LEARNING WHEN TO LOOK: ADAPTIVE VISUAL COMPUTATION FOR VIDEO MLLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Video multimodal large language models (MLLMs) process long visual sequences, yet fixed token budgets ignore variation in the evidence needed by different video–question pairs. We introduce Route2K, which organizes visual tokens into nested prefixes and uses a low-rank predictor initialized from one real seed state to choose a route without repeated full-decoder decisions. The route sets early decoder width before question-conditioned late pruning. On a frozen 1,000-question NExT-QA validation subset, Route2K reaches 71.7% accuracy at 1,599 executed visual token-layers, versus 71.4% at 2,324 for a measured FastVID operating point; narrow execution reaches 69.0%. On MVBench and TempCompass, it executes fewer visual token-layers than selected FastVID points with reported accuracy within one percentage point. Validation ablations support route-conditioned entry widths. These results show the potential of sample-conditioned routing to improve the accuracy–computation balance of video question answering.

1 INTRODUCTION

Video multimodal large language models (Video-MLLMs) have emerged as a powerful interface for video understanding, supporting video question answering, temporal reasoning, and instruction following over open-ended video inputs (Lin et al., 2024; Bai et al., 2025). However, this capability comes at substantial inference cost. Multiple frames are encoded into a long sequence of visual tokens and processed together with the textual prompt. A growing body of work reduces this cost by pruning or merging redundant visual tokens before or within language-model inference (Yang et al., 2025; Zhang et al., 2024; Shen et al., 2026; Fan et al., 2026). Most such methods focus on *which* tokens to retain under a specified budget. A complementary question remains unresolved: *how much visual evidence does a particular video–question pair require?*

Using one global retention ratio implicitly assigns every input the same evidence budget. A conservative budget wastes computation on easy examples, whereas an aggressive one can remove evidence needed by harder examples. The appropriate budget can also depend on the question. Two questions about the same video may rely on different frames, objects, or temporal relations and tolerate different degrees of compression. An input-aware importance score still does not determine where to stop along the resulting ranking. Recent work adapts retention using video-level redundancy or question-conditioned evidence (Regmi et al., 2026; Ding et al., 2026), but neither video-only allocation nor question-conditioned image pruning fully addresses progressive video evidence and its execution cost. Exploratory retention sweeps (Section 4.2) show substantial variation in when predictions stabilize. They also show that adding tokens can either correct or damage an answer, so agreement with a full-budget prediction is not a correctness oracle. Figure 1 illustrates the question dependence on one video: the eight-token seed answers one question, while the routed final pass corrects its mistake on a second.

Adaptive budgeting helps only when its decisions cost less than the computation they avoid. Repeated language-model calls can make a small final token budget slower than fixed-budget inference. A recent break-even study confirms that fewer visual tokens need not lower end-to-end latency (Dou & Tian, 2026). We therefore measure runtime alongside visual token-layer work rather than infer speed from the final retained count.

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.

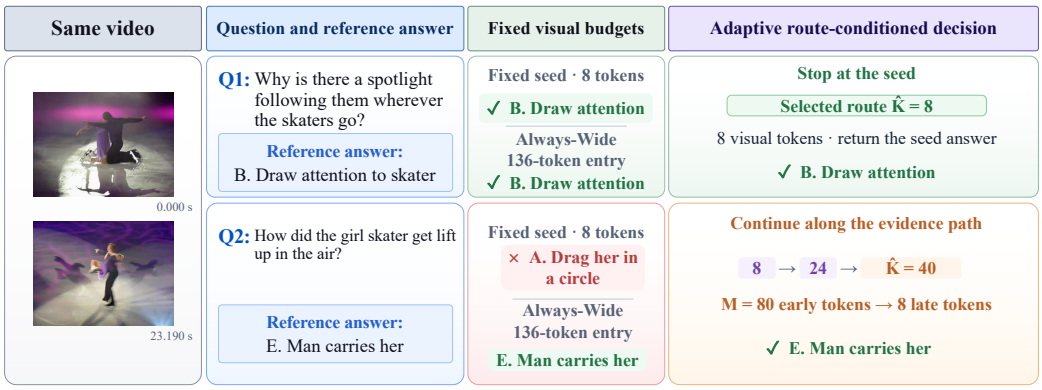

Figure 1: Two post-hoc NExT-QA examples from the same video. The eight-token seed answers the spotlight question but misses the lifting question. For the latter, the routed final pass enters with 80 visual tokens, prunes to eight in later layers, and answers correctly. Always-Wide enters with 136 tokens for both.

We introduce a progressive visual computation framework organized around a *nested evidence path*. A single master ordering induces increasingly large visual-token prefixes, allowing inference to begin with a compact seed. A low-rank transition predictor starts from the seed's real answer state and estimates the effect of scheduled evidence increments without another full decoder pass at each step. Exact prefix nesting keeps those increments consistent, and retained tokens preserve their original spatiotemporal coordinates. The predicted route sets the width of the early decoder context. After the question has interacted with that context, attention-based pruning selects a compact visual tail for the remaining layers.

