PARK: ACCURATE BLOCK RETRIEVAL FOR SPARSE ATTENTION IN VIDEO DIFFUSION TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Transformers (DiTs) have become a dominant architecture for video generation, but their efficiency is limited by the quadratic complexity of full attention. Sparse attention reduces this cost by retrieving important blocks and computing attention only within them, but inaccurate retrieval can either degrade generation quality or yield unnecessary computation. We identify two retrieval mismatches in methods that retrieve blocks using the averaged representations of query and key blocks: (i) query-side aggregation mismatch, where averaging queries before Softmax fails to preserve their individual attention preferences, and (ii) key-side clustering metric mismatch, where standard Euclidean clustering in the original key space can group keys with dissimilar QK scores under the current query, so their average representation may not accurately represent how the current query scores individual keys. These mismatches can lead to inaccurate block retrieval. To address these mismatches, we propose PARK, a training-free sparse attention method for accurate block retrieval. PARK retains every original query, independently normalizes its attention over key blocks, and then averages these distributions within each query block. It also uses information from the current queries to transform keys before clustering, so that keys receiving similar QK scores are grouped together. A fused GPU kernel further reduces the overhead of block retrieval. Experiments on HunyuanVideo and Wan demonstrate that PARK improves block retrieval accuracy and preserves generation quality while accelerating inference, achieving the best quality-efficiency trade-off among the compared sparse attention methods.

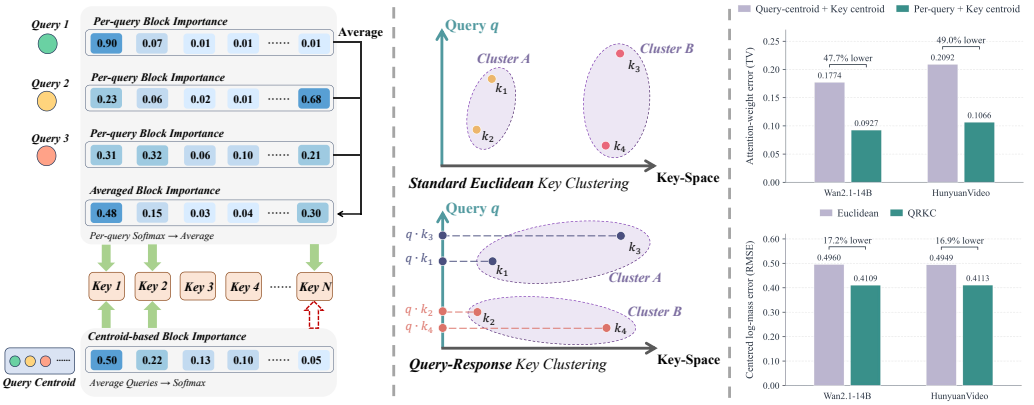
1 INTRODUCTION

Diffusion Transformers (DiTs) (Peebles & Xie (2023)) have emerged as a dominant paradigm for modern video generation, demonstrating strong scalability and enabling high-fidelity, temporally coherent synthesis (Yang et al. (2025b); Kong et al. (2024); Wan et al. (2025)). However, generating high-resolution and long videos involves a large number of spatio-temporal tokens. Since 3D full attention scales quadratically with token count, it becomes a major computational bottleneck in video generation (Xi et al. (2025); Zhang et al. (2025b); Vaswani et al. (2017)).

Fortunately, attention maps in video DiTs are inherently sparse, with only a small fraction of token interactions contributing significantly to the output (Sun et al. (2026); Zhang et al. (2026)). Sparse attention exploits this property by computing attention only over these critical token interactions, substantially reducing computation with little degradation in generation quality. In practice, query and key tokens are partitioned into blocks by grouping consecutive tokens, and attention is computed only for a selected subset of blocks. These blocks are retrieved by estimating their importance and selecting those with the highest scores (Guo et al. (2024); Li et al. (2025c)). This process, referred to as block retrieval, produces a *sparse mask* that specifies which blocks are computed.

Existing training-free sparse attention methods broadly fall into static and dynamic methods. Static methods (Li et al. (2025a); Xi et al. (2025); Li et al. (2025b); Zhang et al. (2025c)) derive sparse masks from offline attention profiling or predefined rules and apply them to individual layers or attention heads during inference. Dynamic methods (Xia et al. (2025); Zhang et al. (2025a); Xu et al. (2025)) perform block retrieval at runtime using lightweight block importance estimation, adapting the sparse masks to the current attention pattern. Additionally, recent token-reordering methods

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.



(a) Query-side Aggregation Mismatch (b) Key-side Clustering Metric Mismatch (c) Two-sided Approximation Error

Figure 1: Two-sided retrieval mismatches and approximation errors in centroid-based block retrieval. (a) Averaging queries into a single centroid before Softmax normalization ignores their individual key preferences, leading to inaccurate block retrieval. (b) Standard Euclidean clustering considers only distances between keys in feature space and does not ensure that keys within the same cluster receive similar scores from the current query, leading to inaccurate key-centroid estimates. (c) Retaining the original queries instead of using query centroids (top) and clustering keys by QK-score similarity (bottom) both reduce approximation errors. More experimental details are provided in Appendix A.

(Yang et al. (2025a); Zhao et al. (2025); Tan et al. (2026a); Luo et al. (2026)) cluster semantically similar tokens and permute them into contiguous blocks. This helps retain more critical interactions with fewer blocks, improving the efficiency-quality trade-off.

Even with token reordering, the effectiveness of sparse attention depends on how accurately it retrieves important blocks. Missing important blocks can degrade generation quality, while selecting unimportant blocks wastes computation. To compensate for inaccurate retrieval, more blocks may need to be retained to preserve generation quality, limiting the achievable speedup. Accurate block retrieval is therefore essential to improving the efficiency-quality trade-off of sparse attention.

To keep block retrieval efficient, many existing methods estimate block importance from the average-pooled representations (or *centroids*) of query and key blocks, Our theoretical analysis of the resulting estimation errors reveals two retrieval mismatches in this common centroid-based design:

- **M1: Query-side aggregation mismatch.** Each query independently yields an attention distribution over all key blocks, and these query-specific preferences jointly determine which blocks should be selected for the sparse mask. As analyzed in Sec. 4.1, averaging the original queries into a single query centroid before Softmax normalization ignores these individual preferences and can misestimate the attention assigned to each key block, leading to an inaccurate sparse mask. Figure 1(c) shows that retaining the original queries instead of using query centroids reduces attention-weight approximation error by 47.7–49.0%.
- **M2: Key-side clustering metric mismatch.** Existing reordering methods cluster key tokens using standard Euclidean clustering in the original key space and use the key block centroids for importance estimation. As analyzed in Sec. 4.2, centroid-based estimation is more accurate when keys in the same block receive similar dot-product scores from the current query. However, this clustering metric compares only the keys themselves and does not account for how the current query scores them. Consequently, standard Euclidean clustering may group keys that are scored differently by the current query, making key centroids less accurate for block importance estimation. Figure 1(c) shows that clustering keys by QK-score similarity reduces centered log-mass error by 16.9–17.2% compared with standard Euclidean clustering.

