OMNIPARTITION: QUERY-GUIDED REGION REFINEMENT FOR AUDIO–VISUAL TOKEN PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Omnimodal large language models (OmniLLMs) jointly process audio, video, and text through a shared decoder sequence, but the resulting long multimodal context makes prefill increasingly expensive. Training-free token reduction offers an effective solution, yet existing methods often overlook the spatial and temporal structure of multimodal inputs or rely on fixed modality allocations and auxiliary models. We introduce *OmniPartition*, a training-free framework for query-guided, structure-aware audio–visual token pruning. OmniPartition first performs pruning after the first decoder block, where contextualized representations provide a reliable query-affinity signal. It then partitions video and audio tokens along their native spatial and temporal coordinates and adaptively refines these regions according to their query-weighted representation error. A shared integer budget is allocated through the marginal gain of each refinement, allowing the method to jointly determine which regions to preserve and how many tokens to retain without predefined modality ratios or temporal windows. The resulting compact sequence can be directly processed by standard FlashAttention. Across six audio–visual benchmarks and multiple OmniLLM backbones, OmniPartition consistently preserves performance under substantial token reduction. On Qwen2.5-Omni-7B, it retains 95.27% of full-token performance at 20% retention while achieving a 3.34× prefill speedup.

1 INTRODUCTION

Omnimodal large language models (OmniLLMs) jointly process visual, audio, and textual inputs, enabling tasks such as audio–visual question answering, video understanding, and multimodal dialogue Xu et al. (2025b); Fu et al. (2024); Li et al. (2024). By integrating speech, environmental sounds, and visual content, OmniLLMs can capture complementary evidence across modalities, which is particularly valuable for video understanding where the answer may depend on both audio and visual cues. However, encoding video frames and audio segments produces long token sequences, substantially increasing LLM prefill latency and GPU memory usage and limiting efficient inference on long videos Tao et al. (2025); Shao et al. (2025).

Token pruning offers an effective means of reducing prefill cost in vision-language models (VLMs). Existing approaches select or merge visual tokens based on attention scores Chen et al. (2024); Yang et al. (2025a), feature similarity Shang et al. (2025); Alvar et al. (2025), or temporal redundancy Huang et al. (2025); Tao et al. (2025). Recent efforts extend token compression to OmniLLMs, where visual and audio tokens should be pruned jointly. OmniZip Tao et al. (2026) leverages audio saliency to guide video-token budget allocation, while OmniSelect Yang et al. (2026) employs AudioCLIP to estimate modality relevance to the query and derive modality-specific pruning policies. These methods perform pruning before the LLM, whereas OmniDrop Park et al. (2026) performs query-guided pruning across decoder layers.

Despite these advances, effective audio–visual token pruning remains challenging. OmniZip and OmniSelect rely on query-independent token-ranking criteria, limiting their ability to adapt token selection to the specific question. OmniSelect introduces query awareness through an external AudioCLIP model, but the additional inference incurs non-trivial latency. OmniDrop brings query-guided selection into the decoder, yet repeated attention computation and token selection across layers introduce substantial overhead, resulting in limited practical speedup in our measurements. More

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.

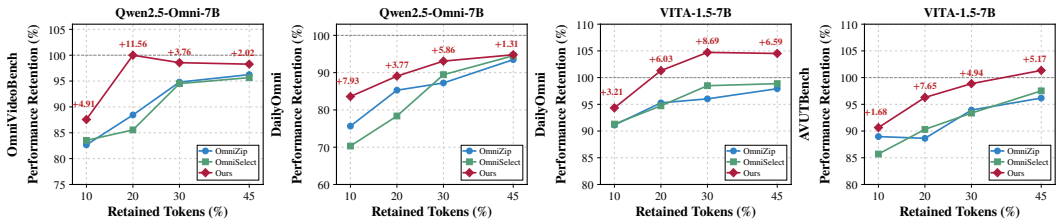


Figure 1: Performance relative to full-token inference. OmniPartition better preserves performance than OmniZip across audio–visual benchmarks and two OmniLLM backbones under reduced token budgets.

fundamentally, existing approaches do not explicitly account for spatial coverage within individual video frames. Our experiments show that uniform sampling with spatial and temporal coverage can match or outperform OmniZip and OmniSelect in the evaluated settings (Figure 4), suggesting that token relevance alone is insufficient for effective pruning and that the structural organization of multimodal inputs should also be preserved.

To address these limitations, we propose OmniPartition, a training-free method for query-guided audio–visual token pruning that preserves spatial and temporal structure. First, Query-Affinity Weighting (QAW) models query relevance within the LLM to guide audio–visual token selection. Second, Native-Coordinate Partition (NCP) preserves spatial and temporal coverage through hierarchical partitioning along the native coordinates of video and audio tokens. Third, Representation-Error Driven Partitioning and Allocation (RE-PA) jointly considers query relevance and feature redundancy to adaptively refine regions and allocate a shared audio–visual token budget. A fused Triton kernel further reduces the overhead of dynamic budget allocation.

Our contributions are summarized as follows:

- We introduce OmniPartition, a training-free framework for query-guided audio–visual token pruning that jointly captures cross-modal relevance and native input structure, enabling structure-preserving compression tailored to the query.
- We propose Native-Coordinate Partition and Representation-Error Driven Partitioning and Allocation, which organize multimodal tokens into spatial and temporal regions and dynamically allocate a shared budget according to their marginal compression gains.
- We develop a fused Triton implementation that precomputes candidate-region distortions, avoiding repeated feature aggregation and substantially reducing the overhead of adaptive token selection.

2 Related Work

Omnimodal Large Language Models. OmniLLMs, such as VITA Fu et al. (2024) and Qwen2.5-Omni Xu et al. (2025a), integrate audio and visual perception with language models for multimodal understanding. Modality-specific encoders extract audio and visual features, which are projected into the LLM embedding space. Qwen2.5-Omni Xu et al. (2025a) groups audio–visual tokens into fixed-duration time windows and arranges them chronologically for joint processing with text tokens. The resulting long audio–visual sequences create a computational bottleneck during inference.

Token Compression for OmniLLMs. Earlier work on vision–language models showed that redundant visual encoder outputs can be removed while preserving much of the information needed for downstream understanding; VisionZip is one representative example Yang et al. (2025a). For OmniLLMs, compression must additionally preserve complementary audio and video evidence under a shared token budget. OmniZip identifies salient audio segments to guide video-token pruning within temporal groups Tao et al. (2026). OmniSIFT first reduces spatial and temporal video redundancy, then uses retained visual information to select audio tokens Ding et al. (2026). Other methods adapt modality allocation to the query: OmniSelect estimates modality preference to choose pruning policies Yang et al. (2026), while OmniScope scores audio and video relevance separately and

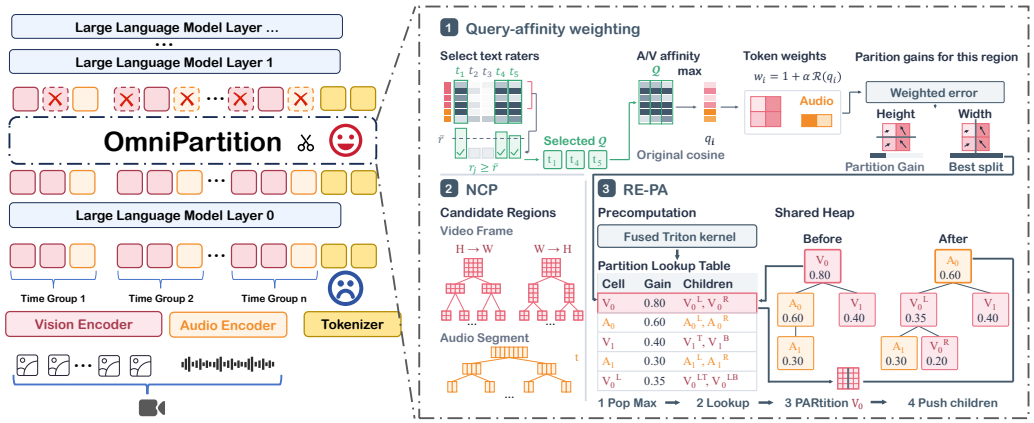


Figure 2: Overview of OmniPartition. Left: Audio–visual tokens are pruned after the first decoder layer. Right: (1) QAW weights tokens by their affinity to selected query anchors. (2) NCP constructs spatial video regions and temporal audio regions. (3) RE-PA uses precomputed split gains and a shared max-heap to refine regions under a shared audio–visual budget. Each final region retains its highest-affinity token.

allocates modality-specific budgets Su et al. (2026). Unlike methods that use one modality to guide the other or determine modality-specific budgets from query relevance, OmniPartition jointly determines where to retain tokens and how to allocate the audio–visual budget by refining spatially and temporally coherent regions according to query relevance and feature redundancy.

In-LLM Token Pruning. Beyond which tokens to retain, an important question is when query relevance becomes sufficiently informative to guide pruning. In-LLM methods exploit contextualized representations for token selection Chen et al. (2024); Xing et al. (2025). SparseVLM Zhang et al. (2025) uses text-to-visual attention for progressive pruning, while OmniDrop Park et al. (2026) applies text-to-audiovisual attention across decoder layers. TransPrune Li et al. (2026) combines representation transitions with instruction attention at multiple stages. These approaches raise an efficiency trade-off: deeper representations may provide stronger query signals, but delaying pruning leaves more layers processing the full sequence. Extracting attention scores and repeating token selection can further offset the savings, particularly for long audio–visual inputs. This motivates examining how early query relevance becomes useful and whether it can guide pruning without costly attention extraction.

3 Method

3.1 Problem Formulation

Different questions require different audio–visual information: some depend primarily on speech or environmental sounds, while others rely on visual content or both modalities. Fixed audio and video retention rates may therefore fail to reflect the information needed to answer a given query. We instead consider a shared audio–visual token budget, allowing the allocation between modalities to adapt to the input and query. Figure 2 provides an overview of our approach.

Retention depth is not a free parameter. The token budget also depends on the pruning position. Let N denote the original number of audio–visual tokens and M the number of decoder layers. If pruning retains k tokens before the first decoder layer, the retention rate is k/N throughout the decoder. If pruning occurs after d layers, the first d layers process all N tokens, while the remaining $M - d$ layers process k tokens. We therefore define the layer-average audio–visual retention rate as

$$R(k; d) = \frac{dN + (M - d)k}{MN}. \tag{1}$$

For a feasible target retention rate r, the required post-pruning retention rate is

$$\rho(d) = \frac{k}{N} = \frac{Mr - d}{M - d}. \tag{2}$$

When $d = 0$, this reduces to $\rho(0) = r$. At a fixed layer-average retention rate, pruning at a deeper layer requires retaining fewer tokens in subsequent layers. This trade-off motivates considering both the pruning position and the allocation of the shared audio–visual budget.

3.2 Query-Affinity Weighting

Where the Query Signal Becomes Readable. Let H^0 denote the input embeddings to the LLM decoder, including projected audio–visual features and text embeddings, and let H^1 denote the hidden states after the first decoder block. At H^0, we observe low cosine similarity between query and audio–visual tokens, whereas their affinity increases at H^1 (Figure 3). Meanwhile, Eq. equation 2 shows that delaying pruning reduces the number of tokens available to subsequent layers under the same layer-average budget. We therefore compute query affinity and prune at H^1, allowing all remaining decoder layers to process the reduced sequence.

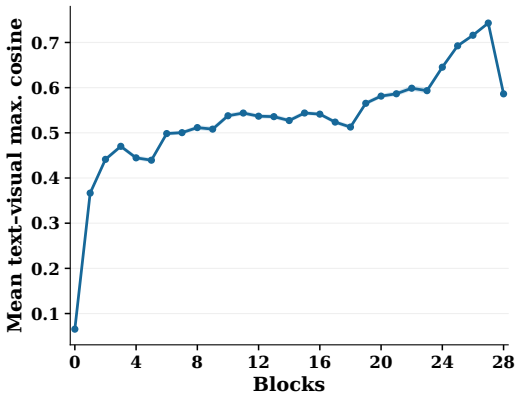


Figure 3: Mean text–visual cosine is weak in H^0 and usable after the first block.

Not every query token is a useful anchor. Let $H_{\mathrm{AV}} \in \mathbb{R}^{N \times D}$ collect the H^1 states of the audio–visual tokens and $H_{\mathrm{query}} \in \mathbb{R}^{N_q \times D}$ those of the query tokens of the current sample, where N_q is the query token count and D is the hidden dimension. With norm denoting row-wise L_2 normalization and $^\top$ denoting transpose, the cosine-affinity matrix is

$$A = \mathrm{norm}(H_{\mathrm{AV}})\,\mathrm{norm}(H_{\mathrm{query}})^\top \in \mathbb{R}^{N \times N_q}. \tag{3}$$

Let $\mathcal{V}$ be the visual token index set, with $|\cdot|$ denoting set cardinality. We compute query-token scores r_j and select the anchor index set $\mathcal{Q}$ as

$$r_j = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \mathrm{softmax}(A_{i,:})_j, \qquad \mathcal{Q} = \{\, j : r_j \geq \bar{r} \,\}, \tag{4}$$

where $A_{i,:}$ is the row of similarities between audio–visual token i and all query tokens, softmax operates over query tokens, and $\bar{r}$ is the mean of $(r_j)_{j=1}^{N_q}$.

Query-Guided Token Weighting. We measure each audio–visual token's relevance to the query by its maximum affinity to the selected query tokens:

$$q_i = \max_{j \in \mathcal{Q}} A_{ij}, \qquad i = 1, \ldots, N. \tag{5}$$

Let $q_{\min}$ and $q_{\max}$ denote the minimum and maximum affinity scores over all audio–visual tokens in the sample. We apply min–max normalization, $\mathcal{R}(q_i) = (q_i - q_{\min})/(q_{\max} - q_{\min})$, and define the token weights as

$$w_i = 1 + \alpha\, \mathcal{R}(q_i), \tag{6}$$

where α controls the influence of query relevance and is set to 1. Higher-affinity tokens receive larger weights, while the constant term ensures that low-affinity tokens still contribute to the region distortion. These weights guide the subsequent partitioning, and the affinity scores determine the representative token within each final region.

3.3 Native-Coordinate Partition (NCP)

Spatial and Temporal Coverage. Figure 4 shows that uniformly sampling visual and audio tokens within their respective frames and segments achieves comparable or higher mean performance retention than OmniZip and OmniSelect. Although these methods organize compression into temporal groups, their within-frame selection does not explicitly constrain spatial coverage. This observation motivates preserving spatial and temporal structure alongside query relevance. We therefore partition the input into regions and select a query-relevant token from each region.