We evaluate the frozen execution policy on NExT-QA, MVBench, and TempCompass, with component ablations on video-disjoint validation data. The results distinguish the visual width entering the decoder from the final active count, and executed token-layers from wall-clock latency. On NExT-QA, route-conditioned execution improves over a narrow predecessor and saves visual token-layer work relative to an always-wide control.

This paper makes three contributions:

- Exploratory fine-grid trajectories reveal heterogeneous suffix-stable budgets and opposing correctness flips, motivating sample-conditioned routing without treating prediction stability as correctness.
- We introduce a nested visual evidence path and a low-rank state-transition predictor that chooses a route without repeated full-decoder decisions. The route controls early decoder width before late visual pruning.
- We evaluate route-conditioned execution across three video question-answering benchmarks, with validation ablations and measured latency. This separates reductions in visual token-layer work from end-to-end runtime gains.

2 Related Work

2.1 Visual Token Compression for Video MLLMs

Visual-token compression reduces the long multimodal context presented to the language model by pruning, merging, or reorganizing redundant visual representations. VisionZip selects informative visual tokens for both image and video inputs (Yang et al., 2025), while FasterVLM ranks tokens using vision-encoder `[CLS]` attention without requiring language-model cross-attention (Zhang et al., 2024). Video-specific methods further exploit temporal redundancy. FastVID partitions a video into temporally ordered segments and applies density-based pruning (Shen et al., 2026), whereas FlashVID combines attention-and-diversity selection with tree-based spatiotemporal merging (Fan

et al., 2026). These methods improve how a prescribed visual-token budget is used, but their reported operating points commonly apply a selected retention ratio across samples. Our work asks how much visual evidence should be allocated to each video–question pair.

2.2 Sample-Adaptive Visual Computation

Recent methods replace uniform allocation with input-dependent visual budgets to avoid spending computation on detail that a particular input does not need. CARES predicts a sufficient image resolution from an image–query pair before running the target VLM (Kimhi et al., 2026), and ResAdapt learns an input-side allocator that distributes visual budget across frames before encoding (Liao et al., 2026). For post-encoding reduction, Adaptive Two-Stage Visual Token Pruning determines a content-dependent retention ratio from video-token redundancy (Regmi et al., 2026). ET-Prune derives question-conditioned evidence for text-rich images, sets a sample-specific token floor at an intermediate layer, and progressively prunes toward that floor (Ding et al., 2026). These studies demonstrate the value of sample-dependent allocation. Most select the target resolution or retention level before the target model completes inference, while ET-Prune fixes its token floor at an intermediate layer before subsequent pruning. They do not address whether a Video-MLLM should request additional question-specific temporal evidence after observing an initially compact state. Our framework organizes visual tokens into a nested evidence path so that each video–question pair can begin with a compact prefix and acquire more evidence when needed.

2.3 Progressive Inference and Computation Reuse

Adaptive inference reduces computation by allowing different inputs to terminate before full model execution. DeeBERT adds intermediate classifiers that support early exit at different Transformer layers (Xin et al., 2020). CALM extends confidence-based early exit to autoregressive language models (Schuster et al., 2022), and LayerSkip combines early exit with self-speculative verification that reuses the remaining layers (Elhoushi et al., 2024). These methods adapt the depth of language-model computation rather than the amount of visual evidence. Progressive visual inference also introduces a distinct cost because each stopping decision may require an additional multimodal state. Recent break-even analysis shows that this overhead can offset the benefit of processing fewer visual tokens (Dou & Tian, 2026). Our framework predicts intermediate state transitions without repeated full-model calls and uses the resulting route to choose how much visual evidence enters the early decoder layers.

3 Method

Our goal is to allocate visual computation to each video–question pair without deciding its budget in advance. To make successive decisions comparable, expanding the evidence must preserve the tokens already considered and add a well-defined suffix. In a causal decoder, this exact-prefix structure also permits KV-cache reuse when the execution backend supports it; reselecting or reordering earlier tokens would invalidate the cached prefix. We therefore organize projected visual tokens into a deterministic, append-only sequence of nested evidence sets. A lightweight transition predictor estimates the effect of the next suffix on the model's answer state, and the policy continues only when that evidence is predicted to cause a consequential change. The predicted route subsequently controls how much evidence enters the early decoder layers, before a compact visual tail is selected. Figure 2 summarizes the path, stopping decision, and route-conditioned execution.

