COMPUTE ONCE, UPDATE MANY: SHARING COMPUTATION FOR EFFICIENT VIDEO-LLM UNDERSTANDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Video-language models spend substantial computation repeatedly processing visual tokens during language decoder prefill. Existing acceleration methods often reduce this cost by removing or merging tokens, but retaining a token's individual representation need not require computing its update independently. We introduce a carrier–receiver computation sharing method that reduces repeated computation while updating each visual token's own hidden state. Selected visual carriers compute Attention and feed-forward network (FFN) updates, which receivers add to their own residual states. All token positions and key/value entries are retained, and non-visual tokens are processed normally. Input-dependent assignments identify which tokens share updates, while calibrated layer-wise ratios control the amount of sharing. Across three model families, our method maintains accuracy close to dense inference on MVBench, VideoMME, and LongVideoBench, while delivering measured decoder prefill speedups of 1.38–$1.51\times$ at 32 frames and consistent acceleration from 8 to 64 frames. Beyond standalone acceleration, our method consistently outperforms further token reduction alone when combined with four visual token reduction methods, improving accuracy by up to 10.13 percentage points at matched decoder prefill FLOPs across the three benchmarks. These results show that sharing updates offers an effective way to reduce decoder computation while keeping visual tokens available for subsequent attention.

1 INTRODUCTION

Video-language models encode video frames into visual tokens that an autoregressive language decoder processes alongside text (Bai et al., 2025; Wang et al., 2024; Li et al., 2025a). During prefill, the decoder repeatedly applies causal Attention and feed-forward networks (FFNs) to every visual token. This incurs substantial computation despite spatial and temporal redundancy, suggesting that not every token needs an independently computed update at every layer.

Most acceleration methods shorten the visual sequence through pruning, compression, or cross-frame reduction (Chen et al., 2024; Xing et al., 2025; Chen et al., 2026; Li et al., 2025b; Yang et al., 2025; Fu et al., 2025b), or merge redundant states (Bolya et al., 2023; Cao et al., 2023). Others freeze or skip selected visual updates (Zeng et al., 2025; Yuan et al., 2025; Luo et al., 2026). Yet retaining a token's individual representation need not require computing its update independently: tokens carrying different information may require similar updates. Each token could therefore retain its own residual state while adding an update computed for another token. Can we reduce repeated decoder computation by sharing updates while keeping visual states separate? Figure 1 contrasts these alternatives.

We introduce carrier–receiver computation sharing for decoder prefill. Selected visual *carriers* compute operator updates, and *receivers* add assigned carrier updates to their own residual states. Attention retains every token's key/value entry while computing visual queries and outputs only for carriers; the FFN likewise evaluates only visual carriers. Non-visual tokens remain dense, and the visual encoder and adaptor are unchanged (Figure 2).

Input features determine separate carrier assignments for Attention and FFN, with positional and causal constraints for Attention. Assignments are reused across layers with the same carrier ratio, while each layer computes fresh updates. A small held-out calibration set determines layer-wise sharing ratios, which remain fixed during evaluation.

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.

Across Qwen3-VL-8B, Qwen2-VL-7B, and LLaVA-OneVision-7B, our method maintains accuracy close to dense inference on the complete MVBench and VideoMME benchmarks and the LongVideoBench validation split (Li et al., 2024; Fu et al., 2025a; Wu et al., 2024), while providing measured decoder prefill acceleration across frame counts. Beyond standalone acceleration, our method consistently outperforms further visual token reduction alone when combined with four representative reduction methods, with accuracy gains of up to 10.13 pp at matched decoder prefill FLOPs across the three benchmarks.

Our contributions are threefold:

- We introduce carrier–receiver sharing of Attention and FFN updates for efficient video-language decoder prefill, retaining separate visual residual states and key/value entries.
- We show that sharing complements four visual token reduction methods, achieving higher accuracy than further token reduction alone at matched decoder prefill FLOPs.
- We demonstrate accuracy close to dense inference across three model families and three video benchmarks, together with measured decoder prefill speedups of 1.38–1.51× at 32 frames and consistent acceleration from 8 to 64 frames.

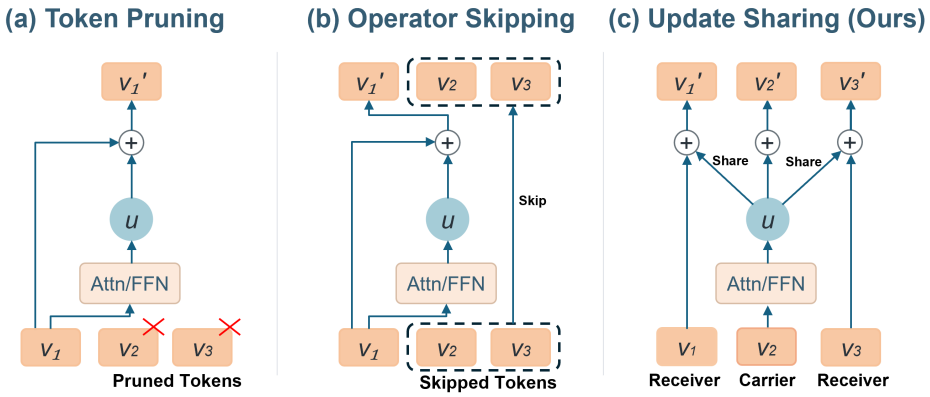

Figure 1: **Comparison of decoder acceleration strategies.** Token pruning removes selected visual states, while operator skipping retains their residual states but omits selected operator updates. Our method retains every token state and reuses nonzero updates from computed carrier tokens for assigned receiver tokens; non-visual tokens remain dense.

2 Related Work

Visual token reduction. Reducing the visual sequence is a common route to efficient vision-language inference. DynamicViT and TokenLearner establish input-adaptive token reduction in vision models (Rao et al., 2021; Ryoo et al., 2021); FastV, PyramidDrop, and V2Drop extend this idea to LVLMs by pruning visual tokens inside the decoder (Chen et al., 2024; Xing et al., 2025; Chen et al., 2026). TokenPacker and VisionZip instead compress visual features before language model processing, while ToMe, PuMer, and FrameFusion exploit redundancy through token merging and cross-frame compression (Li et al., 2025b; Yang et al., 2025; Bolya et al., 2023; Cao et al., 2023; Fu et al., 2025b). iLLaVA and VidCom^2 further combine visual token reduction with model-specific or frame-adaptive computation (Hu et al., 2026; Liu et al., 2025). These methods reduce the number of visual states processed by later computation. Our method takes a complementary route: it retains the full visual sequence and each token's residual state, but reduces how many tokens independently evaluate an operator, making it directly composable with visual token reduction.