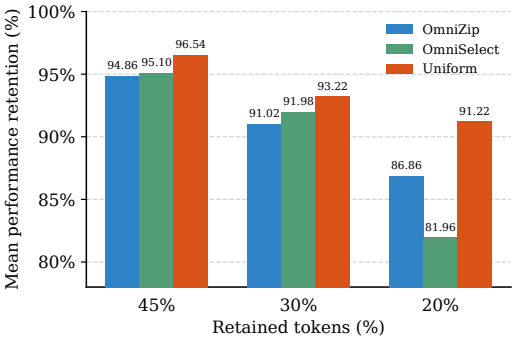


Figure 4: Uniform structural sampling retains more than OmniZip and OmniSelect at 45%, 30%, and 20% A/V retention.

How Represent A Region. A video region consists of tokens within a two-dimensional area of a frame, while an audio region consists of tokens within a contiguous temporal interval. Each final region C retains the original token with the highest query affinity:

$$i_C^\star = \arg\max_{i \in C} q_i. \tag{7}$$

Is the Region Greatly Represented Regions with similar features are easier to represent, whereas those containing diverse features may require more representative tokens. We quantify this difficulty using a query-weighted representation error. Let $x_i = h_i/\|h_i\|_2$ denote the normalized token feature and w_i is query-guided weight (equation 6). For region C, define the weighted feature sum $s_C = \sum_{i \in C} w_i x_i$, the total weight $W_C = \sum_{i \in C} w_i$, and the normalized mean direction $\mu_C = s_C/\|s_C\|_2$. The representation error is

$$D(C) = \sum_{i \in C} w_i \big(1 - x_i^\top \mu_C\big) = W_C - \|s_C\|_2. \tag{8}$$

Here, q_i determines which token is retained, while $D(C)$ measures the difficulty of representing the region with a single feature direction.

Native-Coordinate Partitioning. To accommodate differences in region representation difficulty, we recursively partition regions along their native coordinates. Each video frame is initialized as a complete two-dimensional token grid and partitioned along height or width. Each contiguous audio segment is initialized as a temporal interval and partitioned along time. These constraints preserve spatial locality within frames and temporal continuity within audio segments. For a partition $C = C^- \cup C^+$ with $C^- \cap C^+ = \varnothing$, the weighted feature sums and total weights are additive. The triangle inequality therefore gives

$$D(C) - D(C^-) - D(C^+) = \|s_{C^-}\|_2 + \|s_{C^+}\|_2 - \|s_{C^-} + s_{C^+}\|_2 \geq 0. \tag{9}$$

Thus, partitioning cannot increase the sum of regional representation errors. The following section uses this error reduction to select partitions and allocate the shared audio–visual token budget.

3.4 Representation-Error Driven Partitioning and Allocation (RE-PA)

Query Relevance and Feature Redundancy. Questions differ in their reliance on audio and visual evidence, so we allocate a shared token budget across both modalities. QAW incorporates query relevance into regional representation errors through token weights. The resulting split gains therefore depend on both query affinity and feature variation. Two cases illustrate the role of redundancy. When normalized features converge to a common unit direction,

$$x_i \to u \quad (\forall i \in C), \qquad \|u\|_2 = 1 \quad \Longrightarrow \quad D(C) \to 0. \tag{10}$$

A single feature direction then represents the region with vanishing error. For mutually orthogonal features, $x_i^\top x_j = 0$ for all $i \neq j$, we obtain

$$D(C) = W_C - \sqrt{\sum_{i \in C} w_i^2} = w\big(n - \sqrt{n}\big), \qquad \text{when } w_i = w, \quad n = |C|. \tag{11}$$

The latter case reflects the difficulty of representing distinct feature directions with a single direction. We allocate additional tokens according to the reduction in this representation error achievable through admissible splits.

Adaptive Region Refinement. We form candidate splits at the median of the distinct coordinates along each admissible axis: height or width for video regions, and time for audio regions. For axis a, let C_a^- and C_a^+ denote the two child regions. We score the split by

$$g_a(C) = \left[D(C) - D(C_a^-) - D(C_a^+)\right] \frac{2\min(|C_a^-|, |C_a^+|)}{|C|}. \tag{12}$$

The first factor measures the reduction in representation error from introducing one additional region and is nonnegative by Eq. equation 9. The second factor accounts for unequal child sizes arising from discrete coordinate cuts. We select the preferred axis and its gain as

$$a^\star(C) = \arg\max_a g_a(C), \qquad g(C) = g_{a^\star(C)}(C). \tag{13}$$

All current audio and visual regions participate in the same allocation procedure. At each step, we split the region with the highest gain:

$$C^\star = \arg\max_{C \in \mathcal{L}: |C|>1} g(C), \qquad \mathcal{L} \leftarrow (\mathcal{L} \setminus \{C^\star\}) \cup \{C^{\star,-}, C^{\star,+}\}, \tag{14}$$

where $\mathcal{L}$ is the current partition and $C^{\star,-}, C^{\star,+}$ are the selected children. Each split replaces one region with two, adding one retained token. Refinement continues until the budget $k^\star$ is reached, with audio and visual token counts satisfying $k_a + k_v = k^\star$. Their ratio follows from the selected splits without a prescribed modality quota.

Fused Kernel for Precomputation. For each input, we use a fused Triton kernel to precompute representation errors across candidate regions in parallel. By fusing weighted feature aggregation and norm computation, the kernel avoids materializing intermediate feature tensors. The resulting errors determine the split gains stored in a lookup table, enabling dynamic allocation of the shared audio–visual budget without repeated feature aggregation during refinement (Appendix H).

4 Experiments

4.1 Experimental Setup

Models. We evaluate primarily on Qwen2.5-Omni-3B and Qwen2.5-Omni-7B Xu et al. (2025a), with additional evaluations on VITA-1.5-7B Fu et al. (2025b) and Baichuan-Omni-1.5-7B Li et al. (2024) to assess generalization.

Baselines. We compare against full-token inference, random pruning, DyCoke Tao et al. (2025), OmniZip Tao et al. (2026), and OmniSelect Yang et al. (2026). OmniSIFT Ding et al. (2026) trains modality-asymmetric selectors, so we do not include it among the training-free baselines. OmniZip scores tokens from audio-encoder attention, which has to be materialized and is therefore poorly compatible with FlashAttention Dao (2024). In addition to extracting encoder attention, OmniSelect Yang et al. (2026) relies on an auxiliary AudioCLIP model, making it similarly incompatible with FlashAttention Dao (2024).

Benchmarks. We evaluate on six complementary benchmarks. DailyOmni Zhou et al. (2025) emphasizes temporally distributed everyday events; WorldSense Hong et al. (2025) covers diverse domains and joint audio–visual reasoning; and OmniVideoBench Li et al. (2025) targets synergistic cross-modal understanding. VideoMME Fu et al. (2025a) includes long-form video with audio, AVUTBench Yang et al. (2025b) is audio-centric, and ShortVid-Bench Ge et al. (2025) focuses on fine-grained short-video understanding.

Implementation Details All experiments are conducted on NVIDIA H100 GPUs with 80 GB of memory each. Unless otherwise specified, we sample at most 512 frames per video for VideoMME and 128 frames for all other benchmarks. We retain all text tokens and report accuracy and mean retention relative to full visual-aduio token inference.

4.2 Main Results

Table 1: Qwen2.5-Omni on OmniVideoBench, VideoMME, DailyOmni, and WorldSense. Mean Ret. is the unweighted mean of the four dataset scores relative to Full Tokens, using Avg. for VideoMME.

Method	Retained	Omni VideoBench	VideoMME Short	VideoMME Medium	VideoMME Long	VideoMME Avg.	DailyOmni	World Sense	Mean Ret. (%)
Qwen2.5-Omni-7B									
Full Tokens	100%	34.6	77.1	66.8	55.4	66.43	64.16	46.82	100.00%
Random	55%	32.1	73.9	66.6	55.3	65.27	56.89	43.25	93.02%
DyCoke (V&A)	50%	32.4	73.9	66.4	55.1	65.13	59.48	43.95	94.57%
OmniZip	45%	33.3	75.4	67.2	55.4	66.00	59.98	45.27	96.44%
OmniSelect	45%	33.1	74.9	67.7	56.4	66.33	60.65	46.00	97.07%
OmniPartition	45%	**34.0**	**75.6**	**69.0**	**56.9**	**67.17**	**60.82**	**46.15**	**98.19%**
OmniZip	30%	32.8	73.8	66.3	56.3	65.47	55.97	43.85	93.56%
OmniSelect	30%	32.7	73.8	66.3	56.3	65.47	57.39	44.17	94.21%
OmniPartition	30%	**34.1**	**75.1**	**69.0**	**57.0**	**67.03**	**59.73**	**44.96**	**97.15%**
Qwen2.5-Omni-3B									
Full Tokens	100%	32.9	74.8	64.1	52.8	63.90	62.82	45.62	100.00%
Random	55%	31.0	68.8	61.6	52.4	60.93	53.55	41.68	91.55%
DyCoke (V&A)	50%	31.3	71.1	62.8	52.4	62.10	55.89	43.06	93.92%
OmniZip	45%	32.5	74.0	64.6	52.7	63.77	56.14	44.07	96.14%
OmniSelect	45%	32.7	73.4	**65.0**	53.3	63.90	**58.06**	45.08	97.66%
OmniPartition	45%	**33.4**	**74.1**	**65.0**	**53.5**	**64.20**	**58.06**	**45.40**	**98.48%**
OmniZip	30%	31.4	70.6	61.2	52.7	61.50	53.05	41.83	91.96%
OmniSelect	30%	**33.3**	69.7	61.9	53.2	61.60	54.39	44.42	95.39%
OmniPartition	30%	**33.3**	**73.0**	**64.1**	**53.5**	**63.53**	**55.05**	**44.86**	**96.65%**

Table 1 shows that OmniPartition achieves the highest mean performance retention among the compared compressors at both 45% and 30% retention for both model scales. Notably, the 7B model exceeds full-token inference on VideoMME, while the 3B model surpasses its full-token baseline on OmniVideoBench, at both retention rates. VideoMME spans short to long videos, making the preservation of informative evidence across varying temporal extents important. OmniVideoBench instead emphasizes complementary audio–visual evidence, requiring compression to retain information useful for joint reasoning. Substantial token reduction can therefore preserve task-relevant evidence in both regimes, and the gains over full-token inference are consistent with a benefit from removing redundant context.

Table 2: Generalization results of VITA-1.5 7B.

Method	Ret.	Daily Omni	AVUT Bench	ShortVid Bench	Mean Ret. (%)
VITA-1.5 7B					
Full Tokens	100%	44.28	51.21	64.30	100.00%
OmniZip	45%	43.36	49.25	62.70	97.20%
OmniSelect	45%	43.78	49.94	62.40	97.81%
OmniPartition	45%	**46.28**	**51.90**	**63.60**	**101.59%**
OmniZip	30%	42.52	48.10	61.80	95.35%
OmniSelect	30%	43.61	47.81	62.70	96.45%
OmniPartition	30%	**46.37**	**50.63**	**62.90**	**100.47%**
OmniZip	20%	42.19	45.39	61.50	93.19%
OmniSelect	20%	41.94	46.25	62.00	93.82%
OmniPartition	20%	**44.86**	**49.31**	**62.60**	**98.32%**
OmniZip	10%	40.35	45.56	61.00	91.65%
OmniSelect	10%	40.43	43.89	59.70	89.95%
OmniPartition	10%	**41.77**	**46.42**	**61.40**	**93.49%**

Table 3 examines more aggressive compression, where OmniPartition outperforms OmniZip and OmniSelect on all five datasets at both 20% and 10% retention. The spread of these tasks matters: AVUTBench is audio-centric while OmniVideoBench requires complementary audio–visual reasoning, and leading on both is what a shared budget is supposed to deliver. DailyOmni stresses temporal relations among everyday events and ShortVid-Bench fine-grained detail in short clips, so both probe whether discriminative evidence survives a tight budget; WorldSense extends the pattern across real-world domains.

Table 3: Qwen2.5-Omni-7B at 20% and 10% retention.

Method	Ret.	Omni VideoBench	World Sense	Daily Omni	ShortVid Bench	AVUT Bench	Mean Ret. (%)
Full Tokens	100%	34.6	46.82	64.16	70.5	64.59	100.00%
OmniZip	20%	30.6	41.52	54.72	68.1	57.50	89.60%
OmniSelect	20%	29.6	40.06	50.28	64.2	53.40	84.64%
OmniPartition	20%	**34.6**	**44.58**	**57.14**	**69.5**	**60.38**	**95.27%**
OmniZip	10%	28.6	37.83	48.54	65.3	52.54	82.62%
OmniSelect	10%	28.9	35.69	45.11	61.2	46.08	77.64%
OmniPartition	10%	**30.3**	**40.70**	**53.63**	**65.6**	**55.31**	**87.35%**

Generalization. As shown in table 2and 14, OmniPartition transfers consistently beyond Qwen2.5-Omni, achieving the strongest mean retention across the tested VITA-1.5 and Baichuan-Omni settings. On VITA-1.5, it even surpasses Full Tokens at some compression rates, suggesting that removing redundant context can preserve or improve task-relevant evidence. These results demonstrate the generalizability of OmniPartition across model families.

4.3 Ablation Study

Table 4 evaluates OmniPartition on four datasets using Qwen2.5-Omni-7B at 30% layer-average audio–visual token retention.

Table 4: Ablations with Qwen2.5-Omni-7B, 128 frames, and 30% retention.

Variant	WorldSense	DailyOmni	OVB	ShortVid	Mean Ret. (%)
Full Tokens	46.82	64.16	34.60	70.50	100.00%
OmniPartition (After Layer 0)	44.96	**59.73**	34.10	**69.70**	**96.64%**
w/o QAW	44.29	57.48	32.00	67.50	93.10%
w/o RE-PA	43.13	59.40	33.50	68.70	94.74%
w/o Balance	44.92	59.65	33.30	69.50	95.93%
H^0 Space	44.04	58.40	33.40	68.80	94.80%
Random(in Eq.7)	43.22	56.98	31.60	65.80	91.45%
Mean(in Eq. 5)	44.29	57.06	32.00	66.90	92.73%
H^0 & Random(in Eq.7)	43.66	57.48	31.90	66.20	92.23%
H^2, After Layer 1	**45.40**	58.56	33.30	**69.70**	95.84%
H^3, After Layer 2	44.86	57.98	**34.80**	69.60	96.37%
H^4, After Layer 3	44.10	58.23	**34.80**	68.10	95.53%

Query guidance. Removing the query-affinity weights in Eq. 6 from the regional representation error in Eq. 8 (w/o QAW) while retaining affinity-based representative selection reduces mean performance retention from 96.64% to 93.10%, supporting query-aware partitioning and allocation. Replacing maximum affinity in Eq. 5 with mean aggregation (Mean in Eq. 5) lowers retention to 92.73%, suggesting that averaging may dilute strong matches to specific query tokens. Random representative selection in Eq. 7 (Random in Eq.7) causes the largest drop, to 91.45%, highlighting the importance of selecting query-relevant evidence within each region.