3.1 Problem Formulation

Let $x = (V, q)$ denote a video–question pair. A frozen vision encoder and projector map V to a full sequence of projected visual tokens $\mathcal{V}(V) = \{v_i\}_{i=1}^{N}$. Many visual-token compressors can use video content or the question to rank and select tokens, but they are still commonly evaluated with a target cardinality fixed across inputs. Question-conditioned selection determines which tokens occupy that budget, but it does not by itself determine how much evidence the current pair needs. We instead construct an ordered family of evidence sequences

$$S_0 \subset S_1 \subset \cdots \subset S_T \subseteq \mathcal{V}(V), \qquad S_{t+1} = S_t \parallel \Delta V_t, \tag{1}$$

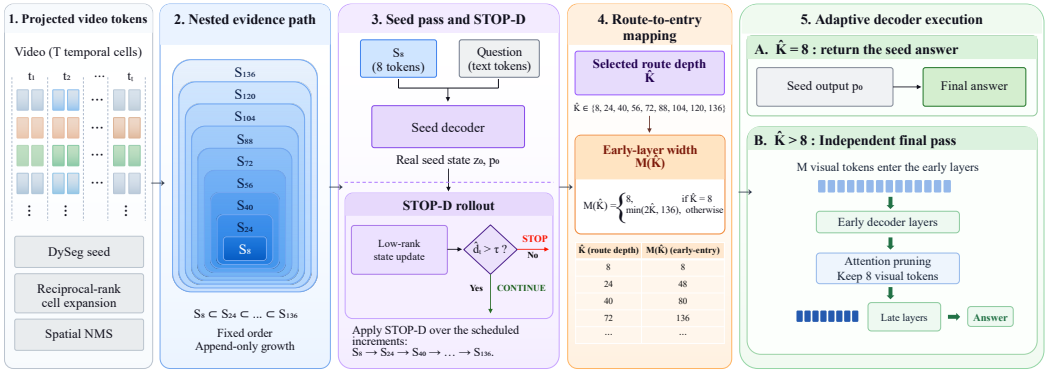

Figure 2: Route-conditioned visual inference. A deterministic nested evidence path defines candidate prefixes. Starting from one real seed state, STOP-D predicts the effect of scheduled increments without intermediate full-decoder passes. Seed stops return the existing answer; continued samples use the selected route to set the early-layer entry width, run an independent final pass, and prune to eight visual tokens for later layers.

where $\|$ denotes append-only concatenation and ΔV_t contains the evidence scheduled for the next expansion. The corresponding visual-token budget is $K_t = |S_t|$. Exact nesting serves two purposes. First, it makes budgets comparable because moving from S_t to S_{t+1} changes only the newly added evidence. Second, it lets the policy select a depth along one path rather than constructing an unrelated token set at every budget; a compatible backend could also reuse the prefix computation.

For a policy π with stopping depth $t_\pi(x)$, let $f_{\text{exec}}(x,t)$ and $C_{\text{exec}}(x,t)$ denote the answer and cost of the route-conditioned decoder execution at depth t, defined below. The deployment objective is to reduce expected inference cost while satisfying a validation-defined quality constraint,

$$\min_{\pi} \mathbb{E}_x\big[C_{\text{exec}}\big(x, t_\pi(x)\big)\big] \quad \text{subject to} \quad \mathbb{E}_{(x,y)}\big[\ell\big(f_{\text{exec}}(x, t_\pi(x)), y\big)\big] \le \mathcal{L}_{\text{ref}} + \epsilon, \tag{2}$$

where C_{exec} accounts for the executed visual and language-model computation, ℓ is the task loss, and $\mathcal{L}_{\text{ref}}$ is the loss of the chosen reference policy. The tolerance $\epsilon \ge 0$ specifies the maximum permitted increase in expected loss relative to that reference and therefore determines the desired accuracy–computation operating point. The stopping decision is causal. At step t, the policy may use the real or recursively predicted state associated with S_t, the question representation, and inexpensive features of the scheduled increment ΔV_t. It may not run a future evidence set merely to observe its true answer state before deciding whether to acquire it. Ground-truth answers and true future states are used only as offline supervision or evaluation signals, while dense correctness and oracle budgets are never inference-time inputs.

3.2 Nested Visual Evidence Path

The evidence path should expose useful information early while avoiding a seed concentrated in a narrow temporal or spatial region. We use the received-attention score from the final vision block, denoted by R_0, as the base token priority. We further adopt dynamic temporal segmentation (DySeg) from FastVID (Shen et al., 2026) to group neighboring temporal content into coherent segments. These segments provide a content-aware unit for distributing the initial evidence across distinct portions of a video. Within each temporal cell, the unselected spatial tokens form a queue ordered by global R_0 priority.

DySeg seed. The initial set S_0 contains eight tokens, an empirical path hyperparameter chosen on the development split to balance segment coverage and initial cost, and fixed before held-out evaluation. For each DySeg segment, we define its anchor as the highest-ranked token in that segment and order the segments by their anchors' global R_0 ranks. If at least eight segments are present, S_0 takes the anchors from the first eight segments in this order. If fewer than eight segments are present, it first takes one anchor from every segment. Each remaining position is then assigned to a segment that currently has the fewest selected tokens; ties are broken by the global R_0 rank of the segment's

next available token. This balanced filling procedure prevents a high-priority segment from consuming all remaining seed positions before the other segments receive coverage. The selected seed tokens are finally serialized in global R_0 order.

