Stability Implies Redundancy: Delta Attention Selective Halting for Efficient Long-Context Prefilling

Anonymous ACL submission

Abstract

Prefilling computational costs pose a significant bottleneck for Large Language Models (LLMs) and Large Multimodal Models (LMMs) in long-context settings. While token pruning reduces sequence length, prior methods rely on heuristics that break compatibility with hardware-efficient kernels like FlashAttention. In this work, we observe that tokens evolve toward *semantic fixing points*, making further processing redundant. To this end, we introduce Delta Attention Selective Halting (DASH), a training-free policy that monitors the layer-wise update dynamics of the self-attention mechanism to selectively halt stabilized tokens. Extensive evaluation confirms that DASH generalizes across language and vision benchmarks, delivering significant prefill speedups while preserving model accuracy and hardware efficiency.

1 Introduction

The capability to process extensive context windows is a key feature of Large Language Models (LLMs) and Large Multimodal Models (LMMs), enabling long-document summarization and high-resolution visual understanding (OpenAI et al., 2024; Team et al., 2025). However, this capability comes with a prohibitive computational cost. While techniques such as quantization (Dettmers et al., 2022) and FlashAttention (Dao et al., 2022) have alleviated memory bottlenecks, the prefill complexity remains quadratic or linear with respect to sequence length. As token sequences grow into the hundreds of thousands, the floating-point operations required for the forward pass become the primary latency bottleneck.

To address this problem, recent research has explored token pruning strategies to reduce the sequence length (Zhang et al., 2023). However, most existing methods rely on heuristic importance scores (e.g., accumulated attention weights) that require access to the attention matrix, making them incompatible with FlashAttention and other efficient attention operators. This leads to a question: instead of asking *which tokens are important*, can we identify *which tokens have already finished their job*?

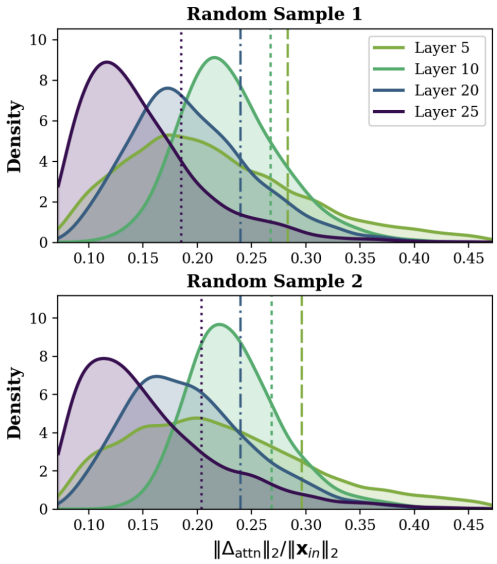

Figure 1: Layer-wise distributions of token-wise relative Δ_{attn} at layers 5, 10, 20, and 25 for Qwen2.5-7B-Instruct-1M, computed on two randomly sampled LongBench-E (2WikiMQA) instances.

To answer this question, we revisit token dynamics across layers through the lens of residual connections, where $x_{l+1} = x_l + \Delta$. We posit that tokens evolve towards a "semantic fixed point", where a diminishing update magnitude Δ signals that the representation has stabilized within the current context. Specifically, we monitor the update of the self-attention layer (Δ_{attn}) to capture global information exchange, rather than local feature variations.

Observation 1: Semantic fixing points exist. This hypothesis is validated by the empirical distribution of Δ_{attn}, as shown in Figure 1. We observe a highly skewed pattern where the vast majority of

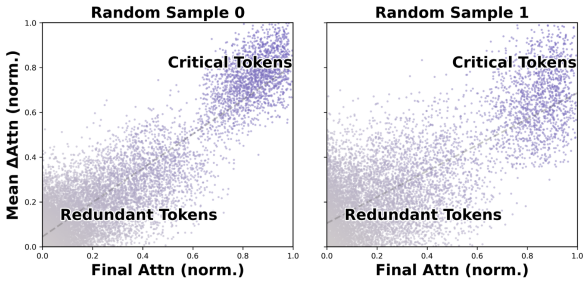

Figure 2: Correlation between normalized final-layer attention scores (x-axis) and the mean Delta Attention Δ_{attn} of each token across layers (y-axis).

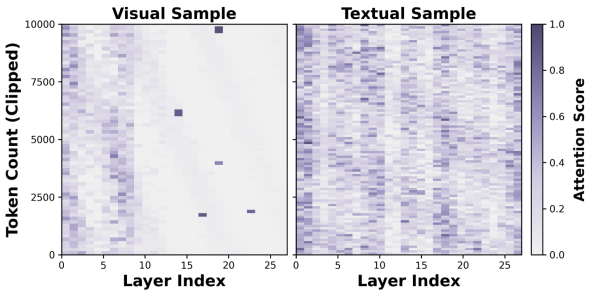

Figure 3: Layer-wise sparsity patterns for visual (left) and textual (right) tokens, showing earlier saturation for visual tokens compared with text tokens.

tokens cluster near zero, indicating they have effectively reached their semantic fixed points. Only a sparse subset of tokens continues to undergo significant updates in deeper layers. This distinct separation reveals a substantial computational redundancy: once a token enters this "saturation regime," further processing contributes little to the global context and yields diminishing returns.

Observation 2: Tokens converged to their fixed point are redundant. To confirm that the semantic fixed point" identified in Observation 1 translates to a loss of influence over the global context, we analyze the correlation between update magnitude and token importance. Figure 2 plots each token's normalized attention score against its Δ_{attn}, revealing a strong positive relationship: tokens that stabilize (low Δ_{attn}) rarely attract significant attention from subsequent layers. This behavior suggests that once a token saturates, it effectively becomes an information sink," allowing us to use Δ_{attn} as a safe, training-free proxy for pruning without computing expensive attention matrices.

Observation 3: Visual tokens saturate earlier than textual tokens. While saturation serves as a general indicator of redundancy, the *rate* of convergence differs fundamentally across modalities. Figure 3 compares the layer-wise attention sparsity patterns for visual and textual tokens within a vision-language model. We observe a distinct disparity: visual tokens tend to reach their semantic fixed points and exhibit high sparsity significantly earlier in the network, whereas textual tokens require deeper layers to process and stabilize. This modality gap explains a common pitfall in efficient inference: token pruning strategies originally developed for large multimodal language models often fail when directly applied to Large Language Models, calling for token pruning methods designed for both language and vision.

Building on these insights, we propose **DASH** (Delta Attention Selective Halting), a training-free inference-time policy for fast prefill. DASH computes token-wise Δ_{attn} once at a start layer l_s, selects a compact active set by retaining the $topK = \lfloor (1-\rho)T \rfloor$ tokens, and reuses this fixed set for all subsequent prefill layers. Tokens outside the active set are halted at l_s and skip both self-attention and FFN computation in deeper layers. This single-shot schedule is compatible with efficient attention operators (e.g., FlashAttention (Dao et al., 2022)) since it does not require materializing full attention matrices. We evaluate DASH on both long-context text and vision–language benchmarks. Across these settings, DASH consistently outperforms representative prefill-time token compression baselines under the same token reduction setting, while remaining close to the uncompressed backbone. Moreover, the vision–language results exhibit a larger performance margin under more aggressive reduction ratios, indicating that DASH retains accuracy more effectively when prompts are inflated with dense visual tokens.

In summary, our contributions are three-fold:

- We identify per-layer Δ_{attn} as a first-order indicator of token saturation and provide empirical evidence of its correlation with token importance and the difference between vision and language (Figures 1, 2, and Figure 3).
- We introduce **DASH**, a training-free framework that leverages Δ_{attn} to perform single-shot selective token halting at l_s, substantially reducing prefill latency without retraining.
- Extensive experiments on long-context language and vision–language benchmarks with Qwen2.5-7B-Instruct-1M and Qwen2-VL-7B validate **DASH**, delivering consistent prefill efficiency gains with minimal performance degradation and strong accuracy–efficiency trade-offs against competitive baselines.