To address these mismatches, we propose **PARK**, a training-free sparse attention method for accurate block retrieval via **P**er-query **A**ttention-Mass Aggregation (PAMA) and Query-**R**esponse **K**ey Clustering (QRKC). First, PAMA retains original queries, computes a separate Softmax distribution over the key blocks for each query, and aggregates these distributions by averaging them within each

query block to estimate block importance for sparse mask selection. This preserves query-specific attention preferences and eliminates the aggregation mismatch introduced by query centroids. We fuse per-query normalization and averaging into a custom kernel for efficient execution. Second, QRKC uses a linear transformation derived from the current queries to map the keys into a query-conditioned space for clustering, where Euclidean distance reflects how similarly those queries score them. By grouping keys that are scored similarly by the current queries, QRKC reduces the approximation error introduced by key centroids. Together, PAMA and QRKC improve block retrieval with negligible overhead. Our contributions are summarized as follows:

- We identify two retrieval mismatches in existing centroid-based block retrieval methods: query-side aggregation mismatch and key-side clustering metric mismatch, and analyze how they introduce errors in block-importance estimation.
- We propose PARK, a novel training-free sparse attention method for fast video generation. It addresses the two identified mismatches through Per-query Attention-Mass Aggregation and Query-Response Key Clustering.
- Extensive experiments on HunyuanVideo and Wan demonstrate that PARK consistently improves block retrieval accuracy and preserves generation quality with high end-to-end speedup, achieving the best quality-efficiency trade-off among the compared training-free sparse attention methods.

2 Related Work

Training-based Sparse Attention. Training-based sparse attention methods fine-tune video diffusion models with task-specific sparse attention patterns. VSA (Zhang et al. (2026)) employs a coarse-to-fine routing mechanism to select important attention blocks, while VMoBA (Wu et al. (2026)) cycles through 1D, 2D, and 3D block partitions across layers and retrieves blocks using a global similarity threshold. Other methods, such as DSV (Tan et al. (2026b)), BSA (Zhan et al. (2025)), and MoGA (Jia et al. (2026)), train lightweight predictors, routers, or cluster assignments to estimate sparse attention patterns before computation. However, they generally require additional training or fine-tuning and may introduce auxiliary prediction modules.

Training-free Sparse Attention. Training-free methods directly exploit attention sparsity in pre-trained video DiTs without additional optimization. Static methods (Xi et al. (2025); Zhang et al. (2025c); Li et al. (2025a;b)) derive sparse masks from offline profiling or predefined spatio-temporal structures. Dynamic methods (Jiang et al. (2024); Zhang et al. (2025a); Xu et al. (2025); Xia et al. (2025)) instead construct sparse masks at runtime by estimating the importance of attention blocks. Recent methods further improve sparse attention through token reordering. PAROAttention (Zhao et al. (2025)) performs pattern-aware token reordering for sparse and quantized attention. SVG2 (Yang et al. (2025a)) and AdaCluster (Tan et al. (2026a)) apply K-means clustering to query and key tokens and reorder semantically similar tokens into contiguous blocks, while SVOO (Luo et al. (2026)) incorporates query–key interactions through bidirectional co-clustering.

3 Preliminary

3.1 Attention and Sparse Attention

For a single attention head, the input tokens are projected into query, key, and value matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L\times d}$, where L is the sequence length and d is the head dimension. Full attention (Vaswani et al. (2017); Dao et al. (2022)) is computed as

$$\mathbf{Z} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}, \qquad \mathbf{A} = \mathrm{Softmax}(\mathbf{Z}), \qquad \mathbf{O} = \mathbf{A}\mathbf{V}, \tag{1}$$

where the Softmax is applied along the key dimension independently for each query. Computing the full attention matrix incurs quadratic complexity with respect to the sequence length.

Given a sparse mask $\mathbf{M} \in \{0,1\}^{L\times L}$, sparse attention is computed as

$$\mathbf{O}^{\mathrm{sp}} = (\mathbf{A} \odot \mathbf{M})\mathbf{V}, \tag{2}$$

where $\odot$ denotes element-wise masking, and $M_{ij} = 0$ indicates that the interaction from query i to key j is skipped.

In practice, query and key tokens are partitioned into query blocks $\{\mathcal{Q}_u\}_{u=1}^{N_Q}$ and key blocks $\{\mathcal{K}_v\}_{v=1}^{N_K}$, respectively. Each query–key block consists of all attention interactions between the queries in $\mathcal{Q}_u$ and the keys in $\mathcal{K}_v$, corresponding to a submatrix of $\mathbf{A}$. The sparse mask selects or skips each query–key block as a whole. The block sizes are denoted by $n_u^Q = |\mathcal{Q}_u|$ and $n_v^K = |\mathcal{K}_v|$, with $\sum_{u=1}^{N_Q} n_u^Q = \sum_{v=1}^{N_K} n_v^K = L$.

3.2 Block Importance and Attention Recall

For a query block $\mathcal{Q}_u$ and a key block $\mathcal{K}_v$, we define their exact block importance W_{uv} as the average attention weight that the queries in $\mathcal{Q}_u$ assign to $\mathcal{K}_v$:

$$W_{uv} = \frac{1}{n_u^Q} \sum_{i \in \mathcal{Q}_u} \sum_{j \in \mathcal{K}_v} A_{ij}. \tag{3}$$

Here, A_{ij} denotes the dense post-Softmax attention weight from query token i to key token j. For every query block, the block importance is normalized over all key blocks, i.e., $\sum_{v=1}^{N_K} W_{uv} = 1$. Under Top-$p$ retrieval, key blocks are retrieved for each query block until their cumulative estimated importance reaches the threshold p. In practice, key blocks are retrieved in descending order of $\widehat{W}_{uv}$, which estimates the exact block importance W_{uv}. The retained query–key block pairs form the block-level sparse mask $\mathbf{M}$.

The block retrieval accuracy of sparse attention is measured using attention recall, defined as

$$\mathrm{Recall}(\mathbf{A}, \mathbf{M}) = \frac{\sum_{i=1}^{L} \sum_{j=1}^{L} A_{ij} M_{ij}}{\sum_{i=1}^{L} \sum_{j=1}^{L} A_{ij}}. \tag{4}$$

Attention recall (Treviso et al. (2022)) measures the proportion of full-attention weights covered by the block-level sparse mask. A higher recall indicates more accurate block retrieval.

4 Motivation

Centroid-based block retrieval compresses both query and key blocks before estimating their importance. Although this design is efficient, the two compressions introduce fundamentally different approximations. We analyze them relative to the exact block importance in Eq. (3) and show how they lead to the two retrieval mismatches.