Reciprocal-rank cell expansion. After the seed, the path must decide which temporal regions should receive the next tokens. We use a deterministic reciprocal-rank cell scheduler that prioritizes cells whose next few available tokens appear early in the global R_0 ordering. This avoids exhausting one cell or allocating the same number of tokens to every cell. Let $Q_c^{(t)}$ be the remaining R_0-ordered queue for temporal cell c at step t, and let $H_c^{(t)}$ be its next group of at most four tokens. For the one-indexed global R_0 rank $r(v)$, the cell score is

$$m_t(c) = \frac{1}{|H_c^{(t)}|} \sum_{v \in H_c^{(t)}} \frac{1}{r(v)}. \tag{3}$$

The reciprocal weighting emphasizes the highest-ranked candidates, while averaging the next four ranks favors cells that can supply a complete high-priority group. At each expansion, the scheduler selects the four non-exhausted cells with the largest $m_t(c)$. It is deterministic once R_0, the temporal cells, and the previously selected tokens are fixed.

Spatial non-maximum suppression. High-ranked candidates within the same temporal cell can occupy neighboring patches and provide spatially redundant evidence. SpatialNMS is designed to spread a fixed token quota across more distinct spatial locations while preserving the underlying R_0 priority. For each of the four cells selected by the reciprocal-rank scheduler, it examines the first 12 unselected candidates in R_0 order and scans them greedily. After a token at coordinate (h, w) is accepted, a candidate (h', w') is deferred when its Chebyshev distance $\max(|h-h'|, |w-w'|)$ from any token already accepted in that cell is at most one. The scan stops after accepting four tokens. If suppression leaves fewer than four, the method scans the full remaining cell queue in its original R_0 order and fills the quota without suppression. Selecting four cells and four tokens per cell adds approximately 16 tokens at each expansion, after which the new tokens are appended in global R_0 order.

The resulting legal budget ladder in our implementation is

$$(K_0, K_1, \ldots, K_T) = (8, 24, 40, 56, 72, 88, 104, 120, 136). \tag{4}$$

Every expansion is nested and append-only. For any $m \leq 136$, let $S^{(m)}$ denote the first m tokens of the same master order, so $S_t = S^{(K_t)}$; this notation also covers entry widths between the route-ladder values. Each retained token keeps its original temporal, height, and width mRoPE coordinates, so its position in the video remains unchanged when it is appended to the evidence path.

3.3 Low-Rank Future State-Transition Prediction

Evaluating the full language-model decoder after every expansion would reveal the true effect of additional evidence, but it would also require a separate expensive forward pass at every candidate budget. A decision made only from visual features, on the other hand, does not observe how the current evidence and question have already shaped the model's answer. We use the real answer state computed at the seed as an initial summary of this interaction, then predict how the scheduled visual increments would update that state. To keep this simulation inexpensive, the update is represented in a shared low-rank subspace rather than directly regressing a full hidden-state vector.

Let $z_t \in \mathbb{R}^d$ denote the answer state associated with the current evidence sequence, and let q_{nat} be the backbone's native question representation. Before the predictor runs at step t, the deterministic scheduler has already selected the next increment ΔV_t. A question-conditioned pooling operation summarizes the projected features of these scheduled tokens, while a context vector c_t contains the inference-time path-position and cardinality features used by the implementation. We denote the learned transition predictor by D_θ and refer to the resulting stopping module as STOP-D. It predicts eight coefficients in a shared update basis $U_8 \in \mathbb{R}^{d \times 8}$,

$$\widehat{\alpha}_t = D_\theta(\widetilde{z}_t, q_{\mathrm{nat}}, \Delta V_t, c_t)\,, \tag{5}$$

$$\widehat{z}_{t+1} = \widetilde{z}_t + U_8 \widehat{\alpha}_t. \tag{6}$$

Here $\widetilde{z}_0 = z_0$ is the real state produced from the seed. The shared basis U_8 is fitted offline from state-transition trajectories and then frozen. The parameters θ are learned on trajectories generated by the final evidence path, using the true next states as training supervision for coefficient reconstruction, answer-distribution reconstruction, and routing. These true future states are training targets only and are unavailable during inference.

STOP-D predicts the effect of the specific next increment already chosen by the deterministic scheduler; it neither selects tokens nor directly predicts a final budget or answer. When the policy continues beyond one predicted step, it sets $\widetilde{z}_{t+1} = \widehat{z}_{t+1}$ and uses this predicted state to evaluate the following scheduled increment. Re-running the full decoder at every intermediate budget would turn a T-step decision into as many as T expensive decoder calls and could erase the saving from selecting fewer tokens. Recursive state prediction instead performs the budget search with the small transition model. The full decoder is invoked again only after the final evidence depth has been chosen, so predicted states guide allocation but do not replace the final answer computation for continued samples.