Structure and allocation. Replacing RE-PA with preset token budgets for each video frame and audio segment, and selecting the highest-affinity tokens within each frame or segment up to its assigned budget (w/o RE-PA), reduces mean performance retention to 94.74%. This supports allocating the shared budget according to regional representation error. Preset budgets may retain redundant evidence in some frames or audio segments while assigning insufficient capacity to others with greater feature diversity. Removing the child-size balance factor in Eq. 12 (w/o Balance) lowers mean performance retention to 95.93%, with small accuracy decreases across all four datasets, supporting this correction for unequal child sizes.

Pruning position. Pruning in H^0 before the first decoder block achieves 94.80% mean performance retention, compared with 96.64% after the first block. This comparison reflects changes in both feature space and pruning position under the calibrated budget. Delaying pruning until after the second, third, or fourth block yields 95.84%, 96.37%, and 95.53%, respectively. Although some later positions improve WorldSense or OVB accuracy, pruning after the first block achieves the highest mean retention. As Eq. 2 shows, delaying pruning under a fixed layer-average budget requires retaining fewer tokens in subsequent layers.

4.4 Efficiency Analysis

As shown in Table 5, OmniPartition achieves increasing prefill speedups as the retention budget decreases. At 20% and 10% retention, its Triton implementation achieves $3.34\times$ and $4.54\times$ speedups over full-token inference, reducing total latency from 397.85 ms to 119.17 ms and 87.56 ms, respectively. Peak GPU memory also decreases from 22.80 GiB to 18.17 GiB and 18.09 GiB. OmniZip's feature aggregation and related pruning operations incur substantial computational overhead, limiting its speedups to $1.88\times$ and $2.78\times$ at the same budgets. OmniSelect is slower than full-token inference because of the additional cost of its auxiliary AudioCLIP model. OmniPartition's lightweight selection and fused Triton implementation keep pruning overhead low, allowing token reduction to translate into larger latency savings. These measurements include pruning overhead and cover prefill only, excluding subsequent decoding.

Table 5: H100 prefill efficiency of Qwen2.5-Omni-7B on WorldSense. Latencies are in ms, including pruning overhead in Total. Speedup is relative to Full Tokens. Qwen2.5-Omni-3B results are in Appendix I.5 Table 16.

Method	Ret.	Peak Mem. (GiB)	LLM Prefill	Extra attn.	Prune	Total	Speedup
Qwen2.5-Omni-7B (Model: 16.64 GiB)							
Full Tokens	100%	22.80	397.85	–	–	397.85	1.00×
OmniZip	45%	20.20	188.27	13.27	149.53	351.06	1.13×
OmniSelect	45%	20.19	163.44	9.33	11.38 (+1259.38)	1443.53	0.28×
OmniPartition (PyTorch)	45%	19.53	166.33	–	58.87	225.20	1.77×
OmniPartition (Triton)	45%	19.53	166.20	–	55.96	222.16	1.79×
OmniZip	30%	19.41	149.21	14.08	127.17	290.46	1.37×
OmniSelect	30%	19.32	109.80	9.56	11.05 (+1256.73)	1387.14	0.29×
OmniPartition (PyTorch)	30%	19.07	112.27	–	56.86	169.13	2.35×
OmniPartition (Triton)	30%	19.07	112.16	–	45.33	157.48	2.53×
OmniZip	20%	18.24	85.01	13.16	113.37	211.54	1.88×
OmniSelect	20%	18.79	99.78	14.64	30.98 (+2242.06)	2387.46	0.17×
OmniPartition (PyTorch)	20%	18.17	76.80	–	50.01	126.82	3.14×
OmniPartition (Triton)	20%	18.17	77.77	–	41.40	119.17	3.34×
OmniZip	10%	17.93	48.36	11.98	82.87	143.21	2.78×
OmniSelect	10%	18.62	44.82	11.53	16.68 (+1376.89)	1449.92	0.27×
OmniPartition (PyTorch)	10%	18.09	48.42	–	41.00	89.42	4.45×
OmniPartition (Triton)	10%	18.09	49.75	–	37.81	87.56	4.54×

5 Conclusion

We presented OmniPartition, a training-free method for audio–visual token pruning in OmniLLMs. OmniPartition prunes the hidden states after the first decoder block, combining query affinity with structure-preserving region selection to allocate a shared budget across audio and video. Results across six benchmarks support the value of combining query relevance and structural coverage, particularly under aggressive compression.

AI use statement

We did not use generative AI tools for tasks requiring disclosure under the ICLR 2027 AI Policy. We used generative AI tools only to edit author-written text for readability, including grammar, clarity, and style. The authors reviewed all AI-assisted edits and take responsibility for the final content of this work.

Ethics statement

This work adheres to the ICLR Code of Ethics. Our study does not involve human subjects, nor personal or sensitive data. All datasets utilized in this paper are publicly available and widely adopted within the research community, and we strictly follow their respective licenses and intended usage.

Reproducibility statement

We strive to ensure the reproducibility of our results. Full details are provided in the main paper and the appendix. Our implementation is built on PyTorch, Triton and standard open-source libraries. We provide key code implementations to facilitate reproducibility and further research.

REFERENCES

Saeed Ranjbar Alvar, Gursimran Singh, Mohammad Akbari, and Yong Zhang. Divprune: Diversity-based visual token pruning for large multimodal models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9392–9401, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Alvar_DivPrune_Diversity-based_Visual_Token_Pruning_for_Large_Multimodal_Models_CVPR_2025_paper.html.

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.

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

Yue Ding, Yiyan Ji, Jungang Li, Xuyang Liu, Xinlong Chen, Junfei Wu, Bozhou Li, Bohan Zeng, Yang Shi, Yushuo Guan, et al. OmniSIFT: Modality-asymmetric token compression for efficient omni-modal large language models. *arXiv preprint arXiv:2602.04804*, 2026. URL https://arxiv.org/abs/2602.04804.

Chaoyou Fu, Haojia Lin, Zuwei Long, Yunhang Shen, Yuhang Dai, Meng Zhao, Yi-Fan Zhang, Shaoqi Dong, Yangze Li, Xiong Wang, et al. VITA: Towards open-source interactive omni multimodal LLM. *arXiv preprint arXiv:2408.05211*, 2024. URL https://arxiv.org/abs/2408.05211.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, et al. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025a.

Chaoyou Fu, Haojia Lin, Xiong Wang, Yi-Fan Zhang, Yunhang Shen, Xiaoyu Liu, Haoyu Cao, Zuwei Long, Heting Gao, Ke Li, et al. Vita-1.5: Towards gpt-4o level real-time vision and speech interaction. *arXiv preprint arXiv:2501.01957*, 2025b.

Yuying Ge, Yixiao Ge, Chen Li, Teng Wang, Junfu Pu, Yizhuo Li, Lu Qiu, Jin Ma, Lisheng Duan, Xinyu Zuo, et al. Arc-hunyuan-video-7b: Structured video comprehension of real-world shorts. *arXiv preprint arXiv:2507.20939*, 2025.

Jack Hong, Shilin Yan, Jiayin Cai, Xiaolong Jiang, Yao Hu, and Weidi Xie. Worldsense: Evaluating real-world omnimodal understanding for multimodal llms. *arXiv preprint arXiv:2502.04326*, 2025.

Xiaohu Huang, Hao Zhou, and Kai Han. Prunevid: Visual token pruning for efficient video large language models. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Ao Li, Yuxiang Duan, Jinghui Zhang, Congbo Ma, Yutong Xie, Gustavo Carneiro, Mohammad Yaqub, and Hu Wang. Transprune: Token transition pruning for efficient large vision-language model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 39529–39538, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Li_TransPrune_Token_Transition_Pruning_for_Efficient_Large_Vision-Language_Model_CVPR_2026_paper.html.

Caorui Li, Yu Chen, Yiyan Ji, Jin Xu, Zhenyu Cui, Shihao Li, Yuanxing Zhang, Wentao Wang, Zhenghao Song, Dingling Zhang, et al. Omnivideobench: Towards audio-visual understanding evaluation for omni mllms. *arXiv preprint arXiv:2510.10689*, 2025.

Yadong Li, Haoze Sun, Mingan Lin, Tianpeng Li, Guosheng Dong, Tao Zhang, Bowen Ding, Wei Song, Zhenglin Cheng, Yuqi Huo, Song Chen, Xu Li, Da Pan, Shusen Zhang, Xin Wu, Zheng Liang, Jun Liu, Tao Zhang, Keer Lu, Yaqi Zhao, Yanjun Shen, Fan Yang, Kaicheng Yu, Tao Lin, Jianhua Xu, Zenan Zhou, and Weipeng Chen. Baichuan-omni technical report. *arXiv preprint arXiv:2410.08565*, 2024.

Yeo Jeong Park, Hyemi Jang, Minseo Choi, Jongsun Lee, Jooyoung Choi, and Yongkweon Jeon. Omnidrop: Layer-wise token pruning for omni-modal llms via query-guidance, 2026. URL https://arxiv.org/abs/2605.14458.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. Llava-prumerge: Adaptive token reduction for efficient large multimodal models. In *IEEE/CVF International Conference on Computer Vision*, 2025.

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. *arXiv preprint arXiv:2507.20198*, 2025.

Jinsen Su, Yongdong Luo, Yuexiao Ma, Yibo Hu, Meiguang Jin, and Xiawu Zheng. OmniScope: Modality-decoupled token compression for omnimodal large language models. *arXiv preprint arXiv:2607.23193*, 2026. URL https://arxiv.org/abs/2607.23193.

Keda Tao, Can Qin, Haoxuan You, Yang Sui, and Huan Wang. Dycoke: Dynamic compression of tokens for fast video large language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Keda Tao, Kele Shao, Bohan Yu, Weiqiang Wang, Jian Liu, and Huan Wang. OmniZip: Audio-guided dynamic token compression for fast omnimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17682–17692, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Tao_OmniZip_Audio-Guided_Dynamic_Token_Compression_for_Fast_Omnimodal_Large_Language_CVPR_2026_paper.html.

Long Xing, Qidong Huang, Xiaoyi Dong, Jiajie Lu, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, Jiaqi Wang, Feng Wu, et al. Pyramiddrop: Accelerating your large vision-language models via pyramid visual redundancy reduction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, et al. Qwen2.5-Omni technical report. *arXiv preprint arXiv:2503.20215*, 2025a. URL https://arxiv.org/abs/2503.20215.

Jin Xu, Zhifang Guo, Hangrui Hu, Yunfei Chu, Xiong Wang, Jinzheng He, Yuxuan Wang, Xian Shi, Ting He, Xinfa Zhu, et al. Qwen3-omni technical report. *arXiv preprint arXiv:2509.17765*, 2025b.

Morunliu Yang, Ruotao Xu, Le Li, Yue Wang, Jianxin Zhang, Juntao Li, Yihang Lou, Siwei Feng, and Peifeng Li. OmniSelect: Dynamic modality-aware token compression for efficient omni-modal large language models. *arXiv preprint arXiv:2605.18041*, 2026. URL https://arxiv.org/abs/2605.18041.

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*, pp. 19792–19802, 2025a. URL https://openaccess.thecvf.com/content/CVPR2025/html/Yang_VisionZip_Longer_is_Better_but_Not_Necessary_in_Vision_Language_CVPR_2025_paper.html.

Yudong Yang, Jimin Zhuang, Guangzhi Sun, Changli Tang, Yixuan Li, Peihan Li, Yifan Jiang, Wei Li, Zejun Ma, and Chao Zhang. Audio-centric video understanding benchmark without text shortcut. In *Conference on Empirical Methods in Natural Language Processing*, 2025b.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis A Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and Shanghang Zhang. SparseVLM: Visual token sparsification for efficient vision-language model inference. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 74840–74857. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/zhang25s.html.

Ziwei Zhou, Rui Wang, Zuxuan Wu, and Yu-Gang Jiang. Daily-omni: Towards audio-visual reasoning with temporal alignment across modalities. *arXiv preprint arXiv:2505.17862*, 2025.

A AGORITHM

Algorithm 1 summarizes the OmniPartition pipeline. Given the hidden states after the first decoder block, we determine the audio–visual token budget $k^\star$ that best matches the target layer-average retention rate and compute query affinities and token weights using the selected anchors. We initialize regions as video-frame grids and contiguous audio intervals. If the initial region count exceeds the budget, we retain the $k^\star$ regions with the largest unweighted representation errors $D_0(C)$. Otherwise, using precomputed split gains, we repeatedly partition the highest-gain region across both modalities until $k^\star$ regions are obtained. This procedure jointly allocates the shared budget to audio and video. Finally, we select the token with the highest query affinity from each region and return the retained hidden states and indices, preserving all text tokens and the original sequence order.

Algorithm 1 OmniPartition

Require: $H^1 \in \mathbb{R}^{(T+N)\times D}$, layers M, retention r, labels t (0: text; > 0: A/V)
Require: Coordinates c, frame/segment IDs, anchors $\mathcal{Q}$, query-to-sequence map π_q
$h_i \leftarrow H^1[i,:]$ (full-sequence indexing)
$N \leftarrow |\{i : t_i > 0\}|$; $k^\star \leftarrow \arg\min_{1\le k\le N,\, k\in\mathbb{Z}} |R(k;1) - r|$
$q_i \leftarrow \max_{j\in\mathcal{Q}} \cos(h_i, h_{\pi_q(j)})$ $(t_i > 0)$
$w_i \leftarrow 1 + \mathcal{R}(q_i)$; $x_i \leftarrow \mathrm{norm}(h_i)$ $(t_i > 0)$
$\mathcal{L}_0 \leftarrow \{\text{video-frame grids}\} \cup \{\text{contiguous audio intervals}\}$
if $|\mathcal{L}_0| > k^\star$ **then**
  $D_0(C) \leftarrow |C| - \|\sum_{i\in C} x_i\|_2$ $(C \in \mathcal{L}_0)$
  $\mathcal{L}_0 \leftarrow \mathrm{TopK}(\mathcal{L}_0, D_0, k^\star)$
end if
Precompute $\mathcal{T}(C) = (g(C), C^-, C^+)$ for candidate splits within $\mathcal{L}_0$ (Appendix H).
$\mathcal{L} \leftarrow \mathcal{L}_0$
while $|\mathcal{L}| < k^\star$ **do**
  $C^\star \leftarrow \arg\max_{C\in\mathcal{L}:|C|>1} g(C)$
  $(g(C^\star), C^-, C^+) \leftarrow \mathcal{T}(C^\star)$
  $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{C^\star\}) \cup \{C^-, C^+\}$
end while
$i^\star_C \leftarrow \arg\max_{i\in C} q_i$ $(C \in \mathcal{L})$
$\mathcal{I} \leftarrow \mathrm{sort}\big(\{i : t_i = 0\} \cup \{i^\star_C : C \in \mathcal{L}\}\big)$

return $(H^1[\mathcal{I},:],\ \mathcal{I})$

B TEMPORAL TOKEN ALLOCATION

Figure 5 shows the temporal token-retention profiles of OmniPartition on a single sample using Qwen2.5-Omni-7B at a 30% audio–visual retention target. Video retention exhibits pronounced local fluctuations, whereas audio retention remains comparatively stable over much of the sequence and decreases toward the end. These distinct profiles illustrate non-uniform allocation across modalities and temporal regions under a shared budget, consistent with the representation-error-driven refinement.