4.1 Query-Side Aggregation Mismatch

Averaging queries into a centroid before Softmax can lead to inaccurate block-importance estimates. To analyze this error, we retain all original keys. The exact block importance W_{uv} in Eq. (3) is computed by first normalizing attention for each query and then averaging the attention weights of queries in the same block. Query-centroid estimation instead replaces these queries with their centroid before Softmax normalization. A Taylor expansion expresses the approximation error as

$$\Delta_{uv}^Q = W_{uv} - \widehat{W}_{uv}^{\mathrm{centroid}} = \frac{1}{2} \left\langle \mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u), \mathbf{\Sigma}_{Q,u} \right\rangle_F + \mathcal{R}_{uv}^Q. \tag{5}$$

Here, $\widehat{W}_{uv}^{\mathrm{centroid}} = \phi_v(\overline{\mathbf{q}}_u)$ is the query-centroid estimate, where $\phi_v(\mathbf{q})$ is the exact attention weight that query $\mathbf{q}$ assigns to key block $\mathcal{K}_v$, and $\overline{\mathbf{q}}_u$ is the query-block centroid. $\mathbf{\Sigma}_{Q,u}$ is the within-block query covariance, and $\mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u)$ is the Hessian of ϕ_v, capturing how ϕ_v curves as the query varies near its centroid. $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product, and $\mathcal{R}_{uv}^Q$ is the higher-order remainder. The full derivation is provided in Appendix D.1.

Accurate query-centroid estimation requires a small $|\Delta_{uv}^Q|$, whose leading error is determined by $\langle \mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u), \mathbf{\Sigma}_{Q,u} \rangle_F$. A sufficient condition for a small leading error is that the product of the norm of $\mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u)$ and the total within-block query variance is small. Standard Euclidean query clustering

Table 1: Top-p block-retrieval recall ($p = 0.9$) under different query/key representations. Mean, P05, and Worst denote the mean, fifth percentile, and minimum attention recall across all attention heads and layers, respectively. All values are percentages. Higher is better.

Ordering	Query	Key	HunyuanVideo (Kong et al. (2024))			Wan 2.1-14B (Wan et al. (2025))		
			Mean	P05	Worst	Mean	P05	Worst
Clustered	Full	Full	91.29	90.02	90.01	91.71	90.02	90.01
	Centroid	Centroid	88.06	81.53	64.21	90.43	87.11	73.89
	Centroid	Full	89.10	84.44	68.40	91.00	88.64	79.06
	Full	Centroid	**90.70**	**87.62**	**81.64**	**91.56**	**89.36**	**83.61**
Original	Full	Full	90.64	90.02	90.01	90.89	90.03	90.02
	Centroid	Centroid	85.59	72.40	22.94	90.01	83.96	69.11
	Centroid	Full	86.18	56.17	15.02	89.24	74.92	39.94
	Full	Centroid	**87.80**	**76.35**	**24.30**	**91.32**	**87.02**	**73.55**

reduces the total within-block query variance, captured by $\mathbf{\Sigma}_{Q,u}$, but does not generally eliminate it. Since query clustering does not account for the Hessian $\mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u)$, it does not by itself ensure that this product is sufficiently small. Empirically, Table 4 shows that query-centroid estimation with full keys incurs even higher approximation error than retaining original queries with key centroids.

We further evaluate whether retaining the original queries remains beneficial when keys are represented by centroids. Figure 1(c) shows that retaining original queries reduces approximation error when both estimates use the same key centroids. Table 1 further shows that retaining full queries with key centroids achieves the highest recall among the compressed representations across both ordering schemes and models. These empirical results support preserving query-specific attention preferences in practical block retrieval and motivate Per-Query Attention-Mass Aggregation (§5.1).

4.2 Key-Side Clustering Metric Mismatch

With the original queries retained, the exact attention weight $\phi_v(\mathbf{q}_i)$ is obtained by applying Softmax to the block log-masses: $\phi_v(\mathbf{q}_i) = \exp(S^\star_{iv}) / \sum_{r=1}^{N_K} \exp(S^\star_{ir})$, where $S^\star_{iv}$ is the exact log-mass of key block $\mathcal{K}_v$ for query $\mathbf{q}_i$. The exact block log-mass and its key-centroid estimate are

$$S^\star_{iv} = \log \sum_{j \in \mathcal{K}_v} \exp\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}}\right), \qquad \widehat{S}_{iv} = \log n_v^K + \frac{\mathbf{q}_i^\top \mathbf{c}_v}{\sqrt{d}}. \tag{6}$$

Here, $\mathbf{c}_v = (1/n_v^K) \sum_{j \in \mathcal{K}_v} \mathbf{k}_j$ is the key-block centroid, and n_v^K is the number of keys in $\mathcal{K}_v$. The term $\log n_v^K$ accounts for key-block size. An expansion expresses the key-centroid approximation error as

$$\Delta^K_{iv} = S^\star_{iv} - \widehat{S}_{iv} = \frac{1}{2d} \mathbf{q}_i^\top \mathbf{\Sigma}_{K,v} \mathbf{q}_i + \mathcal{R}^K_{iv}. \tag{7}$$

Here, $\mathbf{\Sigma}_{K,v}$ is the within-block key covariance, $\epsilon^K_{iv} = \frac{1}{2d} \mathbf{q}_i^\top \mathbf{\Sigma}_{K,v} \mathbf{q}_i$ is the leading term of Δ^K_{iv}, and $\mathcal{R}^K_{iv}$ is the higher-order remainder. The full derivation is provided in Appendix D.2.

Accurate key-centroid log-mass estimation requires a small $|\Delta^K_{iv}|$. The leading term ϵ^K_{iv} is proportional to the within-block key variance after projection onto the current query, which is the variance of the unscaled QK scores within that block. Standard euclidean clustering aims to reduce within-block key variation, captured by $\mathbf{\Sigma}_{K,v}$, but does not account for the current query $\mathbf{q}_i$ and therefore does not directly minimize ϵ^K_{iv}. Additional empirical evidence for this metric mismatch is provided in Appendix B. Therefore, key clustering should account for QK-score similarity under the current query, motivating Query-Response Key Clustering (§5.2).

