FLASHGAZE: TRAINING-FREE MULTI-SCALE PATCH PRUNING FOR EFFICIENT VIDEO UNDERSTANDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal Large Language Models (MLLMs) have demonstrated strong performance in video understanding, yet efficiently processing long, high-resolution videos remains challenging. Such videos often contain substantial spatiotemporal redundancy, and processing redundant visual tokens can incur avoidable computational overhead. Many existing methods prune visual tokens during or after vision transformer (ViT) encoding, leaving much of the encoding cost unaddressed. Some approaches prune patches before encoding but rely on learned auxiliary networks for patch selection, incurring additional training and inference overhead. To address these limitations, we propose FlashGaze, a training-free method that reduces spatiotemporal redundancy before ViT encoding without introducing auxiliary networks. FlashGaze uses pixel-space differences as a proxy for information loss and employs Quadtree Dynamic Programming to jointly optimize patch dropping, merging, and keeping under a fixed budget. Experiments on two MLLM backbones across multiple benchmarks demonstrate substantial efficiency gains while largely preserving accuracy. On Qwen3-VL-8B, FlashGaze retains **98%** of the full-input baseline accuracy on LongVideoBench while achieving up to **5.4**$\times$ and **17**$\times$ speedups in ViT encoding and MLLM prefill, respectively, and reducing peak GPU memory usage by a factor of **1.8**. These efficiency gains enable the model to process videos with more frames and higher resolutions on the same GPU hardware, unlocking video understanding at scales previously out of reach.

1 INTRODUCTION

Recent advances in multimodal large language models (MLLMs) have demonstrated strong capabilities in video understanding (Qian et al., 2024; Bai et al., 2025a; Zhang et al., 2024), with growing attention to long-form and high-resolution scenarios (Mangalam et al., 2023; Wu et al., 2024; Zhou et al., 2025; Fu et al., 2025; Hu et al., 2026). Emerging applications such as embodied intelligence and autonomous driving further demand accurate, low-latency processing of such videos. However, long, high-resolution videos produce lengthy visual token sequences that incur substantial computational and memory overhead and may exceed the context limits of MLLMs, hindering their deployment in real-world applications.

MLLMs typically process all visual tokens in a video, despite substantial temporal redundancy from static content across frames and spatial redundancy from large low-information regions. These redundant tokens impose considerable unnecessary computation. Existing methods therefore seek to reduce visual tokens while preserving informative content. For example, ToMe (Bolya et al., 2022) reduces tokens within the ViT, VisionZip (Yang et al., 2025) compresses tokens after the ViT, and FastV (Chen et al., 2024) prunes tokens inside the MLLM.

Although these methods have shown promising results, as illustrated in Fig. 1, they prune visual tokens only before or inside the MLLM. Consequently, part or all of the ViT, and even several MLLM layers in some cases, still process the full set of visual tokens. The later token pruning is applied, the more unnecessary computation redundant tokens incur, limiting the ability of MLLMs to scale to longer and higher-resolution videos. Another line of work explores token pruning at the video input stage. AutoGaze (Shi et al., 2026), for example, employs an autoregressive network to select visual patches, removing redundant ones before the ViT. However, such methods typically introduce additional network components and may rely on specially constructed training data, resulting

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.

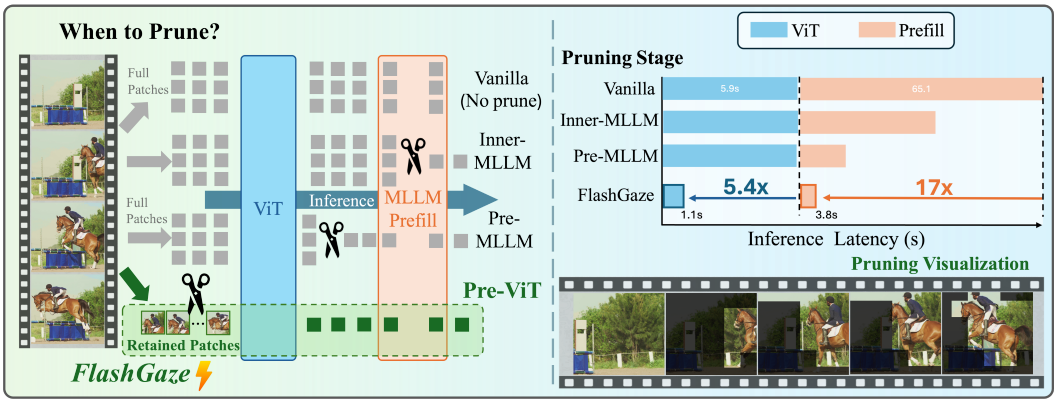

Figure 1: We propose FlashGaze, a training-free method that enables MLLMs to process 1.5K-frame and 4K-resolution videos. Existing methods prune visual tokens before or inside the MLLM, leaving substantial redundant computation in the ViT. In contrast, FlashGaze prunes patches directly at the video input stage, retaining **98%** of the full-input baseline accuracy while speeding up the ViT and prefill stages by up to **5.4**× and **17**×, respectively. We also visualize the pruning results, showing FlashGaze selectively retains informative visual patches.

in non-negligible training and inference overhead. In contrast, we observe that temporal and spatial redundancy can be assessed directly in pixel space, as static regions exhibit minimal pixel variation across frames and low-information areas display high local similarity, enabling effective patch selection before the ViT without a learned selector.

Based on this observation, we propose FlashGaze, a training-free method that prunes visual patches directly at the video input stage. For each patch, FlashGaze considers three actions: dropping it, merging it with neighboring patches, or retaining it at its original resolution. FlashGaze estimates the cost of each action in pixel space: the dropping cost is measured by the difference between the current patch and its corresponding historical patch, while the merging cost is measured by the difference before and after merging. A quadtree naturally captures the hierarchical structure of multi-scale merging, and dynamic programming (DP) efficiently solves the selection problem under a given patch budget. FlashGaze thus employs Quadtree DP to determine the action assignment, dropping temporally redundant patches while allocating fine-scale patches to detail-rich areas and coarse-scale patches to low-information areas (Fig. 2).

We evaluate FlashGaze on two representative MLLMs, Qwen3-VL-8B and NVILA-8B-Video, across a range of video understanding benchmarks. Notably, FlashGaze retains 98% of the full-input baseline accuracy while accelerating the ViT and prefill stages by up to $5.4\times$ and $17\times$, respectively.

Our major contributions are as follows:

- We propose FlashGaze, a training-free method that performs patch pruning at the video input stage via Quadtree DP, guided by pixel-space dropping and merging costs, to simultaneously reduce temporal and spatial redundancy.
- Extensive experiments on two representative MLLMs and multiple video understanding benchmarks demonstrate the effectiveness and efficiency of FlashGaze, while unlocking longer visual contexts for improved video understanding.
- Our analysis reveals that FlashGaze largely preserves visual representations, enables MLLMs to allocate more attention to dynamic visual tokens, and assigns fine-scale patches to information-rich regions, thereby maintaining high accuracy.

