gradually reduces the joint distillation and balance objective, while Stage 3 reduces the distillation loss on hard-routed mixed-granularity sequences.

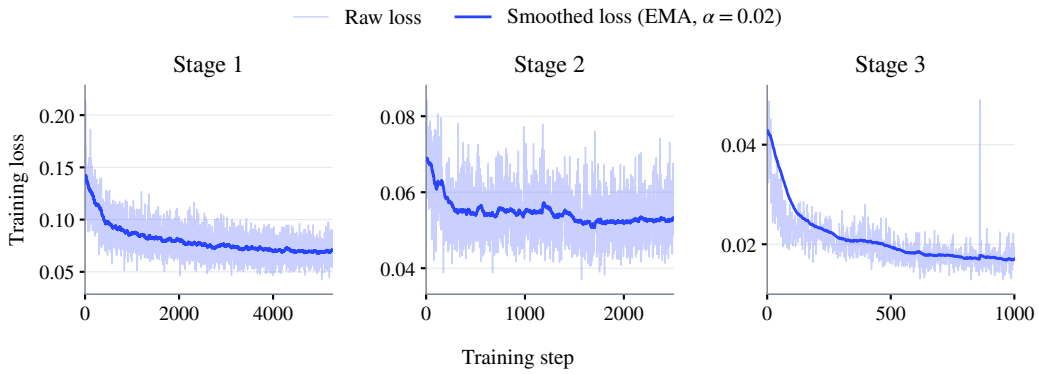

Figure 15: Raw and smoothed training losses of VisionWeave on Qwen3.8-27B.

In Stage 2, training router entropy decreases from 0.658 to 0.427 nats (first and last 100-step means), indicating more decisive granularity allocation (Figure 16). From step 50 to 2,500, the validation

share of blocks with $p_{IJ} \in [0.4, 0.6)$ falls from 64.8% to 14.6%, while the share with $p_{IJ} \geq 0.9$ rises to 28.3%, reflecting a shift away from ambiguous mixtures toward confident coarse-grained predictions.

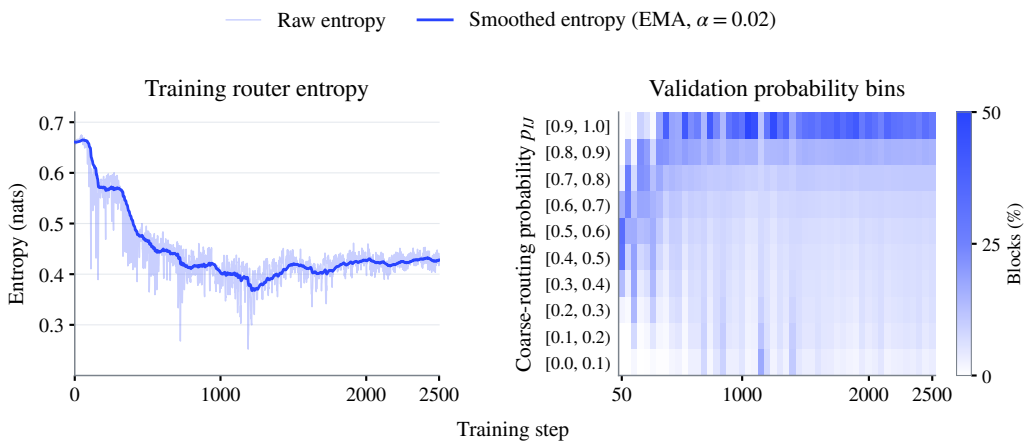

Figure 16: Stage 2 router entropy and validation distributions of coarse-routing probabilities.

E Additional Experimental Details

E.1 Ablation of Distillation Policy

We compare on- and off-policy self-distillation on Qwen3.5-4B (Figure 17) in Stage 3. The two variants achieve similar mean scores across eight benchmarks, with each leading on four benchmarks. These results support our choice of off-policy self-distillation for training efficiency.