Selective visual computation. Another line of work selectively omits visual token updates. SkipVision bypasses visual FFNs and removes skipped key/value entries during inference, ShortV freezes visual tokens in ineffective layers, and Op-Skip selectively bypasses Attention or FFN while retaining the full visual sequence (Zeng et al., 2025; Yuan et al., 2025; Luo et al., 2026). EarlyTom instead uses early layer signals for encoder-side video token compression (Wang et al., 2026). These methods

show that visual computation is layer and operator dependent. Our method builds on this insight by replacing the omitted computation with a useful update: visual carrier tokens execute the original operator and provide outputs to receiver tokens, which add the update to their own residual states.

Sharing computation across tokens. Clustered Attention reuses outputs computed from query centroids (Vyas et al., 2020), while ToMeSD temporarily merges tokens and restores the outputs before residual addition (Bolya & Hoffman, 2023). These methods demonstrate that tokens need not be processed independently to retain distinct residual states. Our method computes reusable updates from selected visual carrier tokens rather than averaged representations. Applied to both Attention and FFN during video-language decoder prefill, this reduces computation while keeping every visual token available to subsequent attention.

3 METHOD

Our method addresses three questions: how to share updates while retaining separate token states, how to select carriers and assign receivers, and how much sharing to apply at each decoder layer.

3.1 COMPUTATION SHARING MECHANISM

Our method operates only during decoder prefill (Figure 2). The input contains N tokens: V visual tokens and $N - V$ non-visual tokens, including text and special tokens. Let $\mathbf{h}_{\ell,i}$ be token i's state at decoder layer ℓ. A standard block applies Attention and FFN residual branches,

$$\widetilde{\mathbf{h}}_{\ell,i} = \mathbf{h}_{\ell,i} + \mathbf{u}^{A}_{\ell,i}, \qquad \mathbf{h}_{\ell+1,i} = \widetilde{\mathbf{h}}_{\ell,i} + \mathbf{u}^{F}_{\ell,i}, \tag{1}$$

where $\mathbf{u}^{A}_{\ell,i}$ and $\mathbf{u}^{F}_{\ell,i}$ are updates computed independently for every token under dense inference.

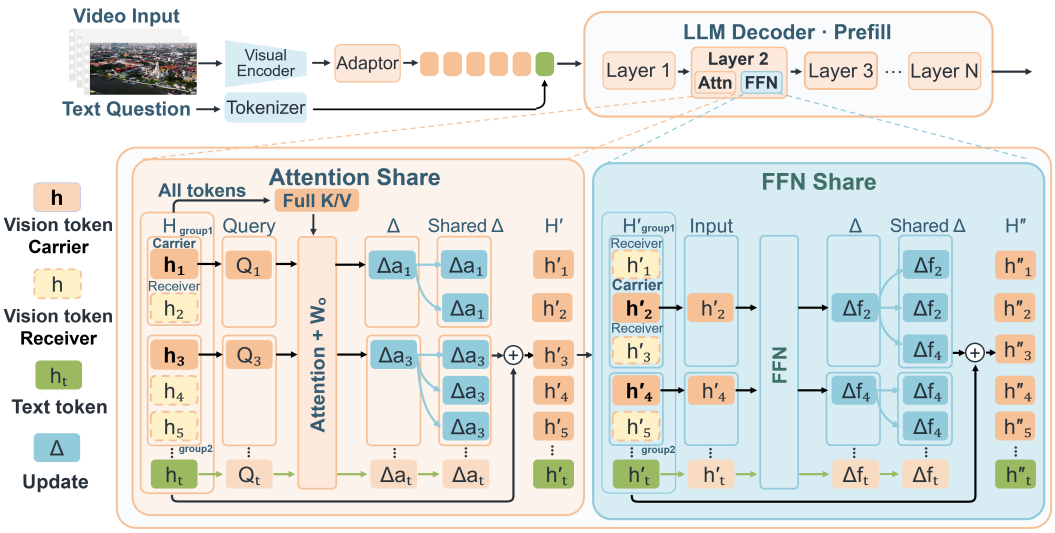

Figure 2: **Our method during language decoder prefill.** The visual encoder and adaptor remain unchanged, and non-visual tokens remain dense. Selected visual carriers evaluate the original Attention or FFN operator; visual receivers retain their own residual states and sequence positions while reusing the assigned carrier update. Attention retains all key/value entries.

For one Attention or FFN branch, let $\mathcal{V}$ and $\mathcal{T}$ denote visual and non-visual tokens. A carrier set $\mathcal{C} \subseteq \mathcal{V}$ and assignment $\pi : \mathcal{V} \to \mathcal{C}$ satisfy $\pi(c) = c$ for each carrier. With states $\mathbf{h}_i$ and $\mathbf{h}'_i$ before and after the branch and original operator update $\mathbf{u}_i$, we apply

$$\mathbf{h}'_i = \begin{cases} \mathbf{h}_i + \mathbf{u}_i, & i \in \mathcal{C} \quad \text{(carrier)}, \\ \mathbf{h}_i + \mathbf{u}_{\pi(i)}, & i \in \mathcal{V} \setminus \mathcal{C} \quad \text{(receiver)}, \\ \mathbf{h}_i + \mathbf{u}_i, & i \in \mathcal{T} \quad \text{(non-visual token)}. \end{cases} \tag{2}$$

Carriers and non-visual tokens therefore receive updates computed from their own states, whereas each receiver reuses the update computed for its assigned carrier. Attention and FFN use separate carrier sets and assignments. Receivers assigned to the same carrier share its update while retaining distinct residual states.

Attention. Our method computes queries for all non-visual tokens and the selected visual carriers, while computing keys and values for every token. Each computed query uses its original position and causal mask. Receivers reuse their assigned carrier's Attention output after the output projection. Their own states remain available as keys and values, without computing individual queries.

FFN. Our method evaluates the original gated FFN on all non-visual tokens and the selected visual carriers. Each receiver adds its carrier's FFN output to its own post-Attention state. This follows the same sharing rule as Attention, but uses a separate carrier assignment.

3.2 Carrier and Receiver Assignment

For operator $o \in \{A, F\}$ and visual carrier ratio r, let $\mathcal{C}_r^o \subseteq \mathcal{V}$ and $\pi_r^o : \mathcal{V} \to \mathcal{C}_r^o$ denote the carrier set and assignment. Selection uses input features projected into a fixed low-dimensional space, centered, and normalized as $\mathbf{z}_{r,i}^o$. These features are used only for selection and assignment; operators evaluate current full-dimensional carrier states. The carrier ratio specifies the fraction of visual tokens selected to compute an operator update.

Attention carrier selection and receiver assignment. Similar content alone is insufficient for sharing Attention updates, because each update also depends on the token's position and visible context. We therefore select carriers within each frame to maintain coverage across the video. Each frame is divided into groups along its spatial token order, and the token closest to each group's normalized mean feature is selected as its carrier. We always include the first visual token as a carrier, ensuring that every visual receiver has at least one carrier available. Let $\boldsymbol{\phi}_i$ denote the normalized positional feature of token i, and let $\mathcal{E}_r(i)$ contain candidate carriers at the same or an earlier sequence position. Each receiver is assigned using both feature and positional similarity:

$$\pi_r^A(i) = \arg\min_{c \in \mathcal{E}_r(i)} \left[1 - (\mathbf{z}_{r,i}^A)^\top \mathbf{z}_{r,c}^A + \lambda_p \big(1 - \boldsymbol{\phi}_i^\top \boldsymbol{\phi}_c\big)\right]. \tag{3}$$

Here $\lambda_p \geq 0$ controls the positional term. The nonfuture candidate set preserves causality and is restricted to a small window of preceding carriers to limit assignment cost.