5 Methodology

In this section, we introduce PARK, a training-free sparse attention framework designed for accurate block retrieval via Per-Query Attention-Mass Aggregation (PAMA) and Query-Response Key

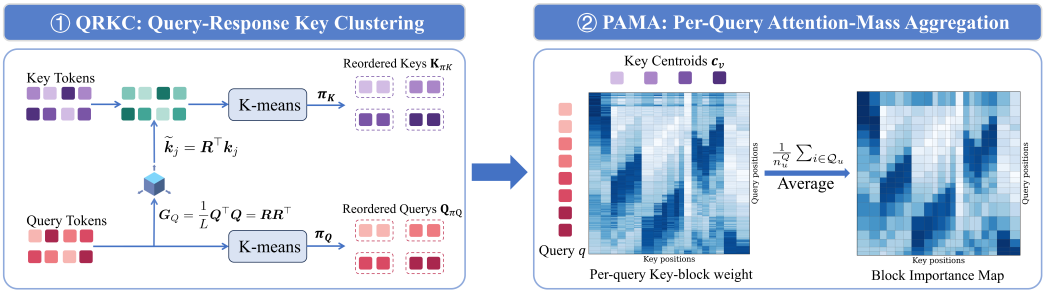


Figure 2: Overview of PARK. QRKC transforms keys using a metric derived from the current queries, so that keys receiving similar QK scores are grouped together, while queries are clustered using standard Euclidean K-means. PAMA retains every original query, computes its Softmax-normalized weights over the key centroids, and averages them within each query block to produce the block-importance map.

Clustering (QRKC). PAMA preserves all original queries, computes their normalized attention over key blocks separately, and then averages these attention weights within each query block for block-importance estimation (§5.1). QRKC uses information from the current queries to transform the keys before clustering, so that keys receiving similar QK scores are grouped together (§5.2). These two modules improve block retrieval on both the query and key sides. An overview of our proposed PARK framework is shown in Figure 2.

5.1 PAMA: Per-Query Attention-Mass Aggregation

As shown in Eq. (3), exact block importance is computed by normalizing attention independently for every query and then averaging the attention weights within each query block. Following this order, PAMA retains all original queries during block-importance estimation. For each key block $\mathcal{K}_v$, we compute its centroid $\mathbf{c}_v$ in the original key space and approximate its log-mass $\widehat{S}_{iv}$ for query $\mathbf{q}_i$ as

$$\mathbf{c}_v = \frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} \mathbf{k}_j, \qquad \widehat{S}_{iv} = \frac{\mathbf{q}_i^\top \mathbf{c}_v}{\sqrt{d}} + \log n_v^K. \tag{8}$$

Here, $\mathcal{K}_v$ denotes a key block containing n_v^K keys. The block-size term $\log n_v^K$ accounts for the different numbers of keys represented by the block centroids.

PAMA then normalizes the estimated scores over all key blocks for each query and averages the block weights within query block $\mathcal{Q}_u$:

$$\widehat{P}_{iv} = \frac{\exp(\widehat{S}_{iv})}{\sum_{r=1}^{N_K} \exp(\widehat{S}_{ir})}, \qquad \widehat{W}_{uv}^{\mathrm{PAMA}} = \frac{1}{n_u^Q} \sum_{i \in \mathcal{Q}_u} \widehat{P}_{iv}. \tag{9}$$

Here, N_K is the number of key blocks, and $\mathcal{Q}_u$ is a query block containing n_u^Q queries. $\widehat{P}_{iv}$ is the estimated attention weight that query $\mathbf{q}_i$ assigns to key block $\mathcal{K}_v$, and $\widehat{W}_{uv}^{\mathrm{PAMA}}$ is the importance of the query–key block.

Efficient Kernel Implementation. For each attention head, PAMA requires $\mathcal{O}(LN_K d)$ operations for QK-centroid scores and $\mathcal{O}(LN_K)$ for row-wise Softmax and averaging within query blocks, far below full attention's $\mathcal{O}(L^2 d)$ cost since $N_K \ll L$. A straightforward implementation stores $L \times N_K$ logits and probabilities in HBM, incurring $\mathcal{O}(LN_K)$ intermediate storage and additional memory access. To reduce this memory overhead, we fuse the QK-centroid matrix multiplication, row-wise Softmax, and averaging within query blocks into a single GPU kernel. The intermediate logits and probabilities remain in on-chip registers, and only the final $N_Q \times N_K$ block-importance matrix is written to HBM. The pseudocode and overhead analysis are provided in Appendix E.

5.2 QRKC: Query-Response Key Clustering

QRKC aims to reduce the leading key-centroid approximation error ϵ_{iv}^K identified in Sec. 4.2. To construct a key-clustering objective that accounts for all current queries, we average the squared

QK-score error caused by replacing keys with their centroids over these queries:

$$\mathcal{J}_{\mathrm{QRKC}} = \frac{1}{L}\sum_{i=1}^{L}\sum_{v=1}^{N_K}\sum_{j\in\mathcal{K}_v}\left[\mathbf{q}_i^\top(\mathbf{k}_j-\mathbf{c}_v)\right]^2 = \sum_{v=1}^{N_K}\sum_{j\in\mathcal{K}_v}\|\mathbf{k}_j-\mathbf{c}_v\|^2_{\mathbf{G}_Q}. \tag{10}$$

Here, $\mathbf{G}_Q = (1/L)\mathbf{Q}^\top\mathbf{Q} \in \mathbb{R}^{d\times d}$ is the uncentered second-moment matrix of the current queries, and $\|\mathbf{x}\|^2_{\mathbf{G}_Q} = \mathbf{x}^\top\mathbf{G}_Q\mathbf{x}$. The relation $\mathcal{J}_{\mathrm{QRKC}} = 2d\sum_v n_v^K(1/L)\sum_i \epsilon_{iv}^K$ connects this clustering objective $\mathcal{J}_{\mathrm{QRKC}}$ directly to the leading key-centroid approximation error ϵ_{iv}^K.

To minimize $\mathcal{J}_{\mathrm{QRKC}}$ efficiently, we reuse a standard Euclidean K-means implementation. However, standard K-means uses the squared Euclidean distance $\|\mathbf{k}_j-\mathbf{c}_v\|_2^2$, whereas our objective in Eq. (10) uses $\|\mathbf{k}_j-\mathbf{c}_v\|^2_{\mathbf{G}_Q} = (\mathbf{k}_j-\mathbf{c}_v)^\top\mathbf{G}_Q(\mathbf{k}_j-\mathbf{c}_v)$. We therefore transform the keys into a space where the squared Euclidean distance between each key and its centroid satisfies the following equality:

$$\|\widetilde{\mathbf{k}}_j-\widetilde{\mathbf{c}}_v\|_2^2 = (\mathbf{k}_j-\mathbf{c}_v)^\top\mathbf{G}_Q(\mathbf{k}_j-\mathbf{c}_v). \tag{11}$$

Here, $\widetilde{\mathbf{k}}_j$ and $\widetilde{\mathbf{c}}_v$ denote a transformed key and its cluster centroid, respectively. The sum of these squared distances over all keys equals $\mathcal{J}_{\mathrm{QRKC}}$ in Eq. (10). Thus, standard Euclidean K-means in the transformed space directly minimizes this objective $\mathcal{J}_{\mathrm{QRKC}}$.