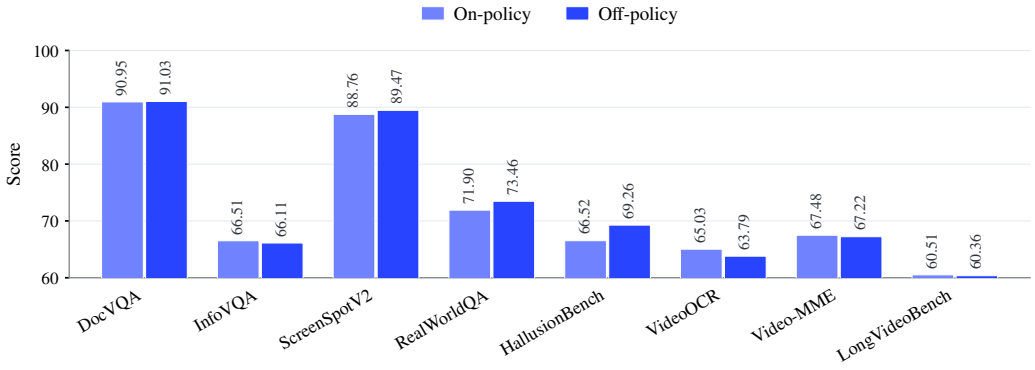

Figure 17: On- and off-policy self-distillation yield comparable performance on Qwen3.5-4B.

E.2 Ablation of Stage3: Mixed-Granularity Self-Distillation

We compare Qwen3.8-27B before and after Stage 3 on the eight main-text benchmarks (Figure 18). Stage 3 improves seven of eight scores and raises the average from 73.15 to 76.73. The largest gain is on ScreenSpotV2 (66.59 to 91.43); excluding this benchmark, the average still improves by 0.54 points, despite a 1.18-point decrease on RealWorldQA. These results support the value of self-distillation on the hard-routed mixed-granularity sequences used at inference.

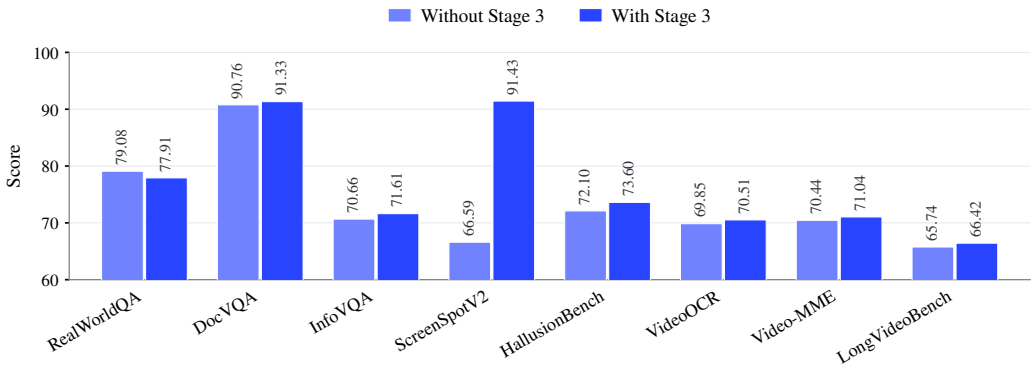

Figure 18: Stage 3 improves Qwen3.8-27B performance on seven of eight benchmarks.

E.3 Ablation of Learned Granularity Router

We shuffle the router's block-wise scores within each image or video before applying the default threshold $t = 0.5$. The permutation preserves the number of coarse-grained selections for each input while breaking their alignment with visual content. It should be noted that, under this ablation, the router's ability to produce dynamically varying savings ratios is still preserved, making it still a strong baseline. We use the same 512-token image/frame input budgets as the main comparison. Shuffling reduces mean scores by 2.31 and 5.36 points on Qwen3.5-4B and Qwen3.8-27B, respectively (Table 6). Even excluding ScreenSpotV2, which shows the largest losses, the mean decreases remain 1.30 and 1.53 points. These results support the value of learning where to allocate fine- and coarse-grained representations.

Table 6: Learned granularity allocation improves performance over shuffled routing.

	Qwen3.5-4B			**Qwen3.8-27B**		
Benchmark	VisionWeave	Shuffled	Δ	VisionWeave	Shuffled	Δ
RealWorldQA	**73.46**	**73.46**	0.00	**77.91**	76.60	−1.31
DocVQA	**91.03**	88.70	−2.32	**91.33**	88.35	−2.98
InfoVQA	**66.11**	64.83	−1.28	**71.61**	69.04	−2.58
ScreenSpotV2	**89.47**	80.03	−9.43	**91.43**	59.28	−32.15
HallusionBench	**69.26**	67.32	−1.95	**73.60**	71.66	−1.95
VideoOCR	**63.79**	61.85	−1.95	**70.51**	69.44	−1.08
Video-MME	**67.22**	66.78	−0.44	**71.04**	70.67	−0.37
LongVideoBench	**60.36**	59.24	−1.12	**66.42**	65.97	−0.45
Average	**72.59**	70.28	−2.31	**76.73**	71.37	−5.36

E.4 Evaluation Details

Task metrics. All task scores are reported on a 0–100 scale, using ANLS for DocVQA and InfoVQA, aAcc for HallusionBench, and the task-specific metric for each remaining benchmark.