2 Related Work

2.1 Memory-Oriented Inference

Memory-oriented approaches primarily aim to compress the key-value (KV) cache to reduce memory footprint and bandwidth. Common strategies include eviction policies based on accumulated attention or recency (Zhang et al., 2023; Li et al., 2024b; Adnan et al., 2024), and more refined criteria leveraging head-level or value-aware signals (Ge et al., 2024; Guo et al., 2024). While effective for decoding efficiency, these methods offer limited savings for the dominant prefill computation, as the full prompt must typically be processed to construct the initial cache.

2.2 Structural Compute Reduction

Complementary to cache compression, structural approaches reduce inference FLOPs by pruning the computation graph. Techniques range from exploiting specific attention patterns like attention sinks (Xiao et al., 2024) to dynamic token pruning strategies (Fu et al., 2024; Long et al., 2025; Yan et al., 2025) and depth-wise early exiting (Tang et al., 2023). While token pruning is well-explored in vision (Wei et al., 2023), training-free token reduction for autoregressive text generation remains understudied. Our work addresses this gap by proposing a lightweight policy that targets prefill FLOPs and generalizes to multimodal inputs.

2.3 Information Propagation and Redundancy

Recent analyses of deep Transformers reveal significant redundancy, where deeper layers often introduce only minor representation updates (He et al., 2024). Mechanistic studies further suggest that attention heads perform iterative computation, gradually refining intermediate states (Brinkmann et al., 2024; Musat, 2025). These findings motivate shifting from static importance metrics to criteria based on dynamic updates. Accordingly, we leverage the magnitude of the attention residual update as a layer-local proxy for marginal computational return to guide our training-free selective halting.

3 Methodology

3.1 Preliminaries

Long-context and multimodal inference. Inference for autoregressive Transformers consists of a *prefill* stage and a *decoding* stage. Given an input sequence of length T, prefill processes all T tokens once to produce hidden states and initialize the KV cache, after which decoding generates tokens autoregressively while reusing cached keys and values. In long-context settings, prefill can dominate latency because it executes attention and feed-forward computation over the entire prompt, and the cost grows rapidly with T (quadratically for self-attention). In practice, T is often inflated beyond plain text due to retrieval-augmented prompts and multimodal inputs with dense visual tokens, further increasing time-to-first-token. We therefore focus on reducing prefill computation for both long-context text generation and multimodal generation.

Transformer blocks. We consider a Transformer with L blocks. Let $\mathbf{H}^{(l)} \in \mathbb{R}^{T\times d}$ denote the token representations entering the self-attention sublayer at block l. The self-attention output *before* residual addition is denoted by $\mathbf{U}^{(l)} \in \mathbb{R}^{T\times d}$, yielding the residual update $\widetilde{\mathbf{H}}^{(l)} = \mathbf{H}^{(l)} + \mathbf{U}^{(l)}$.

Token pruning and pruning ratio. Token pruning implements conditional computation by halting a subset of tokens in intermediate and deep layers, so that only the remaining tokens are propagated through subsequent blocks. We denote by $\rho \in [0, 1)$ the *pruning ratio*, meaning that $(1-\rho)T$ tokens are retained at layers where pruning is activated. Prior evidence suggests that information can diffuse across depth, motivating depth dependent selective computation that reduces the effective sequence length in later layers while preserving overall behavior.

3.2 Overview of DASH

Motivated by the prefill bottleneck in long-context and multimodal inference, we introduce **DASH**, a training-free inference-time policy for reducing prefill FLOPs via token-level selective halting in intermediate and deep Transformer blocks, as illustrated in Figure 4. We consider an input token sequence of length T, including both text-only prompts and multimodal prompts formed by concatenating text and visual tokens. DASH makes no modality-specific assumptions and instead instantiates a unified halting criterion based on a layer-local signal available during the forward pass. Halting is activated from a start layer l_s. For $l < l_s$, the model processes all T tokens as usual. At the start layer l_s, DASH computes a per-token Δ_{attn} score, ranks tokens by this score, and retains a fixed fraction determined by the pruning ratio ρ to form an active set. The same active set is then reused for

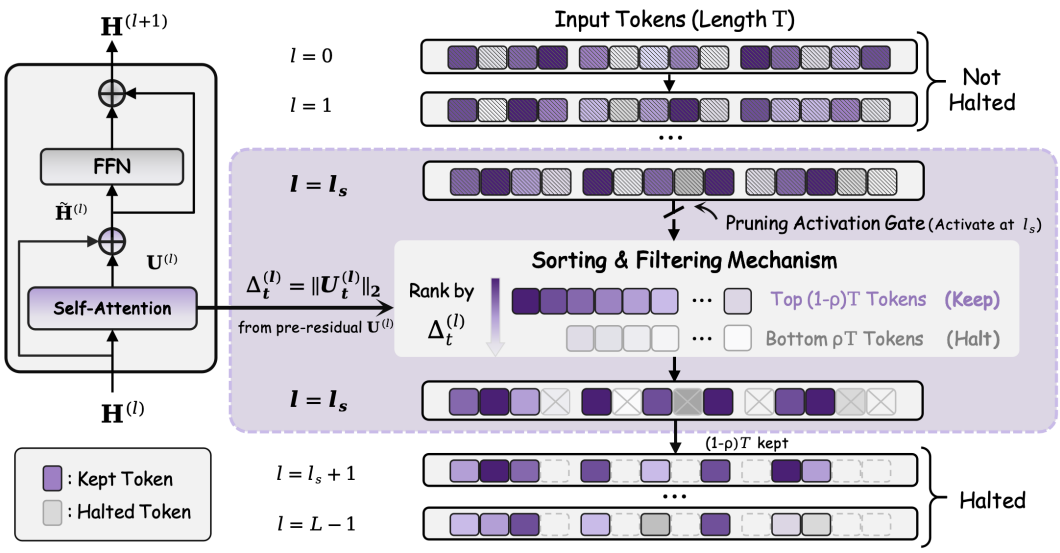

Figure 4: Overview of Δ_{attn} guided token halting during prefill. **Left:** in block l, self-attention produces the pre-residual output $\mathbf{U}^{(l)}$, and we define a per-token Δ_{attn} score $\Delta_t^{(l)} = \|\mathbf{U}_t^{(l)}\|_2$. **Right:** given an input of length T, all tokens are processed normally for layers $l < l_s$. At the activation layer l_s, we rank tokens by $\Delta_t^{(l_s)}$ and apply a fixed pruning ratio ρ, keeping the top $(1-\rho)T$ tokens (purple) and halting the remaining tokens (gray). Halted tokens are removed from the active set and, in all deeper layers, *skip the entire Transformer block computation*, including both self-attention and the feed-forward network (FFN), thereby reducing prefill FLOPs. The figure highlights the kept tokens (filled purple), halted tokens (gray with cross), and the pruning gate activated at l_s.

all subsequent layers $l > l_s$, so halted tokens skip both self-attention and feed-forward computation throughout the remaining prefill blocks.

Consequently, the sequence length drops at l_s and remains reduced in deeper layers, reducing the computational cost of both attention and feed-forward sublayers during prefill.

3.3 Δ_{attn} Signal and Active-Set Update Rule

Let $\mathbf{H}^{(l)} \in \mathbb{R}^{T\times d}$ denote the token representations entering block l during prefill. We define the attention output *before* the residual addition as

$$\mathbf{U}^{(l)} = \mathrm{Attn}\Big(\mathrm{LN}(\mathbf{H}^{(l)})\Big)\,, \tag{1}$$

and compute a layer-local delta-attn score for each token t as

$$\Delta_t^{(l)} = \Big\|\mathbf{U}_t^{(l)}\Big\|_2\,. \tag{2}$$

We interpret $\Delta_t^{(l)}$ as a proxy for the marginal benefit of further updating token t at depth l.