3.4 State-Aware Adaptive Stopping

The frozen final normalization and language-model head define an option readout $\mathcal{R}$ over the answer choices. At the current and predicted-next states, respectively,

$$p_t = \mathcal{R}(\widetilde{z}_t), \qquad \widehat{p}_{t+1} = \mathcal{R}(\widehat{z}_{t+1}). \tag{7}$$

We measure how much the next increment is predicted to alter the answer distribution using Jensen–Shannon divergence. Writing $m_t = \frac{1}{2}(p_t + \widehat{p}_{t+1})$, the stopping score is

$$\widehat{d}_t = \mathrm{JS}(p_t, \widehat{p}_{t+1}) = \frac{1}{2}\,\mathrm{KL}(p_t \| m_t) + \frac{1}{2}\,\mathrm{KL}(\widehat{p}_{t+1} \| m_t). \tag{8}$$

The policy stops at S_t when $\widehat{d}_t \le \tau$, indicating that the next scheduled evidence is predicted to have little effect on the current answer distribution. Otherwise, it carries $\widehat{z}_{t+1}$ forward and evaluates the following increment. The selected stopping depth and predicted route are

$$\widehat{t} = \min\Big(\{t < T : \widehat{d}_t \le \tau\} \cup \{T\}\Big), \qquad \widehat{K} = K_{\widehat{t}}. \tag{9}$$

The threshold τ specifies the accuracy–computation operating point of the stopping policy. It is selected once on a video-disjoint validation split under the prescribed quality constraint, after D_θ has been trained, and is then frozen. Held-out evaluation labels are not used to choose it. Inference begins with one real seed state and uses STOP-D to simulate successive state changes along the deterministic path. No full-decoder computation is performed during this intermediate simulation. The resulting $\widehat{K}$ is an evidence-demand route, which the execution stage converts to a decoder entry width rather than treating as the final retained-token count. Algorithm 1 summarizes the complete inference procedure.

3.5 Route-Conditioned Adaptive Execution

The earlier execution fed $S^{(\widehat{K})}$ directly to the decoder, excluding every later-ranked candidate before it could interact with the question. We now use the predicted route to set the width of the early decoder context:

$$M(\widehat{K}) = \begin{cases} 8, & \widehat{K} = 8, \\ \min(2\widehat{K}, 136), & \widehat{K} > 8. \end{cases} \tag{10}$$

The routes $8, 24, 40, 56$, and 72 or above thus map to entry widths $8, 48, 80, 112$, and 136. Each $S^{(M(\widehat{K}))}$ remains a prefix of the frozen master order. The eight-token case returns the existing seed answer without a second decoder pass; the factor of two gives continuing samples additional candidates, capped by the path's 136-token endpoint. The entry-width mapping and the within-decoder pruning configuration were fixed before evaluating route-conditioned accuracy and are not claimed to be optimal.

For a continued sample, $S^{(M(\widehat{K}))}$ remains active through an initial portion of the decoder. This gives the question state access to a broader set of visual candidates before question-to-visual attention is

Algorithm 1 Route-conditioned adaptive visual inference

Require: Video V, question q, scheduler G, predictor D_θ, basis U_8, threshold τ
$\mathcal{V}(V) \leftarrow \textsc{EncodeProject}(V)$
$(S_0, \ldots, S_T) \leftarrow \textsc{BuildNestedPath}(\mathcal{V}(V), G)$
$(z_0, p_0) \leftarrow \textsc{SeedPass}(S_0, q)$
$\widetilde{z}_0 \leftarrow z_0$; $\widehat{t} \leftarrow T$
for $t = 0, \ldots, T-1$ **do**
  $\Delta V_t \leftarrow S_{t+1} \setminus S_t$
  $\widehat{z}_{t+1} \leftarrow \widetilde{z}_t + U_8 D_\theta(\widetilde{z}_t, q_{\mathrm{nat}}, \Delta V_t, c_t)$
  if $\mathrm{JS}(\mathcal{R}(\widetilde{z}_t), \mathcal{R}(\widehat{z}_{t+1})) \le \tau$ **then**
    $\widehat{t} \leftarrow t$; **break**
  end if
  $\widetilde{z}_{t+1} \leftarrow \widehat{z}_{t+1}$
end for
if $\widehat{t} = 0$ **then**
  return p_0
end if
$\widehat{K} \leftarrow K_{\widehat{t}}$; $M \leftarrow \min(2\widehat{K}, 136)$
$p_{\mathrm{final}} \leftarrow \textsc{ExpandedPass}(S^{(M)}, q, \text{prune to } 8)$
return p_{final}

used to retain eight tokens for the remaining layers. The reduction limits later-layer computation after the model has processed the wider context. Eight is the *final active* visual count, not the number processed throughout the decoder.