Directly mapping each key to $\mathbf{Q}\mathbf{k}_j/\sqrt{L}$ also realizes this distance in Eq. (11), but increases the feature dimension from d to L. To construct a transformation that avoids this increase in feature dimension while satisfying Eq. (11), we apply Cholesky decomposition and transform each key as

$$\mathbf{G}_Q = \mathbf{R}\mathbf{R}^\top, \qquad \widetilde{\mathbf{k}}_j = \mathbf{R}^\top\mathbf{k}_j \in \mathbb{R}^d. \tag{12}$$

Here, $\mathbf{R}$ is a Cholesky factor when $\mathbf{G}_Q$ is positive definite. Since the transformation is linear, $\widetilde{\mathbf{c}}_v = \mathbf{R}^\top\mathbf{c}_v$ is the centroid of the transformed keys in block v. This transformation satisfies the distance equality in Eq. (11). For numerical stability, we use a regularized metric, which yields the corresponding regularized clustering objective. The derivation and stabilization details are provided in Appendix D.2, and the QRKC pseudocode is provided in Appendix F.

Clustering and Reuse Strategy. We apply standard Euclidean K-means to cluster and reorder the queries, while using QRKC to cluster and reorder the keys. Prior studies (Hu et al. (2026); Luo et al. (2026); Xia et al. (2025)) have shown that sparse masks and clustering results change little across adjacent diffusion steps. To reduce online clustering and block-retrieval overhead, we reuse the sparse masks and clustering results across diffusion steps and recompute them every R steps.

Difference from SVG2 and SVOO. SVG2 (Yang et al. (2025a)) independently clusters queries and keys using Euclidean K-means, while SVOO (Luo et al. (2026)) couples the two partitions through iterative bidirectional co-clustering. Both methods then estimate block importance using query and key centroids. In contrast, PAMA retains every original query, normalizes its attention over key blocks separately, and then averages these weights within each query block to estimate block importance. QRKC derives a query-dependent metric from all current queries and transforms the keys before clustering, so that Euclidean K-means minimizes the average squared QK-score error introduced by key centroids. PARK therefore adopts an asymmetric retrieval design that preserves query-specific preferences while clustering keys using a theoretically derived metric.

6 Experiments

6.1 Experimental Setup

Models. We evaluate PARK on four open-source DiT models, including Wan2.1-T2V-1.3B, Wan2.1-T2V-14B, Wan2.2-T2V-14B (Wan et al. (2025)), and HunyuanVideo-T2V-13B (Kong et al. (2024)). All models generate videos at 720p resolution. In latent space, the Wan2.1 models process 21 frames, whereas HunyuanVideo processes 32 frames, with 3,600 tokens per frame in all cases.

Metrics. We use PSNR (OpenCV Contributors (2025)), SSIM (Wang et al. (2004)), and LPIPS (Zhang et al. (2018)) to measure the similarity between videos generated with sparse and dense

Table 2: Quality and efficiency benchmark results for PARK and the baselines.

Model	Method	Quality							Efficiency	
		PSNR↑	SSIM↑	LPIPS↓	SubCons.↑	BackCons.↑	ImageQual.↑	AesQual.↑	Latency↓	Speedup↑
Wan2.2-14B	Dense	–	–	–	97.13%	97.22%	73.78%	65.56%	3792s	1.00×
	SpargeAttn	18.32	0.714	0.295	97.10%	97.33%	73.73%	65.55%	2614s	1.45×
	XAttention	17.31	0.640	0.353	96.89%	97.10%	**73.87%**	**66.07%**	2819s	1.34×
	SVG	17.33	0.663	0.342	96.99%	96.96%	73.73%	65.85%	2668s	1.42×
	SVG2	18.77	0.730	0.288	96.88%	97.11%	73.71%	65.20%	2672s	1.42×
	SVOO	19.08	0.753	0.267	**97.15%**	**97.34%**	73.76%	65.31%	2722s	1.39×
	PARK	**19.19**	**0.754**	**0.265**	97.11%	97.23%	73.73%	65.29%	**2452s**	**1.55×**
Wan2.1-1.3B	Dense	–	–	–	97.36%	96.99%	70.59%	61.23%	745s	1.00×
	SpargeAttn	25.15	0.857	0.231	97.28%	96.95%	71.44%	**61.58%**	478s	1.57×
	XAttention	23.43	0.786	0.296	96.68%	96.36%	68.71%	60.82%	495s	1.51×
	SVG	22.39	0.754	0.336	96.73%	96.47%	**72.16%**	60.99%	506s	1.47×
	SVG2	25.98	0.867	0.230	96.45%	96.03%	67.78%	58.90%	435s	1.71×
	SVOO	27.36	0.901	0.187	**97.32%**	**97.07%**	70.94%	61.25%	472s	1.58×
	PARK	**27.52**	**0.908**	**0.177**	97.22%	96.87%	70.48%	61.02%	**423s**	**1.76×**
Wan2.1-14B	Dense	–	–	–	97.95%	97.83%	72.46%	61.83%	3595s	1.00×
	SpargeAttn	20.55	0.757	0.295	94.88%	95.49%	**72.80%**	61.73%	2732s	1.32×
	XAttention	21.67	0.794	0.266	97.82%	97.73%	71.94%	62.13%	2710s	1.33×
	SVG	20.87	0.785	0.277	97.74%	**97.90%**	71.99%	**63.18%**	2747s	1.31×
	SVG2	22.98	0.848	0.219	97.82%	97.51%	72.24%	61.61%	2490s	1.44×
	SVOO	23.66	0.871	0.188	97.89%	97.76%	72.42%	61.63%	2754s	1.31×
	PARK	**24.01**	**0.875**	**0.186**	**97.91%**	97.74%	72.28%	62.03%	**2354s**	**1.53×**
HunyuanVideo	Dense	–	–	–	96.95%	96.98%	69.69%	55.58%	3058s	1.00×
	SpargeAttn	29.32	0.923	0.187	96.83%	96.44%	**69.90%**	**55.44%**	2064s	1.48×
	XAttention	29.08	0.909	0.198	96.80%	96.28%	69.44%	55.27%	2243s	1.36×
	SVG	27.88	0.898	0.221	96.71%	96.38%	68.96%	55.32%	2066s	1.48×
	SVG2	29.27	0.914	0.213	96.60%	96.05%	66.50%	54.05%	1978s	1.55×
	SVOO	30.31	0.931	0.172	**96.88%**	96.39%	69.19%	55.05%	1975s	1.55×
	PARK	**30.69**	**0.936**	**0.166**	96.87%	**96.50%**	69.41%	55.07%	**1837s**	**1.66×**

attention. For video quality evaluation, we report Subject Consistency, Background Consistency, Image Quality, and Aesthetic Quality from VBench (Huang et al. (2024)). To assess efficiency, we report the end-to-end latency and the overall speedup.

Datasets. For all the text-to-video generation experiments, we use the original prompts from VBench (Huang et al. (2024)) for both HunyuanVideo and Wan.

Baselines. We compare PARK with state-of-the-art training-free sparse attention methods, including SpargeAttention (Zhang et al. (2025a)), XAttention (Xu et al. (2025)), SVG (Xi et al. (2025)), SVG2 (Yang et al. (2025a)), and SVOO (Luo et al. (2026)). All baselines are evaluated under the same hardware configuration and input settings as PARK.