Given a pruning ratio $\rho \in [0, 1)$, DASH performs *single-shot* selection at the start layer l_s. Let $S = \{1, \ldots, T\}$ be token indices. At layer l_s, we select

$$\begin{aligned} S^\star &= \mathrm{TopK}\Big(S,\, K,\, \Delta^{(l_s)}\Big)\,, \\ K &= \lfloor (1-\rho)T \rfloor\,, \end{aligned} \tag{3}$$

reuse $S^\star$ for all subsequent layers, and halt tokens in $S \setminus S^\star$ at l_s. Unless stated otherwise, the rule is applied uniformly to all tokens, including multimodal visual tokens when present. Implementation details and pseudo-code are provided in Appendix B.1 (Alg. 1).

3.4 Execution Semantics, Scheduling, and Complexity

A token is *halted* once it is excluded from the active set. Halted tokens are removed from the active set, skip both self-attention and feed-forward computation in all subsequent layers, and their hidden states remain fixed at their last updated values. We implement halting by executing each block on the compacted active sequence, so computation scales with the active length rather than the prompt length.

Under the single-shot schedule, layers $0{:}l_s-1$ process the full prefill sequence of length T, while layers $l_s{:}L{-}1$ reuse a fixed active set of length $\hat{T} = \lfloor (1-\rho)T \rfloor$. Using the standard per-layer FLOPs proxy $A(T)$, the baseline and DASH prefill FLOPs are $C_{\text{full}}(T) = L\,A(T)$ and $C_{\text{ours}}(T) = l_s A(T) + (L - l_s)A(\hat{T})$, yielding a theoretical prefill FLOPs speedup $s_{\text{FLOPs}}(T) = C_{\text{full}}(T)/C_{\text{ours}}(T)$. For our typical text-only setting on QWEN2.5-7B-INSTRUCT-1M with $l_s = \lfloor 0.4L \rfloor$ and $\rho = 0.667$, the theoretical speedup is $1.83\times$ at $T = 16{,}384$ to-

Table 1: LongBench-E results on QWEN2.5-7B-INSTRUCT-1M under the same prefill-time token reduction setting. We compare DASH with representative baselines, including SnapKV adapted to a pruning variant (SnapKV-pr), D^3, LLMLingua2, and a long-context adaptation of FastV. Avg. (%) denotes the average score across tasks; higher is better. Bold and underlined numbers indicate the best and the second-best results in each column, respectively.

Method	Avg. (%)	Sing. QA		Multi. QA		Summ.		Few-shot			Synth. Task		Code Comp.	
		Qasper	MFQA	Hotp.QA	2Wiki.	GovRep	MNews	TREC	Triv.QA	S.Sum	Count	Retrieval	LCC	Rep.-P
Qwen2.5-7B-Instruct-1M	48.87	44.19	51.13	62.97	51.07	6.97	6.57	65.00	85.75	32.95	15.00	99.33	59.86	54.52
+ FastV	43.99	40.44	42.63	57.67	43.48	6.96	6.53	59.33	84.79	**33.98**	7.33	83.67	52.71	52.31
+ LLMLingua2	44.16	40.41	42.91	51.58	43.29	6.44	6.41	61.67	80.87	32.52	6.33	99.00	49.64	**53.05**
+ D^3	45.00	40.18	44.49	60.95	**50.16**	6.19	6.01	**64.67**	85.01	27.14	15.00	**99.33**	45.54	40.34
+ SnapKV (pr.)	46.15	38.14	42.98	**61.54**	48.31	7.00	**6.70**	63.67	**85.25**	30.87	16.60	97.67	51.89	49.37
+ DASH (ours)	**46.76**	**40.58**	**49.38**	61.00	48.03	**7.01**	6.47	59.00	84.21	31.82	**16.67**	98.00	**54.58**	51.14

kens. Full derivations and length-dependent results are provided in Appendix A.1.

4 Experiments

4.1 Experimental Setup

Evaluation focus. We evaluate DASH as a *prefill-time* token compression method for long-context inference. Given an input prompt of length T, DASH reduces prefill computation by halting a subset of prompt tokens after a designated start layer l_s and executing subsequent layers only on the compacted active set. Unless otherwise stated, decoding is kept identical across methods; token compression is applied exclusively to the prompt tokens in prefill.

Backbones. For long-context text experiments, we use QWEN2.5-7B-INSTRUCT-1M (Yang et al., 2025). For vision–language (VL) experiments reported later in §4.3, we use QWEN2-VL-7B (Wang et al., 2024).

Benchmarks. For long-context text, we consider (i) LONGBENCH-E (Bai et al., 2024), which covers single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion (Table 1), and (ii) LOOGLE (Li et al., 2024a), which stresses long-context understanding and includes short/long QA, long summarization, and cloze-style evaluation (Table 2). We follow the official evaluation protocols of each benchmark and report their standard task scores; higher is better.

Baselines. We compare against representative prefill-time compression baselines: (i) SnapKV (Li et al., 2024b) adapted to a token-pruning variant (SNAPKV(PR.)), (ii) D_3 (Fan et al., 2025), (iii) LLMLINGUA2 (Pan et al., 2024), and (iv) a long-context adaptation of FASTV (Chen et al., 2024). All methods are evaluated under the same backbone and inference configuration.

Operating-point selection (accuracy–speedup trade-off). Many baselines expose tunable hyperparameters that induce different accuracy–efficiency trade-offs. Therefore, in Table 1 and Table 2, we report each method at a *selected operating point* from its sweep results, chosen to represent its best observed accuracy–speedup trade-off under our evaluation setting. We report detailed end-to-end quality–efficiency breakdowns separately (Figure 6). For DASH, the sweep is over the start layer l_s and pruning ratio ρ (and any scheduling parameters described in Appendix B).

4.2 Long-Context Text Prefill Compression Results

4.2.1 LongBench-E Main Results

Table 1 summarizes LONGBENCH-E results on QWEN2.5-7B-INSTRUCT-1M at the selected operating point for each method. Among the prefill-time compression baselines, DASH achieves the strongest overall average score (Avg. = 46.76), outperforming SNAPKV(PR.) (46.15), D^3 (45.00), LLMLINGUA2 (44.16), and FASTV (43.99), while remaining close to the uncompressed backbone (48.87). At this operating point, DASH also achieves a higher speedup (lower FLOPs) than competing methods at comparable accuracy (Table 4).

Beyond the overall average, DASH remains competitive across diverse task families. Notably, it is robust on information-intensive settings such as multi-field QA and code completion (e.g., MFQA and LCC in Table 1), and it preserves strong performance on synthetic retrieval-style tasks where maintaining salient prompt evidence is essential. These results support that delta-attention-guided halting can reduce prefill computation without dis-

Table 2: LooGLE results on QWEN2.5-7B-INSTRUCT-1M under the same prefill-time token reduction setting. We compare DASH with SnapKV adapted to a token pruning variant (SnapKV pr.), D^3, LLMLingua2, and a long-context adaptation of FastV. Avg denotes the overall benchmark score; higher is better. Bold and underlined numbers indicate the best and the second-best results in each column, respectively.