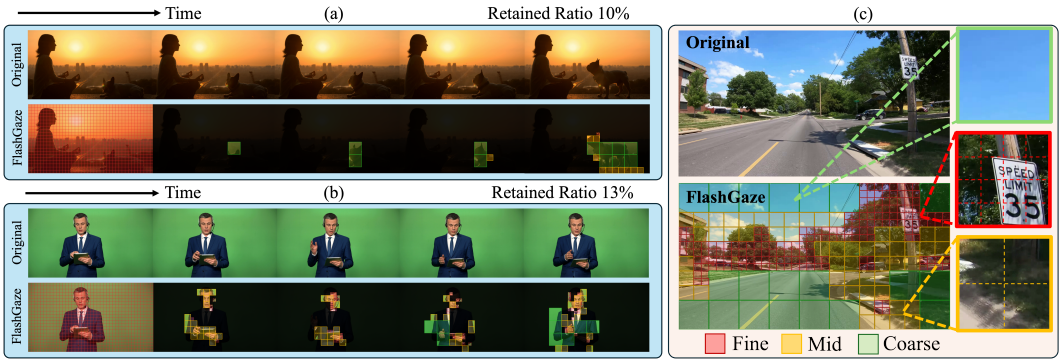

Figure 2: **Visualization of pruned videos.** FlashGaze mainly retains patches in regions that change over time in (a) and (b). In the continuously changing driving scene in (c), FlashGaze uses coarse-scale patches for low-information areas such as the sky, while preserving fine-scale patches for detail-rich regions such as traffic signs.

2 Related Work

2.1 Long-Form Video Understanding

Long-form video understanding requires integrating information over extended temporal horizons (Mangalam et al., 2023; Wu et al., 2024; Chen et al., 2026; Xie et al., 2026; Rege et al., 2026), yet processing longer videos substantially increases the number of visual tokens (Chen et al., 2026), raising computational and memory costs and potentially exceeding the context capacity of MLLMs. Prior work addresses this by extending context capacity (Chen et al., 2025), maintaining compact memories (Song et al., 2024; He et al., 2024), or selectively retrieving relevant visual evidence (Wang et al., 2024). These methods improve how long-range information is represented or accessed, but efficiently processing the underlying visual input remains a challenge when dense sampling introduces substantial spatiotemporal redundancy.

2.2 Visual Token Pruning and Compression

Visual token reduction can occur at several stages. DynamicViT and ToMe prune or merge tokens within vision encoders (Rao et al., 2021; Bolya et al., 2022); VisionZip compresses encoder outputs (Yang et al., 2025), while FastV prunes tokens within the language model (Chen et al., 2024). These approaches reduce downstream computation, leaving part or all of the encoding cost unaddressed. Video methods further exploit temporal redundancy: LongVU, VideoChat-Flash, and FastVID compress visual tokens after encoding (Shen et al., 2024; Li et al., 2026; Shen et al., 2025), while TimeChat-Online removes inter-frame redundancy (Yao et al., 2025). Addressing encoding cost directly, STC reuses encoder features (Wang et al., 2026c) and EarlyTom applies compression early within the encoder (Wang et al., 2026a). Selection before encoding avoids processing redundant content altogether: PS3 selectively encodes high-resolution regions (Shi et al., 2025), PixelPrune is limited to static document and GUI images (Wang et al., 2026b), and AutoGaze learns to select multi-scale patches autoregressively (Shi et al., 2026). In contrast, FlashGaze jointly reduces temporal and spatial redundancy with pixel-space costs and Quadtree DP, without training or auxiliary networks

3 Method

3.1 Problem Formulation

Scaling MLLMs to long, high-resolution videos calls for reducing redundant computation not only in the LLM backbone, but also in the vision encoder. We therefore target spatiotemporal redundancy at the video input stage. The central challenge is to decide which regions require new visual information and at what spatial resolution, without relying on encoded features or a learned selector. We

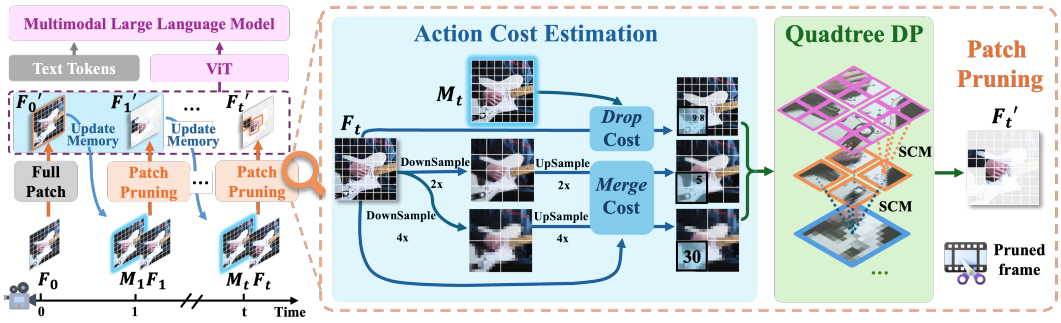

Figure 3: **Processing pipeline of FlashGaze.** FlashGaze performs patch pruning on video frames sequentially, based on visual memory and current frames (left). During patch pruning, FlashGaze estimates the costs of Drop and Merge using current frame F_t and memory M_t (middle). Given the costs and budget, FlashGaze determines the action assignment via Quadtree DP (right).

formulate this challenge as budget-constrained patch selection. Given a video frame F_t at time t and a patch budget $\mathcal{B}$, we seek a compressed representation F'_t that minimizes information loss while retaining at most $\mathcal{B}$ patches:

$$F'^{*}_t = \arg\min_{F'_t} \mathcal{L}(F'_t, F_t) \quad \text{s.t.} \quad N_t \leq \mathcal{B}, \tag{1}$$

where F'^{*}_t denotes a compressed representation obtained under this objective, N_t the number of retained patches in F'_t, and $\mathcal{L}$ the information discrepancy between the compressed representation and the original frame.

3.2 FlashGaze

To instantiate this formulation, FlashGaze constructs F'_t through a joint assignment of three patch-level actions: Drop, Merge, and Keep. Drop omits temporally redundant patches, Merge represents neighboring patches at a coarser spatial scale, and Keep preserves patches at their original resolution. We represent the action assignment for frame F_t as a set $\mathcal{A}_t$ of patch-action pairs (p, a).

We use the sum of pixel-space action costs as a measure of the information discrepancy in Eq. 1. We further introduce a patch-count penalty λN_t to favor more compact representations without using the full budget $\mathcal{B}$. This yields the following regularized objective:

$$\mathcal{A}^*_t = \arg\min_{\mathcal{A}_t} \left[\sum_{(p,a)\in\mathcal{A}_t} \text{Cost}(p, a) + \lambda N_t \right], \quad \text{s.t.} \quad N_t \leq \mathcal{B}. \tag{2}$$

Here, N_t denotes the number of patches retained under $\mathcal{A}_t$, and λ controls the trade-off between information preservation and the number of retained patches.