The implementation uses at most two real decoder passes. The initial pass processes $[S_0, Q_{\mathrm{seed}}]$ and returns the real seed state z_0 and distribution p_0. STOP-D predicts $\widehat{K}$ without further full-decoder refreshes. If $\widehat{K} = 8$, the method returns p_0. Otherwise, one independent final pass processes $[S^{(M(\widehat{K}))}, Q_{\mathrm{final}}]$, applies the within-decoder pruning, and supplies the answer. The current pruning implementation selects among tokens in that final pass, so it does not reuse the seed KV cache. We account for both passes when reporting computation.

The predicted states determine the route and hence the early entry width; every continued sample obtains its answer from the independent final decoder pass.

4 Experiments

4.1 Experimental Setup

Backbone and data. We evaluate Qwen2.5-VL-7B-Instruct (Bai et al., 2025) using 32 uniformly sampled frames, a video-before-question prompt, and fixed multiple-choice scoring. We developed the nested path on 240 NExT-QA (Xiao et al., 2021) videos and checked the selected path on a separate set of 280 videos. STOP-D was fitted and its threshold selected using video-disjoint, official-train-derived data. The evidence order, predictor, threshold, and execution mapping were fixed before the reported evaluations. Our NExT-QA evaluation comprises a frozen subset of 1,000 questions from the official validation set, not the entire benchmark. We also evaluate all 4,000 MVBench questions and TempCompass (Liu et al., 2024). NExT-QA was used repeatedly during method development, so its frozen subset should not be interpreted as an untouched external test set.

Baselines and accounting. FastVID (Shen et al., 2026) and FlashVID (Fan et al., 2026) provide measured external operating points. We additionally compare the historical narrow execution and an Always-Wide control that uses the same late pruning. We report task accuracy, the mean visual width entering the answer-producing decoder pass, the final active width, and executed visual token-layers. The latter charges both independent decoder passes of Route2K, including its seed pass; it is a computation proxy rather than measured latency. All cross-method comparisons use the same backbone and frame protocol, but the listed operating points are not necessarily matched in quality or runtime. TempCompass calibration details and measured latency appear in Appendix A.

4.2 Main Accuracy–Computation Results

Table 1: Selected measured operating points on the frozen 1,000-question NExT-QA validation subset, ordered by executed visual token-layers. Entry and Final are mean decoder widths. Route2K's cost includes its seed pass. The points are not matched in accuracy or runtime.

Method	Entry	Final	Visual token-layers	Accuracy (%)
FastVID	41.968	41.968	1,175.104	67.4
Route2K	83.144	8.000	1,599.008	71.7
FastVID	58.336	58.336	1,633.408	69.4
FastVID	83.008	83.008	2,324.224	71.4
FlashVID	135.888	41.176	3,047.168	73.1

Route2K reaches 71.7% accuracy on the frozen NExT-QA subset, compared with 69.0% for the historical narrow execution. The paired gain is 2.7 points: 49 narrow errors become correct and 22 correct answers become wrong. At nearly identical mean decoder entry width, FastVID reaches 71.4% but executes 2,324 rather than 1,599 visual token-layers on average. This proxy advantage does not imply a latency advantage. Always-Wide attains 73.0% accuracy with 2,272 token-layers, and FlashVID reaches 73.1% at a wider entry. Route2K thus saves 29.6% of the executed token-layer proxy relative to Always-Wide while giving up 1.3 accuracy points on this subset.

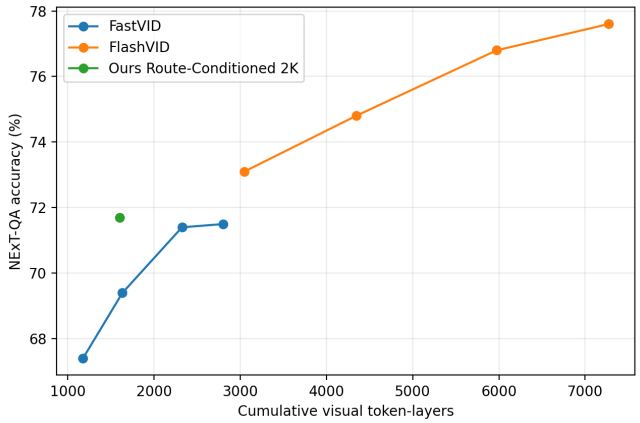

Figure 3: Accuracy versus executed visual token-layers on frozen NExT-QA-1000. Route2K's point charges both decoder passes. Token-layers are a computation proxy, not measured latency.

Figure 3 shows the measured accuracy–computation trade-off: Route2K uses fewer token-layers than a similarly accurate FastVID point, while higher-budget FlashVID points attain higher accuracy.