FFN carrier selection and receiver assignment. The FFN processes each token independently, so its selection and assignment procedure uses feature similarity without the positional constraints used for Attention. We cluster the visual token features and select an actual token nearest to each cluster center as a carrier. Each remaining token is assigned to its nearest carrier:

$$\pi_r^F(i) = \arg\min_{c \in \mathcal{C}_r^F} \left\| \mathbf{z}_{r,i}^F - \mathbf{z}_{r,c}^F \right\|_2^2. \tag{4}$$

Reusing assignments across layers. For each operator and ratio, we construct carrier indices and receiver assignments at the first layer using that ratio and reuse them at subsequent layers to limit overhead. Every layer computes fresh updates from its current carrier states. Appendix A details selection and assignment; Appendix B.4 evaluates reuse.

3.3 Layer-wise Sharing Ratios

We calibrate the Attention and FFN carrier ratios across decoder layers on a small labeled video set disjoint from all evaluations, keeping the pretrained model frozen. Calibration measures how much sharing changes the negative log-likelihood of the correct answer relative to dense inference. A smaller increase indicates that the tested layers can tolerate more sharing.

We divide the decoder into contiguous groups of layers and evaluate several carrier ratios for each group. Attention sharing, FFN sharing, and their joint application are tested to account for their combined effect. These measurements are used to estimate the loss increase for candidate allocations.

Dynamic programming then selects the Attention and FFN ratios for each group to minimize the estimated loss increase at a prescribed overall sharing level. The selected ratios are fixed before evaluation; carrier selection and receiver assignment remain input dependent.

For a decoder with L layers, we define the normalized sharing level as

$$S = 1 - \frac{1}{2L} \sum_{\ell=1}^{L} \left(r_\ell^A + r_\ell^F \right), \tag{5}$$

where r_ℓ^A and r_ℓ^F are the visual carrier ratios for Attention and FFN at layer ℓ. Thus, S measures the average fraction of visual token operator evaluations replaced by shared updates. It does not directly specify the FLOPs reduction, because Attention and FFN have different computational costs and key/value computation is retained. Appendix A.4 describes the calibration and allocation procedure.

3.4 Computational Cost

Sharing reduces visual query/output projections, Attention products, and FFN evaluations while retaining all key/value projections. We report analytical decoder prefill FLOPs using per-layer carrier counts (Appendix A.5). Measured latency additionally includes selection, assignment, gather/scatter, and update copying.

4 Experiments

4.1 Experimental setup

Models. We evaluate our method on Qwen3-VL-8B-Instruct, Qwen2-VL-7B-Instruct, and LLaVA-OneVision-7B, covering three multimodal decoder architectures (Bai et al., 2025; Wang et al., 2024; Li et al., 2025a).

Benchmarks. We evaluate the complete MVBench and VideoMME sets, containing 4,000 and 2,700 multiple-choice questions, respectively, and all 1,337 questions in the LongVideoBench validation split (Li et al., 2024; Fu et al., 2025a; Wu et al., 2024). LongVideoBench is evaluated in the video-only setting without subtitles. We report question-level accuracy on all three benchmarks.

Baselines. We compare our method with dense inference and four visual token reduction methods: VisionZip (Yang et al., 2025), VidCom^2 (Liu et al., 2025), iLLaVA (Hu et al., 2026), and an encoder-side port of EarlyTom to Qwen3-VL (Wang et al., 2026). We call the unmodified token reduction pipeline the *base method*. For each base method, we construct a control that uses further visual token reduction alone by continuing the same reduction rule until its analytical decoder prefill FLOPs match those of the base method combined with our method.

Implementation details. Accuracy experiments use 32 uniformly sampled frames; the separate latency scaling study uses 8, 16, 32, and 64 frames. All models run with batch size one in BF16. Unless otherwise stated, our method uses a 32-dimensional fixed signed projection, one deterministic k-means iteration for FFN carrier selection, an Attention candidate window of $M = 16$, and position weight $\lambda_p = 1$. For each input and operator, one carrier–receiver assignment map is constructed for every distinct carrier ratio and reused by all layers assigned that ratio. We use model-specific schedules calibrated on a labeled subset of VideoChat2-IT with a normalized sharing level of $S = 0.5$; the calibration and evaluation videos are disjoint. We report analytical decoder prefill FLOPs, counting one multiplication and one addition as two floating-point operations. This count covers the dominant matrix multiplications under the same full attention matrix convention for all methods and excludes selection and assignment overhead. Compute percentages are relative to dense decoder prefill, and accuracy differences are reported in percentage points (pp). Measured decoder prefill latency includes selection and assignment overhead.

4.2 Main results

Table 1 reports the standalone results of our method under the calibrated $S = 0.5$ schedules across the three model families and video benchmarks.

Table 1: **Standalone accuracy and decoder prefill FLOPs.** Q3, Q2, and L-OV denote Qwen3-VL-8B, Qwen2-VL-7B, and LLaVA-OneVision-7B; MV, VM, and LVB denote MVBench, VideoMME, and LongVideoBench. FLOPs are analytical percentages of dense prefill.

(a) Accuracy (%)

	MVBench		VideoMME		LongVideoBench	
Model	Dense	Ours	Dense	Ours	Dense	Ours
Q3	66.60	66.63	62.48	61.81	59.31	59.16
Q2	65.03	64.90	56.11	55.44	53.78	52.51
L-OV	55.50	55.35	56.15	55.56	54.30	55.12

(b) Decoder prefill FLOPs (% of dense)

Model	MV	VM	LVB
Q3	58.19	58.23	59.42
Q2	56.59	56.68	57.72
L-OV	50.03	50.11	50.60

Retained decoder prefill FLOPs vary across benchmarks because sequence lengths and integer carrier counts differ across inputs.

4.3 Combination with token reduction

Base and +Ours retain identical visual tokens. The token-only control further reduces tokens with the same reduction rule to match +Ours decoder prefill FLOPs per input, using token geometry only.

Table 2: **Sharing versus further token reduction at matched decoder prefill FLOPs.** Base and +Ours retain identical tokens. Token-only further reduces tokens without sharing to match +Ours FLOPs. Δ Acc. is +Ours minus Token-only; retention is the mean fraction retained.