As shown in Fig. 3, FlashGaze performs patch pruning on video frames sequentially. To provide historical information, FlashGaze maintains a visual memory M_t at each time step. At each spatial location, M_t stores the most recently retained patch from previously processed frames. The first frame F_0 is fully retained, with $F'_0 = F_0$, and initializes the visual memory M_1. This initialization frame is exempt from the patch budget. For each subsequent frame F_t ($t \geq 1$), FlashGaze uses M_t as a reference to prune F_t and obtain F'_t. After obtaining F'_t, its retained patches update corresponding locations in the memory, producing M_{t+1} for the next frame.

3.3 Patch Pruning

During patch pruning, FlashGaze estimates the costs of different actions for each patch based on the current frame F_t and the visual memory M_t. Given these action costs and the budget, FlashGaze solves Eq. 2 via Quadtree DP to determine an action assignment.

Action Cost Estimation To reduce temporal redundancy, we estimate the dropping cost of each patch using the historical visual information stored in the visual memory. For a basic patch p_{f}, we compare its content in the current frame F_t with the corresponding content in the visual memory M_t to obtain $\mathrm{Cost}(p_{\mathrm{f}}, \mathrm{Drop})$:

$$\mathrm{Cost}(p_{\mathrm{f}}, \mathrm{Drop}) = D_{\mathrm{pix}}(F_t(p_{\mathrm{f}}), M_t(p_{\mathrm{f}})), \tag{3}$$

where D_{pix} denotes the pixel-wise difference between two aligned regions of the same size. To mitigate the effects of slight camera motion, we divide each region into small blocks and search a local neighborhood in the reference image for the displacement that minimizes the mean pixel-wise difference within each block. We then align the reference region accordingly and compute the patch-level difference. $\mathrm{Cost}(p_{\mathrm{f}}, \mathrm{Drop})$ reflects the difference between the current patch and the historical patch, where a higher value indicates a larger difference.

To reduce spatial redundancy, we evaluate the information loss caused by merging neighboring patches. We consider two spatial merge scales, $s \in \{\mathrm{m}, \mathrm{c}\}$, where smaller and larger groups of neighboring fine-scale patches are merged into p_{m} and p_{c}, respectively. For each scale s, we spatially downsample F_t and then upsample the result to the original resolution, obtaining the reconstructed frame $\widehat{F}_t^{(s)}$. Similar to the dropping cost, the merging cost is computed as follows:

$$\mathrm{Cost}(p_s, \mathrm{Merge}) = D_{\mathrm{pix}}\Big(F_t(p_s), \widehat{F}_t^{(s)}(p_s)\Big), \quad s \in \{\mathrm{m}, \mathrm{c}\}, \tag{4}$$

where $\mathrm{Cost}(p_s, \mathrm{Merge})$ reflects the information difference before and after merging, scaled by the number of fine-scale patches covered by p_s to ensure cross-scale comparability. Keeping a fine-scale patch unchanged incurs zero cost, i.e., $\mathrm{Cost}(p_{\mathrm{f}}, \mathrm{Keep}) = 0$.

Quadtree DP Candidate patches at different scales may overlap, so they cannot be selected independently. We organize them into a three level quadtree with fine, mid, and coarse scales. Each mid-scale node p_{m} contains four fine-scale patches p_{f}, and each coarse-scale node p_{c} contains four mid-scale nodes. Quadtree DP computes the minimum-cost arrays from fine to mid to coarse scales, and determines the action for each region under the budget.

We denote $T_{\mathrm{f}}(p_{\mathrm{f}}, n)$, $T_{\mathrm{m}}(p_{\mathrm{m}}, n)$, and $T_{\mathrm{c}}(p_{\mathrm{c}}, n)$ as the minimum costs of representing regions covered by p_{f}, p_{m}, and p_{c} with exactly n retained patches. For $s \in \{\mathrm{f}, \mathrm{m}, \mathrm{c}\}$, $T_s(p_s, \cdot)$ denotes the entire cost array for the corresponding region. The local patch count n is distinct from the frame-level patch count N_t. For a leaf node p_{f}, the patch can either be dropped or kept unchanged. Therefore, $n \in \{0, 1\}$, and the corresponding costs are as follows:

$$T_{\mathrm{f}}(p_{\mathrm{f}}, n) = \begin{cases} \mathrm{Cost}(p_{\mathrm{f}}, \mathrm{Drop}), & n = 0, \\ 0, & n = 1. \end{cases} \tag{5}$$

We use **Subtree Cost Merging (SCM)** to combine the cost arrays of child nodes. For a mid-scale node p_{m} with four fine-scale children $p_0, \ldots, p_3$, SCM enumerates all allocations $(n_0, \ldots, n_3) \in \{0, 1\}^4$ and records the minimum sum of child costs for each total retained patch count $n = \sum_{i=0}^{3} n_i$, yielding $\widetilde{T}_{\mathrm{m}}(p_{\mathrm{m}}, n)$. Algorithm 1 illustrates this fine-to-mid aggregation. At higher levels, SCM uses the corresponding feasible child patch counts.

We then account for the option of merging the entire mid-scale region into a single patch:

$$T_{\mathrm{m}}(p_{\mathrm{m}}, n) = \begin{cases} \widetilde{T}_{\mathrm{m}}(p_{\mathrm{m}}, n), & n \neq 1, \\ \min\big(\widetilde{T}_{\mathrm{m}}(p_{\mathrm{m}}, 1), \mathrm{Cost}(p_{\mathrm{m}}, \mathrm{Merge})\big), & n = 1. \end{cases} \tag{6}$$

For $n = 1$, DP compares the cost of retaining one fine-scale patch and dropping the other three with the cost of merging the entire region into a single mid-scale patch.

For each coarse-scale node p_{c}, we apply the same procedure to its four mid-scale children, including the comparison with $\mathrm{Cost}(p_{\mathrm{c}}, \mathrm{Merge})$ at $n = 1$, to obtain T_{c}. We then recursively combine the coarse-scale cost arrays using SCM to obtain the frame-level cost array. Finally, following Eq. 2, we select the retained patch count that minimizes the total action cost plus λN_t subject to $N_t \leq \mathcal{B}$, and backtrack the recorded choices to obtain the final action assignment. See Appendix A for details.

Algorithm 1 Subtree Cost Merging (SCM)

Require: Child cost arrays $T_{\mathrm{f}}(p_i, \cdot)$, $i = 0, \ldots, 3$, for the four children of p_{m}
Ensure: Combined cost array $\widetilde{T}_{\mathrm{m}}(p_{\mathrm{m}}, \cdot)$ for $n = 0, \ldots, 4$

```
for n = 0, ..., 4 do
    T̃_m(p_m, n) = +∞
end for
for all (n_0, n_1, n_2, n_3) ∈ {0, 1}^4 do
    n = 0,   T_total = 0
    for i = 0, ..., 3 do
        n = n + n_i,   T_total = T_total + T_f(p_i, n_i)
    end for
    T̃_m(p_m, n) = min(T̃_m(p_m, n), T_total)
end for
```

Integration with MLLMs The resulting action assignment determines which patches are sent to the ViT and at what scale. A fine-scale patch p_{f} keeps its original content, while p_{m} and p_{c} represent the corresponding area after $2\times$ and $4\times$ spatial downsampling. FlashGaze assigns each retained patch a positional embedding based on the position of its center. Then, selected patches are encoded by the ViT and fed into MLLM as usual. See Appendix B for details.