Method	Avg.	sQA		lQA		lSum		sCloze	
		BF1	RL	BF1	RL	BF1	RL	Ex.	Part.
Qwen2.5-7B-Instruct-1M	22.69	0.842	0.234	0.830	0.123	0.842	0.043	73.5	83.3
+ D^3	19.49	0.830	0.124	0.822	0.078	0.831	0.038	72.1	81.1
+ LLMLingua2	19.56	**0.841**	0.180	0.828	0.113	**0.842**	**0.049**	73.1	80.5
+ FastV	19.78	0.840	**0.202**	**0.829**	0.106	0.840	0.040	73.0	82.4
+ SnapKV (pr.)	19.87	0.836	0.171	0.827	**0.116**	0.841	0.043	73.2	82.9
+ **DASH (ours)**	**19.94**	0.839	0.172	**0.829**	0.112	0.841	0.040	**73.6**	**83.1**

proportionately harming long-context accuracy.

4.2.2 LooGLE Results

LooGLE complements LongBench-E by stressing long-range evidence tracking under QA, long summarization, and cloze-style evaluation. Table 2 reports LOOGLE results under the same setting.

DASH delivers the best overall score among the compared compression methods (Avg. = 19.94), slightly exceeding SNAPKV(PR.) (19.87) and improving over FASTV, LLMLINGUA2, and D3. On cloze-style evaluation, DASH attains the highest Exact and Partial scores (73.6 and 83.1, respectively), indicating that it better preserves the long-context evidence required for precise span-level inference. Overall, the LOOGLE results further corroborate that DASH maintains long-context understanding under prefill-time token reduction.

4.3 Vision–Language Prefill Compression Results

Figure 5 summarizes vision–language token compression results on QWEN2-VL-7B across six VL benchmarks under varying token reduction ratios. We report performance using the average decline ratio (ADR), where the uncompressed model is normalized to 100.0 and higher values indicate better retention of task accuracy.

Across all reduction ratios, DASH consistently achieves the highest ADR among the compared methods. Notably, its advantage becomes more pronounced under aggressive compression. At moderate reduction (e.g., 75%–88.9%), DASH already matches or exceeds prior methods, while at extreme reduction levels (96% and 99%), it exhibits substantially slower degradation than FASTV, VISIONZIP, and DART. This trend indicates that delta-attention-guided halting is particularly effective in identifying and preserving the small subset of visual–textual tokens that remain influential in deeper layers.

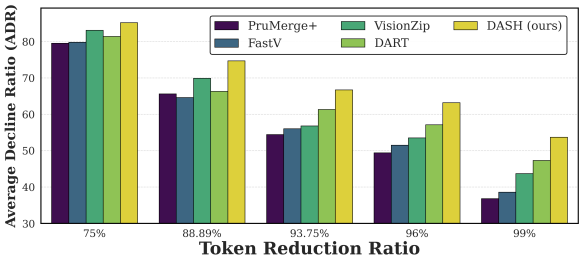

Figure 5: Vision–language token compression on Qwen2-VL-7B across six vision–language benchmarks. We compare PruMerge+, FastV, VisionZip, DART, and DASH (ours) under different token reduction ratios. Performance is reported as the average decline ratio (ADR), averaged over benchmarks, with the uncompressed model normalized to 100.0. Full per-benchmark results are provided in Table 5 in the appendix.

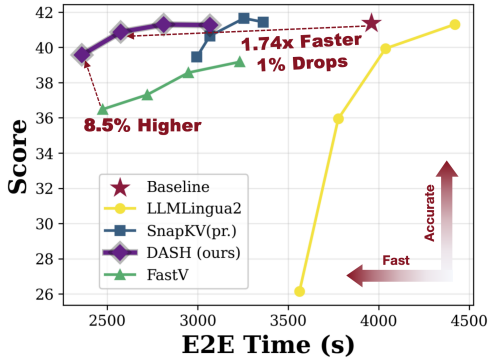

Figure 6: Overall LongBench-E performance trade-off between end-to-end (E2E) time and score. DASH achieves higher accuracy at lower latency, outperforming FastV by 8.5% in score and running 1.74× faster than the baseline under comparable accuracy.

This robustness is consistent with two observations. First, VL prompts typically contain a large number of visually redundant tokens whose contributions saturate early, making them suitable candidates for early halting. Second, by relying on attention-branch deltas rather than heuristic importance scores, DASH may better capture cross-modal interactions that remain active across layers. Full per-benchmark results are provided in Table 5 in the appendix.

4.4 End-to-End Efficiency and Accuracy Trade-off

While the above results focus on task accuracy at selected operating points, prefill-time compression ultimately aims to improve end-to-end (E2E) in-

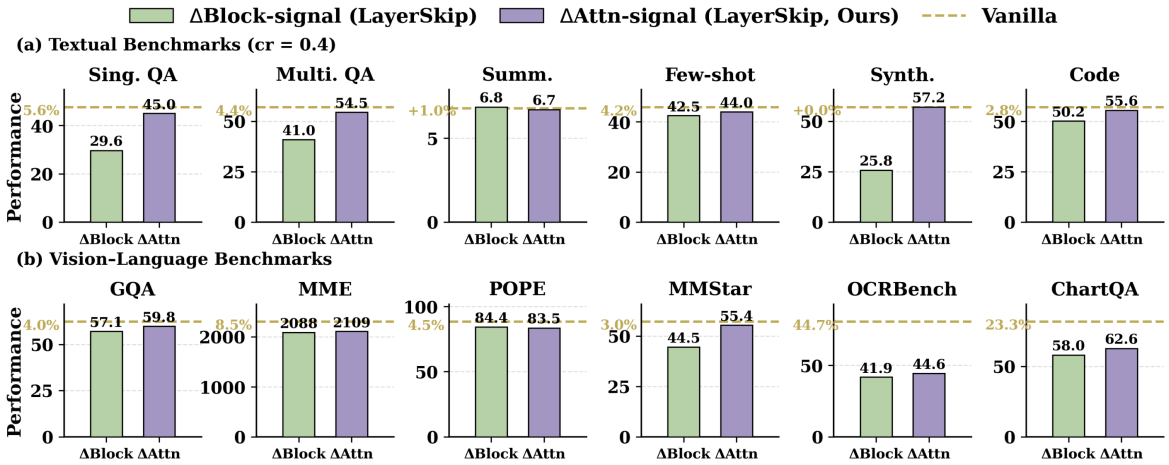

Figure 7: Ablation of delta-signal choices for layer-level token halting. **(a)** Textual benchmarks from LongBench-E at compression ratio $\mathrm{cr} = 0.4$, evaluated on Qwen2.5-7B-Instruct-1M. **(b)** Vision–language benchmarks evaluated on Qwen2-VL-7B with a fixed token removal ratio of 75%. Bars compare Δ*Block-signal* (layer-level delta from the whole Transformer block) and Δ*Attn-signal* (delta from the attention branch only), where both variants apply the same token halting mechanism and differ only in the delta signal used for decision making. The dashed horizontal line denotes the *Vanilla* (no layer skipping) baseline performance for each benchmark, and the percentage annotated above it indicates the relative performance change of the best layer-skipping variant with respect to the Vanilla baseline.

ference efficiency. Figure 6 plots the trade-off between overall LongBench-E score and E2E inference time. Because prefill dominates time-to-first-token in long-context settings, reductions in prefill FLOPs translate directly into end-to-end latency improvements.

DASH occupies a favorable region of the trade-off curve, achieving higher accuracy at lower latency than all baselines. Under comparable E2E time, it outperforms FastV by 8.5%. Conversely, at comparable accuracy to the uncompressed baseline, DASH runs 1.74× faster with only a marginal performance drop. These gains stem primarily from reduced prefill computation, while decoding remains unchanged across methods.

Importantly, these improvements are not achieved by trading accuracy for speed in a narrow regime. Instead, DASH offers a smooth accuracy–efficiency frontier, allowing practitioners to select operating points that balance latency and quality according to deployment constraints. A detailed breakdown of task-wise E2E quality and efficiency metrics is reported in Appendix Figure 6.