Implementations. We set the number of query blocks to $N_Q = 128$ for all models, and the number of key blocks to $N_K = 1{,}024$ for HunyuanVideo and $N_K = 512$ for Wan. All models generate videos with 50 denoising steps. We implement the fused PAMA kernel with Triton (Tillet et al. (2019)) and use a FlashInfer (Ye et al. (2025)) kernel with dynamic block sizes for sparse attention computation. All models use dense attention during the first 20% of the diffusion steps and a fixed Top-p threshold of $p = 0.9$. We recompute sparse masks and clustering results every $R = 10$ diffusion steps. All experiments are conducted on a single NVIDIA A800 GPU.

6.2 Quality and Efficiency Evaluation

As shown in Table 2, PARK consistently achieves the best similarity to dense attention across all four models, attaining the highest PSNR and SSIM and the lowest LPIPS among the compared sparse attention methods. It also remains close to dense attention on the VBench metrics, showing that PARK preserves high generation quality. Compared with SVG2, PARK improves Image Quality by 2.91 and 2.70 percentage points on HunyuanVideo and Wan2.1-1.3B, respectively, with smaller improvements on Wan2.1-14B and Wan2.2-14B. As shown in Figure 3, PARK produces videos that more closely match dense attention than SVG2, with better-preserved fine-grained visual details.

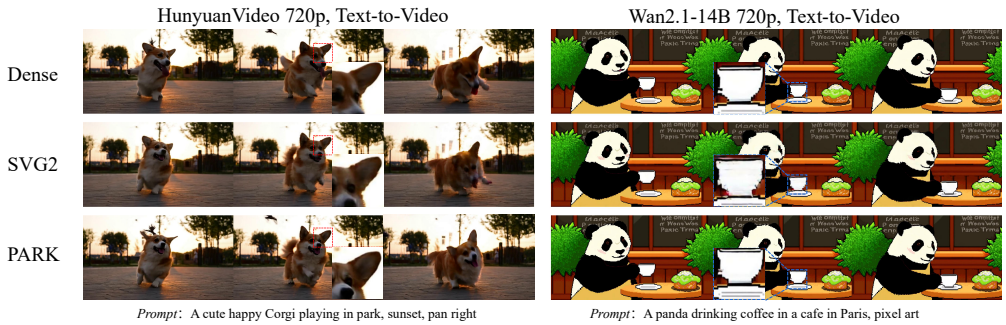


Figure 3: Examples of videos generated by PARK and SVG2 on HunyuanVideo and Wan2.1-14B.

While preserving this level of generation quality, PARK delivers $1.55\times$, $1.76\times$, $1.53\times$, and $1.66\times$ end-to-end speedups over dense attention on Wan2.2-14B, Wan2.1-1.3B, Wan2.1-14B, and HunyuanVideo, respectively, achieving the highest speedup among the compared methods on all four models. Overall, PARK achieves the best quality-efficiency trade-off among the compared sparse-attention methods.

Table 3: Ablation study of PAMA and QRKC on HunyuanVideo and Wan2.1-14B.

Model	Variant	PSNR ↑	SSIM ↑	LPIPS ↓	Latency ↓
HunyuanVideo	PARK	**29.85**	**0.931**	**0.168**	1827s
	w/o PAMA	28.40	0.911	0.205	1824s
	w/o QRKC	29.54	0.928	0.173	1826s
Wan2.1-14B	PARK	**23.90**	**0.871**	**0.189**	2359s
	w/o PAMA	23.48	0.864	0.199	2353s
	w/o QRKC	23.80	0.869	0.190	2354s

6.3 Ablation Study

We evaluate the contributions of PAMA and QRKC on a fixed subset of VBench prompts by removing each module individually, while keeping the model configurations and inference settings identical to those used in the main evaluation. Removing PAMA replaces per-query estimation with query-centroid estimation, while removing QRKC replaces query-response key clustering with standard Euclidean K-means.

As shown in Table 3, removing PAMA consistently degrades PSNR, SSIM, and LPIPS, especially on HunyuanVideo, confirming the importance of preserving query-specific attention preferences during block retrieval. Removing QRKC also reduces similarity to dense attention on both models, demonstrating the benefit of grouping keys according to their QK-score similarity under the current queries. Meanwhile, the latency differences among the three variants remain within 0.3%, indicating that PAMA and QRKC improve generation quality from the query and key sides, respectively, with negligible overhead.

7 Conclusion

In this paper, we proposed PARK, a training-free sparse attention method designed for fast diffusion-based video generation. We identified query-side aggregation mismatch and key-side clustering metric mismatch in centroid-based block retrieval. To address them, PAMA preserves query-specific attention preferences, while QRKC clusters keys using a query-dependent metric. Experiments across four models show that PARK preserves high generation quality while achieving $1.53\times$–$1.76\times$ end-to-end speedups over dense attention.

AI USE STATEMENT

In this work, we did not use generative AI tools for any tasks with required disclosure. We have not used generative AI tools for other tasks with required disclosure, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to aid and polish writing, including improving grammar, clarity, and readability. We have reviewed all AI-assisted work. All AI-assisted edits were manually reviewed by the authors to ensure that they did not alter the technical meaning, scientific claims, or conclusions of the paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

PARK improves the computational efficiency of video generation. Like other video-generation technologies, it could be misused to produce misleading or harmful content. We recommend retaining the underlying models' existing safety safeguards and clearly labeling AI-generated videos.

REPRODUCIBILITY STATEMENT

Detailed experimental settings and implementation parameters are provided in Section 6.1. We also provide our code in the supplementary materials to facilitate reproduction of our results.

REFERENCES

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Jialin Guo, Haotian Tang, Shang Yang, Zhao Zhang, Zirui Liu, and Song Han. Block sparse attention. `https://github.com/mit-han-lab/Block-Sparse-Attention`, 2024.

Jie Hu, Zixiang Gao, Yutong He, and Kun Yuan. Dfsattn: Dynamic fine-grained sparse attention for efficient video generation. *arXiv preprint arXiv:2605.23445*, 2026.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Weinan Jia, Yuning Lu, Mengqi Huang, Hualiang Wang, Binyuan Huang, Nan Chen, Jidong Jiang, Zhendong Mao, et al. Moga: Mixture-of-groups attention for end-to-end long video generation. In *International Conference on Learning Representations*, volume 2026, pp. 115944–115964, 2026.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Weijie Kong, Qi Tian, Zijian Zhang, Rox Min, Zuozhuo Dai, Jin Zhou, Jiangfeng Xiong, Xin Li, Bo Wu, Jianwei Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

Qirui Li, Guangcong Zheng, Qi Zhao, Jie Li, Bin Dong, Yiwu Yao, and Xi Li. Compact attention: Exploiting structured spatio-temporal sparsity for fast video generation. *arXiv preprint arXiv:2508.12969*, 2025a.