4 Experiments

4.1 Experimental Setup

Benchmarks We evaluate FlashGaze on six video understanding benchmarks spanning general, long-video, and high-resolution settings: VideoMME (Fu et al., 2025), NExT-QA (Xiao et al., 2021), MLVU (Zhou et al., 2025), VideoMMMU (Hu et al., 2026), LongVideoBench (Wu et al., 2024), and HLVid (Shi et al., 2026). Among them, we use the last three for main comparisons. In addition, we compare efficiency under different retained ratios on two of these benchmarks. Across all six benchmarks, we further evaluate FlashGaze with larger visual inputs across diverse tasks.

Comparison Methods We compare FlashGaze with five token compression methods: ToMe (Bolya et al., 2022), FastV (Chen et al., 2024), VisionZip (Yang et al., 2025), FlashVID (Fan et al., 2026), and AutoGaze (Shi et al., 2026), in terms of accuracy and TTFT. On Qwen3-VL-8B, we further compare accuracy, peak GPU memory, and TTFT under different retained ratios.

Implementation Details We conduct all experiments on NVIDIA H200 GPUs with Qwen3-VL-8B (Bai et al., 2025a) and NVILA-8B-Video (Liu et al., 2025) backbones. For a fair comparison, we define the retained ratio as the fraction of visual tokens remaining after the final pruning stage. TTFT includes the latency of pre-ViT video processing, pruning, ViT encoding, and MLLM prefill.

4.2 Main comparisons

Performance Comparison with Other Methods We compare FlashGaze with five methods on Qwen3-VL-8B and NVILA-8B-Video, using retained ratios of 50% and 10%, respectively. For LongVideoBench and VideoMMMU, the maximum number of sampled frames is set to 768. For HLVid, we evaluate high resolution performance using 2K-resolution videos. Under the same backbone, we use the same retained ratio across methods and compare their accuracy and TTFT latency. As shown in Table 1, FlashGaze achieves the highest accuracy and the lowest TTFT on both backbones for high resolution and long video understanding.

Scalability across various benchmarks As shown in Table 2, FlashGaze scales Qwen3-VL-8B to 1.5K-frame and 4K-resolution videos. With richer visual inputs, FlashGaze outperforms the vanilla model on all six benchmarks and achieves the highest accuracy. These results demonstrate that FlashGaze enables MLLMs to scale to longer and higher resolution videos, leading to better video understanding.

Table 1: **Performance comparison with other methods.** FlashGaze achieves the highest accuracy and lowest TTFT across high-resolution and long-video benchmarks. *FastV is omitted for NVILA-8B-Video due to pre-pruning context overflow.

Method	LongVideoBench				VideoMMMU					HLVid	
	Perc.	Relat.	Overall	TTFT(s)	Perc.	Comp.	Adapt.	Overall	TTFT(s)	Overall	TTFT(s)
Qwen3-VL-8B											
+ToMe	**73.2**	60.4	66.4	24.3	78.0	69.0	48.3	65.1	23.7	71.6	25.7
+FastV	72.8	60.4	66.2	28.7	79.6	67.0	**50.7**	65.8	28.2	70.9	30.1
+VisionZip	72.6	60.3	66.0	25.4	79.0	68.3	49.0	65.4	24.6	71.2	27.5
+FlashVID	72.0	59.7	65.4	26.5	77.3	64.7	49.3	63.8	25.8	72.3	28.9
+AutoGaze	72.0	60.3	65.8	34.9	77.7	69.7	47.7	65.0	37.2	69.8	34.6
+FlashGaze	72.3	**61.8**	**66.7**	**20.7**	**80.9**	**71.3**	48.7	**67.0**	**20.4**	**73.2**	**20.5**
*NVILA-8B-Video**											
+ToMe	61.1	48.0	54.2	30.7	46.7	34.7	41.0	40.8	29.4	49.6	58.0
+VisionZip	61.3	47.3	53.9	34.9	43.3	34.0	41.3	39.6	32.8	49.3	65.4
+FlashVID	61.4	48.9	54.8	38.9	45.3	37.3	38.0	40.2	42.4	52.6	77.4
+AutoGaze	65.4	**53.4**	58.8	17.5	**53.0**	36.3	**42.7**	44.0	15.9	49.3	32.2
+FlashGaze	**65.8**	53.0	**59.0**	**13.7**	52.3	**38.0**	**42.7**	**44.3**	**13.6**	**53.0**	**27.3**

Table 2: **Scalability across various benchmarks.** FlashGaze scales Qwen3-VL-8B to 1.5K-frame and 4K-resolution videos, unlocking longer context for better video understanding.

Models	Max #F	Max Res	*general video* VideoMME (w/o sub)	*general video* NExT-QA (mc)	*long video* VideoMMMU (overall)	*long video* LongVideoBench (overall)	*long video* MLVU (m-avg)	*high-res & long* HLVid (test)
LLaVA-OV-8B (Li et al., 2025)	32	384	58.2	79.4	-	56.5	64.7	41.1
LLaVA-Video-7B (Zhang et al., 2024)	64	384	63.3	83.2	36.1	58.2	70.8	-
VideoLLaMA3-7B (Zhang et al., 2025)	180	384	66.2	84.5	47.0	59.8	73.0	38.8
InternVL3.5-8B (Wang et al., 2025)	64	448	66.0	-	-	62.1	70.2	39.9
VideoChat-Flash (Li et al., 2026)	10000	448	65.3	-	41.7	64.7	74.7	46.6
LongVILA-7B (Chen et al., 2025)	2048	384	60.1	80.7	-	57.1	-	41.4
NVILA-8B-Video (Liu et al., 2025)	256	448	64.2	82.2	-	57.7	70.1	42.5
Qwen2.5-VL-7B (Bai et al., 2025b)	48	896	65.1	-	47.4	56.0	70.2	48.1
Qwen3-VL-8B (Bai et al., 2025a)	768	1792	71.5	83.6	65.3	66.7	77.3	71.6
+ **FlashGaze**	1536	3840	**71.8**	**84.7**	**67.0**	**66.9**	**77.6**	**73.2**

Efficiency under Different Retained Ratios To further evaluate the efficiency of FlashGaze, we compare it with three methods under different retained ratios on two benchmarks. As shown in Table 3, FlashGaze achieves the highest accuracy in seven settings while achieving the lowest TTFT and peak GPU memory in all settings. At a 20% retained ratio, FlashGaze still preserves 98% of the full input accuracy on LongVideoBench, while reducing peak GPU memory by 1.8×. It also achieves a 13× TTFT speedup over the vanilla model, including 5.4× and 17× speedups in ViT encoding and MLLM prefill. See Appendix D for details. Compared with other pruning methods, FlashGaze reduces both TTFT and peak GPU memory usage.