4.5 Ablation: Choice of Delta Signal for Token Halting

We ablate the choice of delta signal used for token-level halting applied from layer l_s. Specifically, we compare Δ_{block}-*signal*, which measures representation change across the entire Transformer block, against Δ_{attn}-*signal*, which isolates the contribution from the attention branch. Both variants employ identical layer-skipping mechanisms and differ only in the signal used for halting decisions.

As shown in Figure 7(a), on textual benchmarks from LongBench-E at compression ratio $\mathrm{cr} = 0.4$, Δ_{attn}-signal consistently outperforms Δ_{block}-signal across most task categories, including single- and multi-document QA, few-shot learning, synthetic tasks, and code completion, while maintaining comparable summarization performance. A similar pattern is observed on vision–language benchmarks (Figure 7(b)), where Δ_{attn}-signal yields higher accuracy on tasks such as GQA, MMStar, OCRBench, and ChartQA.

These results suggest that attention-branch deltas provide a more faithful indicator of token relevance in deeper layers. While block-level deltas conflate attention updates with feed-forward transformations that may reflect local reparameterization rather than contextual influence, attention deltas more directly capture whether a token continues to participate in global information aggregation. This property makes Δ_{attn}-signal better suited for guiding token halting under aggressive compression.

5 Discussion

Token dynamics: a heavy-tailed view from Δ_{attn}. Figure 8 provides an intuitive explanation of why DASH can aggressively reduce prefill com-

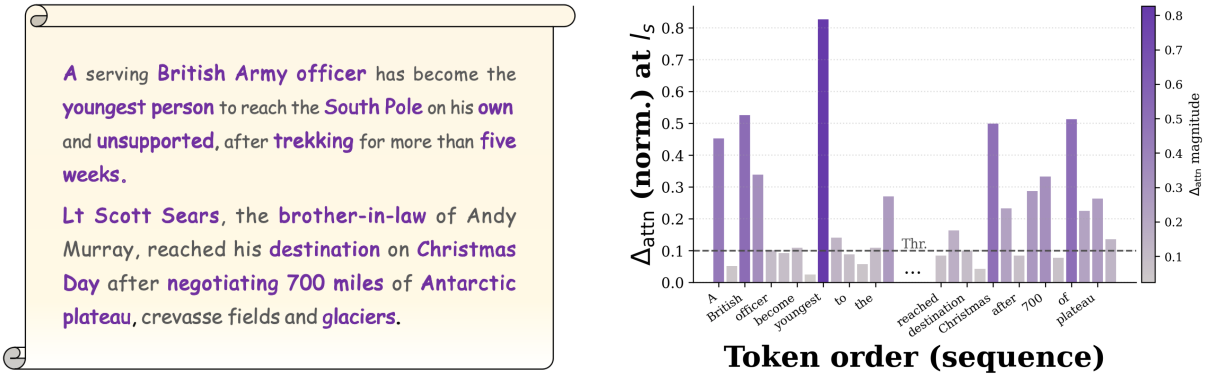

Figure 8: Qualitative example from LongBench-E (MultiNews) showing that token-wise Δ_{attn} at the decision layer l_s tends to assign higher scores to semantically informative tokens. The dashed line indicates the pruning threshold and ellipses denote omitted tokens.

putation while retaining accuracy. At the decision layer l_s, token-wise Δ_{attn} exhibits a pronounced heavy-tailed pattern: most tokens lie in a narrow low-Δ_{attn} band, while only a small fraction forms a long tail with substantially larger updates. This structure naturally supports supports a natural separation for single-shot TopK selection at l_s. Tokens with small Δ_{attn} at l_s are unlikely to benefit from further updates, so they can be halted at l_s with minor impact, while tail tokens are retained in the active set. In this sense, Δ_{attn} acts as a layer-local indicator of whether a token remains *active* in shaping attention updates beyond l_s.

Why the attention-branch delta matters. The ablation in Figure 7 further clarifies that not all delta signals are equally suitable for halting decisions. Using Δ_{attn}-signal consistently outperforms the block-level alternative across both long-context text and vision–language benchmarks under the same layer-skipping mechanism. A plausible interpretation is that attention-branch deltas directly reflect a token's marginal participation in cross-token (and cross-modal) aggregation, whereas block-level deltas conflate attention updates with feed-forward transformations that may alter representations without indicating continued global influence. Combined with the heavy-tailed dynamics observed in Figure 8, this explains why Δ_{attn} yields a more reliable separation between tokens that can be halted and tokens that should remain in the active set.

Implications for robustness under aggressive compression. The heavy-tailed separation in Figure 8 also sheds light on the empirical trend that DASH degrades more gracefully at higher reduction ratios. When compression becomes aggressive, the primary failure mode is to mistakenly discard the small set of tokens that carry persistent global influence. A signal with clearer tail separation is therefore crucial in the high-compression regime. This aligns with the vision–language results in Figure 5, where DASH maintains a larger margin over prior methods as token reduction increases.

From prefill savings to end-to-end gains. Finally, Figure 6 shows that the token-level decisions guided by Δ_{attn} translate into a favorable end-to-end accuracy–latency frontier. Because DASH targets prefill-time computation while keeping decoding identical, improvements in prefill efficiency can be directly reflected in reduced E2E time at comparable quality. Together, the qualitative evidence in Figure 8 and the ablation in Figure 7 provide a coherent explanation for why DASH achieves strong accuracy retention (Tables 1, 2) while improving practical serving efficiency.

6 Conclusion

We introduced DASH, a training-free inference-time method that reduces long-context prefill computation via delta-attention-guided selective token halting: DASH selects a compact active set at a start layer l_s using token-wise Δ_{attn} and reuses it in subsequent layers, while keeping decoding unchanged. Our analyses further show that attention-branch deltas are more reliable than block-level deltas for halting decisions, and token-level visualizations reveal a heavy-tailed Δ_{attn} pattern that separates a small set of persistently influential tokens from largely redundant ones. Overall, DASH offers a practical and effective approach to accelerating prefill for long-context and multimodal inference, making AI applications affordable.

Limitations

This paper focuses on the core design and evaluation of Δ_{attn}–guided selective token halting for prefill acceleration. While we benchmark DASH on a broad set of long-context and vision–language tasks, some additional ablations and per-setting results are omitted from the main text due to space constraints and can be included in supplementary material. Moreover, we do not exhaustively explore all model scales, architectures, or deployment configurations; studying these extensions and their interactions with system-level optimizations is left for future work.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Preprint*, arXiv:2403.09054.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. *Preprint*, arXiv:2308.14508.

Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. 2024. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. *Preprint*, arXiv:2402.11917.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. 2024. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. *Preprint*, arXiv:2403.06764.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Preprint*, arXiv:2205.14135.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Preprint*, arXiv:2208.07339.

Siqi Fan, Xuezhi Fang, Xingrun Xing, Peng Han, Shuo Shang, and Yequan Wang. 2025. Position-aware depth decay decoding (d^3): Boosting large language model inference efficiency. *Preprint*, arXiv:2503.08524.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. 2024. Lazyllm: Dynamic token pruning for efficient long context llm inference. *Preprint*, arXiv:2407.14057.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. *Preprint*, arXiv:2406.12335.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. 2024. What matters in transformers? not all attention is needed. *Preprint*, arXiv:2406.15786.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. 2024a. Loogle: Can long-context language models understand long contexts? *Preprint*, arXiv:2311.04939.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. Snapkv: Llm knows what you are looking for before generation. *Preprint*, arXiv:2404.14469.

Lingkun Long, Rubing Yang, Yushi Huang, Desheng Hui, Ao Zhou, and Jianlei Yang. 2025. Sliminfer: Accelerating long-context llm inference via dynamic token pruning. *Preprint*, arXiv:2508.06447.

Tiberiu Musat. 2025. Mechanism and emergence of stacked attention heads in multi-layer transformers. *Preprint*, arXiv:2411.12118.