Model	Data	Tokens ret. (%)	Base (%)	+Ours (%)	Token-only (%)	Δ Acc. (pp)	+Ours FLOPs (% dense)
VisionZip (CVPR 2025) (Yang et al., 2025)							
Qwen3-VL-8B	MV	10.00	57.70	57.73	50.45	+7.28	14.81
Qwen3-VL-8B	VM	9.83	52.81	51.81	49.59	+2.22	14.78
Qwen3-VL-8B	LVB	9.83	51.53	51.08	49.89	+1.20	18.41
Qwen2-VL-7B	MV	9.84	59.98	59.53	54.93	+4.60	8.76
Qwen2-VL-7B	VM	9.73	49.52	48.33	46.33	+2.00	8.88
Qwen2-VL-7B	LVB	9.73	48.17	47.49	46.75	+0.75	11.87
LLaVA-OneVision-7B	MV	10.20	50.33	50.05	39.93	+10.13	5.30
LLaVA-OneVision-7B	VM	10.20	48.96	48.52	42.19	+6.33	5.42
LLaVA-OneVision-7B	LVB	10.20	46.82	46.90	42.41	+4.49	7.46
VidCom^2 (EMNLP 2025) (Liu et al., 2025)							
Qwen3-VL-8B	MV	25.03	63.80	63.60	58.28	+5.33	21.71
Qwen3-VL-8B	VM	24.82	58.89	57.70	55.33	+2.37	21.67
Qwen3-VL-8B	LVB	24.84	57.74	57.14	54.00	+3.14	23.77
Qwen3-VL-8B	MV	10.05	55.33	55.35	50.53	+4.83	14.83
Qwen3-VL-8B	VM	9.78	54.41	53.70	50.85	+2.85	14.76
Qwen3-VL-8B	LVB	9.83	51.16	51.08	49.89	+1.20	17.05
iLLaVA (ICLR 2026) (Hu et al., 2026)							
Qwen3-VL-8B	MV	31.32	66.13	65.90	62.93	+2.98	45.85
Qwen3-VL-8B	VM	30.68	60.93	60.30	56.48	+3.81	45.13
Qwen3-VL-8B	LVB	30.40	58.12	58.26	56.99	+1.27	46.29
EarlyTom-Q3 (CVPR 2026) (Wang et al., 2026)							
Qwen3-VL-8B	MV	20.00	62.23	61.38	60.00	+1.38	19.38
Qwen3-VL-8B	VM	20.13	55.85	55.85	54.00	+1.85	19.50
Qwen3-VL-8B	LVB	19.72	54.45	53.85	52.80	+1.05	21.47

EarlyTom-Q3 is an encoder-side adaptation of the released EarlyTom method for Qwen3-VL.

At matched analytical decoder prefill FLOPs, +Ours outperforms further token reduction alone in all 21 comparisons, by 0.75–10.13 pp. The gains span all four methods and three benchmarks, supporting the complementarity of token reduction and update sharing. Sixteen comparisons remain

significant after Holm correction across the 21 configurations; the five nonsignificant comparisons are on LongVideoBench. Appendix C reports the paired analysis.

4.4 Accuracy–Efficiency Trade-off

Table 3 evaluates Qwen3-VL-8B at different sharing levels on the complete MVBench and VideoMME evaluation sets. Increasing S from 0.25 to 0.50 reduces retained decoder prefill FLOPs from approximately 78% to 58%, while accuracy remains close to dense inference. Increasing S to 0.625 further reduces computation to approximately 46% of dense decoder prefill FLOPs, with some reduction in accuracy on both benchmarks. We use $S = 0.5$ as the default sharing level in our experiments.

Table 3: **Effect of the sharing level on Qwen3-VL-8B.** Accuracy is reported on the complete evaluation sets. FLOPs are percentages of dense decoder prefill.

	Accuracy (%)		FLOPs (% dense)	
S	MVBench	VideoMME	MVBench	VideoMME
0.000	66.60	62.48	100	100
0.250	66.63	62.37	77.76	77.78
0.375	66.23	62.26	67.97	68.00
0.500	66.63	61.81	58.19	58.23
0.625	65.15	59.19	45.73	45.78

4.5 Measured Decoder Prefill Acceleration

We measure a GPU-synchronized decoder prefill interval on an RTX 3090 with batch size one, BF16 precision, and PyTorch scaled dot-product attention (SDPA). The timer includes projection, carrier selection, receiver assignment, gather/scatter, and update copying operations, and excludes video decoding, the vision encoder, tokenization, host-to-device transfer, and subsequent autoregressive decoding. Each point pools 50 fixed MVBench and 50 fixed VideoMME examples. Before measurement, we perform a warm-up pass for each input shape and rotate method order across examples.

Table 4: **Measured decoder prefill acceleration at 32 frames.** Panel (a) compares our standalone method with dense inference. Panel (b) evaluates our method on three base token reduction methods; VisionZip and VidCom^2 retain 10% and 25% of visual tokens, respectively. For panel (b), visual token reduction preparation is outside the timer, while decoder computation, carrier selection, receiver assignment, the language model head, and selection of the first generated token are included.

(a) Standalone Ours

Model	Dense (ms)	Ours (ms)	Speedup
Q3	532.7	385.7	1.381×
Q2	566.3	394.9	1.434×
L-OV	1779.1	1180.9	1.506×

(b) Base method + Ours

Method	Base (ms)	+Ours (ms)	Speedup
VisionZip	135.3	123.9	1.092×
VidCom^2	222.7	173.7	1.282×
iLLaVA	468.2	410.1	1.142×

Our standalone method provides 1.38–1.51× speedup across the three model families. At 32 frames, our method provides 1.09–1.28× speedup over the corresponding base token reduction method for VisionZip retaining 10% of visual tokens, VidCom^2 retaining 25%, and iLLaVA.

Frame Count Scaling. We repeat the same GPU-synchronized protocol at 8, 16, 32, and 64 frames, with one unmeasured warm-up pass per input shape. Figure 3 shows 1.21–1.51× speedup across all tested models and frame counts. Appendix B.5 reports the median latencies and visual token counts for the 100 fixed inputs (50 per benchmark).

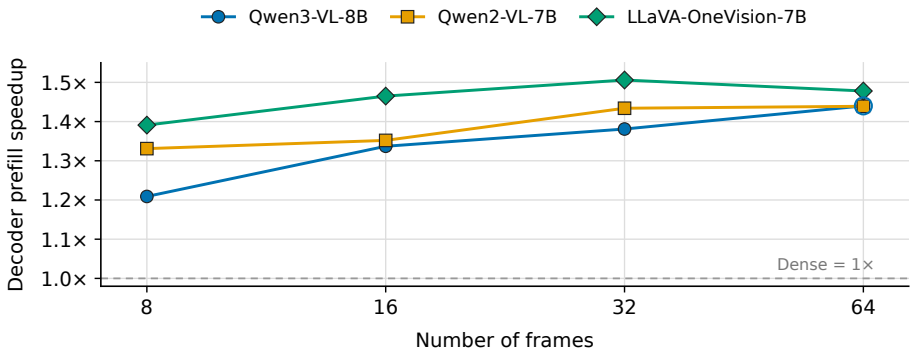

Figure 3: **Decoder prefill speedup across frame counts.** Speedup is dense median latency divided by our method's median latency over 100 fixed inputs. The dashed line marks 1× speedup.