C MODALITY-WISE TOKEN RETENTION

Table 6 reports the audio and video token retention rates under different joint compression budgets. Across all evaluated settings, a larger fraction of audio tokens is retained than video tokens. At the 30% target budget, audio retention ranges from 32.29% to 33.57% across DailyOmni, OmniVideoBench, and WorldSense, while video retention ranges from 28.76% to 29.37%. The same pattern holds on AVUTBench at the 10% and 20% budgets, where the 7B model consistently retains a higher proportion of audio tokens and a lower proportion of video tokens than the 3B model. These results show that the joint budget yields different retention rates for the two modalities.

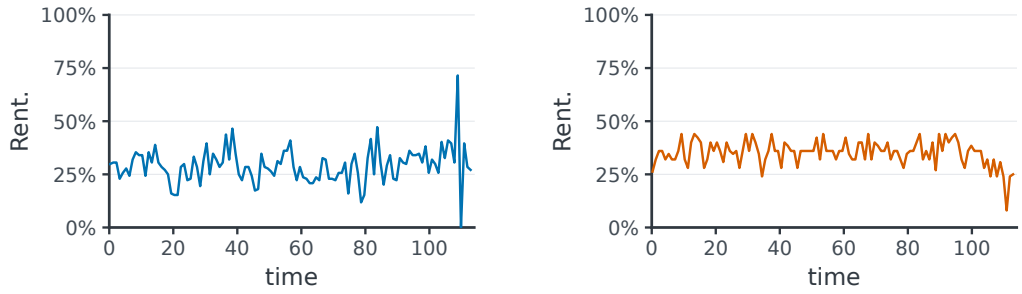


Figure 5: **Temporal variation in token retention.** Video (left) and audio (right) retention rates for the same input. Both modalities exhibit non-uniform retention over time, reflecting adaptive allocation across temporal regions.

Table 6: Audio and video token retention on different benchmarks using Qwen2.5-Omni with a 128-frame setting. Target denotes the joint audio–video retention budget. Audio and video retention are computed relative to the original token count of each modality and then averaged across samples.

Dataset	Model	Target (%)	Audio (%)	Video (%)
DailyOmni	7B	30	33.57	29.37
OmniVideoBench	7B	30	32.29	28.76
WorldSense	7B	30	33.05	28.98
AVUTBench	3B	10	15.45	8.88
AVUTBench	3B	20	25.91	18.75
AVUTBench	7B	10	16.53	8.63
AVUTBench	7B	20	27.95	18.31

D Case Study

We examine two VideoMME examples comparing OmniPartition with OmniZip at the same 30% target layer-averaged audio–visual retention. Both require preserving localized evidence together with its temporal context. The visualizations show the original frames, retained visual patches, and predicted answers; red boxes mark the diagnostic regions.

Preserving the state needed for counterfactual reasoning. Figure 6 asks what would have happened if Harrison Barnes' game-winning shot had missed. The decisive evidence is the tied score, 111–111 with 1.3 seconds remaining, followed by the expired clock and the eventual update to 114–111. A miss would leave the game tied and lead to overtime, rather than a defeat for Sacramento. OmniPartition preserves all four highlighted score-region patches and answers correctly, while OmniZip removes these patches and selects the defeat option. This example illustrates why retaining the players and court alone may be insufficient: a small scoreboard region establishes the game state on which the counterfactual depends. Preserving that region across the displayed moments also helps distinguish the score before the shot from the final result.

Preserving context that explains an observed reaction. Figure 7 asks why the man screams at the beginning of the video. His expression and the scream identify the reaction, but the preceding statement, "Only one more point and I win," explains its cause: the rabbit is about to win. OmniPartition retains four of the eight highlighted subtitle-region patches and selects this explanation. OmniZip retains none of these patches and instead chooses the generic description that the man is angry. The contrast illustrates the value of preserving evidence across related moments: the reaction frame alone does not specify the situation that prompted it. Here, the subtitle provides a localized visual cue to the spoken context.

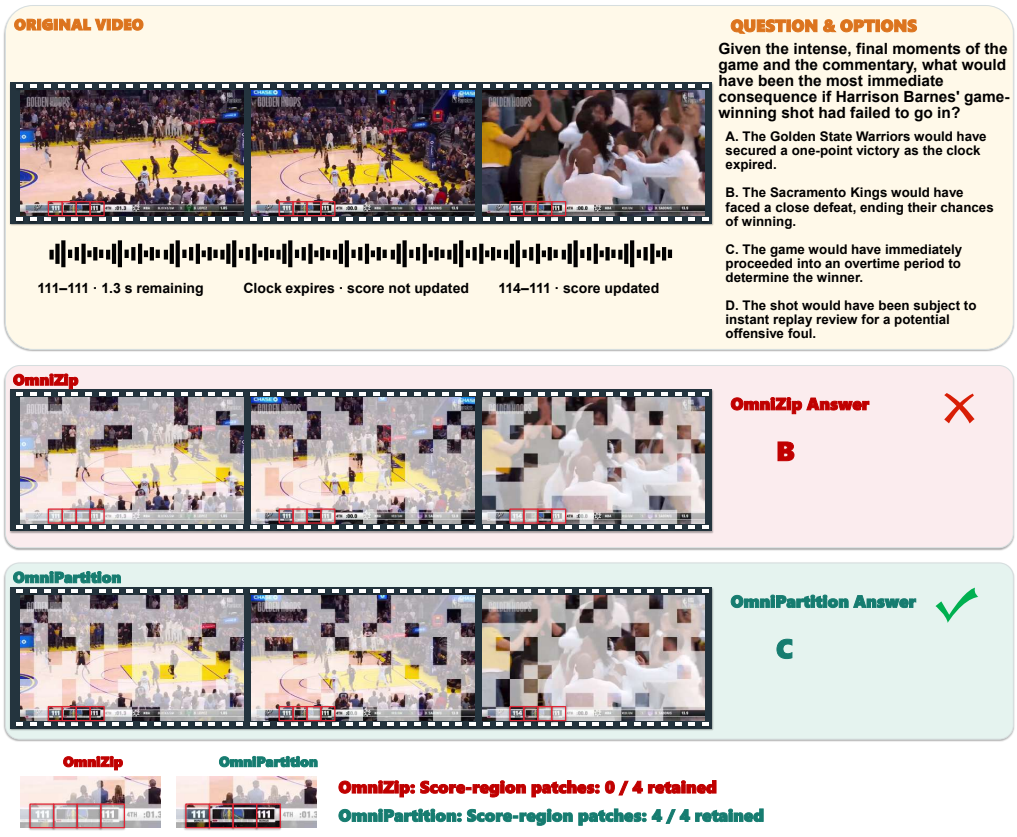


Figure 6: **Counterfactual reasoning from the score and game clock.** The game is tied at 111–111 before the final shot and ends at 114–111 after the score updates. In the highlighted score region, OmniPartition retains 4/4 patches, whereas OmniZip retains 0/4. OmniPartition correctly selects C (overtime if the shot had missed); OmniZip selects B (a close defeat).

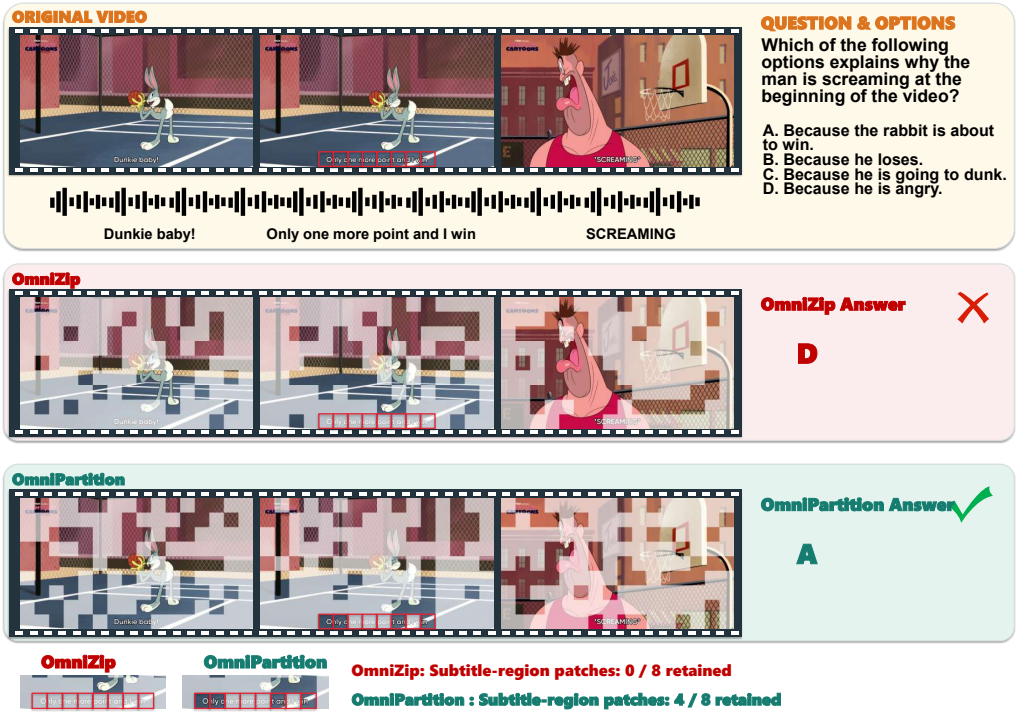


Figure 7: **Identifying the cause of a reaction from preceding context.** The subtitle "Only one more point and I win" provides context for the man's scream. OmniPartition retains 4/8 patches in the highlighted subtitle region and correctly selects A (the rabbit is about to win), whereas OmniZip retains 0/8 and selects D (the man is angry).

E LAYER-AVERAGED TOKEN BUDGET ANALYSIS

Layer-averaged token retention. Let $M \geq 2$ denote the number of decoder layers, $N \geq 1$ the original audio–visual token count, $d \in \{0, \ldots, M-1\}$ the number of complete layers before pruning, and k the number of audio–visual tokens retained afterwards. All text tokens remain and are excluded from this audio–visual budget. The target $r \in (0, 1]$ is a layer-averaged token retention rate; the corresponding pruning rate is $1 - r$.

The first d layers each process N audio–visual tokens, while each of the remaining $M - d$ layers processes k. Dividing by the count MN without pruning gives the layer-averaged token retention rate

$$R(k; d) = \frac{\overbrace{dN}^{\text{layers before pruning}} + \overbrace{(M-d)k}^{\text{layers after pruning}}}{MN}. \tag{15}$$

Let $\rho(d)$ denote the token retention rate after pruning at depth d. Solving $R(k; d) = r$ in the continuous relaxation yields

$$\begin{aligned} dN + (M-d)k &= MNr, \\ (M-d)k &= N(Mr - d), \\ \rho(d) := \frac{k}{N} &= \frac{Mr - d}{M - d}. \end{aligned} \tag{16}$$

This derives Eqs. equation 1 and equation 2 of Section 3.1. At $d = 0$, every layer sees the same retention rate k/N, so $\rho(0) = r$. At $k = N$, $R(N; d) = 1$ for every depth. Fixing only k/N while changing d therefore does not fix the token–layer budget.

Dependence on pruning depth. To quantify this dependence, compare pruning after d and $d + 1$ decoder layers. For $0 \leq d \leq M - 2$, both choices leave at least one layer after pruning, and

$$\begin{aligned} \rho(d+1) - \rho(d) &= \frac{Mr - d - 1}{M - d - 1} - \frac{Mr - d}{M - d} \\ &= -\frac{M(1-r)}{(M-d-1)(M-d)}. \end{aligned} \tag{17}$$

For $r < 1$, this difference is negative wherever both choices are feasible. Its magnitude grows with d: under a fixed token budget, pruning at a deeper layer requires a larger reduction in the token retention rate of subsequent layers. For $r = 1$, the difference is zero.

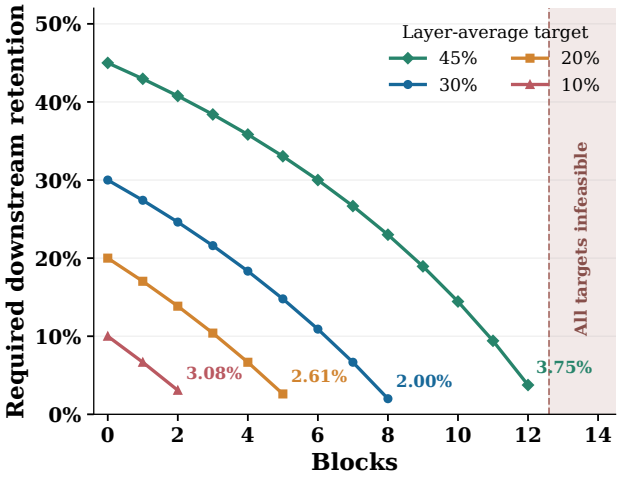


Figure 8: Token retention rate after pruning, $\rho(d)$, as a function of pruning depth d for a decoder with $M = 28$ layers and layer-averaged token retention targets $r \in \{45\%, 30\%, 20\%, 10\%\}$.

Pruning Depth Constraints. The expression for $\rho(d)$ also determines which pruning depths are feasible. Since $M-d>0$, retaining a positive number of audio–visual tokens requires $Mr-d>0$. Otherwise, the layers before pruning already require at least the target token budget. The largest integer pruning depth satisfying this condition is $\lceil Mr \rceil - 1$. For $M = 28$, the maximum depths are 12, 8, 5, and 2 for the 45%, 30%, 20%, and 10% targets, respectively (Figure 8). The shaded region includes all integer depths $d \geq 13$, where none of the four budgets is feasible. Within the feasible range, deeper pruning reduces the number of tokens retained in subsequent layers. For example, at $r = 30\%$,

$$\rho(1) = \frac{8.4-1}{27} \approx 27.41\%, \qquad \rho(4) = \frac{8.4-4}{24} \approx 18.33\%. \tag{18}$$

F Computational Complexity Analysis

F.1 Notation and Assumptions

Scope. Our analysis covers the decoder prefill stage and a single invocation of the token selector. The modality encoders and autoregressive decoding are outside its scope.

Token counts. Let M denote the number of decoder layers and N the number of audio–visual tokens before pruning. Pruning is applied after the first $d \in \{0, \ldots, M-1\}$ layers, which therefore process the full sequence. The number of retained audio–visual tokens is k. All T text tokens are retained. Let L denote the total sequence length processed by a decoder layer. Each of the first d layers processes $L = T + N$ tokens, and each of the remaining $M - d$ layers processes $L = T + k$ tokens.