4.3 Ablation Studies

Input Scale and Retained Ratio By varying the patch budget $\mathcal{B}$ and penalty coefficient λ, we evaluate the efficiency of FlashGaze on Qwen3-VL-8B across different retained ratios, frame counts, and input resolutions. As shown in Fig. 4, Qwen3-VL-8B quickly reaches its context limit as the number of frames and input resolution increase, supporting up to 256 frames at a resolution of 1792 in our setting. At this input scale, with $\lambda = 4 \times 10^{-3}$, FlashGaze reaches a 20% retained ratio at 256 frames and 1792 resolution, achieving 7.3× and 19.7× speedups in ViT encoding and MLLM prefill, and a 15× TTFT speedup.

Cost Estimation Method We compare several cost estimation methods to study their impact on downstream accuracy. Besides pixel differences, we use ViT Patch Embedding (Dosovitskiy et al.,

Table 3: **Efficiency under different retained ratios.** FlashGaze achieves the lowest TTFT and peak GPU memory across all settings while largely preserving accuracy.

Method	Retained Ratio	L-VidBench Acc.	Peak GPU Mem. (GiB)	TTFT (s)
Qwen3-VL-8B	100%	66.7	106.4	71.7
+ToMe	50%	66.4	82.5	24.3
+FlashVID		65.4	101.2	26.5
+AutoGaze		65.8	81.0	34.9
+FlashGaze		**66.7**	**80.6**	**20.7**
+ToMe	40%	66.6	82.5	18.6
+FlashVID		64.9	101.2	20.7
+AutoGaze		65.7	74.0	25.3
+FlashGaze		**66.7**	**73.1**	**14.5**
+ToMe	30%	65.9	82.5	13.8
+FlashVID		64.4	101.2	16.0
+AutoGaze		65.1	65.5	18.2
+FlashGaze		**66.3**	**65.2**	**9.5**
+ToMe	20%	65.1	82.5	10.3
+FlashVID		63.6	101.2	12.2
+AutoGaze		64.1	59.0	12.1
+FlashGaze		**65.5**	**58.2**	**5.4**

Method	Retained Ratio	HLVid Acc.	Peak GPU Mem. (GiB)	TTFT (s)
Qwen3-VL-8B	100%	71.6	105.6	72.3
+ToMe	50%	71.6	82.3	25.7
+FlashVID		72.3	101	28.9
+AutoGaze		69.8	80.2	34.6
+FlashGaze		**73.2**	**80.0**	**20.5**
+ToMe	40%	71.2	82.3	20.0
+FlashVID		**71.3**	101	23.0
+AutoGaze		68.2	72.9	25.4
+FlashGaze		**71.3**	**72.4**	**14.3**
+ToMe	30%	71.6	82.5	15.4
+FlashVID		71.6	101.2	18.2
+AutoGaze		66.7	65.5	19.0
+FlashGaze		**71.7**	**65.2**	**9.1**
+ToMe	20%	67.9	82.3	12.1
+FlashVID		**70.9**	101.0	14.8
+AutoGaze		54.9	58.4	13.5
+FlashGaze		68.2	**58.0**	**5.3**

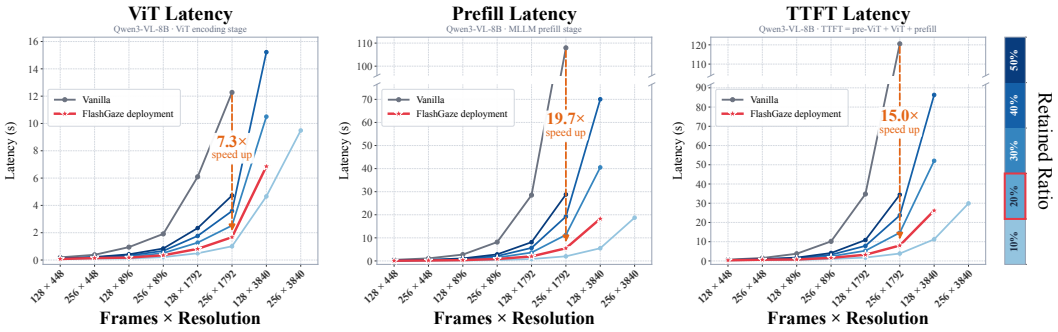

Figure 4: **Latency across input scales and retained ratios.** FlashGaze can choose different retained ratios to balance accuracy and efficiency. At a 20% retained ratio, FlashGaze still maintains strong accuracy. At this ratio, with 256 frames and 1792 resolution, it achieves 7.3× ViT, 19.7× prefill, and 15× TTFT speedups.

2021), SigLIP (Zhai et al., 2023), and LPIPS (Zhang et al., 2018) to measure feature, semantic, and perceptual differences, respectively. Under a 30% retained ratio with 768 sampled frames, pixel differences achieve the best scores on both HLVid and LongVideoBench, reaching 71.7 and 66.3. The other methods perform slightly worse while introducing additional feature extraction and higher inference latency. Overall, pixel differences provide a better tradeoff between accuracy and efficiency.

Optimization Method We compare four methods for solving Eq. 2 and report their downstream accuracy and cumulative action cost, $\sum \text{Cost}$. As shown in Table 4, Quadtree DP achieves both the lowest cost and the highest accuracy, at a 30% retained ratio. Notably, among the other three algorithms, lower cumulative cost is consistently associated with higher accuracy. This trend is consistent with our objective of reducing visual information loss by minimizing action costs.

4.4 Analysis and Visualization

Preserving Visual Representations To evaluate whether pruning preserves visual representations, we compare attention weighted representations from FlashGaze and the full input on 20 HLVid videos. To handle different token counts in corresponding regions, we use the last prefill token as the query and aggregate first layer visual value vectors by their attention weights. We then compute their

Table 4: **Ablation on cost estimation and optimization methods.**

Method	Type	HLVid Acc.	L-VidBench Acc.
Patch embedding	Feature-space	69.3	65.8
SigLIP	Semantic	65.1	64.2
LPIPS	Perception	69.0	66.0
Pixel (ours)	Pixel	**71.7**	**66.3**

Method	Candidates fine	mid	coarse	$\sum$ Cost ($\times 10^3$)	HLVid Acc.
Random	✓			16.50	60.8
Single-scale Greedy	✓			7.5	69.3
Multi-scale Greedy	✓	✓	✓	3.3	70.6
Quadtree DP (ours)	✓	✓	✓	**1.96**	**71.7**

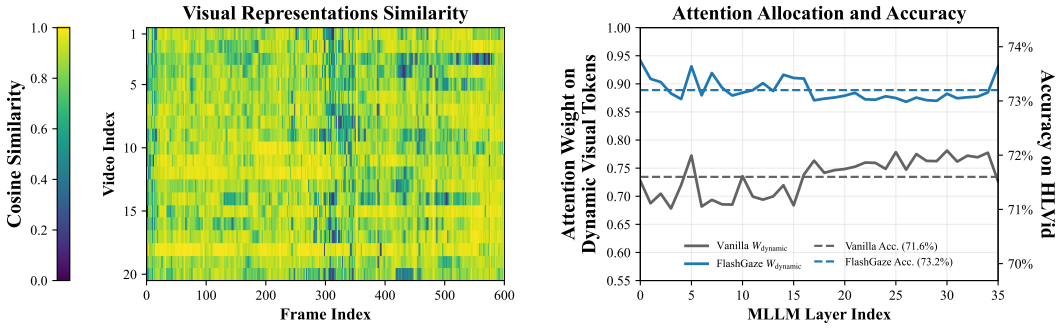

Figure 5: **Analysis of how FlashGaze enables strong video understanding.** Visual representations remain highly similar between FlashGaze and the vanilla setting (left). FlashGaze enables MLLMs to allocate more attention to dynamic tokens and achieves better accuracy, suggesting that increased attention to dynamic tokens may contribute to the accuracy improvement (right).

cosine similarity. See Appendix E for details. As shown in Fig. 5 (left), the visual representations remain highly similar between FlashGaze and the vanilla setting.