4.6 Ablation Studies

For 500-example ablations, we randomly sample equally across MVBench tasks and stratify VideoMME sampling by video duration. All variants within a comparison use the same subset.

Contributions of Attention and FFN. Table 5 evaluates our method on Attention only, FFN only, and both operators on fixed stratified random subsets of 500 examples per benchmark. Sharing both operators combines their computational savings with almost no change in accuracy relative to the better single-operator variant on either subset, supporting the joint use of Attention and FFN sharing.

Table 5: **Attention and FFN component ablation.** Results use fixed stratified random subsets of 500 examples per benchmark.

	MVBench		VideoMME	
Variant	Decoder prefill FLOPs (% dense)	Acc. ↑	Decoder prefill FLOPs (% dense)	Acc. ↑
Dense	100.00	66.60	100.00	60.20
Attention only	88.92	66.00	88.94	59.00
FFN only	69.26	66.80	69.30	59.60
Attention + FFN	**58.18**	66.80	**58.23**	59.40

Calibrated depth allocation. Uniform depth allocation uses the same carrier ratio at every decoder layer for each operator. With total Attention and FFN sharing levels matched and nearly identical decoder prefill FLOPs, calibrated depth allocation improves accuracy over this uniform setting by 4.10 pp on MVBench and 3.41 pp on VideoMME (Table 6).

Table 6: **Calibrated versus uniform depth allocation.** Results on the complete evaluation sets for Qwen3-VL-8B at matched Attention and FFN sharing levels. Δ is calibrated minus uniform; FLOPs are listed as calibrated/uniform.

Dataset	Calibrated (%)	Uniform (%)	Δ (pp)	Decoder prefill FLOPs (% dense)
MVBench	66.63	62.53	+4.10	58.19/58.18
VideoMME	61.81	58.41	+3.41	58.23/58.22

Shared update versus zero update. We replay the same carrier–receiver groups while zeroing receiver updates, preserving carrier and non-visual computation, residual states, and all key/value entries. Across four complete Qwen2-VL and Qwen3-VL evaluations, accuracy drops by 28.13–36.78 pp (Appendix B.2). Appendix B.3 isolates sublayers with a 0.125 carrier ratio.

Reusing assignments across layers. Reusing carrier–receiver assignments reduces decoder prefill latency from 441.42 to 389.62 ms compared with recomputing assignments at every shared layer, with no statistically detectable accuracy difference on the two 500-example subsets. Appendix B.4 reports the paired confidence intervals and timing protocol.

Calibration set size. Our main Qwen3-VL experiments use 24 calibration videos. Appendix B.1 reports results with different calibration set sizes.

5 Conclusion

We introduced carrier–receiver computation sharing for video-language decoder prefill. Visual carriers compute Attention and FFN updates that receivers add to their own hidden states. This separates the decision to retain a visual token from the decision to compute its update independently.

Across three model families and three video benchmarks, our method maintains accuracy close to dense inference, with measured acceleration from 8 to 64 frames. Combined with four token reduction methods, it consistently outperforms further token reduction alone at matched decoder prefill FLOPs. Sharing updates thus offers a complementary way to use a limited computation budget. Extending the approach beyond decoder prefill to autoregressive decoding and broader deployment settings remains future work.

Ethics Statement

This work studies computational efficiency in video-language models using existing research datasets. Our experiments do not involve recruiting human participants or collecting new personal data. The proposed method reduces inference computation but does not address biases, privacy concerns, or potentially harmful outputs inherited from the underlying models and datasets. Applications should respect dataset licenses and applicable privacy requirements, and retain appropriate safeguards for the intended use.

AI Use Statement

Generative AI tools were used for language polishing and LaTeX formatting. The authors take responsibility for the final text, claims, and presentation.

Reproducibility Statement

Section 3 defines the update-sharing mechanism and calibration procedure. Section 4.1 specifies models, benchmarks, and implementation settings, and Section 4.5 describes latency measurements. Appendix A details selection, assignment, calibration, and FLOP accounting; Appendices B and C provide additional ablations, timing protocols, and paired statistical analyses. We plan to release our code and experimental configurations to facilitate reproduction of the reported results.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025. URL https://arxiv.org/abs/2511.21631.

Daniel Bolya and Judy Hoffman. Token merging for fast Stable Diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 4599–4603, 2023. URL https://arxiv.org/abs/2303.17604.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=JroZRaRw7Eu.

Qingqing Cao, Bhargavi Paranjape, and Hannaneh Hajishirzi. PuMer: Pruning and merging tokens for efficient vision language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12890–12903, 2023. doi: 10.18653/v1/2023.acl-long.721. URL https://aclanthology.org/2023.acl-long.721/.

Junjie Chen, Xuyang Liu, Zichen Wen, Yiyu Wang, Siteng Huang, and Honggang Chen. Variation-aware vision token dropping for faster large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3489–3499, 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 (ECCV)*, 2024. URL https://arxiv.org/abs/2403.06764.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, Peixian Chen, Yanwei Li, Shaohui Lin, Sirui Zhao, Ke Li, Tong Xu, Xiawu Zheng, Enhong Chen, Caifeng Shan, Ran He, and Xing Sun. Video-MME: The first-ever comprehensive evaluation benchmark of multi-modal LLMs in video analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24108–24118, 2025a. URL https://arxiv.org/abs/2405.21075.

Tianyu Fu, Tengxuan Liu, Qinghao Han, Guohao Dai, Shengen Yan, Huazhong Yang, Xuefei Ning, and Yu Wang. FrameFusion: Combining similarity and importance for video token reduction on large vision language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22654–22663, 2025b. URL https://arxiv.org/abs/2501.01986.

Lianyu Hu, Liqing Gao, Fanhua Shang, Liang Wan, and Wei Feng. iLLaVA: An image is worth fewer than 1/3 input tokens in large multimodal models. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2412.06263.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. LLaVA-OneVision: Easy visual task transfer. *Transactions on Machine Learning Research*, 2025a. URL https://arxiv.org/abs/2408.03326.

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, Limin Wang, and Yu Qiao. MVBench: A comprehensive multi-modal video understanding benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22195–22206, 2024. URL https://arxiv.org/abs/2311.17005.

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*, 133:6794–6812, 2025b. doi: 10.1007/s11263-025-02491-7. URL https://doi.org/10.1007/s11263-025-02491-7.

Xuyang Liu, Yiyu Wang, Junpeng Ma, and Linfeng Zhang. Video Compression Commander: Plug-and-play inference acceleration for video large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1910–1924, 2025. URL https://arxiv.org/abs/2505.14454.

Zhaoyang Luo, Runmin Dong, Miao Yang, Fan Wei, Yushan Lai, Bin Luo, and Haohuan Fu. Attend, transform, or silence: Operator-level visual skipping for efficient multimodal LLM inference. *arXiv preprint arXiv:2606.31903*, 2026. URL https://arxiv.org/abs/2606.31903.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021. URL https://arxiv.org/abs/2106.02034.