Layer dimensions. The input to a decoder layer is $X \in \mathbb{R}^{L\times D}$, where D is the model dimension, and D_{ff} denotes the intermediate dimension of the gated feed-forward network (FFN). Each layer has h_q query heads and h_{kv} key–value (KV) heads of dimension d_h. We define the KV-sharing ratio $\eta = h_{\text{kv}}/h_q \in (0, 1]$, where $\eta = 1$ corresponds to multi-head attention and $\eta < 1$ to grouped-query attention. As is standard, we assume that the query heads partition the model dimension. It follows that

$$h_q d_h = D, \qquad h_{\text{kv}} d_h = \eta D. \tag{19}$$

Counting convention. All shapes refer to a single sample, with the batch dimension omitted. We write $[a, b]$ for a matrix with a rows and b columns. For positive integers m, n, p, the product $[m, n]\times [n, p]$ requires mnp multiplications and $mp(n-1)$ additions. Following standard practice (**?**), we use the leading-order estimate $2mnp$, which counts each multiply–accumulate as two floating-point operations (FLOPs). We denote the resulting estimate for one decoder layer by $F_{\text{layer}}(L)$.

F.2 Computational Cost of a Decoder Layer

Attention projections. Let $W_Q, W_O \in \mathbb{R}^{D\times D}$ be the query and output projection weights, and $W_K, W_V \in \mathbb{R}^{D\times \eta D}$ the key and value projection weights. The resulting query, key, and value matrices are $Q = XW_Q$, $K = XW_K$, and $V = XW_V$, respectively. Concatenating the attention outputs of all query heads gives the matrix $O \in \mathbb{R}^{L\times D}$, followed by the output projection OW_O. Write $F_{\text{proj}}(L)$ for the combined cost of these four projections. Their shapes and individual costs are

Operation	Matrix product	Output shape	FLOPs
XW_Q	$[L, D] \times [D, D]$	$[L, D]$	$2LD^2$
XW_K	$[L, D] \times [D, \eta D]$	$[L, \eta D]$	$2\eta LD^2$
XW_V	$[L, D] \times [D, \eta D]$	$[L, \eta D]$	$2\eta LD^2$
OW_O	$[L, D] \times [D, D]$	$[L, D]$	$2LD^2$

Adding the four projections explains the coefficient $4(1 + \eta)$:

$$\begin{aligned} F_{\text{proj}}(L) &= 2LD^2 + 2\eta LD^2 + 2\eta LD^2 + 2LD^2 \\ &= 4(1+\eta)LD^2. \end{aligned} \tag{20}$$

For ordinary multi-head attention, $\eta = 1$ and this reduces to $8LD^2$.

Attention scores and outputs. For a query-head index $j \in \{1, \dots, h_q\}$, let $g(j) \in \{1, \dots, h_{\text{kv}}\}$ identify the KV head shared by that query head. The matrices $Q_j, K_{g(j)}, V_{g(j)}$ are the corresponding head slices of Q, K, V, each with shape $[L, d_h]$. Let S_j denote the unscaled attention-score matrix and O_j the output of query head j; concatenating all O_j gives O. With $\mathrm{softmax}$ normalizing each row over key positions, these matrices are

$$S_j = Q_j K_{g(j)}^\top \in \mathbb{R}^{L \times L}, \qquad O_j = \mathrm{softmax}(S_j/\sqrt{d_h}) V_{g(j)} \in \mathbb{R}^{L \times d_h} \tag{21}$$

Their computation requires two matrix products: $[L, d_h] \times [d_h, L]$ and $[L, L] \times [L, d_h]$. Each costs $2L^2 d_h$ per head. Writing $F_{\text{attn}}(L)$ for their total over all query heads gives

$$F_{\text{attn}}(L) = \underbrace{2h_q L^2 d_h}_{QK^\top} + \underbrace{2h_q L^2 d_h}_{\text{attention-weighted } V} = 4L^2 D. \tag{22}$$

Sharing K/V does not eliminate the distinct scores and outputs of the query heads; the quadratic term is therefore not multiplied by η.

Gated feed-forward network. Let W_{gate} and W_{up} be the gate and up-projection weights, and W_{down} the down-projection weights of the feed-forward network. Here $\odot$ denotes elementwise multiplication. For a scalar z, $\mathrm{SiLU}(z) = z/(1 + e^{-z})$; this activation is applied elementwise to matrix inputs. Suppressing normalization and residual connections, the gated FFN is

$$\mathrm{FFN}(X) = \big[\mathrm{SiLU}(XW_{\text{gate}}) \odot (XW_{\text{up}})\big] W_{\text{down}}, \tag{23}$$

where $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{D \times D_{\text{ff}}}$ and $W_{\text{down}} \in \mathbb{R}^{D_{\text{ff}} \times D}$. Both the gate and up products have shape $[L, D_{\text{ff}}]$ and cost $2LDD_{\text{ff}}$ each. The down product maps $[L, D_{\text{ff}}] \times [D_{\text{ff}}, D]$ back to $[L, D]$ and has the same cost. Denoting their combined matrix-operation cost by $F_{\text{FFN}}(L)$, we obtain

$$F_{\text{FFN}}(L) = \underbrace{2LDD_{\text{ff}}}_{\text{gate}} + \underbrace{2LDD_{\text{ff}}}_{\text{up}} + \underbrace{2LDD_{\text{ff}}}_{\text{down}} = 6LDD_{\text{ff}}. \tag{24}$$

The coefficient six thus comes from three matrix products. The activation and elementwise gate multiplication add $O(LD_{\text{ff}})$ operations and are omitted from this leading matrix-operation estimate.

Total per-layer cost. Summing equation 20, equation 22, and equation 24 gives

$$\begin{aligned} F_{\text{layer}}(L) &= F_{\text{proj}}(L) + F_{\text{attn}}(L) + F_{\text{FFN}}(L) \\ &= 4(1+\eta)LD^2 + 4L^2 D + 6LDD_{\text{ff}}. \end{aligned} \tag{25}$$

The projection and FFN costs are linear in L, because the same weight matrices are applied to every token independently. The attention cost is quadratic in L, because every query position interacts with every key position. Grouping terms by powers of L, we obtain

$$F_{\text{layer}}(L) = \gamma L + 4DL^2, \qquad \gamma = 4(1+\eta)D^2 + 6DD_{\text{ff}}, \tag{26}$$

where γ is the per-token cost of the projections and the FFN and is independent of L.

F.3 Computational Savings from Token Pruning

Prefill cost with and without pruning. Let F_{dense} denote the total decoder prefill cost without pruning. Every dense layer receives $T + N$ tokens, giving

$$F_{\text{dense}} = M\, F_{\text{layer}}(T + N). \tag{27}$$

With a single pruning boundary, let F_{pruned} denote the decoder-only cost, F_{select} the cost of one selector call, and F_{method} their sum. Then

$$\begin{aligned} F_{\text{pruned}} &= d\, F_{\text{layer}}(T + N) + (M - d)\, F_{\text{layer}}(T + k), \\ F_{\text{method}} &= F_{\text{pruned}} + F_{\text{select}}. \end{aligned} \tag{28}$$

The selector overhead F_{select} is expanded in Section F.4. Define ΔF as the net FLOPs saved relative to dense prefill. Subtracting cancels the shared dense prefix:

$$\begin{aligned}\Delta F &= F_{\text{dense}} - F_{\text{method}} \\ &= (M-d)\big[F_{\text{layer}}(T+N) - F_{\text{layer}}(T+k)\big] - F_{\text{select}} \\ &= (M-d)\big[\gamma(N-k) + 4D\big((T+N)^2 - (T+k)^2\big)\big] - F_{\text{select}}.\end{aligned} \tag{29}$$

The attention difference factors as

$$\begin{aligned}(T+N)^2 - (T+k)^2 &= [(T+N) - (T+k)][(T+N) + (T+k)] \\ &= (N-k)(2T+N+k).\end{aligned} \tag{30}$$

Let ΔF_{proj}, ΔF_{FFN}, and ΔF_{attn} denote the projection, FFN, and attention savings, respectively, before subtracting selection overhead. The first d layers process the same sequence in both cases and contribute no savings. In each of the remaining $M-d$ layers, the sequence length decreases from $T+N$ to $T+k$. Thus, each component's total savings equal its per-layer cost difference multiplied by $M-d$. Substituting Eqs. equation 20, equation 24, and equation 22, respectively, gives

$$\begin{aligned}\Delta F_{\text{proj}} &= (M-d)\big[F_{\text{proj}}(T+N) - F_{\text{proj}}(T+k)\big] \\ &= 4(1+\eta)(M-d)\big[(T+N) - (T+k)\big]D^2 \\ &= 4(1+\eta)(M-d)(N-k)D^2, \\ \Delta F_{\text{FFN}} &= (M-d)\big[F_{\text{FFN}}(T+N) - F_{\text{FFN}}(T+k)\big] \\ &= 6(M-d)\big[(T+N) - (T+k)\big]DD_{\text{ff}} \\ &= 6(M-d)(N-k)DD_{\text{ff}}, \\ \Delta F_{\text{attn}} &= (M-d)\big[F_{\text{attn}}(T+N) - F_{\text{attn}}(T+k)\big] \\ &= 4D(M-d)\big[(T+N)^2 - (T+k)^2\big] \\ &= 4D(M-d)(N-k)(2T+N+k).\end{aligned} \tag{31}$$

The final attention expression uses the difference of squares in Eq. equation 30. Summing these three components and subtracting the selector cost yields $\Delta F = \Delta F_{\text{proj}} + \Delta F_{\text{FFN}} + \Delta F_{\text{attn}} - F_{\text{select}}$. Define s_{FLOPs} as the net saved fraction of dense prefill cost. The net savings and this fraction are

$$\boxed{\begin{aligned}\Delta F &= (M-d)(N-k)[\gamma + 4D(2T+N+k)] - F_{\text{select}}, \\ s_{\text{FLOPs}} &= \frac{\Delta F}{F_{\text{dense}}}.\end{aligned}} \tag{32}$$

For OmniPartition, denote the net savings by $\Delta F_{\text{OmniPartition}}$. Substituting $d=1$ gives

$$\Delta F_{\text{OmniPartition}} = (M-1)(N-k)[\gamma + 4D(2T+N+k)] - F_{\text{select}}. \tag{33}$$

Net savings are positive when the removed decoder work exceeds the selector cost. All these expressions use the realized integer token count.

F.4 Computational Overhead of the OmniPartition Selector

We analyze the query-affinity, shared-budget, lookup/fused path. Let $N_q \geq 1$ be the number of user-query tokens used to score relevance. These tokens form a subset of the T retained text tokens. Let $N_s = |\mathcal{V}|$ denote the number of visual tokens used to select query anchors.

Notation. For an audio–visual token index $i \in \{1, \ldots, N\}$, let $h_i \in \mathbb{R}^D$ be its hidden state after the first decoder block, and $x_i = h_i/\|h_i\|_2$ its normalized feature. Let $z_j \in \mathbb{R}^D$ be the state of query token $j \in \{1, \ldots, N_q\}$ and $u_j = z_j/\|z_j\|_2$ its normalized feature. Here j indexes query tokens, rather than the attention heads used in the preceding subsection. The cosine-affinity matrix is $A \in \mathbb{R}^{N \times N_q}$ with entry $A_{ij} = x_i^\top u_j$. The selected query-anchor set $\mathcal{Q} \subseteq \{1, \ldots, N_q\}$ contains query tokens whose scores are at least the mean, as defined locally below; it is distinct from the attention query matrix Q.

A region $C \subseteq \{1, \ldots, N\}$ is a set of audio–visual token indices. The initial regions are those present before recursive splitting; regions remaining at the end of splitting are leaves. If the number of initial regions exceeds the token budget, only the regions with the largest unweighted representation errors are retained. Let $\mathcal{C}$ be the family of candidate regions evaluated within the retained initial regions. Define

$$\mathcal{C}_+ = \{C \in \mathcal{C} : |C| > 1\}, \qquad P_+ = |\mathcal{C}_+|, \qquad E_+ = \sum_{C \in \mathcal{C}_+} |C|, \tag{34}$$

where P_+ counts non-singleton candidate regions and E_+ counts their memberships with multiplicity: a token in two candidates is counted twice. In the cost decomposition below, the subscripts norm, $\mathrm{affinity}$, anchor, weight, and region denote feature normalization, affinity-matrix multiplication, query-anchor selection, token-weight calculation, and candidate-region statistics, respectively:

$$\begin{aligned} F_{\mathrm{select}} \approx\ & F_{\mathrm{norm}} + F_{\mathrm{affinity}} + F_{\mathrm{anchor}} + F_{\mathrm{weight}} \\ & + F_{\mathrm{region}}. \end{aligned} \tag{35}$$

These terms count logical arithmetic on valid entries.

Feature normalization. Normalizing a vector of width D requires D squares, $D-1$ additions, one square root, and D divisions, totaling $3D$ scalar operations. The current selector separately normalizes the N structural features and the N audio–visual plus N_q query features for affinity. Consequently,

$$F_{\mathrm{norm}} \approx 3D(2N + N_q). \tag{36}$$

Affinity computation. Multiplying normalized audio–visual features $[N, D]$ by the transpose of the query features $[N_q, D]$ produces $[N, N_q]$, at cost

$$F_{\mathrm{affinity}} \approx 2NN_qD. \tag{37}$$

The same matrix supplies query-anchor selection and the maximum affinity to selected anchors. No second feature matrix product is required; the maximum is a comparison reduction.

Query-anchor selection. Let $\mathcal{V}$ be the visual token index set defined in Section 3.2, with $|\mathcal{V}| = N_s$. For the affinity matrix $A \in \mathbb{R}^{N \times N_q}$, the entry $A_{i,j}$ is the cosine similarity between audio–visual token i and query token j. The notation $A_{i,:}$ selects all columns of row i, giving the vector

$$A_{i,:} = [A_{i,1}, \ldots, A_{i,N_q}] \in \mathbb{R}^{N_q}. \tag{38}$$

Thus, this vector contains the similarities of one audio–visual token to all N_q query tokens. Softmax is applied independently to each selected row, normalizing over query tokens. Define p_{ij} as the resulting weight for query token j in row i, and m_i as that row's largest affinity:

$$p_{ij} = \mathrm{softmax}(A_{i,:})_j = \frac{\exp(A_{i,j} - m_i)}{\sum_{t=1}^{N_q} \exp(A_{i,t} - m_i)}, \qquad m_i = \max_{1 \le t \le N_q} A_{i,t}, \tag{39}$$

where t indexes query tokens in the sum and maximum.

We then average each column over the N_s selected audio–visual tokens. The resulting score r_j measures the average normalized weight assigned to query token j. Let $\bar{r}$ be the mean of these query-token scores; the anchor set $\mathcal{Q}$ retains tokens whose scores are at least this mean:

$$r_j = \frac{1}{N_s} \sum_{i \in \mathcal{V}_s} p_{ij}, \qquad \bar{r} = \frac{1}{N_q} \sum_{j=1}^{N_q} r_j, \qquad \mathcal{Q} = \{j : r_j \ge \bar{r}\}. \tag{40}$$

Here r_j denotes a query-token score. Since each normalized row satisfies $\sum_{j=1}^{N_q} p_{ij} = 1$, averaging the rows gives $\sum_{j=1}^{N_q} r_j = 1$ and hence $\bar{r} = 1/N_q$ in exact arithmetic. The selection rule therefore retains query tokens receiving at least the uniform share of the normalized affinity.