E.5 Additional Resolution Results

We supplement the main-text spatial-resolution study with InfoVQA, HallusionBench, Video-MME, and LongVideoBench results (Figures 19–22). All four sweeps use input budgets of 64, 128, 256, and 512 tokens per image or video frame. The video sweeps fix the frame cap at 64 and sampling rate at 2 FPS, varying only the spatial input budget rather than repeating the main-text frame-budget study. At 512 tokens per image on InfoVQA, VisionWeave saves 10.6%/16.5% of tokens

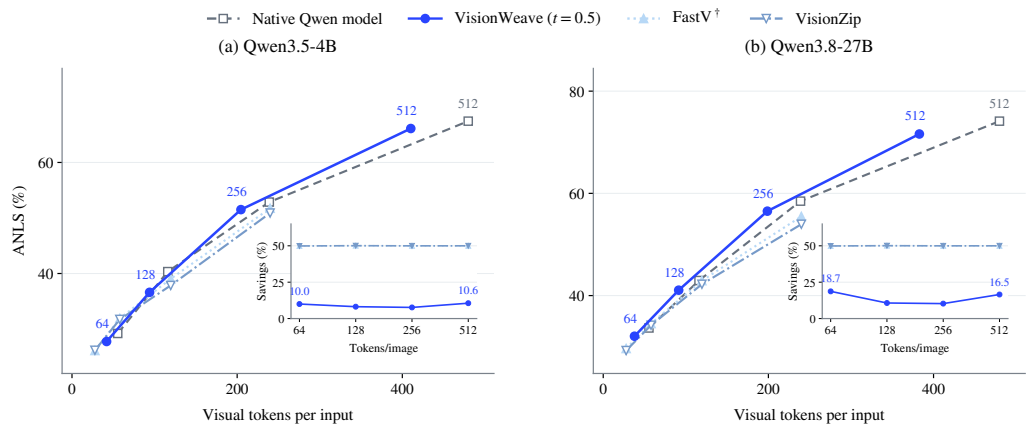

Figure 19: VisionWeave adapts token savings to input resolution on InfoVQA.

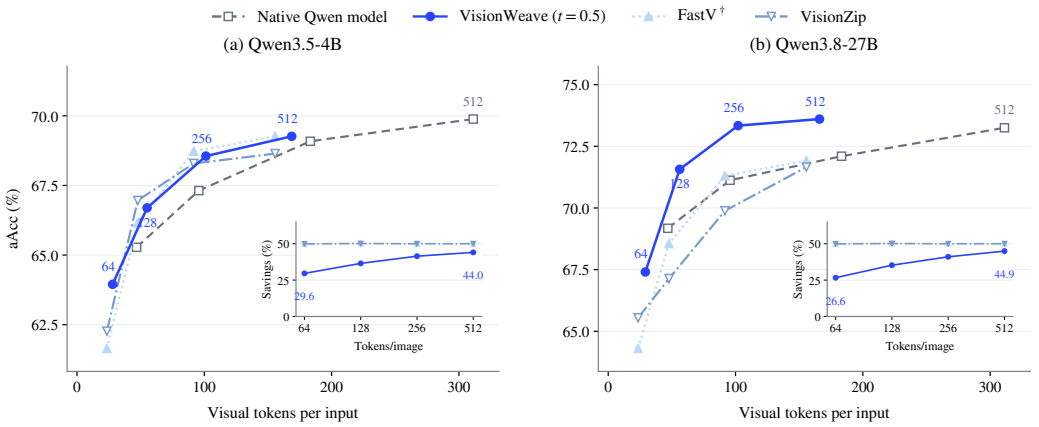

Figure 20: VisionWeave adapts token savings to input resolution on HallusionBench.

on Qwen3.5-4B/Qwen3.8-27B, achieving 66.11/71.61 ANLS compared with 67.44/74.13 for the corresponding native models. The additional datasets also exhibit increasing token savings with spatial resolution. On Qwen3.8-27B, savings on HallusionBench rise from 26.6% to 44.9% as the input budget increases from 64 to 512 tokens per image. Increasing the video budget from 64 to 512 tokens per frame raises savings from 44.2% to 54.2% on Video-MME and from 41.0% to 53.0% on LongVideoBench. At 512 tokens per frame, VisionWeave scores 71.04 and 66.42, respectively, compared with 70.89 and 66.49 for the native model.