Michael S. Ryoo, A. J. Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. TokenLearner: Adaptive space-time tokenization for videos. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 12786–12797, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/6a30e32e56fce5cf381895dfe6ca7b6f-Abstract.html.

Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2007.04825.

Hesong Wang, Xin Jin, Lu Lu, Chenhaowen Li, Jian Chen, Qiang Liu, and Huan Wang. EarlyTom: Early token compression completes fast video understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 40559–40568, 2026. URL https://arxiv.org/abs/2605.30010.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024. URL https://arxiv.org/abs/2409.12191.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. LongVideoBench: A benchmark for long-context interleaved video-language understanding. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. doi: 10.52202/079017-0907. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/329ad516cf7a6ac306f29882e9c77558-Abstract-Datasets_and_Benchmarks_Track.html.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. NExT-QA: Next phase of question-answering to explaining temporal actions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9777–9786, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Xiao_NExT-QA_Next_Phase_of_Question-Answering_to_Explaining_Temporal_Actions_CVPR_2021_paper.html.

Long Xing, Qidong Huang, Xiaoyi Dong, Jiajie Lu, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, Jiaqi Wang, Feng Wu, and Dahua Lin. Conical visual concentration for efficient large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14593–14603, 2025. URL https://arxiv.org/abs/2410.17247.

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 (CVPR)*, pp. 19792–19802, 2025. URL https://arxiv.org/abs/2412.04467.

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 (ICCV)*, pp. 329–339, 2025. URL https://arxiv.org/abs/2504.00502.

Weili Zeng, Ziyuan Huang, Kaixiang Ji, and Yichao Yan. Skip-Vision: Efficient and scalable acceleration of vision-language models via adaptive token skipping. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 21384–21397, 2025. URL https://arxiv.org/abs/2503.21817.

A SELECTION, ASSIGNMENT, CALIBRATION, AND COMPUTATION DETAILS

A.1 FEATURES FOR SELECTION AND ASSIGNMENT REUSE

Sections 3.1–3.3 define the visual token set $\mathcal{V}$, carrier sets $\mathcal{C}_r^o$, assignments π_r^o, and layer-wise carrier ratios r_ℓ^o. Let d denote the hidden state width. For each operator and ratio, let $a^o(r) = \min\{\ell : r_\ell^o = r\}$ be the first layer using ratio r, and let $\mathbf{x}_{\ell,i}^o$ denote the corresponding operator input. Our method constructs $(\mathcal{C}_r^o, \pi_r^o)$ once at layer $a^o(r)$ and caches the assignment map for every later layer using the same ratio. The features used for selection and assignment are obtained through a fixed signed feature hashing projection. The sparse matrix $\mathbf{R}_r^o \in \mathbb{R}^{d_s \times d}$ maps each hidden channel to one output coordinate with a fixed sign, giving

$$\mathbf{z}_{r,i}^o = \mathrm{norm}\Big(\mathbf{R}_r^o \mathbf{x}_{a^o(r),i}^o - \mathrm{mean}(\mathbf{R}_r^o \mathbf{x}_{a^o(r),i}^o)\Big). \tag{6}$$

The mapping and signs are generated deterministically from the operator and ratio identifiers. These features are used only for carrier selection and assignment; original operator evaluations consume current full-dimensional carrier states. The integer carrier count rounds the relevant visual token count while retaining at least one carrier, and $r_\ell^o = 1$ recovers the dense branch.

A.2 ATTENTION CARRIER SELECTION AND RECEIVER ASSIGNMENT

Suppose the visual sequence represents T_v sampled frames with $P = H_p \times W_p$ spatial tokens per frame. For carrier ratio r, we retain

$$K_{r,\mathrm{frame}}^A = \max(1, \min(P, \mathrm{round}(rP))) \tag{7}$$

carriers in every frame. Within a frame, spatial index $p \in \{0, \ldots, P-1\}$ is assigned to stratum

$$g_r(p) = \left\lfloor \frac{pK_{r,\mathrm{frame}}^A}{P} \right\rfloor. \tag{8}$$

For each stratum, we average the projected features from the first layer using the corresponding ratio and select the observed token nearest to the normalized centroid by cosine distance, breaking ties by original order. This medoid construction, stratified by frame, gives every sampled frame carrier coverage while adapting carrier identity to the input. If the first visual token is not selected, it replaces a selected carrier, keeping the total carrier count unchanged and ensuring that every visual receiver has a nonfuture carrier available.

Receiver assignment follows Eq. (3). The positional feature $\boldsymbol{\phi}_i$ is the normalized concatenation of the cosine and sine components of the model's multimodal rotary position embedding at position i. The candidate set $\mathcal{E}_r(i)$ contains the M most recent carriers whose original positions do not exceed i, thereby preserving causality. At every layer that uses ratio r, Attention and the output projection are evaluated on the cached carrier layout using that layer's current states, and each receiver reuses the complete output of its assigned carrier.

A.3 FFN CARRIER SELECTION AND RECEIVER ASSIGNMENT

Because the FFN processes each token independently, its selection and assignment procedure does not require the temporal and causal constraints used by Attention. For carrier ratio r, we choose

$$K_r^F = \max(1, \min(V, \mathrm{round}(rV))) \tag{9}$$

visual carriers from the projected FFN input features at the first layer using ratio r, denoted by $a^F(r)$. A fixed number of deterministic k-means iterations produces centroids, which are mapped to their nearest observed tokens. Duplicate selections are replaced by tokens farthest from the current carrier set until K_r^F carriers are obtained. Receiver assignment follows Eq. (4). At every layer that uses ratio r, the original FFN is evaluated on the current carrier states, and each receiver adds its assigned carrier's output to its own residual state according to Eq. (2).

A.4 SCHEDULE CALIBRATION

Decoder layers are partitioned into contiguous relative depth bands $\{\mathcal{G}_b\}_{b=1}^{B}$, with carrier ratios shared within each band. For every band, we probe Attention sharing, FFN sharing, and their joint application at several carrier ratios. Perturbation is measured by the increase in labeled answer negative log-likelihood relative to dense inference. We average examples within question categories and then average category risks so that frequent categories do not dominate. This yields bandwise curves $R_b^A(k)$, $R_b^F(k)$, and $R_b^{AF}(k)$. On a grid of resolution Q, saving level $k \in \{0, \ldots, Q-1\}$ corresponds to $r(k) = 1 - k/Q$. Linear interpolation and a cumulative maximum produce monotone risk estimates on the grid.

For Attention and FFN saving levels (k_b^A, k_b^F) in band b, let $m_b = \min(k_b^A, k_b^F)$. We estimate their cost from the measured joint risk at m_b plus the marginal Attention and FFN risks beyond m_b, clipped at zero. For each target saving K, dynamic programming minimizes the summed bandwise cost subject to the saving across bands weighted by band length. The resulting allocation is expanded to per-layer ratios $\mathbf{r}^A$ and $\mathbf{r}^F$, and duplicate schedules are removed. Layers assigned the same operator ratio reuse one assignment map.