For each selected row, stable softmax requires N_q subtractions of m_i, N_q exponentiations, $N_q - 1$ additions to form the denominator, and N_q divisions. Counting each exponentiation and division as one scalar operation gives $4N_q - 1$ operations per row, excluding maximum comparisons. Computing r_j requires $N_s - 1$ additions and one division for each of the N_q columns, totaling

$N_q[(N_s-1)+1] = N_sN_q$ operations. Explicitly computing $\bar{r}$ from these scores requires a further $N_q - 1$ additions and one division, or N_q operations. Thus,

$$\begin{aligned} F_{\text{anchor}} &\approx \underbrace{N_s(4N_q-1)}_{\text{row-wise softmax}} + \underbrace{N_sN_q}_{\text{column means}} + \underbrace{N_q}_{\text{mean query score}} \\ &= N_s(5N_q-1) + N_q. \end{aligned} \tag{41}$$

Query-guided token weighting. For audio–visual token i, define its relevance score $q_i = \max_{j\in\mathcal{Q}} A_{ij}$. Let $q_{\min} = \min_{1\le i\le N} q_i$ and $q_{\max} = \max_{1\le i\le N} q_i$ be the smallest and largest relevance scores. Its scalar distortion weight is $w_i = 1 + \alpha(q_i - q_{\min})/(q_{\max} - q_{\min})$, where α controls query weighting and is set to 1 in OmniPartition. Forming the common denominator takes one subtraction; each token then needs one subtraction, division, multiplication by α, and addition:

$$F_{\text{weight}} \approx 4N + 1. \tag{42}$$

Candidate-region statistics. For a non-singleton region C, let $n_C = |C|$ be its token count, $s_C = \sum_{i\in C} w_i x_i \in \mathbb{R}^D$ its weighted feature sum, and $W_C = \sum_{i\in C} w_i$ its total scalar weight. Its weighted representation error (distortion) is $D(C) = W_C - \|s_C\|_2$; the function $D(C)$ is distinct from the scalar hidden width D. Starting from zero accumulators, each member requires D multiplications and D additions for s_C, plus one addition for W_C. The weighted feature sum therefore costs $2n_CD$ operations, and the weight sum costs n_C operations. Computing $\|s_C\|_2$ and subtracting it from W_C requires D squares, $D-1$ additions, one square root, and one subtraction, totaling $2D+1$ operations. Summing these three contributions over the set $\mathcal{C}_+$ of evaluated non-singleton candidate regions gives

$$\begin{aligned} F_{\text{region}} &\approx \sum_{C\in\mathcal{C}_+} \left[\underbrace{2n_CD}_{\text{weighted feature sum}} + \underbrace{n_C}_{\text{weight sum}} + \underbrace{2D+1}_{\text{norm and subtraction}} \right] \\ &= (2D+1)\sum_{C\in\mathcal{C}_+}(n_C+1) \\ &= (2D+1)(E_+ + P_+), \end{aligned} \tag{43}$$

where $E_+ = \sum_{C\in\mathcal{C}_+} n_C$ counts token memberships across these regions and $P_+ = |\mathcal{C}_+|$ is their number. A token belonging to multiple candidates is counted once for each candidate, since its contribution is aggregated separately in each region. Thus, E_+ cannot in general be replaced by N.

Total selector cost. Substituting these terms into Eq. equation 35 gives an input-dependent estimate that can be evaluated from token and candidate counts:

$$\boxed{\begin{aligned} F_{\text{select}} \approx\ & 3D(2N+N_q) + 2NN_qD \\ & + N_s(5N_q-1) + N_q + 4N + 1 \\ & + (2D+1)(E_+ + P_+). \end{aligned}} \tag{44}$$

The corresponding net savings estimate is

$$\begin{aligned} \Delta F \approx\ & (M-d)(N-k)[\gamma + 4D(2T+N+k)] \\ & - \Big[3D(2N+N_q) + 2NN_qD + N_s(5N_q-1) + N_q \\ & + 4N + 1 + (2D+1)(E_+ + P_+)\Big]. \end{aligned} \tag{45}$$

G Query Affinity

The role of α. Eq. equation 6 interpolates between two extremes. At $\alpha = 0$ all tokens carry equal mass and $D(C)$ becomes a purely geometric coverage residual, so the partition ignores the query. As $\alpha \to \infty$ the unit term is negligible, low-affinity tokens stop influencing whether a region needs a second representative, and the allocation degenerates towards query-only Top-k within the structural scaffold. We use $\alpha = 1$ at all models.

Affinity after the first decoder block. Figure 9 compares pairwise cosine similarity distributions at the decoder input H^0 and after the first decoder block H^1. At H^0, the text–audio and text–video distributions are concentrated near zero, whereas within-modality similarities are predominantly positive. At H^1, both cross-modal distributions shift toward higher cosine similarity, indicating increased affinity between text and audio–visual representations after the first block. This observation is consistent with the query-affinity analysis in Section 3.2 and supports computing affinity from H^1 rather than the input embeddings. Together with the pruning-depth constraint in Eq. equation 2, it motivates our choice to compute query affinity and prune immediately after the first decoder block.

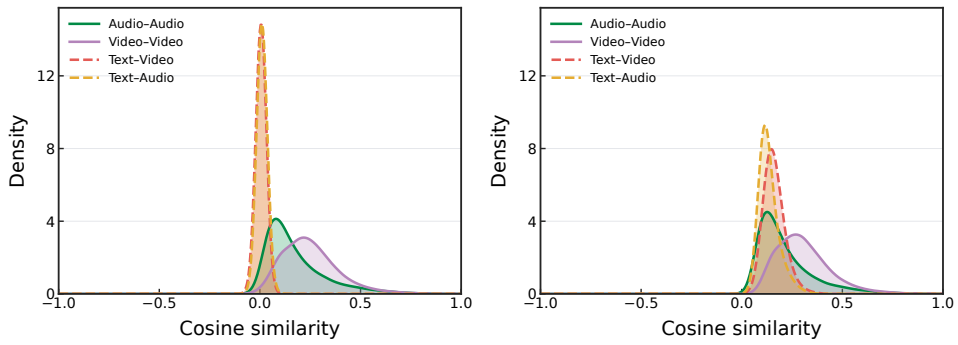


Figure 9: **Modality-pair cosine similarity before and after the first decoder block. Left:** decoder input representations H^0. **Right:** hidden states H^1 after the first decoder block. Solid curves show audio–audio and video–video similarities; dashed curves show text–video and text–audio similarities. Both cross-modal distributions shift toward higher cosine similarity after the first block.

H Lookup and Fused Evaluation

OmniPartition evaluates candidate-region representation errors before iterative partitioning. The candidate regions and their possible splits depend only on token coordinates, whereas their errors depend on the current sample's hidden states and query weights. We first describe the candidate set and the quantities to be evaluated, then derive the data representation used for parallel evaluation.

Candidate region counts. Let P denote the total number of distinct candidate regions, including single-token regions, and let $\mathcal{C} = \{C_0, \ldots, C_{P-1}\}$ denote their set. Each C_c is a set of token indices obtained from an initial audio interval or video frame by the median-coordinate splits in Section 3.4. Regions with the same token set are counted once, regardless of the order of splits; distinct regions may share tokens.

For an audio interval containing $n \geq 1$ tokens at distinct temporal positions, let $P_{\mathrm{aud}}(n)$ denote its number of candidate intervals, including the initial interval. A split at the lower median produces two disjoint intervals of sizes $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$. Hence,

$$\begin{aligned} &P_{\mathrm{aud}}(1) = 1, \\ &P_{\mathrm{aud}}(n) = 1 + P_{\mathrm{aud}}(\lceil n/2 \rceil) + P_{\mathrm{aud}}(\lfloor n/2 \rfloor), \qquad n > 1. \end{aligned} \tag{46}$$

Since descendants of the two disjoint children are distinct, induction gives

$$\begin{aligned} P_{\mathrm{aud}}(n) &= 1 + (2\lceil n/2 \rceil - 1) + (2\lfloor n/2 \rfloor - 1) \\ &= 2n - 1, \qquad n > 1. \end{aligned} \tag{47}$$

The result also holds for $n = 1$. These $2n - 1$ intervals comprise n single-token intervals and $n - 1$ non-singleton intervals, all generated by the prescribed median splits.

For a video frame, let H and W denote the numbers of rows and columns in its complete token grid, with one token at each position. Let $\mathcal{I}_H$ and $\mathcal{I}_W$ denote the one-dimensional interval sets generated by median splits along height and width, respectively. Their sizes are $2H - 1$ and $2W - 1$ by the same one-dimensional argument. A candidate region is a Cartesian product of a height interval

$I_h \in \mathcal{I}_H$ and a width interval $I_w \in \mathcal{I}_W$. Every such product can be obtained by successive height and width splits. Thus, the frame's candidate set is

$$\mathcal{C}_{H,W} = \{I_h \times I_w : I_h \in \mathcal{I}_H,\ I_w \in \mathcal{I}_W\}. \tag{48}$$

Let $P_{\mathrm{vid}}(H, W)$ denote the number of candidates in this frame. Different interval pairs define different token sets on the complete grid, whereas reversing the order of splits leaves the region unchanged. Therefore,

$$P_{\mathrm{vid}}(H, W) = |\mathcal{C}_{H,W}| = (2H - 1)(2W - 1) < 4HW. \tag{49}$$

This expression applies to complete spatial grids. For irregular layouts, the count is the number of distinct token sets obtained by the admissible splits. A 4×4 frame has 49 candidate regions: 16 single-token regions and 33 non-singleton regions. Only the latter require feature aggregation to evaluate their representation errors.

Let $\mathcal{F}$ and $\mathcal{A}$ index the video frames and audio intervals included in precomputation, respectively. Frame $f \in \mathcal{F}$ has a complete $H_f \times W_f$ grid, and interval $a \in \mathcal{A}$ contains n_a tokens at distinct temporal positions. Since the initial regions contain disjoint tokens, the total candidate count is

$$\begin{aligned} P &= \sum_{f \in \mathcal{F}} P_{\mathrm{vid}}(H_f, W_f) + \sum_{a \in \mathcal{A}} P_{\mathrm{aud}}(n_a) \\ &= \sum_{f \in \mathcal{F}} (2H_f - 1)(2W_f - 1) + \sum_{a \in \mathcal{A}} (2n_a - 1). \end{aligned} \tag{50}$$

Region representation error. For this section, index the N audio–visual tokens by $i = 0, \ldots, N-1$, following their order in the feature matrix. Let $x_i \in \mathbb{R}^D$ be the normalized first-layer feature of token i, where D is the hidden dimension, and let w_i be its query weight from Eq. equation 6. Collect these features and weights as

$$X = \begin{bmatrix} x_0^\top \\ \vdots \\ x_{N-1}^\top \end{bmatrix} \in \mathbb{R}^{N \times D}, \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_{N-1} \end{bmatrix} \in \mathbb{R}^N.$$

The matrix B indicates which tokens belong to each candidate region. For candidate C_c, with $c = 0, \ldots, P - 1$, its entries are defined by

$$B \in \{0, 1\}^{P \times N}, \qquad B_{c,i} = \begin{cases} 1, & i \in C_c, \\ 0, & i \notin C_c. \end{cases}$$

The weighted feature sums and total weights of all candidates are

$$S = B\,\mathrm{diag}(\mathbf{w})X \in \mathbb{R}^{P \times D}, \qquad \mathbf{W} = B\mathbf{w} \in \mathbb{R}^P, \tag{51}$$

where $\mathrm{diag}(\mathbf{w})$ denotes the diagonal matrix of token weights. Row $S_{c,:}$ is the transpose of the weighted feature sum s_{C_c}, and $\mathbf{W}_c$ is the total weight W_{C_c}. The vector $\boldsymbol{\delta} \in \mathbb{R}^P$ collects the candidate representation errors:

$$\delta_c = D(C_c) = \mathbf{W}_c - \|S_{c,:}\|_2. \tag{52}$$

Here $D(C_c)$ is the representation error defined in Eq. equation 8. These matrix expressions specify the quantities to compute; the kernel does not materialize B, $\mathrm{diag}(\mathbf{w})$, or the full matrix S.

Region indices and offsets. The arrays I and o specify which tokens belong to each candidate region, expressing the same relation as the matrix B. Let $n_c = |C_c|$ denote the number of tokens in region C_c, and let

$$E = \sum_{c=0}^{P-1} n_c$$

be the total number of tokens counted across all candidate regions. A token occurring in multiple regions contributes once to each region's count.

Concatenating the token indices of $C_0, \ldots, C_{P-1}$ gives

$$I = (I_0, \ldots, I_{E-1}) \in \{0, \ldots, N-1\}^E.$$

Within each region, tokens retain their original sequence order. The subscript j denotes a position in this concatenated array, whereas the value I_j denotes a token index. Thus, I_j identifies the feature row $X_{I_j,:}$ and the scalar weight w_{I_j}.

For example, consider

$$C_0 = \{0, 1\}, \qquad C_1 = \{0\}, \qquad C_2 = \{1\}.$$

Here $P = 3$ and $E = 4$, and the index array is

$$I = \big(\underbrace{0, 1}_{C_0}, \underbrace{0}_{C_1}, \underbrace{1}_{C_2}\big).$$

The entries $I_0 = 0$ and $I_1 = 1$ identify the two tokens in C_0; $I_2 = 0$ identifies the token in C_1, and $I_3 = 1$ identifies the token in C_2. Repeated values therefore indicate that the same token occurs in different regions, while each region contains each of its tokens only once.

The offset array

$$o = (o_0, \ldots, o_P) \in \{0, \ldots, E\}^{P+1}$$

specifies the boundaries between consecutive regions in the concatenated index array I. Since the token indices of $C_0, \ldots, C_{P-1}$ are stored consecutively, the starting position of region C_c equals the total number of indices occupied by the preceding regions $C_0, \ldots, C_{c-1}$. Therefore,

$$\begin{aligned} o_0 &= 0, \\ o_c &= \sum_{u=0}^{c-1} n_u = n_0 + \cdots + n_{c-1}, \qquad 1 \le c \le P. \end{aligned} \tag{53}$$

Here, u indexes the preceding regions, and $n_u = |C_u|$ is the number of tokens in region C_u. The first region starts at position zero. Each subsequent region starts immediately after the previous region, so the same definition can be written recursively as

$$o_{c+1} = o_c + n_c, \qquad 0 \le c < P.$$

Thus, region C_c occupies the n_c consecutive positions

$$o_c,\ o_c + 1,\ \ldots,\ o_{c+1} - 1$$

in I. Its token indices are $I_{o_c}, \ldots, I_{o_{c+1}-1}$, giving

$$C_c = \{I_j : o_c \le j < o_{c+1}\}, \qquad n_c = o_{c+1} - o_c.$$

The offsets are positions in I; the values stored at those positions are token indices.