At almost the same executed token-layer budget, Route2K reaches 71.7% at 1,599, compared with FastVID's 69.4% at 1,633. This is a descriptive comparison across different token-selection and layerwise execution rules. Route2K's final eight-token width would obscure its wider early-layer computation if used alone as the cost measure.

Exploratory budget trajectories. A separate 13-budget sweep of 1,000 NExT-QA questions found 828 suffix-stable at 5% and 16 only at 100%. Suffix stability means agreement with the full-budget prediction at every larger measured budget, not ground-truth correctness. Correctness is non-monotonic for 96 questions; at 30%, compression corrects 30 full-budget errors but degrades 22 full-budget correct answers. These observations motivate sample-dependent allocation rather than evaluate Route2K.

4.3 CROSS-BENCHMARK EVALUATION

Table 2: Cross-benchmark operating points. MVBench baseline readout (generation/parser) differs from Route2K (option logits). TempCompass global and Target100 populations differ. External token-layers use measured widths and frozen layer schedules; Route2K costs are recorded directly.

Method / protocol	Entry	Final	Visual token-layers	Accuracy (%)
MVBench (4,000 questions)				
FastVID	48.450	48.450	1,356.586	51.225
FastVID	59.738	59.738	1,672.657	51.475
FastVID	92.454	92.454	2,588.712	54.200
FlashVID	158.829	47.649	3,557.764	57.800
FlashVID	211.872	63.562	4,745.938	59.425
FlashVID	303.101	90.930	6,789.468	62.050
Route2K	76.958	8.000	1,496.056	51.575
TempCompass (global: 4,033 questions; Target100: 3,533 questions)				
Route2K / global	41.396	8.000	854.931	55.939
Route2K / Target100	107.315	8.000	2,013.328	59.921
FastVID / Target100	96.960	96.960	2,714.883	60.826
FlashVID / Target100	329.874	99.412	7,392.778	68.723

On MVBench, Route2K reaches 51.575% at 1,496 token-layers versus FastVID's 51.475% at 1,673; differing answer readouts preclude a controlled accuracy claim. On the same TempCompass Target100 questions, Route2K reaches 59.921% at 2,013 versus FastVID's 60.826% at 2,715. The global row uses a different population; higher-budget baselines score higher.

4.4 VALIDATION ABLATIONS

Table 3: Full pre-registered ablation grid on 300 train-derived, video-disjoint validation questions. Routes, STOP-D, threshold, path, and eight-token tail are frozen. Executed token-layers include the seed pass; these are not frozen-test results.

Execution variant	Entry	Branch TL	Executed TL	Accuracy (%)
$M = K,\ L = 16$	48.747	875.947	1,046.187	83.000
$M = \lceil 1.5K \rceil,\ L = 16$	70.547	1,224.747	1,448.747	86.000
$M = 2K,\ L = 16$ **(Route2K)**	**83.760**	**1,436.160**	**1,606.400**	**87.667**
$M = 2K$ also for $K = 8$	85.680	1,466.880	1,690.880	87.333
$M = 80,\ L = 16$	80.000	1,376.000	1,600.000	85.333
$M = 136,\ L = 16$	136.000	2,272.000	2,496.000	87.667
$M = 2K,\ L = 12$	83.760	1,133.120	1,303.360	79.667
$M = 2K,\ L = 20$	83.760	1,739.200	1,909.440	86.667

Table 3 shows Route2K gaining 4.67 points over $M = K$ (15 corrections, one degradation) and 2.33 points over fixed $M = 80$ at nearly equal cost (ten corrections, three degradations). Always-Wide matches 87.67% with 55.4% more two-call token-layers; standalone fixed-width execution would omit the seed pass. Pruning earlier loses 8.00 points, while later pruning and seed-stop expansion add cost without improving accuracy. Reducing expansion to $1.5K$ trades 1.67 points for lower cost. These validation comparisons explain the frozen design. An older cached-feature Always-Wide run reported 88.0%; within-grid comparisons use the recomputed-feature result of 87.667%.

5 CONCLUSION

We studied adaptive visual computation through a nested evidence path and a lightweight transition predictor that selects early decoder width before late visual pruning. On NExT-QA, Route2K improves over narrow execution and reaches higher accuracy than a selected FastVID point with fewer visual token-layers. On MVBench and TempCompass, it remains within one reported accuracy point of selected FastVID settings at lower token-layer cost. These results support adaptive visual budgeting across video QA; aligned answer readouts and lower seed overhead are next steps toward wall-clock gains.

Reproducibility Statement

The paper specifies the evidence path, stopping policy, execution procedure, data splits, scoring rules, and cost measures. Our experiment records contain frozen configurations, provenance manifests with checksums, per-sample predictions for the main policy, baseline result tables, aggregation scripts, and timing logs. We distinguish validation analyses from frozen evaluation and identify when a comparison uses different populations or operating points. Reproducing model inference additionally requires the cited pretrained backbone and benchmark media, which are distributed by their original providers.