More Attention to Dynamic Visual Tokens To analyze attention to dynamic visual tokens, we identify regions that change across frames and mark their corresponding tokens as dynamic. On HLVid with a 50% retained ratio, we use the last prefill token as the query and compute the normalized attention assigned to dynamic tokens, W_{dynamic}, at each layer. As shown in Fig. 5 (right), FlashGaze consistently allocates more attention to dynamic tokens than the vanilla model, while improving accuracy from 71.6% to 73.2%, suggesting that allocating more attention to dynamic visual tokens may contribute to better video understanding.

Visualization of pruned videos We visualize FlashGaze pruning on three video clips. In Fig. 2 (a) and (b), most backgrounds are static, and FlashGaze mainly retains regions that change over time, achieving retained ratios of only 10% and 13%. In contrast, the continuously changing driving scene in Fig. 2 (c) requires more patches. FlashGaze uses coarse-scale patches for low-information areas such as the sky, while preserving fine-scale patches for regions rich in detail such as traffic signs.

5 Conclusion

Long and high-resolution videos produce lengthy visual token sequences that impose substantial computational overhead on MLLMs, yet much of this cost stems from spatiotemporally redundant content. We propose FlashGaze, a training-free method that prunes redundant visual patches directly at the video input stage, before ViT encoding. Based on the observation that such redundancy can be assessed in pixel space, FlashGaze estimates the costs of dropping, merging, and keeping patches, and applies Quadtree DP to determine the action assignment under a given budget. Experiments on two representative MLLMs and six benchmarks show that FlashGaze retains 98% of the full-input accuracy while achieving up to $5.4\times$ and $17\times$ speedups in ViT encoding and MLLM prefill, with $1.8\times$ lower peak GPU memory. FlashGaze further scales MLLMs to 1.5K-frame, 4K-resolution videos, unlocking longer and higher-resolution visual contexts for improved video understanding.

AI USE STATEMENT

In this work, we used generative AI tools for retrieval and discovery, including literature search, related-work identification, and summarization of relevant papers. We also used generative AI tools to aid and polish writing by editing the manuscript for grammar, clarity, conciseness, and readability. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the experimental implementation in Sec. 4.1 and C. We will also provide source code in the supplementary materials.

REFERENCES

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

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025b. URL https://arxiv.org/abs/2502.13923.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*, 2022.

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*, pp. 19–35. Springer, 2024.

Tao Chen, Kun Zhang, Qiong Wu, Xiao Chen, Chao Chang, Xiaoshuai Sun, Yiyi Zhou, and Rongrong Ji. Scaling the long video understanding of multimodal large language models via visual memory mechanism. *arXiv preprint arXiv:2603.29252*, 2026.

Yukang Chen, Fuzhao Xue, Dacheng Li, Qinghao Hu, Ligeng Zhu, Xiuyu Li, Yunhao Fang, Haotian Tang, Shang Yang, Zhijian Liu, et al. Longvila: Scaling long-context visual language models for long videos. In *International Conference on Learning Representations*, volume 2025, pp. 18227–18246, 2025.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Ziyang Fan, Keyu Chen, Ruilong Xing, Yulin Li, Li Jiang, and Zhuotao Tian. Flashvid: Efficient video large language models via training-free tree-based spatiotemporal token merging. *arXiv preprint arXiv:2602.08024*, 2026.

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 *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24108–24118. IEEE, 2025.

Bo He, Hengduo Li, Young Kyun Jang, Menglin Jia, Xuefei Cao, Ashish Shah, Abhinav Shrivastava, and Ser-Nam Lim. Ma-lmm: Memory-augmented large multimodal model for long-term video understanding. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13504–13514. IEEE, 2024.

Kairui Hu, Penghao Wu, Fanyi Pu, Wang Xiao, Xiang Yue, Bo Li, Yuanhan Zhang, and Ziwei Liu. Video-mmmu: Evaluating knowledge acquisition from multidisciplinary professional videos. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27798–27828, 2026.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. Llava-onevision: Easy visual task transfer. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://openreview.net/forum?id=zKv8qULV6n.

Xinhao Li, Yi Wang, Jiashuo Yu, Xiangyu Zeng, Yuhan Zhu, Haian Huang, Jianfei Gao, Kunchang Li, Yinan He, Chenting Wang, et al. Videochat-flash: Hierarchical compression for long-context video modeling. In *International Conference on Learning Representations*, volume 2026, pp. 109089–109117, 2026.

Zhijian Liu, Ligeng Zhu, Baifeng Shi, Zhuoyang Zhang, Yuming Lou, Shang Yang, Haocheng Xi, Shiyi Cao, Yuxian Gu, Dacheng Li, et al. Nvila: Efficient frontier visual language models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4122–4134. IEEE, 2025.

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. Egoschema: A diagnostic benchmark for very long-form video language understanding. *Advances in Neural Information Processing Systems*, 36:46212–46244, 2023.

Rui Qian, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Shuangrui Ding, Dahua Lin, and Jiaqi Wang. Streaming long video understanding with large language models. *Advances in Neural Information Processing Systems*, 37:119336–119360, 2024.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949, 2021.

Aniket Rege, Arka Sadhu, Yuliang Li, Kejie Li, Ramya Korlakai Vinayak, Yuning Chai, Yong Jae Lee, and Hyo Jin Kim. Agentic very long video understanding. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 46575–46602, 2026.

Leqi Shen, Guoqiang Gong, Tao He, Yifeng Zhang, Pengzhang Liu, Sicheng Zhao, and Guiguang Ding. Fastvid: Dynamic density pruning for fast video large language models. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/b2e63e36c57e153b9015fece2352a9f9-Abstract-Conference.html.

Xiaoqian Shen, Yunyang Xiong, Changsheng Zhao, Lemeng Wu, Jun Chen, Chenchen Zhu, Zechun Liu, Fanyi Xiao, Balakrishnan Varadarajan, Florian Bordes, et al. Longvu: Spatiotemporal adaptive compression for long video-language understanding. *arXiv preprint arXiv:2410.17434*, 2024.

Baifeng Shi, Boyi Li, Han Cai, Yao Lu, Sifei Liu, Marco Pavone, Jan Kautz, Song Han, Trevor Darrell, Pavlo Molchanov, et al. Scaling vision pre-training to 4k resolution. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9631–9640. IEEE, 2025.