For the example above, the region sizes are $n_0 = 2$, $n_1 = 1$, and $n_2 = 1$. Their cumulative sizes give

$$\begin{aligned} o_0 &= 0, & o_1 &= n_0 = 2, \\ o_2 &= n_0 + n_1 = 3, & o_3 &= n_0 + n_1 + n_2 = 4. \end{aligned}$$

Consequently,

$$\begin{aligned} o &= (0, 2, 3, 4), \\ I[0:2] = (0, 1), \qquad I[2:3] &= (0), \qquad I[3:4] = (1). \end{aligned} \tag{54}$$

The right endpoint of each slice is excluded. These three slices recover the token indices of C_0, C_1, and C_2, respectively.

There are $P+1$ offsets for P regions because each region requires a starting and an ending boundary, and consecutive regions share a boundary. The final offset

$$o_P = \sum_{u=0}^{P-1} n_u = E$$

marks the end of the entire index array. It does not correspond to an additional region or a valid position from which to read a token index.

Using these indices, the regional quantities in Eq. equation 51 can equivalently be written as

$$S_{c,:} = \sum_{j=o_c}^{o_{c+1}-1} w_{I_j} X_{I_j,:}, \qquad \mathbf{W}_c = \sum_{j=o_c}^{o_{c+1}-1} w_{I_j}. \tag{55}$$

Thus, I and o determine the features and weights used to evaluate each region's representation error.

PyTorch implementation. Algorithm 2 gives a direct PyTorch reference implementation of Eq. equation 52, processing one region at a time. Here $\mathcal{J}_c$ contains the token indices of C_c; X_c and $\mathbf{w}_c$ are its features and weights; $\widetilde{X}_c$ contains the weighted features; and s_c and v_c are their feature sum and total weight. The symbol $\mathbf{1}_m$ denotes a length-m all-ones vector. The expression $\mathbf{w}_c \mathbf{1}_D^\top$ broadcasts each weight across feature dimensions.

Algorithm 2 PyTorch Implementation of Region Representation Error

Require: Features $X \in \mathbb{R}^{N \times D}$, weights $\mathbf{w} \in \mathbb{R}^N$
Require: Token indices $I \in \{0, \dots, N-1\}^E$, offsets $o \in \{0, \dots, E\}^{P+1}$
Ensure: Region errors $\boldsymbol{\delta} \in \mathbb{R}^P$

$\boldsymbol{\delta} \leftarrow \mathbf{0}_P$ # $\boldsymbol{\delta} \in \mathbb{R}^P$
for $c = 0, \dots, P-1$ **do**
  $n_c \leftarrow o_{c+1} - o_c$ # number of tokens in C_c
  $\mathcal{J}_c \leftarrow I[o_c : o_{c+1}]$ # $\mathcal{J}_c \in \{0, \dots, N-1\}^{n_c}$
  if $n_c \le 1$ **then**
    $\delta_c \leftarrow 0$
  else
    $X_c \leftarrow X[\mathcal{J}_c, :]$ # $X_c \in \mathbb{R}^{n_c \times D}$
    $\mathbf{w}_c \leftarrow \mathbf{w}[\mathcal{J}_c]$ # $\mathbf{w}_c \in \mathbb{R}^{n_c}$
    $\widetilde{X}_c \leftarrow X_c \odot (\mathbf{w}_c \mathbf{1}_D^\top)$ # $\widetilde{X}_c \in \mathbb{R}^{n_c \times D}$
    $s_c \leftarrow \widetilde{X}_c^\top \mathbf{1}_{n_c}$ # $s_c \in \mathbb{R}^D$
    $v_c \leftarrow \mathbf{1}_{n_c}^\top \mathbf{w}_c$ # $v_c \in \mathbb{R}$
    $\delta_c \leftarrow v_c - \|s_c\|_2$ # $\delta_c \in \mathbb{R}$
  end if
end for

return $\boldsymbol{\delta}$

For a nonempty candidate set, let $n_{\max} = \max_{0 \le c < P} n_c$. The principal temporary tensor storage is $O(n_{\max} D)$ when intermediates are retained only for the current region, excluding shared inputs, indices, and output errors.

Triton-based fused evaluation. Since δ_c depends only on the features and weights of the tokens in C_c, the errors of different candidates can be evaluated independently. Substituting the weighted sums of Algorithm 2 into Eq. equation 52 expresses each error directly in terms of the shared inputs and the region indices:

$$\delta_c = \sum_{j=o_c}^{o_{c+1}-1} w_{I_j} - \left\| \sum_{j=o_c}^{o_{c+1}-1} w_{I_j} X_{I_j,:} \right\|_2. \tag{56}$$

Eq. equation 56 requires only a running weighted sum and a running total weight while traversing $I_{o_c}, \dots, I_{o_{c+1}-1}$. We therefore assign one region to each parallel program and evaluate the equation in a single pass, so that each region needs only a length-D accumulator and writes a single scalar to $\boldsymbol{\delta}$. The region-specific matrices X_c and $\widetilde{X}_c$ and the sums s_c of Algorithm 2 are never stored.

Both formulations yield the same errors in exact arithmetic and require $O(ED + PD)$ operations, with $O(ED)$ for regional aggregation and $O(PD)$ for the norms. The benefit lies in parallelism

across regions and in avoiding intermediate tensors, not in lower arithmetic complexity. Listing 1 gives the Triton kernel and its calling function.

Listing 1: Triton implementation of Eq. equation 56, with one program per region. `X`, `W`, `Indices`, `Offsets`, and `Out` correspond to X, $\mathbf{w}$, I, o, and $\boldsymbol{\delta}$; `SX` is the row stride of X, and `BD` is the smallest power of two not less than D.

```
import torch
import triton
import triton.language as tl


@triton.jit
def _region_error(
    X, W, Indices, Offsets, Out,
    D: tl.constexpr,
    SX: tl.constexpr,
    BD: tl.constexpr,
):
    # X: [N, D], W: [N], Indices: [E]
    # Offsets: [P+1], Out: [P]
    region = tl.program_id(0)
    begin = tl.load(Offsets + region)
    end = tl.load(Offsets + region + 1)

    if end - begin <= 1:
        tl.store(Out + region, 0.0)
    else:
        columns = tl.arange(0, BD)
        total = tl.full((BD,), 0, tl.float32)  # [BD]
        mass = tl.full((), 0, tl.float32)      # scalar

        for position in range(begin, end):
            token = tl.load(Indices + position)
            weight = tl.load(W + token)
            values = tl.load(
                X + token * SX + columns,
                mask=columns < D, other=0,
            )                                  # [BD]
            total += values * weight
            mass += weight

        norm = tl.sqrt(tl.sum(total * total, axis=0))
        tl.store(Out + region, mass - norm)


def region_error(X, w, indices, offsets):
    # X: [N, D], w: [N], indices: [E], offsets: [P+1]
    assert X.dtype == w.dtype == torch.float32
    assert X.is_cuda and X.stride(1) == 1
    assert w.device == indices.device == offsets.device == X.device
    assert all(t.is_contiguous() for t in (w, indices, offsets))

    P = offsets.numel() - 1
    delta = torch.empty(P, device=X.device, dtype=torch.float32)
    if P == 0:
        return delta                           # [0]

    D = X.shape[1]
    _region_error[(P,)](
        X, w, indices, offsets, delta,
        D=D,
        SX=X.stride(0),
        BD=triton.next_power_of_2(D),
        num_warps=8,
```

```
        enable_fp_fusion=False,
    )
    return delta                                    # [P]
```

I ADDITIONAL RESULTS

Uniform Sampling as a Strong Baseline. Table 7 provides detailed results supporting Figure 4. Before the tokens enter the LLM, uniform stride selects visual tokens at evenly spaced positions within each video frame and audio tokens at evenly spaced positions within each audio segment. Despite its simplicity, this baseline achieves performance comparable to or better than OmniZip and OmniSelect, with higher mean performance retention at all four evaluated retention ratios. At 45% retention, it matches OmniSelect on DailyOmni (60.65) and improves OmniVideoBench from 33.10 to 34.10. At 20% retention, it outperforms both methods on both benchmarks, reaching 91.22% mean performance retention. These results suggest that preserving spatial and temporal coverage is an important complement to token relevance, motivating the structure-aware selection design of OmniPartition.

Table 7: Uniform stride sampling versus OmniZip and OmniSelect on Qwen2.5-Omni-7B with 128 input frames at different audio–visual token-retention ratios. Uniform stride downsamples tokens within each video frame and audio segment before they enter the LLM.

Method	Retained	DailyOmni	OmniVideoBench	Mean Ret. (%)
Full Tokens	100%	64.16	34.60	100.00%
OmniZip	45%	59.98	33.30	94.86%
OmniSelect	45%	**60.65**	33.10	95.10%
uniform stride	45%	**60.65**	**34.10**	**96.54%**
OmniZip	30%	55.97	**32.80**	91.02%
OmniSelect	30%	57.39	32.70	91.98%
uniform stride	30%	**58.98**	32.70	**93.22%**
OmniZip	20%	54.72	30.60	86.86%
OmniSelect	20%	50.28	29.60	81.96%
uniform stride	20%	**55.31**	**33.30**	**91.22%**
OmniZip	10%	**48.54**	28.60	79.16%
OmniSelect	10%	45.11	28.90	76.92%
uniform stride	10%	47.70	**29.90**	**80.38%**

I.1 DETAILED PERFORMANCE RESULTS

Tables 8, 9, 10, 11, and 12 provide detailed results complementing the main-paper comparisons. These tables include category-level performance on DailyOmni, WorldSense, ShortVid-Bench, and AVUTBench, as well as additional OmniVideoBench results, covering different model scales and token-retention ratios.

I.2 WORLDSENSE WITH FEWER SAMPLED FRAMES

Table 13 reports WorldSense results using Qwen2.5-Omni-3B with 32 and 64 sampled frames. OmniPartition achieves the highest average accuracy among the evaluated compression methods at both frame budgets and token-retention ratios. At 45% and 30% token retention, it improves over OmniSelect by 1.07 and 0.16 points with 32 frames, and by 0.95 and 0.41 points with 64 frames, respectively. With 32 frames and 45% token retention, OmniPartition achieves 43.44 average accuracy compared with 43.66 for full-token inference, preserving 99.50% of its performance. These results demonstrate that OmniPartition remains effective under reduced frame budgets.

Table 8: DailyOmni category results corresponding to Table 1, using Qwen2.5-Omni-3B and Qwen2.5-Omni-7B with 128 frames. Category abbreviations: Con. = Context Understanding, Event = Event Sequence, AV Event = Audio–Visual Event Alignment, Com. = Comparative, Inf. = Inference, and Rea. = Reasoning.

Method	Retained	Con.	Event	AV Event	Com.	Inf.	Rea.	Avg.
Qwen2.5-Omni-7B								
Full Tokens	100%	56.99	60.13	50.84	71.76	78.57	78.86	64.16
Random	55%	52.33	49.67	41.48	70.99	72.73	71.43	56.89
DyCoke (V&A)	50%	50.26	52.94	44.96	70.23	77.92	76.57	59.48
OmniZip	45%	53.89	51.96	46.22	70.23	**77.27**	76.57	59.98
OmniSelect	45%	53.37	**53.92**	**46.64**	**72.52**	76.62	76.57	**60.65**
OmniPartition	45%	56.48	52.29	46.64	71.76	77.92	76.57	60.82
OmniZip	30%	47.15	47.39	43.70	63.36	76.62	73.71	55.97
OmniSelect	30%	48.19	**50.00**	**46.22**	67.18	74.68	73.14	57.39
OmniPartition	30%	54.40	49.02	48.32	69.47	79.87	74.86	59.73
Qwen2.5-Omni-3B								
Full Tokens	100%	55.44	56.21	53.78	70.99	79.22	74.29	62.82
Random	55%	47.67	44.44	45.38	64.12	71.43	63.43	53.55
DyCoke (V&A)	50%	48.19	49.35	47.48	65.65	72.73	65.14	55.89
OmniZip	45%	47.15	47.71	46.64	64.12	**76.62**	69.71	56.14
OmniSelect	45%	**50.78**	49.67	**52.10**	65.65	**76.62**	69.14	**58.06**
OmniPartition	45%	48.70	**50.00**	49.58	**69.47**	75.32	**70.29**	**58.06**
OmniZip	30%	**46.63**	46.08	42.86	59.54	68.18	**68.00**	53.05
OmniSelect	30%	**46.63**	45.42	**48.32**	62.60	**72.08**	65.14	54.39
OmniPartition	30%	46.11	**48.04**	47.06	**67.18**	71.43	64.57	**55.05**

Table 9: Detailed performance on OmniVideoBench and DailyOmni using Qwen2.5-Omni-3B and Qwen2.5-Omni-7B with 128 input frames at 10% and 20% token-retention ratios.

Method	Retained Ratio (%)	OmniVideo Bench (↑)	DailyOmni (↑)							Mean Ret. (%)
			Con.	Event	AV Event	Com.	Inf.	Rea.	Avg.	
Qwen2.5-Omni-3B										
Full Tokens	100%	32.9	55.44	56.21	53.78	70.99	79.22	74.29	62.82	100.00%
OmniZip	20%	30.5	45.08	42.16	41.60	60.31	66.23	63.43	51.80	87.58%
OmniSelect	20%	30.0	44.56	39.22	40.76	54.20	64.94	63.43	48.87	84.47%
OmniPartition	20%	**31.3**	**46.11**	**47.71**	**45.38**	**65.65**	**66.88**	**65.14**	**53.97**	**90.52%**
OmniZip	10%	28.9	**41.97**	34.64	38.24	59.54	57.79	50.86	44.61	79.43%
OmniSelect	10%	29.1	35.23	**35.62**	32.77	49.62	50.65	46.29	40.02	76.06%
OmniPartition	10%	**31.0**	39.38	**35.62**	**38.66**	**60.31**	**61.04**	**53.71**	**45.45**	**83.28%**
Qwen2.5-Omni-7B										
Full Tokens	100%	34.6	56.99	60.13	50.84	71.76	78.57	78.86	64.16	100.00%
OmniZip	20%	30.6	45.08	**46.73**	**44.96**	62.60	70.78	72.57	54.72	86.86%
OmniSelect	20%	29.6	39.57	45.18	38.38	54.55	67.89	68.61	50.28	81.94%
OmniPartition	20%	**34.6**	**52.33**	**46.73**	43.28	**66.41**	**77.27**	**74.86**	**57.14**	**94.53%**
OmniZip	10%	28.6	45.60	39.54	**41.60**	57.25	61.04	59.43	48.54	79.16%
OmniSelect	10%	28.9	40.93	36.27	36.13	54.96	59.74	57.14	45.11	76.90%
OmniPartition	10%	30.3	46.11	44.12	41.18	64.89	72.73	70.29	53.63	85.58%

Table 10: WorldSense results using Qwen2.5-Omni-7B with 128 input frames across multiple token-retention ratios.