Ethics Statement

This work uses existing video question-answering benchmarks and does not collect new data from human participants. Adaptive visual computation can produce incorrect answers when it discards evidence needed for a particular question, and its failure rate may vary across video content and question types. Deployment in consequential settings would require evaluation on the target distribution, including the groups and failure modes relevant to that setting.

AI Use Statement

We used generative AI tools for research brainstorming, literature search, feedback on method, assistance with code development, and language editing of the manuscript. The authors reviewed all AI-assisted code and text and take responsibility for the content of this work, including AI-assisted figures and other artifacts.

References

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

Zizhong Ding, Junxian Li, Kai Liu, Shaoqiu Zhang, Xiao Xiao, Linghe Kong, and Yulun Zhang. Et-prune: Evidence-aware dynamic budgeting for visual token pruning in text-rich mllms. *arXiv preprint arXiv:2608.01979*, 2026.

Hao Dou and Ruiwen Tian. When do fewer visual tokens accelerate multimodal inference? a break-even study across decision locations and hardware. *arXiv preprint arXiv:2608.03649*, 2026.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

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.

Moshe Kimhi, Nimrod Shabtay, Raja Giryes, Chaim Baskin, and Eli Schwartz. Cares: Context-aware resolution selector for vlms. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2243–2256, 2026.

Huanxuan Liao, Zhongtao Jiang, Yupu Hao, Yuqiao Tan, Shizhu He, Ben Wang, Jun Zhao, Kun Xu, and Kang Liu. Resadapt: Adaptive resolution for efficient multimodal reasoning. *arXiv preprint arXiv:2603.28610*, 2026.

Bin Lin, Yang Ye, Bin Zhu, Jiaxi Cui, Munan Ning, Peng Jin, and Li Yuan. Video-llava: Learning united visual representation by alignment before projection. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 5971–5984, 2024.

Yuanxin Liu, Shicheng Li, Yi Liu, Yuxiang Wang, Shuhuai Ren, Lei Li, Sishuo Chen, Xu Sun, and Lu Hou. TempCompass: Do video LLMs really understand videos? In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 8731–8772, 2024.

Paribesh Regmi, Qingshuang Chen, Chi Zhang, Heba Aly, Yelin Kim, and Hongda Mao. Adaptive two-stage visual token pruning for efficient inference in video-language models. *arXiv preprint arXiv:2608.03112*, 2026.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Leqi Shen, Guoqiang Gong, Tao He, Yifeng Zhang, Pengzhang Liu, Sicheng Zhao, et al. Fastvid: Dynamic density pruning for fast video large language models. *Advances in Neural Information Processing Systems*, 38:123553–123581, 2026.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9772–9781. IEEE, 2021.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 2246–2251, 2020.

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.

Qizhe Zhang, Aosong Cheng, Ming Lu, Zhiyong Zhuo, Minqi Wang, Jiajun Cao, Shaobo Guo, Qi She, and Shanghang Zhang. [CLS] attention is all you need for training-free visual token pruning: Make VLM inference faster. *arXiv preprint arXiv:2412.01818*, 2024.

A Additional Experimental Results

A.1 TempCompass Calibration Details

Target100 sets a separate threshold without labels or accuracy using 500 questions from 48 videos, with no video overlap with the 3,533 evaluation questions. Its observed mean route width on the calibration set is 91.936 rather than exactly 100. The global-threshold run and Target100 run differ in both policy threshold and evaluation population; their accuracy difference cannot be attributed to calibration. No TempCompass correctness labels train STOP-D or select either reported threshold.

A.2 Measured Inference Time

Table 4: Mean latency on the same 100 NExT-QA questions (A800, BF16/FlashAttention-2, three repetitions per question). Post-ViT includes routing and decoder execution; model-side TTFT also includes vision encoding. These points are not a quality-matched runtime frontier.

Method	Post-ViT (ms)	Model TTFT (ms)
Full	392.014	725.128
FastVID	41.492	389.929
FlashVID	115.148	463.596
Route2K	81.339	446.126

Route2K averages 1.75 real decoder calls on this subset. Its model-side TTFT is 38.5% lower than Full and 3.8% lower than FlashVID, but it remains slower than FastVID. The independent seed pass offsets part of the token-layer saving. Runs followed the same timing protocol but were not interleaved; these measurements do not establish a quality-matched runtime advantage.

On this subset, vision encoding averages 364.8 ms; routing and language-model execution take 25.0 and 50.1 ms of Route2K's 81.3-ms post-ViT time. Relative to Full, Route2K cuts post-ViT time by 79.3%, but model-side TTFT by 38.5%. The corresponding reductions relative to FlashVID are 29.4% and 3.8%. The independent decision pass and shared vision-encoding cost limit how much a reduction in visual token-layers can improve end-to-end latency.