Baifeng Shi, Stephanie Fu, Long Lian, Hanrong Ye, David Eigen, Aaron Reite, Jan Kautz, Boyi Li, David M Chan, Trevor Darrell, et al. Attend before attention: Efficient and scalable video understanding via autoregressive gazing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17022–17034, 2026.

Enxin Song, Wenhao Chai, Guanhong Wang, Yucheng Zhang, Haoyang Zhou, Feiyang Wu, Haozhe Chi, Xun Guo, Tian Ye, Yanting Zhang, et al. Moviechat: From dense token to sparse memory for long video understanding. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18221–18232. IEEE, 2024.

Hesong Wang, Xin Jin, Lu Lu, Chenhaowen Li, Jian Chen, Qiang Liu, and Huan Wang. Earlytom: Early token compression completes fast video understanding. *arXiv preprint arXiv:2605.30010*, 2026a.

Nan Wang, Zhiwei Jin, Chen Chen, and Haonan Lu. Pixelprune: Pixel-level adaptive visual token reduction via predictive coding. *arXiv preprint arXiv:2604.00886*, 2026b.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025.

Xiaohan Wang, Yuhui Zhang, Orr Zohar, and Serena Yeung-Levy. Videoagent: Long-form video understanding with large language model as agent. In *European Conference on Computer Vision*, pp. 58–76. Springer, 2024.

Yiyu Wang, Xuyang Liu, Xiyan Gui, Xinying Lin, Boxue Yang, Chenfei Liao, Tailai Chen, and Linfeng Zhang. Accelerating streaming video large language models via hierarchical token compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18523–18533, 2026c.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding. *Advances in Neural Information Processing Systems*, 37:28828–28857, 2024.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 9777–9786. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPR46437.2021.00965. URL https://openaccess.thecvf.com/content/CVPR2021/html/Xiao_NExT-QA_Next_Phase_of_Question-Answering_to_Explaining_Temporal_Actions_CVPR_2021_paper.html.

Yiweng Xie, Bo He, Junke Wang, Xiangyu Zheng, Ziyi Ye, and Zuxuan Wu. Fluxmem: Adaptive hierarchical memory for streaming video understanding. *arXiv preprint arXiv:2603.02096*, 2026.

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 *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19792–19802. IEEE, 2025.

Linli Yao, Yicheng Li, Yuancheng Wei, Lei Li, Shuhuai Ren, Yuanxin Liu, Kun Ouyang, Lean Wang, Shicheng Li, Sida Li, et al. Timechat-online: 80% visual tokens are naturally redundant in streaming videos. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10807–10816, 2025.

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11941–11952. IEEE, 2023.

Boqiang Zhang, Kehan Li, Zesen Cheng, Zhiqiang Hu, Yuqian Yuan, Guanzheng Chen, Sicong Leng, Yuming Jiang, Hang Zhang, Xin Li, et al. Videollama 3: Frontier multimodal foundation models for image and video understanding. *arXiv preprint arXiv:2501.13106*, 2025.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018.

Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. Llava-video: Video instruction tuning with synthetic data. *arXiv preprint arXiv:2410.02713*, 2024.

Junjie Zhou, Yan Shu, Bo Zhao, Boya Wu, Zhengyang Liang, Shitao Xiao, Minghao Qin, Xi Yang, Yongping Xiong, Bo Zhang, et al. Mlvu: Benchmarking multi-task long video understanding. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13691–13701. IEEE, 2025.

A Additional Details of Quadtree DP

This section provides a more detailed description of the overall procedure of the Quadtree DP introduced in Sec. 3.3. Algorithm 2 takes as input the action costs estimated in Sec. 3.3, the patch budget $\mathcal{B}$, the penalty coefficient λ, and the current frame together with its corresponding quadtree $\mathcal{Q}$, and outputs the final action assignment $\mathcal{A}_t^*$. The quadtree structure is illustrated in Fig. 3 (right). During DP, Trace records the minimizing child-count allocations and the decisions between merging a region and representing it through its children. These records are used for backtracking; their updates are omitted from the pseudocode for brevity.

Algorithm 2 Quadtree DP

Require: Quadtree $\mathcal{Q}$, action costs, patch budget $\mathcal{B}$, penalty λ, current frame F_t
Ensure: Action assignment $\mathcal{A}_t^*$

for all fine-scale leaf p_{f} in $\mathcal{Q}$ **do**
  $T_{\mathrm{f}}(p_{\mathrm{f}}, 0) = \mathrm{Cost}(p_{\mathrm{f}}, \mathrm{Drop}),\ T_{\mathrm{f}}(p_{\mathrm{f}}, 1) = 0$
end for
for all mid-scale node p_{m} in $\mathcal{Q}$ with children $p_0, \ldots, p_3$ **do**
  Apply SCM to $T_{\mathrm{f}}(p_0, \cdot), \ldots, T_{\mathrm{f}}(p_3, \cdot)$ to obtain $\widetilde{T}_{\mathrm{m}}(p_{\mathrm{m}}, \cdot)$
  for $n = 0, \ldots, 4$ **do**
    $T_{\mathrm{m}}(p_{\mathrm{m}}, n) = \widetilde{T}_{\mathrm{m}}(p_{\mathrm{m}}, n)$
  end for
  $T_{\mathrm{m}}(p_{\mathrm{m}}, 1) = \min\big(T_{\mathrm{m}}(p_{\mathrm{m}}, 1), \mathrm{Cost}(p_{\mathrm{m}}, \mathrm{Merge})\big)$
end for
Repeat the mid-scale procedure for all coarse-scale nodes p_{c} in $\mathcal{Q}$, using the mid-scale child cost arrays, $n = 0, \ldots, 16$, and $\mathrm{Cost}(p_{\mathrm{c}}, \mathrm{Merge})$ to obtain $T_{\mathrm{c}}(p_{\mathrm{c}}, \cdot)$.
Apply SCM to the coarse-scale cost arrays until obtaining the frame-level array $T_{\mathrm{frame}}(\cdot)$.
$N_t^* = \arg \min_{N_t \in \{0, \ldots, \mathcal{B}\}} \big\{T_{\mathrm{frame}}(N_t) + \lambda N_t\big\}$
return Backtrack($\mathcal{Q}$, Trace, N_t^*)

B Additional Details of Integration with MLLMs

This section provides further implementation details on integrating FlashGaze with different MLLMs. For the Qwen3-VL series, the ViT natively supports multi-scale image inputs, allowing FlashGaze to be seamlessly integrated without any downstream fine-tuning. For NVILA-8B-Video, we enable multi-scale patch inputs by interpolating each frame and its positional embeddings to different scales, allowing patches at different scales to be processed by the ViT encoder.

C Additional Details of Experiment Setup

All latency measurements are conducted on a single NVIDIA H200 GPU with a batch size of 1 and BF16 precision. We consistently use PyTorch native SDPA for attention instead of FlashAttention to ensure a fair comparison across methods. Each configuration is first warmed up with one run, followed by three consecutive runs whose latency is averaged. Peak GPU memory refers to the maximum GPU memory actually allocated during inference. We follow the official prompt protocols of each benchmark as implemented in lmms-eval.