Method	Retained Ratio	Tech & Science	Culture & Politics	Daily Life	Film & TV	Performance	Games	Sports	Music	Avg. (↑)	Mean Ret.(%)
Qwen2.5-Omni-7B											
Full Tokens	100%	49.39	53.72	46.96	44.59	45.69	42.49	42.56	48.03	46.82	100.00%
Random	55%	47.76	49.19	42.86	40.90	38.95	36.48	38.14	48.28	43.25	92.38%
DyCoke (V&A)	50%	46.73	48.54	43.62	39.58	41.57	40.77	41.86	47.29	43.95	93.87%
OmniZip	45%	47.35	49.51	45.14	42.74	**43.07**	41.20	42.33	**49.01**	45.27	96.69%
OmniSelect	45%	47.96	50.81	46.20	**43.01**	**43.07**	**43.78**	**43.72**	48.03	**46.00**	**98.25%**
OmniPartition	45%	**49.18**	**52.75**	**46.96**	42.48	**43.07**	43.35	41.40	48.28	**46.15**	**98.58%**
OmniZip	30%	45.92	46.93	43.92	**43.27**	40.07	39.91	40.23	**48.03**	43.85	93.66%
OmniSelect	30%	46.33	46.93	44.38	42.22	**43.45**	39.91	**42.79**	45.32	44.17	94.34%
OmniPartition	30%	**46.94**	**50.81**	**46.05**	42.22	40.07	**42.49**	41.63	47.04	**44.96**	**96.03%**
OmniZip	20%	45.10	44.98	42.40	36.68	37.83	38.63	38.84	44.58	41.52	88.68%
OmniSelect	20%	41.92	39.91	40.79	**40.73**	38.12	34.27	38.18	42.91	40.06	85.56%
OmniPartition	20%	**48.37**	**49.84**	**45.74**	40.37	**39.70**	**42.49**	**40.70**	**46.55**	**44.58**	**95.22%**
OmniZip	10%	41.02	37.22	38.15	34.83	35.58	34.76	**36.74**	41.13	37.83	80.80%
OmniSelect	10%	40.82	36.57	34.95	31.66	31.46	32.19	34.42	39.90	35.69	76.23%
OmniPartition	10%	**43.67**	**43.69**	**44.38**	**40.37**	**35.96**	**36.05**	34.65	**41.38**	**40.70**	**86.93%**

Table 11: ShortVid-Bench results using Qwen2.5-Omni-3B and Qwen2.5-Omni-7B with 128 frames. Category abbreviations: CIT = Creator Intent Taxonomy, HMD = Humor & Meme Deconstruction, NC = Narrative Comprehension, CIA = Creative Innovation Analysis, AIC = Affective Intent Classification, and TRL = Temporal Reasoning Localization.

Method	Retained Ratio	CIT	HMD	NC	CIA	AIC	TRL	Avg.	Avg. Ret. (%)
Qwen2.5-Omni-3B									
Full Tokens	100%	67.70	63.70	74.14	77.57	74.16	59.52	70.30	100.00%
OmniZip	20%	67.70	59.59	71.55	75.70	70.79	53.57	67.80	96.44%
OmniSelect	20%	64.60	57.53	70.98	70.09	67.42	50.00	65.40	93.03%
OmniPartition	20%	64.16	58.22	72.99	71.03	73.03	59.52	67.50	96.02%
OmniZip	10%	61.50	57.53	67.24	71.96	67.42	52.38	63.80	90.75%
OmniSelect	10%	56.64	56.16	69.54	59.81	59.55	48.81	61.00	86.77%
OmniPartition	10%	61.95	57.53	70.11	66.36	66.29	53.57	64.30	91.47%
Qwen2.5-Omni-7B									
Full Tokens	100%	72.1	64.4	74.7	74.8	67.4	57.1	70.5	100.00%
OmniZip	20%	68.1	**63.7**	72.1	**73.8**	**65.2**	54.8	68.1	96.60%
OmniSelect	20%	61.1	58.9	70.7	69.2	58.4	54.8	64.2	91.06%
OmniPartition	20%	**68.58**	62.33	**76.44**	72.90	64.04	**57.14**	**69.50**	**98.58%**
OmniZip	10%	62.4	61.0	69.5	**70.1**	**67.4**	54.8	65.3	92.62%
OmniSelect	10%	55.3	58.9	67.8	65.4	57.3	52.4	61.2	86.81%
OmniPartition	10%	**62.83**	**63.01**	**69.83**	70.09	61.80	**58.33**	**65.60**	**93.05%**

Table 12: AVUTBench category accuracy using Qwen2.5-Omni-3B and Qwen2.5-Omni-7B with 128 frames. Category abbreviations: AEL = Audio Event Location, AOM = Audio Object Matching, AOCM = Audio OCR Matching, AIE = Audio Information Extraction, ACM = Audio Character Matching, and ACC = Audio Content Counting.

Method	Retained	AEL	AOM	AOCM	AIE	ACM	ACC	Avg.	Mean Ret. (%)
Qwen2.5-Omni-3B									
Full Tokens	100%	31.76	57.65	65.27	85.58	62.11	46.61	62.05	100.00%
OmniZip	20%	29.41	53.32	60.13	72.70	57.55	36.44	55.71	89.78%
OmniSelect	20%	28.24	49.49	58.20	67.18	53.48	39.83	52.60	84.76%
OmniPartition	20%	30.59	54.59	59.49	82.21	59.23	40.68	58.48	94.24%
OmniZip	10%	25.88	49.49	58.84	57.36	54.44	40.68	50.92	82.07%
OmniSelect	10%	27.65	42.09	51.77	46.32	43.88	33.05	43.02	69.33%
OmniPartition	10%	35.29	48.47	56.59	75.77	54.68	38.98	54.61	88.01%
Qwen2.5-Omni-7B									
Full Tokens	100%	36.47	58.16	69.13	86.20	68.35	41.53	64.59	100.00%
OmniZip	20%	**38.82**	50.77	65.27	71.47	**61.15**	34.75	57.50	89.02%
OmniSelect	20%	35.88	47.96	59.81	67.48	55.64	33.05	53.40	82.68%
OmniPartition	20%	35.29	**52.30**	**66.56**	**84.97**	60.43	**38.98**	**60.38**	**93.48%**
OmniZip	10%	36.47	46.94	**60.13**	59.20	**59.95**	29.66	52.54	81.34%
OmniSelect	10%	**40.00**	44.90	52.73	51.53	46.76	23.73	46.08	71.34%
OmniPartition	10%	35.88	**47.19**	59.81	**78.53**	53.72	**39.83**	**55.31**	**85.63%**

Table 13: WorldSense results using Qwen2.5-Omni-3B with 32- and 64-frame budgets.

Method	Retained Ratio	Tech & Science	Culture & Politics	Daily Life	Film & TV	Performance	Games	Sports	Music	Avg. (↑)	Ret. (%)
Frame Budgets 32											
Full Tokens	100%	49.39	47.57	44.07	42.48	38.20	37.77	39.77	45.32	43.66	100.00%
OmniZip	45%	47.35	42.39	41.64	39.58	36.33	37.77	37.67	43.10	41.27	94.53%
OmniSelect	45%	46.94	44.98	42.86	40.90	37.45	38.20	39.30	**44.33**	42.37	97.05%
OmniPartition	45%	**48.16**	**46.93**	**43.16**	**42.48**	**40.07**	**40.77**	**40.93**	42.86	**43.44**	**99.50%**
OmniZip	30%	46.12	38.83	39.51	36.15	34.83	39.48	35.35	44.58	39.75	91.04%
OmniSelect	30%	46.94	**45.95**	**41.19**	**40.63**	**37.83**	36.48	36.74	**47.04**	41.99	96.17%
OmniPartition	30%	**48.16**	**45.95**	**41.19**	40.11	36.70	**39.91**	**39.77**	42.86	**42.15**	**96.54%**
Frame Budgets 64											
Full Tokens	100%	51.02	50.16	43.62	42.22	43.45	39.48	41.63	45.32	44.86	100.00%
OmniZip	45%	48.78	44.66	43.16	40.11	39.33	38.63	38.84	**45.32**	42.84	95.50%
OmniSelect	45%	48.78	46.93	43.62	41.69	37.83	38.20	39.53	44.58	43.19	96.28%
OmniPartition	45%	**49.80**	**49.19**	**43.77**	**42.74**	**40.07**	**40.34**	**41.16**	43.35	**44.14**	**98.39%**
OmniZip	30%	46.33	42.07	40.12	37.47	37.08	39.06	36.05	44.33	40.61	90.53%
OmniSelect	30%	**48.78**	**46.28**	41.64	39.84	37.83	**41.63**	**38.37**	44.33	42.56	94.87%
OmniPartition	30%	48.16	45.95	**41.79**	**42.48**	**40.07**	40.77	**38.37**	**44.83**	**42.97**	**95.79%**

I.3 Generalization to Baichuan-Omni.

Table 14 complements the main-paper generalization results with evaluations on Baichuan-Omni-1.5-7B. OmniPartition achieves the highest mean performance retention among the evaluated compression methods at both 20% and 10% token retention, reaching 88.19% and 82.39%, respectively. At 20% retention, it matches or outperforms both baselines on all three benchmarks. At 10%, it achieves the best results on AVUTBench and ShortVid-Bench while remaining close to OmniZip on DailyOmni (43.69 vs. 43.78). These results further support the generalizability of OmniPartition beyond Qwen2.5-Omni under tight token budgets.

Table 14: Baichuan-Omni-1.5 7B results on DailyOmni, AVUTBench, and ShortVid-Bench.

Method	Retained	DailyOmni	AVUTBench	ShortVidBench	Mean Ret. (%)
Baichuan-Omni-1.5 7B					
Full Tokens	100%	55.97	60.55	71.20	100.00%
OmniZip	20%	46.78	**50.87**	67.10	87.28%
OmniSelect	20%	44.11	46.83	66.40	83.14%
OmniPartition	20%	**47.37**	**50.87**	**68.30**	**88.19%**
OmniZip	10%	**43.78**	45.33	63.90	80.94%
OmniSelect	10%	42.36	43.31	64.80	79.41%
OmniPartition	10%	43.69	**45.67**	**66.70**	**82.39%**

I.4 Hyperparameter Ablation

Table 15 evaluates the additive constant and query-affinity coefficient in the token-weight formulation using Qwen2.5-Omni-7B with 128 frames and 30% token retention. The default setting in Eq. 6, $w_i = 1 + \alpha\mathcal{R}(q_i)$ with $\alpha = 1$, achieves the highest mean performance retention of 96.84%. Removing the constant term lowers mean retention to 96.38%, suggesting that allowing low-affinity tokens to contribute to region partitioning is beneficial. Increasing the affinity coefficient to $\alpha = 5$ improves performance on DailyOmni and ShortVid-Bench but reduces OVB accuracy, yielding a lower mean retention of 96.33%. These results support the default setting as an effective balance between query relevance and structural coverage, with the best aggregate performance among the tested configurations.

Table 15: HyperParameter ablation with Qwen2.5-Omni-7B, 128 frames, and 30% retention.

Variant	DailyOmni	OVB	ShortVid	Mean Ret. (%)
Full Tokens	64.16	34.60	70.50	100.00%
OmniPartition ($w_i = 1 + \mathcal{R}(q_i)$)	59.73	34.10	69.70	**96.84%**
$w_i = \mathcal{R}(q_i)$	59.40	33.80	69.70	96.38%
$w_i = 0.5 + \mathcal{R}(q_i)$	59.57	32.70	69.20	95.17%
$w_i = 1.5 + \mathcal{R}(q_i)$	59.65	33.80	69.20	96.27%
$w_i = 2 + \mathcal{R}(q_i)$	59.40	**34.30**	69.50	96.77%
$w_i = 1 + 2\mathcal{R}(q_i)$	59.57	32.70	69.20	95.17%
$w_i = 1 + 5\mathcal{R}(q_i)$	**60.32**	33.20	**69.80**	96.33%
$w_i = 1 + 10\mathcal{R}(q_i)$	59.48	33.30	69.70	95.94%

I.5 Efficiency on a Smaller Backbone

Table 16 extends the efficiency analysis to Qwen2.5-Omni-3B on WorldSense using an NVIDIA H100 GPU. OmniPartition achieves increasing speedups as token retention decreases, consistent with the 7B results. At 20% and 10% retention, its Triton implementation reduces total prefill latency from 229.36 ms to 91.53 ms and 87.56 ms, corresponding to 2.51× and 2.62× speedups,

respectively. Peak GPU memory decreases from 14.45 GiB to 10.23 GiB at both budgets. In comparison, OmniZip achieves only $1.07\times$ and $0.97\times$ speedups because its attention extraction and pruning overhead offset much of the savings from token reduction. OmniSelect remains slower than full-token inference due to the additional AudioCLIP computation. These results show that OmniPartition also delivers practical efficiency gains on the smaller backbone. All latency measurements include pruning overhead and cover prefill only, excluding subsequent decoding.

Table 16: H100 prefill efficiency for Qwen2.5-Omni-3B on WorldSense. Latencies are in milliseconds, including pruning overhead in Total. Speedup is relative to Full Tokens.

Method	Ret.	Peak Mem. (GiB)	LLM Prefill	Extra attn.	Prune	Total	Speedup
Qwen2.5-Omni-3B (Model: 8.77 GiB)							
Full Tokens	100%	14.45	229.36	–	–	229.36	1.00×
OmniZip	45%	12.04	105.37	12.02	75.92	193.32	1.19×
OmniSelect	45%	12.11	99.82	13.06	21.98 (+1620.88)	1755.75	0.13×
OmniPartition (PyTorch)	45%	11.42	94.79	–	51.92	146.71	1.56×
OmniPartition (Triton)	45%	11.42	94.71	–	51.15	145.86	1.57×
OmniZip	30%	11.29	83.49	12.09	85.71	181.30	1.27×
OmniSelect	30%	11.31	88.93	12.30	26.37 (+1678.20)	1805.81	0.13×
OmniPartition (PyTorch)	30%	10.59	64.41	–	47.19	111.60	2.06×
OmniPartition (Triton)	30%	10.59	64.76	–	46.43	111.20	2.06×
OmniZip	20%	10.22	71.96	13.12	129.77	214.85	1.07×
OmniSelect	20%	10.76	85.79	15.64	27.85 (+1914.87)	2044.16	0.11×
OmniPartition (PyTorch)	20%	10.23	51.31	–	47.56	98.87	2.32×
OmniPartition (Triton)	20%	10.23	50.58	–	40.95	91.53	2.51×
OmniZip	10%	10.00	86.29	12.78	136.22	235.30	0.97×
OmniSelect	10%	10.67	65.54	12.72	20.75 (+1593.76)	1692.76	0.14×
OmniPartition (PyTorch)	10%	10.23	46.16	–	42.96	89.13	2.57×
OmniPartition (Triton)	10%	10.23	47.83	–	39.73	87.56	2.62×