OpenAI, Josh Achiam, and Steven Adler. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *Preprint*, arXiv:2403.12968.

Quan Tang, Bowen Zhang, Jiajun Liu, Fagui Liu, and Yifan Liu. 2023. Dynamic token pruning in plain vision transformers for semantic segmentation. *Preprint*, arXiv:2308.01045.

Gemini Team, Rohan Anil, and Sebastian Borgeaud. 2025. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

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

Siyuan Wei, Tianzhu Ye, Shen Zhang, Yao Tang, and Jiajun Liang. 2023. Joint token pruning and squeezing towards more aggressive compression of vision transformers. *Preprint*, arXiv:2304.10716.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

Yaoyao Yan, Hui Yu, and Weizhi Xu. 2025. Attribution-driven adaptive token pruning for transformers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, and 9 others. 2025. Qwen2.5-1m technical report. *Preprint*, arXiv:2501.15383.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *Preprint*, arXiv:2306.14048.

A Theoretical FLOPs

A.1 Theoretical FLOPs of Single-shot Halting in DASH

FLOPs proxy. Following FastV (Chen et al., 2024), we approximate the per-layer FLOPs of a Transformer block (MHA + FFN) for sequence length n as

$$A(n) \;=\; 4nd^2 + 2n^2d + 2ndm, \qquad (4)$$

where d is the hidden size and m is the FFN intermediate size.

Single-shot halting. Let L be the number of layers and l_s the halting start layer. Under single-shot halting, layers $0{:}l_s{-}1$ process the full sequence of length n, while layers $l_s{:}L{-}1$ reuse a fixed active set with length $\hat{n}$.

Effective kept length. All FLOPs analysis in this section focuses on the prefill stage; decoding-stage costs are unaffected by single-shot halting and thus excluded. Our implementation always keeps a prefix of n_{first} tokens and a suffix of n_{last} tokens, i.e., $n_{\text{fix}} = n_{\text{first}} + n_{\text{last}}$, and drops a fraction c of the remaining eligible tokens. Thus,

$$\hat{n} \;=\; n_{\text{fix}} + (1-c)(n - n_{\text{fix}}) \;=\; (1-c)n + c\,n_{\text{fix}}. \qquad (5)$$

Total FLOPs, reduction, and speedup. The baseline and single-shot FLOPs are

$$C_{\text{full}}(n) = L\,A(n), \qquad (6)$$

$$C_{\text{ours}}(n) = l_s A(n) + (L - l_s)A(\hat{n}). \qquad (7)$$

Therefore,

$$r_{\text{FLOPs}}(n) = 1 - \frac{C_{\text{ours}}(n)}{C_{\text{full}}(n)} = 1 - \frac{l_s A(n) + (L - l_s)A(\hat{n})}{L\,A(n)}, \qquad (8)$$

$$s_{\text{FLOPs}}(n) = \frac{C_{\text{full}}(n)}{C_{\text{ours}}(n)}. \qquad (9)$$

Since $A(n)$ contains both $O(n)$ and $O(n^2)$ terms, $s_{\text{FLOPs}}(n)$ depends on the input length n.

Instantiation (Qwen2.5-7B). We use $L{=}28$, $d{=}3584$, $m{=}18944$, and $l_s{=}\lfloor 0.4L \rfloor{=}11$. With $n_{\text{first}}{=}64$, $n_{\text{last}}{=}32$ ($n_{\text{fix}}{=}96$) and $c{=}0.667$, Eq. (5) gives

$$\hat{n} \;=\; (1 - 0.667)n + 0.667 \cdot 96. \qquad (10)$$

Table 3: Theoretical FLOPs reduction and speedup under single-shot halting for different text-only prefill lengths n. We follow the per-layer FLOPs proxy $A(n)$ in Eq. (4) and compute $r_{\text{FLOPs}}(n)$ and $s_{\text{FLOPs}}(n)$ using Eq. (8)–(9). We instantiate Qwen2.5-7B-Instruct-1M with L=28, l_s=$\lfloor 0.4L \rfloor$=11, and our implementation parameters `compression_ratio` = 0.667, `keep_first_n` = 64, `keep_last_n` = 32. The kept length $\hat{n}$ accounts for the fixed prefix/suffix and the rounding in `drop_target` = $\lfloor \text{round}(c \cdot (n - n_{\text{fix}})) \rfloor$, where $n_{\text{fix}} = 96$. All FLOPs and speedup numbers correspond to the prefill stage.

n	$\hat{n}$ (kept)	$r_{\text{FLOPs}}(n)$	$s_{\text{FLOPs}}(n)$
8,192	2,792	43.28%	1.76×
16,384	5,520	45.49%	1.83×
32,768	10,976	47.90%	1.92×
65,536	21,888	50.09%	2.00×
131,072	43,711	51.72%	2.07×

Length dependence. Because the attention term in $A(n)$ scales as $O(n^2)$, the FLOPs reduction and speedup are length-dependent. Table 3 reports $r_{\text{FLOPs}}(n)$ and $s_{\text{FLOPs}}(n)$ for representative text-only prefill lengths; notably, at n=16,384 the theoretical speedup is $1.83\times$, matching our typical measured setting.

B Implementation Details

B.1 Algorithm Pseudocode

For completeness, we provide the pseudocode of **DASH** used in the prefill stage. The algorithm performs a single-shot token selection at the start layer l_s based on the per-token delta-attn score (the ℓ_2 norm of the pre-residual attention output), and reuses the resulting active set for all subsequent layers under a fixed pruning ratio ρ.

Algorithm 1 DASH for prefill token halting with a fixed pruning ratio

Input: token sequence of length T (text-only or multimodal), Transformer with L blocks, start layer l_s, pruning ratio ρ
Initialize: $S_1 \leftarrow \{1, \ldots, T\}$ and $T_1 \leftarrow T$
for $l = 1$ **to** L **do**
  (Active tokens) form $\mathbf{H}^{(l)} \in \mathbb{R}^{T_l \times d}$ by gathering tokens indexed by S_l
  (Pre-residual attention output) $\mathbf{U}^{(l)} \leftarrow \text{Attn}\Big(\text{LN}(\mathbf{H}^{(l)})\Big)$
  (Block forward) $\widetilde{\mathbf{H}}^{(l)} \leftarrow \mathbf{H}^{(l)} + \mathbf{U}^{(l)}$
  $\mathbf{H}^{(l+1)} \leftarrow \widetilde{\mathbf{H}}^{(l)} + \text{FFN}\Big(\text{LN}(\widetilde{\mathbf{H}}^{(l)})\Big)$
  if $l = l_s$ **then**
    $\Delta_t^{(l_s)} \leftarrow \|\mathbf{U}_t^{(l_s)}\|_2$ for $t \in S$
    $K \leftarrow \lfloor (1-\rho)T \rfloor$
    $S^\star \leftarrow \text{TopK}(S, K, \Delta^{(l_s)})$
  end if
  if $l > l_s$ **then** use $S^\star$ as the active index set
end for

B.2 Experimental Settings

All experiments are conducted on Nvidia GPUs. Qwen2-VL-7B-Instruct is evaluated on image benchmarks using RTX A6000 (48GB). For ToMe and ToFu, we follow the original implementations with a minor modification: tokens are reduced proportionally across layers until the target sparsity is reached. Other baseline settings follow the original papers.

C More Experimental Results

C.1 Long-context Language Benchmarks: Detailed Settings and Results

Backbone and benchmarks. We evaluate long-context text inference on Qwen2.5-7B-Instruct-1M across LongBench-E and LooGLE, following the standard evaluation protocol of each benchmark. We focus on *prefill-time* token halting and report both task performance and efficiency metrics.