D Additional Details of Efficiency under Different Retained Ratios

This section provides a detailed breakdown of the TTFT reported in Table 3. We divide TTFT into three stages: pre-ViT, which includes video processing and pruning overhead; ViT encoding; and MLLM prefill. Stage-wise latency results are reported below.

We define the retained ratio as the number of visual tokens remaining after the final pruning stage divided by the original number of visual tokens. For FlashVID, the ViT-stage pruning reduces the

Table 5: **Stage-wise latency under different retained ratios.**

Latency on LongVideoBench

Method	Retained Ratio	Pre-ViT (s)	ViT encoding (s)	MLLM prefill (s)
Qwen3-VL-8B	100%	0.7	5.9	65.1
+ToMe	50%	0.4	5.9	18.0
+FlashVID		2.2	6.6	17.7
+AutoGaze		14.6	3.6	16.7
+FlashGaze		0.5	2.7	17.5
+ToMe	40%	0.6	5.9	12.1
+FlashVID		2.1	6.7	11.9
+AutoGaze		10.7	2.1	12.5
+FlashGaze		0.5	2.1	11.9
+ToMe	30%	0.5	5.9	7.4
+FlashVID		2.0	6.7	7.3
+AutoGaze		8.9	1.8	7.5
+FlashGaze		0.5	1.6	7.4
+ToMe	20%	0.5	5.8	4.0
+FlashVID		1.8	6.6	3.8
+AutoGaze		7.3	1.1	3.7
+FlashGaze		0.5	1.1	3.8

Latency on HLVid

Method	Retained Ratio	Pre-ViT (s)	ViT encoding (s)	MLLM prefill (s)
Qwen3-VL-8B	100%	0.5	7.9	63.9
+ToMe	50%	0.4	7.8	17.5
+FlashVID		2.2	9.5	17.2
+AutoGaze		14.6	3.1	16.9
+FlashGaze		0.5	3.0	17.0
+ToMe	40%	0.5	7.8	11.7
+FlashVID		2.0	9.4	11.6
+AutoGaze		11.3	2.4	11.7
+FlashGaze		0.5	2.5	11.3
+ToMe	30%	0.5	7.8	7.1
+FlashVID		1.7	9.5	7.0
+AutoGaze		10.0	1.7	7.3
+FlashGaze		0.5	1.6	7.0
+ToMe	20%	0.6	7.7	3.8
+FlashVID		1.6	9.6	3.6
+AutoGaze		8.3	1.5	3.7
+FlashGaze		0.5	1.1	3.7

tokens only to an intermediate ratio, while a subsequent pruning stage further reduces them to the final retained ratio. Therefore, the ViT processes the less-pruned intermediate sequence, resulting in higher ViT latency. AutoGaze has higher pre-ViT latency because it uses an autoregressive network to process the video before ViT encoding. In contrast, FlashGaze performs pruning only once, before the ViT, without introducing additional networks. Thus, FlashGaze achieves the lowest overall latency.

E Additional Details of Analysis

This section describes the detailed computation of visual feature similarity. Since the vanilla input and FlashGaze may contain different numbers of visual tokens for the same spatial area after pruning, their ViT output features cannot be directly compared. We therefore aggregate the visual features using the attention weights from the first MLLM layer to obtain a fixed-dimensional representation for comparison. Let $\mathcal{I}'_t$ denote the indices of the visual tokens corresponding to the pruned frame F'_t. Using the last prefill token as the query, we compute the representative feature as

$$\mathbf{x}_t = \mathbf{W}_{\mathrm{O}} \ \mathrm{Concat}_h \left(\sum_{i \in \mathcal{I}'_t} \alpha_{h,i} \mathbf{v}_{h,i} \right). \tag{7}$$

Here, h indexes attention heads, i indexes visual tokens, $\alpha_{h,i}$ is the attention weight assigned to token i by head h, $\mathbf{v}_{h,i}$ is the corresponding value vector, and $\mathbf{W}_{\mathrm{O}}$ is the output projection matrix.

For the vanilla input, we identify the corresponding spatial area in the original frame and apply the same aggregation to all visual tokens within that area, obtaining $\mathbf{y}_t$. We then compute their cosine similarity as

$$\mathrm{Similarity}(\mathbf{x}_t, \mathbf{y}_t) = \frac{\mathbf{x}_t^\top \mathbf{y}_t}{\|\mathbf{x}_t\|_2 \, \|\mathbf{y}_t\|_2}. \tag{8}$$

F Additional Visual Cases of Pruned Videos

In this section, we present more video clips processed by FlashGaze.

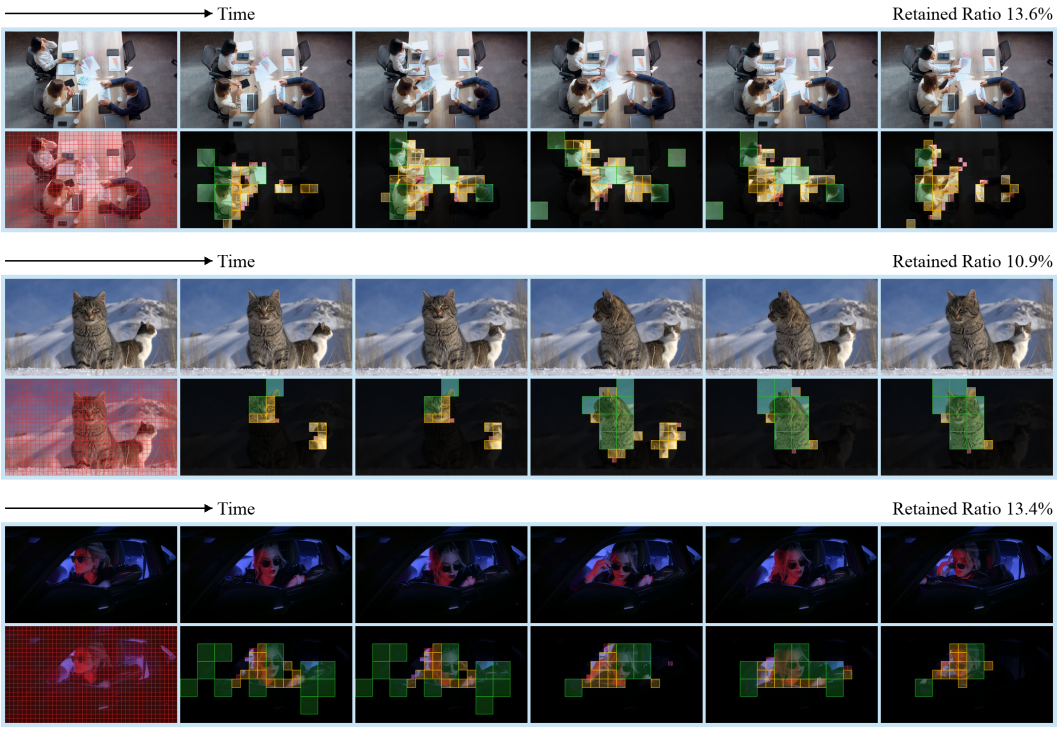

Figure 6: **More cases of videos pruned by FlashGaze.**