Xingyang Li, Muyang Li, Tianle Cai, Haocheng Xi, Shuo Yang, Yujun Lin, Lvmin Zhang, Songlin Yang, Jinbo Hu, Kelly Peng, et al. Radial attention: $O(n \log n)$ sparse attention with energy decay for long video generation. *arXiv preprint arXiv:2506.19852*, 2025b.

Yucheng Li, Huiqiang Jiang, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, et al. Mminference: Accelerating pre-filling for long-context visual language models via modality-aware permutation sparse attention. In *International Conference on Machine Learning*, pp. 34998–35020. PMLR, 2025c.

Jiayi Luo, Jiayu Chen, Jiankun Wang, Cong Wang, Hanxin Zhu, Qingyun Sun, Chen Gao, Zhibo Chen, and Jianxin Li. Attention sparsity is input-stable: Training-free sparse attention for video generation via offline sparsity profiling and online qk co-clustering. *arXiv preprint arXiv:2603.18636*, 2026.

OpenCV Contributors. Opencv: Open source computer vision library. `https://github.com/opencv/opencv`, 2025. Accessed: 2025-02-26.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Wenhao Sun, Rong-Cheng Tu, Yifu Ding, Jingyi Liao, Zhao Jin, Shunyu Liu, and Dacheng Tao. Vorta: Efficient video diffusion via routing sparse attention. *Advances in Neural Information Processing Systems*, 38:7837–7863, 2026.

Haoyue Tan, Shengnan Wang, Yulin Qiao, Juncheng Zhang, Youhui Bai, Ping Gong, Zewen Jin, and Cheng Li. Adacluster: Adaptive query-key clustering for sparse attention in video generation. *arXiv preprint arXiv:2604.18348*, 2026a.

Xin Tan, Yuetao Chen, Yimin Jiang, Xing Chen, Kun Yan, Nan Duan, Yibo Zhu, Daxin Jiang, and Hong Xu. Dynamic sparsity in large-scale video dit training. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 101–116, 2026b.

TileLang. Tilelang github repository. `https://github.com/tile-ai/tilelang`, 2024.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Marcos Treviso, António Góis, Patrick Fernandes, Erick Fonseca, and André FT Martins. Predicting attention sparsity in transformers. In *Proceedings of the sixth workshop on structured prediction for nlp*, pp. 67–81, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Jianzong Wu, Liang Hou, Haotian Yang, Ye Tian, Pengfei Wan, Di ZHANG, and Yunhai Tong. Vmoba: Mixture-of-block attention for video diffusion models. In *International Conference on Learning Representations*, volume 2026, pp. 132534–132548, 2026.

Haocheng Xi, Shuo Yang, Yilong Zhao, Chenfeng Xu, Muyang Li, Xiuyu Li, Yujun Lin, Han Cai, Jintao Zhang, Dacheng Li, et al. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025.

Yifei Xia, Suhan Ling, Fangcheng Fu, Yujie Wang, Huixia Li, Xuefeng Xiao, and Bin Cui. Training-free and adaptive sparse attention for efficient long video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15982–15993, 2025.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *International Conference on Machine Learning*, pp. 69819–69831. PMLR, 2025.

Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Chenfeng Xu, Kelly Peng, et al. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a.

Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. In *International Conference on Learning Representations*, volume 2025, pp. 83048–83077, 2025b.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Chenlu Zhan, Wen Li, Chuyu Shen, Jun Zhang, Suhui Wu, and Hao Zhang. Bidirectional sparse attention for faster video diffusion training. *arXiv preprint arXiv:2509.01085*, 2025.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. In *International Conference on Machine Learning*, pp. 76397–76413. PMLR, 2025a.

Jintao Zhang, Kaiwen Zheng, Kai Jiang, Haoxu Wang, Ion Stoica, Joseph E Gonzalez, Jianfei Chen, and Jun Zhu. Turbodiffusion: Accelerating video diffusion models by 100-200 times. *arXiv preprint arXiv:2512.16093*, 2025b.

Peiyuan Zhang, Yongqi Chen, Runlong Su, Hangliang Ding, Ion Stoica, Zhengzhong Liu, and Hao Zhang. Fast video generation with sliding tile attention. In *International Conference on Machine Learning*, pp. 74714–74731. PMLR, 2025c.

Peiyuan Zhang, Yongqi Chen, Haofeng Huang, Will Lin, Zhengzhong Liu, Ion Stoica, Eric Xing, and Hao Zhang. Faster video diffusion with trainable sparse attention. *Advances in Neural Information Processing Systems*, 38:152509–152534, 2026.

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

Tianchen Zhao, Ke Hong, Xinhao Yang, Xuefeng Xiao, Huixia Li, Feng Ling, Ruiqi Xie, SiQi Chen, Hongyu Zhu, Zhang Yichong, et al. Paroattention: Pattern-aware reordering for efficient sparse and quantized attention in visual generation models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

A Direct Evaluation of Retrieval Mismatches

We evaluate both mismatches on Wan2.1-14B and HunyuanVideo. We measure approximation errors against exact full-query, full-key references, with the results summarized in Figure 4.

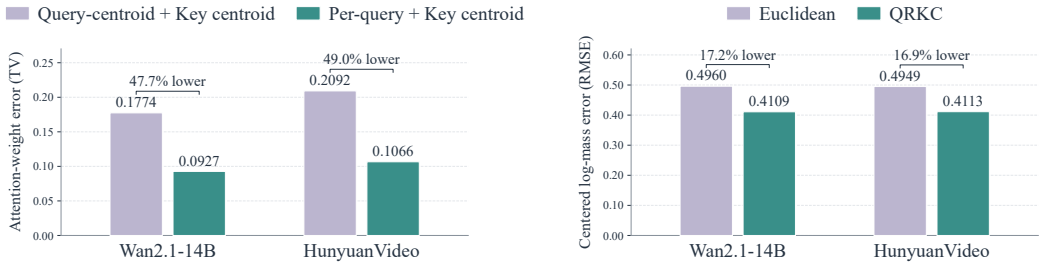


Figure 4: Direct evaluation of the two retrieval mismatches. Left: retaining original queries reduces attention-weight approximation error compared with query-centroid estimation. Right: QRKC reduces centered log-mass error compared with standard Euclidean clustering. Results correspond to Figure 1(c). Lower is better.