A.5 ANALYTICAL DECODER PREFILL FLOPS

Let d_{kv} be the total key/value projection width and m the gated FFN intermediate width. We count one multiplication and one addition as two floating-point operations and count the dominant matrix multiplications under a full attention matrix convention. A dense decoder layer requires

$$F_{\text{dense}} = 2\big(2Nd^2 + 2Ndd_{kv} + 2N^2d + 3Ndm\big)\,. \tag{10}$$

With effective carrier ratios $\bar{r}_\ell^A$ and $\bar{r}_\ell^F$ after integer rounding, define $N_\ell^A = N - V + \bar{r}_\ell^A V$ and $N_\ell^F = N - V + \bar{r}_\ell^F V$. Our method retains dense key/value projection for all tokens, but evaluates query/output projection and the quadratic Attention products only for N_ℓ^A queries and the FFN only for N_ℓ^F inputs:

$$F_{\text{share},\ell} = 2\big(2N_\ell^A d^2 + 2Ndd_{kv} + 2N_\ell^A Nd + 3N_\ell^F dm\big)\,. \tag{11}$$

We sum Eq. (11) over layers and report analytical decoder prefill FLOPs relative to dense inference. Selection and assignment overhead and memory movement operations are excluded from this analytical count; measured decoder prefill time includes them.

B ADDITIONAL CALIBRATION AND MECHANISM ABLATIONS

B.1 EFFECT OF CALIBRATION SET SIZE

Depth bands and schedules. We partition the decoder layers into six contiguous depth bands, denoted by $B_1, \ldots, B_6$ in order from shallow to deep layers. Each band uses one Attention carrier ratio and one FFN carrier ratio, shared by all layers within that band. Thus, each calibrated schedule contains six ratios per operator. These depth bands also define the intervention ranges in Appendix B.3.

Calibration data. We evaluate Qwen3-VL-8B-Instruct with nested calibration sets of 12, 24, 48, and 96 unique videos drawn exclusively from the NExT-QA training split in VideoChat2-IT (Xiao et al., 2021; Li et al., 2024), using one question per video. Each set contains equal numbers of causal, temporal, and descriptive questions (4, 8, 16, and 32 per category, respectively). Selection uses deterministic SHA256 ranking with a fixed manifest seed, stratification by question subtype and supertype, and answer-letter balancing within each selected segment.

The 24-video set is the calibration cohort used for the main Qwen3-VL results; the 12-video set is a balanced subset. The 48-video set extends the 24-video set with 24 additional training videos. The 96-video set adds 48 further training videos under the same selection policy.

Results. Table 7 reports accuracy on the complete evaluation splits, and Table 8 lists the selected schedules over six depth bands. All schedules use the same sharing level $S = 0.5$. Increasing the calibration set from 12 to 24 videos improves both benchmarks. The 24-video configuration achieves

the highest MVBench accuracy among these settings, while the 96-video configuration achieves the highest VideoMME accuracy at 62.15%, improving on the 24-video result by 0.33 pp and remaining 0.33 pp below dense inference. Accuracy does not improve monotonically with calibration set size: the 48-video configuration is below the 24-video result on both benchmarks. Overall, 24 calibration videos already provide strong results, while the benefit of additional calibration data varies across datasets.

Table 7: **Accuracy versus calibration set size on Qwen3-VL.** Δ is the accuracy difference from dense inference in percentage points. The 24-video configuration is the main experimental setting.

	MVBench		VideoMME	
Calibration videos	Accuracy (%)	Δ (pp)	Accuracy (%)	Δ (pp)
Dense	66.60	—	62.48	—
12	65.43	-1.18	60.04	-2.44
24 (main)	66.63	$+0.03$	61.81	-0.67
48	66.10	-0.50	61.44	-1.04
96	66.20	-0.40	62.15	-0.33

Table 8: **Schedules selected with different calibration set sizes.** Entries are visual carrier ratios for depth bands B_1–B_6, ordered from shallow to deep layers.

Calibration videos	Operator	B_1	B_2	B_3	B_4	B_5	B_6
12	Attention	1.000	0.375	1.000	0.250	0.125	0.125
	FFN	1.000	1.000	0.375	0.125	0.500	0.125
24	Attention	0.125	1.000	1.000	0.500	0.125	0.125
	FFN	0.875	0.875	1.000	0.125	0.125	0.125
48	Attention	0.250	1.000	1.000	0.250	0.125	0.250
	FFN	0.750	1.000	1.000	0.125	0.125	0.125
96	Attention	0.500	1.000	1.000	0.125	0.125	0.125
	FFN	0.750	1.000	1.000	0.125	0.125	0.125

B.2 Zeroing Visual Receiver Updates

We record our method's carrier–receiver groups and replay them exactly while replacing every visual receiver update with zero. Non-visual tokens and visual carriers retain their original computation, and both variants preserve all key/value entries and residual states.

Table 9: **Copying carrier updates versus zeroing visual receiver updates.** Complete benchmark results. Δ is Ours minus zero update.

Model	Dataset	Ours (%)	Zero (%)	Δ (pp)
Qwen3-VL-8B	MVBench	66.63	34.00	$+32.63$
	VideoMME	61.81	25.04	$+36.78$
Qwen2-VL-7B	MVBench	64.90	36.78	$+28.13$
	VideoMME	55.44	27.22	$+28.22$

Replacing visual receiver updates with zero causes a 28.13–36.78 pp loss despite identical non-visual and carrier computation. Retaining visual residual states without supplying the shared nonzero update is not equivalent to our method.

B.3 Zeroing Visual Updates at Different Depths

We evaluate visual-only zero update variants on fixed stratified random 500-example subsets. Our baseline uses the default $S = 0.5$ schedules. We use the six contiguous depth bands B_1–B_6 defined

in Appendix B.1, ordered from shallow to deep layers. We replace visual updates with zero only in sublayers whose calibrated carrier ratio is 0.125. The shallow and deep interventions are restricted to bands B_1–B_2 and B_5–B_6, respectively; the full-depth intervention includes eligible sublayers in all six bands (B_1–B_6). Non-visual tokens remain dense in all variants.

Table 10: **Zeroing visual updates at different depths.** Values use fixed stratified random 500-example subsets. Zeroing applies only to sublayers with a carrier ratio of 0.125, within the two shallowest bands (B_1–B_2), the two deepest bands (B_5–B_6), or all six bands (B_1–B_6). Non-visual tokens remain dense. Accuracy is reported in percent.