E.6 Performance at Matched Token Savings

To test whether dataset-level token savings alone explain VisionWeave's advantage, we set FastV† and VisionZip to match its savings at $t = 0.5$ for each backbone and dataset. Among the compression methods, VisionWeave still achieves the highest average score on both backbones: 72.59 versus 72.20/71.90 for FastV†/VisionZip on Qwen3.5-4B, and 76.73 versus 70.10/70.33 on Qwen3.8-27B (Table 7). Rankings vary across tasks, and ScreenSpotV2 contributes substantially to the 27B gap; excluding it, the average scores are still 74.63 versus 73.86/73.85.

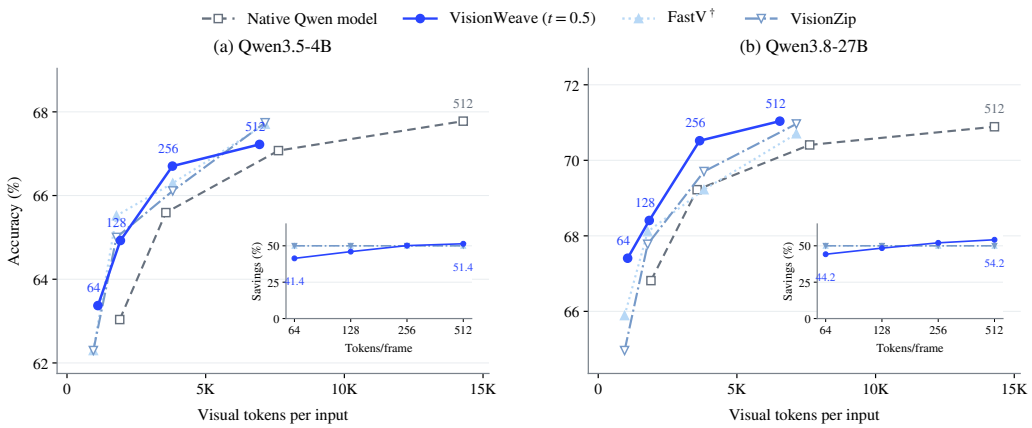

Figure 21: VisionWeave adapts token savings to spatial resolution on Video-MME.

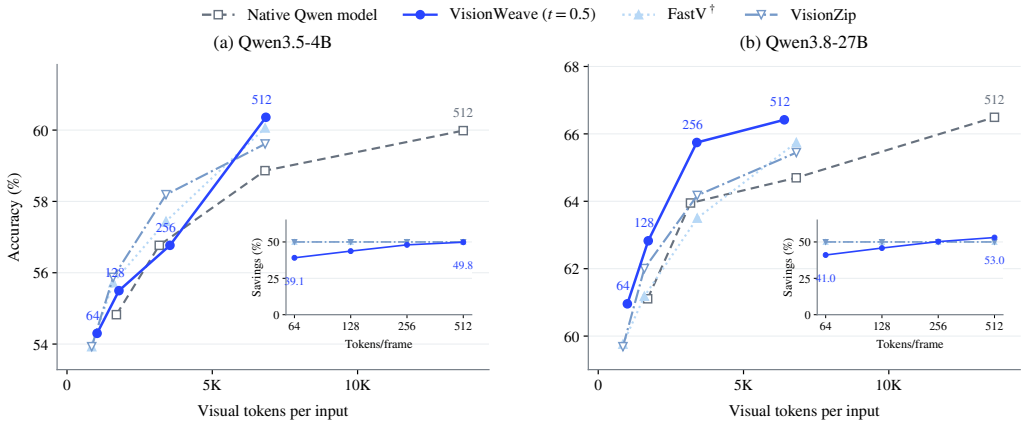

Figure 22: VisionWeave adapts token savings to spatial resolution on LongVideoBench.