DASH settings. DASH is activated from a fixed start layer $l_s = 11$ for all text experiments. At layer l_s, we compute token-wise Δ_{attn} and select a compact active set by retaining a fraction $(1 - \rho)$ of tokens (with the same token reduction ratio definition used throughout the paper). The active set is reused for all subsequent layers $l > l_s$ during prefill. We do not prune generated tokens; however, halting prompt tokens shortens the KV cache and can also reduce the per-token decoding cost.

Baselines and text adaptations. We compare DASH with representative token-compression baselines. For FastV, we use a text-adapted variant: since vanilla FastV is designed for vision–language inputs and typically selects tokens based on vision-heavy attention patterns, we instead derive the retention mask from a text-only attention proxy (computed at the designated prune layer) and apply the same single-shot keep mask to all deeper prefill layers. This yields a comparable *prefill-time* token pruning policy for text without requiring access to full attention matrices under FlashAttention/SDPA.

For SnapKV, we adopt a token-pruning adaptation to ensure a fair comparison in the *prefill-compute* regime: rather than post-hoc KV compression after computing all tokens, we convert SnapKV's selection signal into a single-shot keep mask at the chosen layer and physically reduce

the sequence for subsequent layers (i.e., dropped tokens skip both attention and FFN computation in deeper prefill blocks). This adaptation isolates the effect of token dropping on prefill FLOPs and latency under the same token reduction ratio.

C.2 Vision–Language Benchmarks: Detailed Results and Efficiency Analysis

As summarized in Table 6, we compare DASH with representative advanced token compression methods on Qwen2-VL-7B across multiple general-purpose benchmarks under different compression ratios.

Comparison between Different Methods: Across a wide range of compression ratios and general-purpose benchmarks, **DASH consistently outperforms existing advanced token compression methods**, demonstrating its superior robustness under aggressive token reduction. At a moderate compression ratio of 75.00%, DASH achieves the highest average decline ratio (ADR) of 85.2, surpassing VisionZip, DART, and PruMerge+ by clear margins. Even under more aggressive settings, such as 88.89% compression, DASH maintains a strong overall performance (ADR of 74.7), while other methods exhibit noticeable degradation across multiple benchmarks. These results indicate that DASH is more effective at identifying and preserving task-relevant visual tokens, enabling it to better balance efficiency and accuracy compared to prior approaches.

Comparison between Different Compression Ratios: As the compression ratio increases, all token compression methods inevitably experience performance degradation; however, **DASH demonstrates significantly more graceful degradation**. At 93.75% compression, DASH still achieves an ADR of 66.7, substantially outperforming FastV, VisionZip, and PruMerge+, whose ADR values drop below 60. This trend highlights that DASH remains stable even when the majority of visual tokens are removed, suggesting that its compression strategy successfully retains the most informative visual content under extreme constraints. The results further suggest that naive or heuristic pruning strategies struggle to scale to ultra-high compression regimes, whereas DASH maintains robustness across all evaluated settings.

Comparison between Different Tasks: When analyzing task-level performance, we observe that the advantages of DASH are particularly pronounced on general-purpose benchmarks such as GQA, MME, and MMStar, where global visual information is often sufficient for correct reasoning. However, on tasks requiring fine-grained visual understanding—most notably OCRBench and ChartQA—the performance gap between DASH and other advanced methods narrows. This observation is highly informative: while DASH effectively preserves globally salient visual cues, tasks involving dense text, precise numerical values, or structured visual layouts place higher demands on fine-grained local information. Nonetheless, even under these challenging conditions, DASH remains competitive, indicating that its token selection mechanism captures both global and selectively important local features.

Overall, the comparisons across methods, compression ratios, and task types provide compelling evidence that **DASH achieves a more favorable accuracy–efficiency trade-off than existing token compression approaches**. The consistent performance gains of DASH stem not from aggressive preservation of visual detail, but from its ability to adaptively retain the most informative tokens for a given input. These findings also suggest that current general-purpose benchmarks predominantly reward global visual understanding, while more fine-grained tasks reveal the complementary strengths of sophisticated token compression strategies.

We evaluate inference efficiency on MMBench-EN by reporting total GPU latency and prefilling latency, together with FLOPs and KV cache usage (Table 7). Our method achieves a favorable balance between accuracy and efficiency under both pruning settings. When retaining 35% tokens in both the ViT and LLM stages, our latency drops to 60.8% of the vanilla baseline (prefill latency 54.8%), while maintaining competitive accuracy (77.5) compared with representative LLM-stage methods such as FastV, V^2Drop, and DART. Under the more aggressive setting (ViT retain 20% / LLM retain 20%), our approach further reduces latency to 51.7% (prefill 52.1%), with an accuracy of 73.9, showing a substantially improved speed–accuracy trade-off over ViT-stage pruning baselines (e.g., DART(ViT), ToMe, and ToFu) which achieve lower accuracy despite comparable or higher latency.

We also observe that prefilling latency generally decreases in tandem with total latency, indicating that the proposed pruning strategy effectively reduces the dominant computation in the prefilling stage. While ViT-stage methods tend

to achieve larger reductions in FLOPs, they often incur sharper accuracy degradation, whereas LLM-stage methods provide limited additional speedup once the runtime approaches a plateau. Moreover, our method retains more LLM tokens than pure ViT-stage pruning approaches and thus exhibits a relatively larger KV cache footprint; nevertheless, this overhead remains moderate and practical for deployment. Finally, the results suggest that FLOPs alone do not reliably predict end-to-end GPU latency, as methods with similar FLOPs can lead to noticeably different inference times due to kernel efficiency and memory behavior.

Table 4: LongBench-E quality (Score) and efficiency relative to Qwen2.5-7B-Instruct-1M, including prefill, decoding, end-to-end latency, throughput, and peak GPU memory. **Bold** denotes the best compressed-model result, and underline denotes the second best (excluding the original model).