Model	Dataset	Ours	Zero updates Bands B_1–B_2	Zero updates Bands B_5–B_6	Zero updates Bands B_1–B_6
Qwen3-VL-8B	MVBench	67.4	65.8	66.6	66.2
	VideoMME	59.0	58.6	58.6	58.2
Qwen2-VL-7B	MVBench	65.8	55.6	63.2	56.0
	VideoMME	52.0	37.2	50.4	36.4
LLaVA-OV-7B	MVBench	52.0	50.0	50.8	47.6
	VideoMME	52.8	45.2	51.6	43.6

All three zero-update variants reduce accuracy relative to sharing across the six model–dataset settings, even though non-visual tokens remain dense and the visual sequence is retained. The magnitude is model dependent: zeroing eligible sublayers in bands B_1–B_2 costs 0.4–14.8 pp, while zeroing eligible sublayers across bands B_1–B_6 costs 0.8–15.6 pp. This mechanism test shows that the nonzero visual updates supplied by our method contribute beyond retaining residual states.

B.4 Reusing Assignments Across Layers

We compare our default assignment reuse with recomputing carrier selection and receiver assignment at every shared layer. Both variants use Qwen3-VL-8B-Instruct with 32 frames, the same frozen $S = 0.5$ schedule, integer carrier counts, complete visual key/value sequence, and analytical decoder prefill FLOPs. Accuracy is measured on 500-example subsets sampled equally across MVBench tasks and stratified by video duration for VideoMME. Both variants use the same examples.

Table 11: **Effect of assignment reuse.** Accuracy differences are per-layer recomputation minus reuse, with paired percentile-bootstrap 95% confidence intervals from 10,000 replicates. Latencies are pooled medians over 100 fixed inputs.

(a) Accuracy (%)

Dataset	Reuse	Per-layer	Δ (pp)	95% CI
MVBench	67.2	66.2	-1.0	[-2.6, 0.6]
VideoMME	58.8	59.8	+1.0	[-0.8, 2.8]

(b) Latency (ms)

Measurement	Reuse	Per-layer
Feature projection	1.20	15.49
Carrier selection	1.54	21.64
Receiver assignment	1.27	16.18
Total selection and assignment overhead	4.98	64.47
Decoder prefill	389.62	441.42

The paired accuracy intervals include zero on both benchmarks, so this experiment does not detect an accuracy difference. Assignment reuse reduces total overhead, including feature projection, from 64.47 to 4.98 ms and yields a $1.13\times$ decoder prefill speedup over per-layer recomputation.

Timing uses BF16, batch size one, 30 warm-up iterations, and three repeats per input. We take the repeat median for each input and then the pooled median across 100 inputs (50 from each benchmark).

Component timings and decoder prefill timings come from separate measurement passes; component medians are not additive.

B.5 Detailed Frame Scaling Results

Table 12 provides the absolute latencies and visual token counts corresponding to Figure 3. Measurements follow the protocol in Section 4.5.

Table 12: **Frame count scaling of measured decoder prefill latency.** Measurements use an RTX 3090, batch size one, BF16, and SDPA. Token counts and latencies are medians over 100 inputs.

Model	Frames	Visual tokens	Dense (ms)	Ours (ms)	Speedup
Qwen3-VL-8B	8	448	158.2	130.9	1.209×
	16	896	301.6	225.6	1.337×
	32	1792	532.7	385.7	1.381×
	64	3584	1061.9	737.6	1.440×
Qwen2-VL-7B	8	576	185.5	139.4	1.331×
	16	1152	299.4	221.4	1.352×
	32	2304	566.3	394.9	1.434×
	64	4608	1094.8	760.9	1.439×
LLaVA-OV-7B	8	1568	417.4	300.1	1.391×
	16	3136	874.3	596.9	1.465×
	32	6272	1779.1	1180.9	1.506×
	64	12544	3390.3	2293.0	1.478×

C Statistical Analysis of Combination Results

We compare the correctness of +Ours and the compute-matched token-only control on paired questions using an exact two-sided McNemar test. Accuracies and accuracy differences are computed from unrounded counts and rounded independently; 95% percentile confidence intervals use 20,000 paired bootstrap replicates. Table 13 reports the 21 configurations in Table 2. Holm correction is applied across these 21 comparisons. Labels are used only for this analysis, not for selecting the compute-matched controls.

Table 13: **Paired statistical analysis of combination results.** Δ is +Ours minus the compute-matched control in pp. Holm q values are adjusted across the 21 comparisons. Model and dataset abbreviations follow Table 1.

Method	Model	Data	Δ (pp)	95% CI	Holm q
VisionZip (10%)	Q3	MV	+7.28	[6.00, 8.55]	1.07×10^{-26}
VisionZip (10%)	Q3	VM	+2.22	[1.00, 3.44]	0.0038
VisionZip (10%)	Q3	LVB	+1.20	[-0.22, 2.62]	0.5869
VisionZip (10%)	Q2	MV	+4.60	[3.50, 5.70]	6.92×10^{-15}
VisionZip (10%)	Q2	VM	+2.00	[0.67, 3.30]	0.0236
VisionZip (10%)	Q2	LVB	+0.75	[-0.60, 2.09]	0.6085
VisionZip (10%)	L-OV	MV	+10.13	[8.72, 11.55]	3.81×10^{-43}
VisionZip (10%)	L-OV	VM	+6.33	[4.81, 7.85]	7.77×10^{-15}
VisionZip (10%)	L-OV	LVB	+4.49	[2.62, 6.43]	8.22×10^{-5}
VidCom^2 (25%)	Q3	MV	+5.33	[4.17, 6.48]	3.87×10^{-18}
VidCom^2 (25%)	Q3	VM	+2.37	[1.26, 3.48]	4.42×10^{-4}
VidCom^2 (25%)	Q3	LVB	+3.14	[1.50, 4.79]	0.0026
VidCom^2 (10%)	Q3	MV	+4.83	[3.57, 6.05]	8.69×10^{-13}
VidCom^2 (10%)	Q3	VM	+2.85	[1.63, 4.11]	1.19×10^{-4}
VidCom^2 (10%)	Q3	LVB	+1.20	[-0.45, 2.84]	0.6085
iLLaVA	Q3	MV	+2.98	[1.93, 4.03]	4.28×10^{-7}
iLLaVA	Q3	VM	+3.81	[2.44, 5.19]	5.36×10^{-7}
iLLaVA	Q3	LVB	+1.27	[-0.37, 2.92]	0.6085
EarlyTom-Q3	Q3	MV	+1.38	[0.43, 2.33]	0.0292
EarlyTom-Q3	Q3	VM	+1.85	[0.67, 3.04]	0.0224
EarlyTom-Q3	Q3	LVB	+1.05	[-0.52, 2.62]	0.6085

D LIMITATIONS AND FUTURE WORK

We focus on decoder prefill efficiency and use model-specific schedules calibrated under a fixed protocol. Further evaluation across hardware, batch sizes, end-to-end generation, and additional multimodal architectures will help characterize broader deployment settings.

Future work will investigate fused kernels for carrier selection and receiver assignment, hardware-aware schedule selection, cross-model calibration transfer, and extensions of our method to autoregressive decoding and longer-context multimodal workloads.