E.7 SGLang Serving Configuration

The experiments in Section 3.4 use our SGLang integration, Python 3.12.3, PyTorch 2.13.0+cu129, Transformers 5.12.1, and Triton 3.7.1. Each model runs on two A100-SXM4-80GB GPUs with LLM tensor parallelism of two and one TP=1 vision encoder per GPU, providing encoder data parallelism of two. Both models use BF16, Triton full/linear/ViT attention, and NCCL with custom all-reduce disabled. The context limit is 147,456 tokens, with 4,096-token prefill chunks. Static memory fractions are 0.55 for the language service and 0.06 for each encoder. FCFS and overlap scheduling are enabled; mixed prefill/decode chunks are disabled. Padded decode CUDA graphs capture batch sizes 1, 2 and 4; prefill graphs are disabled. Client concurrency is capped at four outstanding requests.

F Inference Algorithm

Algorithm 1 summarizes the complete inference procedure of VisionWeave. It constructs fine- and coarse-grained visual representations and uses block-wise routing to gather a mixed-granularity sequence for the LLM within the shared MRoPE frame.

Table 7: Performance at matched dataset-level token savings.

	Upper bound	Matched savings		
Qwen3.5-4B	512 tokens	FastV†	VisionZip	VisionWeave
RealWorldQA	74.64	71.76	72.42	**73.46**
DocVQA	91.92	**91.16**	91.09	91.03
InfoVQA	67.44	**66.66**	65.68	66.11
ScreenSpotV2	89.86	87.34	86.08	**89.47**
HallusionBench	69.88	69.09	**69.44**	69.26
VideoOCR	64.51	63.59	63.28	**63.79**
Video-MME	67.78	**67.56**	67.48	67.22
LongVideoBench	59.99	**60.43**	59.76	60.36
Average	73.25	72.20	71.90	**72.59**
Qwen3.8-27B				
RealWorldQA	78.82	78.17	**78.43**	77.91
DocVQA	92.47	88.26	88.51	**91.33**
InfoVQA	74.13	71.29	71.07	**71.61**
ScreenSpotV2	94.42	43.79	45.68	**91.43**
HallusionBench	73.25	72.54	72.54	**73.60**
VideoOCR	70.72	69.79	69.44	**70.51**
Video-MME	70.89	**71.15**	**71.15**	71.04
LongVideoBench	66.49	65.82	65.82	**66.42**
Average	77.65	70.10	70.33	**76.73**

Algorithm 1 VisionWeave inference with elastic visual representations

Require: An $H \times W$ patch grid for an image; textual context c; routing threshold t (default 0.5)

Obtain $X^{(0)}$, $X^{(L/2)}$, $X^{(L)}$ in a single ViT forward pass
$V_{\text{fine}} \leftarrow \mathcal{M}(X^{(L)})$; $V_{\text{coarse}} \leftarrow \mathcal{M}\big(\mathcal{P}(X^{(L)})\big)$ Eqs. (1), (2)
Initialize h_{IJ} from the shared query with block-center RoPE coordinates
for $\ell \in \{0,\ L//2,\ L\}$ **do**
    $h_{IJ} \leftarrow \text{LocalCrossAttn}^{(\ell)}\big(h_{IJ},\ X^{(\ell)}_{IJ}\big)$
    $h_{IJ} \leftarrow \text{GlobalCrossAttn}^{(\ell)}\big(h_{IJ},\ X^{(\ell)}\big)$ Eq. (8)
end for
Compute p_{IJ} using the calibrated router head
for each block (I, J) **do**
    if $p_{IJ} > t$ **then**
        Select $V_{\text{coarse},IJ}$ with its block-center coordinate π_{IJ} from Eq. (3)
    else
        Select $\{V_{\text{fine},2I+a,2J+b}\}_{a,b\in\{0,1\}}$ with their native coordinates from Section 2.1
    end if
end for
Gather selected tokens into V_{routed} in native raster order
return $p_\theta\big(y \mid c,\ V_{\text{routed}}\big)$

G Limitations and Future Works

Our ablations examine on- versus off-policy self-distillation, the contribution of Stage 3, and the pooler architecture. However, the substantial computational cost of training limits a more comprehensive exploration of architectural choices, including alternative router designs. These unexplored choices offer opportunities to further improve elastic visual weaving. This work focuses on two granularities, while we do not foresee significant barriers to extending our approach to diverse levels. We therefore expect that moving beyond the current two-level design to richer multi-scale representations could yield further gains. Moreover, whereas we currently implant this capability via post-training self-distillation, future work could explore introducing elastic visual weaving during base model pre-training. Our long-term vision is a new generation of multimodal foundation

models that jointly develop visual understanding and efficiency, natively modulating computational effort by information content.