	Score				Efficiency vs. Full (↑)				
Method	**D1**	**D2**	**D3**	**Avg**	**Prefill**	**Gen.**	**Total**	**Thrpt.**	**Mem.**
Single-Document QA *(Datasets: qasper_e, multifieldqa_en_e)*									
Qwen2.5-7B-Instruct-1M	44.19	51.13	–	47.66	1.00×	1.00×	1.00×	1.00×	1.00×
+ LLMLingua2	40.41	42.91	–	41.66	1.24×	0.74×	0.77×	0.77×	0.92×
+ SnapKV	40.46	45.77	–	43.11	1.02×	1.10×	1.02×	1.01×	1.01×
+ SnapKV (pr.)	38.14	42.98	–	40.56	1.23×	1.37×	1.26×	1.26×	1.01×
+ D^3	40.18	44.49	–	42.33	1.14×	1.24×	1.12×	1.12×	1.00×
+ **DASH (ours)**	40.58	49.38	–	**44.98**	**1.72**×	**1.56**×	**1.56**×	**1.56**×	**1.02**×
Multi-Document QA *(Datasets: hotpotqa_e, 2wikimqa_e)*									
Qwen2.5-7B-Instruct-1M	62.97	51.07	–	57.02	1.00×	1.00×	1.00×	1.00×	1.00×
+ LLMLingua2	51.58	43.29	–	47.44	1.32×	0.80×	0.73×	0.73×	0.92×
+ SnapKV (pr.)	61.54	48.31	–	54.93	1.30×	1.51×	1.18×	1.18×	1.01×
+ D^3	60.95	50.16	–	**55.56**	1.17×	1.34×	1.00×	1.00×	1.00×
+ **DASH (ours)**	61.00	48.03	–	54.52	**1.74**×	**1.74**×	**1.45**×	**1.45**×	**1.02**×
Summarization *(Datasets: gov_report_e, multi_news_e)*									
Qwen2.5-7B-Instruct-1M	6.97	6.57	–	6.77	1.00×	1.00×	1.00×	1.00×	1.00×
+ LLMLingua2	6.44	6.41	–	6.42	1.47×	0.74×	0.80×	0.80×	0.96×
+ SnapKV (pr.)	7.00	6.70	–	**6.85**	1.27×	1.18×	1.11×	1.11×	1.03×
+ D^3	6.19	6.01	–	6.10	1.26×	1.10×	1.06×	1.06×	1.00×
+ **DASH (ours)**	7.01	6.47	–	6.74	**1.89**×	**1.19**×	**1.28**×	**1.28**×	**1.03**×
Few-shot Learning *(Datasets: trec_e, triviaqa_e, samsum_e)*									
Qwen2.5-7B-Instruct-1M	65.00	85.75	32.95	45.93	1.00×	1.00×	1.00×	1.00×	1.00×
+ LLMLingua2	61.67	80.87	32.53	43.77	1.47×	0.80×	0.79×	0.79×	0.97×
+ SnapKV (pr.)	63.67	85.25	30.87	**44.95**	1.21×	1.21×	1.03×	1.03×	**1.03**×
+ D^3	64.67	85.01	27.14	44.21	1.21×	1.19×	1.01×	1.01×	1.00×
+ **DASH (ours)**	59.00	84.21	31.82	43.76	**1.82**×	**1.53**×	**1.37**×	**1.37**×	1.02×
Synthetic Task *(Datasets: PassageCount_e, PassageRetrieval_en_e)*									
Qwen2.5-7B-Instruct-1M	15.00	99.33	–	57.17	1.00×	1.00×	1.00×	1.00×	1.00×
+ LLMLingua2	6.33	99.00	–	52.67	1.36×	0.85×	0.74×	0.74×	0.92×
+ SnapKV (pr.)	16.60	97.67	–	57.14	1.31×	1.55×	1.10×	1.10×	1.01×
+ D^3	15.00	99.33	–	57.17	1.16×	1.32×	0.99×	0.99×	1.00×
+ **DASH (ours)**	16.67	98.00	–	**57.34**	**1.74**×	**1.87**×	**1.41**×	**1.41**×	**1.02**×
Code Completion *(Datasets: lcc_e, RepoBench-P_e)*									
Qwen2.5-7B-Instruct-1M	59.86	54.52	–	57.19	1.00×	1.00×	1.00×	1.00×	1.00×
+ LLMLingua2	49.64	53.05	–	51.35	1.53×	0.67×	0.84×	0.84×	0.97×
+ SnapKV (pr.)	51.89	49.37	–	50.63	1.39×	1.15×	1.19×	1.19×	1.04×
+ D^3	45.54	40.34	–	42.94	1.20×	1.04×	1.09×	1.09×	1.00×
+ **DASH (ours)**	54.58	51.14	–	**52.86**	**1.86**×	**1.25**×	**1.40**×	**1.46**×	**1.04**×

Table 5: Comparison of Advanced Token Compression Methods on Qwen2-VL-7B. ADR refers to the average decline ratio, which is the average value of the decline ratio of each benchmark (computed over GQA, MME, POPE, MMStar, OCRBench, ChartQA).

Method	**GQA**	**MME**	**POPE**	**MMStar**	**OCRBench**	**ChartQA**	**ADR**
Qwen2-VL-7B	*Upper Bound. All Tokens (100%)*						
Vanilla	62.3	2306	88.4	57.1	80.7	81.6	100.0
Qwen2-VL-7B	*Token Reduction (↓ 75.00%)*						
+ FastV	57.3	2101	83.7	44.8	41.8	58.9	80.0
+ VisionZip	58.6	2058	86.8	47.1	42.5	**66.2**	83.0
+ PruMerge+	59.1	2044	**87.2**	48.1	34.2	55.8	79.5
+ DART	57.1	2056	84.8	46.9	**52.0**	53.1	81.4
+ **DASH (ours)**	**59.8**	**2109**	83.5	**55.4**	44.6	62.6	**85.2**
Qwen2-VL-7B	*Token Reduction (↓ 88.89%)*						
+ FastV	52.1	1866	77.5	40.2	26.1	33.2	65.9
+ VisionZip	53.3	1819	83.3	40.3	24.7	47.9	69.8
+ PruMerge+	**54.5**	1794	**84.1**	38.5	22.4	44.5	68.4
+ DART	52.0	1903	80.2	39.4	41.2	31.0	69.1
+ **DASH (ours)**	53.9	**1928**	78.3	**43.4**	**41.3**	**50.7**	**74.7**
Qwen2-VL-7B	*Token Reduction (↓ 93.75%)*						
+ FastV	49.2	1693	75.0	37.4	18.9	21.1	58.7
+ VisionZip	49.0	1704	**80.1**	35.3	15.7	27.5	59.7
+ PruMerge+	48.7	1668	79.2	33.4	14.3	**30.2**	58.9
+ DART	49.0	1774	78.2	33.2	**34.0**	19.8	61.4
+ **DASH (ours)**	**53.9**	**1901**	78.3	**40.4**	32.5	25.7	**66.7**

Table 6: Comparison of methods on Qwen2-VL. **Avg. Acc.** denotes the average percentage of performance relative to Vanilla.

Methods	**GQA**	**MME**	**POPE**	**SEED**	**VQA^{text}**	**OCRBench**	**Avg. Acc.**
Vanilla	61.5	2319	89.0	76.6	82.1	80.3	100%
ViT retain 50% tokens / LLM retain 50% tokens							
DART (ViT)	56.8	2113	84.2	68.7	63.6	46.9	87.2%
ToMe	58.9	2007	88.2	72.8	61.1	39.1	86.2%
ToFu	59.1	1933	**88.1**	73.2	60.6	**76.7**	92.0%
DASH(ours)	**59.1**	**2116**	81.9	**70.9**	**74.7**	60.0	**97.8%**
ViT retain 35% tokens / LLM retain 35% tokens							
DART (ViT)	55.2	1968	81.1	64.3	59.0	**41.1**	82.3%
ToMe	58.3	1842	87.4	**71.5**	55.5	30.4	82.3%
ToFu	**58.6**	1839	**87.7**	**71.7**	56.4	31.4	82.9%
DASH(ours)	55.4	**1987**	81.7	68.2	**59.3**	39.1	**94.9%**

Table 7: Inference efficiency comparison on MMBench-EN. Latency columns show relative percentage (normalized to Vanilla=100%).

Methods	**Latency↓** (%)	**Prefill Latency↓** (%)	**FLOPs↓** (TFLOPs)	**KV Cache↓** (MB)	**Accuracy↑** (MMB)
Vanilla	100.0%	100.0%	31.9	76.6	80.5
ViT retain 35% tokens / LLM retain 35% tokens					
DART (ViT)	67.1%	58.0%	11.7	29.7	70.6
ToMe	65.9%	54.2%	12.0	30.4	72.6
ToFu	67.5%	54.1%	12.3	29.6	73.9
FastV	82.1%	74.3%	19.5	32.6	80.0
V^2Drop	81.3%	74.6%	21.9	39.0	79.9
SparseVLM	106.6%	94.8%	18.8	29.8	80.4
DART	82.4%	78.6%	20.6	34.5	79.7
DASH(Ours)	70.6%	74.8%	13.1	35.6	77.5
ViT retain 20% tokens / LLM retain 20% tokens					
DART (ViT)	53.0%	53.8%	7.6	19.0	63.9
ToMe	60.9%	53.2%	8.2	19.6	66.8
ToFu	57.3%	53.0%	8.4	19.1	67.7
FastV	76.6%	65.3%	17.0	23.0	77.4
V^2Drop	74.2%	68.0%	18.8	27.6	78.1
SparseVLM	92.4%	80.7%	16.4	20.6	79.2
DART	75.0%	74.7%	18.0	25.0	77.8
DASH(Ours)	61.6%	62.5%	12.6	20.2	73.9