M1: Attention-weight approximation error. We fix the Euclidean query and key partitions and compare per-query and query-centroid estimation using the same key centroids. Unlike the full-key analysis in Section 4.1, this experiment includes key-centroid approximation in both estimates. The per-query estimation normalizes attention separately for each query before averaging the block weights, whereas the query-centroid estimation averages queries before normalization. Both estimates are evaluated against the exact block importance W_{uv} in Eq. (3), computed from the full queries and keys. For each layer–head pair, we measure

$$E_{\mathrm{TV}}^{m} = \sum_{u=1}^{N_Q} \frac{n_u^Q}{L} \left(\frac{1}{2} \sum_{v=1}^{N_K} \left| \widehat{W}_{uv}^{m} - W_{uv} \right| \right), \qquad m \in \{\mathrm{per}, \mathrm{centroid}\}. \tag{13}$$

Here, E_{TV}^{m} is the mean attention-weight TV error for one layer–head pair, with each query block weighted by its number of queries. The superscripts $m = \mathrm{per}$ and $m = \mathrm{centroid}$ denote per-query and query-centroid estimation, respectively, and $\widehat{W}_{uv}^{m}$ is the corresponding block-importance estimate. For each query block u, $\frac{1}{2}\sum_{v=1}^{N_K} |\widehat{W}_{uv}^{m} - W_{uv}|$ is the total variation (TV) distance between the estimated and exact block-weight distributions. We average the resulting errors equally across layer–head pairs. This metric compares each estimate with the exact reference, rather than measuring the difference between the two estimates.

As shown in Figure 4, retaining the original queries reduces mean TV from 0.1774 to 0.0927 on Wan2.1-14B and from 0.2092 to 0.1066 on HunyuanVideo. These correspond to relative reductions of 47.7% and 49.0%, respectively. These results show that using query centroids introduces substantial additional approximation error even when the key representation is unchanged.

Query-centroid error with full keys. To directly evaluate the full-key analysis in Section 4.1, we additionally retain all original keys and replace only the queries with their block centroids. For this configuration, the estimated block importance is $\widehat{W}_{uv} = \phi_v(\overline{\mathbf{q}}_u)$, where ϕ_v uses all original keys as defined in Eq. (17). We use the same Euclidean query and key partitions, exact reference W_{uv}, and TV weighting and averaging as above. Table 4 includes the two key-centroid configurations from Figure 4 for comparison.

As shown in Table 4, query-centroid estimation incurs mean TV errors of 0.1335 and 0.1667 even without key compression, directly measuring the approximation error analyzed in Section 4.1. Retaining the original queries while compressing keys yields lower mean errors than compressing queries while retaining full keys, with relative reductions of 30.6% and 36.0% on Wan2.1-14B and HunyuanVideo, respectively.

Table 4: Mean attention-weight TV under different query and key representations, using the same Euclidean partitions. Full-query, full-key estimation is the exact reference and has zero error by definition. The two key-centroid rows reproduce the M1 results in Figure 4. Lower is better. Bold indicates the lowest error among the compressed representations.

Query	Key	Wan2.1-14B	HunyuanVideo
Full	Full	0.0000	0.0000
Centroid	Centroid	0.1774	0.2092
Centroid	Full	0.1335	0.1667
Full	Centroid	**0.0927**	**0.1066**

M2: Centered log-mass approximation error. We retain all original queries and compare Euclidean key clustering with QRKC. For each partition, we compute the exact log-mass $S^{\star}_{iv}$ using the log-sum-exp expression in Eq. (7) and its key-centroid estimate $\widehat{S}_{iv}$ using Eq. (8). Their difference gives the approximation error:

$$\Delta^{K}_{iv} = S^{\star}_{iv} - \widehat{S}_{iv}. \tag{14}$$

For each query, we subtract the mean error across key blocks, since a common additive offset does not affect Softmax. We then measure the centered error using

$$\delta_{iv} = \Delta^{K}_{iv} - \frac{1}{|\mathcal{V}|}\sum_{r\in\mathcal{V}}\left(S^{\star}_{ir} - \widehat{S}_{ir}\right), \qquad E_{\mathrm{LM}} = \frac{1}{L}\sum_{i=1}^{L}\sqrt{\frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}}\delta^{2}_{iv}}. \tag{15}$$

Here, E_{LM} is the mean centered log-mass RMSE over all queries for one layer–head pair. $\mathcal{V}$ is the set of nonempty key blocks, and r is the index of these blocks in the sum. Thus, δ_{iv} subtracts the mean log-mass error over all nonempty key blocks for query i from its error on block v. We compute E_{LM} across key blocks separately for each query, average over the L queries, and then average equally across layer–head pairs. Each method uses the exact reference under its own key partition.

As shown in Figure 4, QRKC reduces mean centered log-mass RMSE from 0.4960 to 0.4109 on Wan2.1-14B and from 0.4949 to 0.4113 on HunyuanVideo, corresponding to relative reductions of 17.2% and 16.9%, respectively. These results show that clustering keys by QK-score similarity improves log-mass approximation compared with Euclidean clustering.

B Key-Space and QK-Score Approximation Errors

Using the same configuration as Appendix A, we compare the errors caused by replacing each key with its cluster centroid in the original key space and in the scaled QK scores:

$$D_{\mathrm{key}} = \frac{1}{L_K}\sum_{j=1}^{L_K}\|\mathbf{k}_j - \mathbf{c}_{v(j)}\|_2^2, \qquad D_{\mathrm{QK}} = \frac{1}{L_Q L_K}\sum_{i=1}^{L_Q}\sum_{j=1}^{L_K}\left[\frac{\mathbf{q}_i^{\top}(\mathbf{k}_j - \mathbf{c}_{v(j)})}{\sqrt{d}}\right]^2. \tag{16}$$

Here, $v(j)$ is the cluster containing key j, and L_Q and L_K are the numbers of queries and keys. Raw-key MSE D_{key} averages squared Euclidean distances over keys, without dividing by the feature dimension. D_{QK} averages squared scaled-score errors over all query–key pairs. Equivalently, it averages the within-block score variance over queries, weighting key blocks by their sizes. It therefore corresponds to twice the similarly weighted leading error term ϵ^{K}_{iv}, not the full log-mass error. Both metrics are averaged equally across layer–head pairs.

Table 5: Key-space and QK-score errors. Changes are relative to Euclidean clustering.

Model	Metric	Euclidean	QRKC	Change
Wan2.1-14B	Raw-key MSE	66.88	72.59	$+8.5\%$
	QK-score squared error	11.49	**7.28**	$\mathbf{-36.6\%}$
HunyuanVideo	Raw-key MSE	79.51	91.31	$+14.8\%$
	QK-score squared error	2.92	**2.26**	$\mathbf{-22.8\%}$

Table 5 shows that QRKC increases raw-key MSE while reducing QK-score error on both models. Thus, smaller Euclidean key-centroid distances do not necessarily yield more accurate QK scores, supporting the key-side clustering metric mismatch analyzed in Sec. 4.2.

C Effect of Query-Response Key Clustering

We evaluate QRKC by replacing Euclidean key clustering while keeping PAMA and the other settings unchanged. As shown in Figure 5, QRKC improves both mean and P05 attention recall on Wan2.1-14B and HunyuanVideo. These results support clustering keys according to their QK-score similarity under the current queries for more accurate block retrieval.

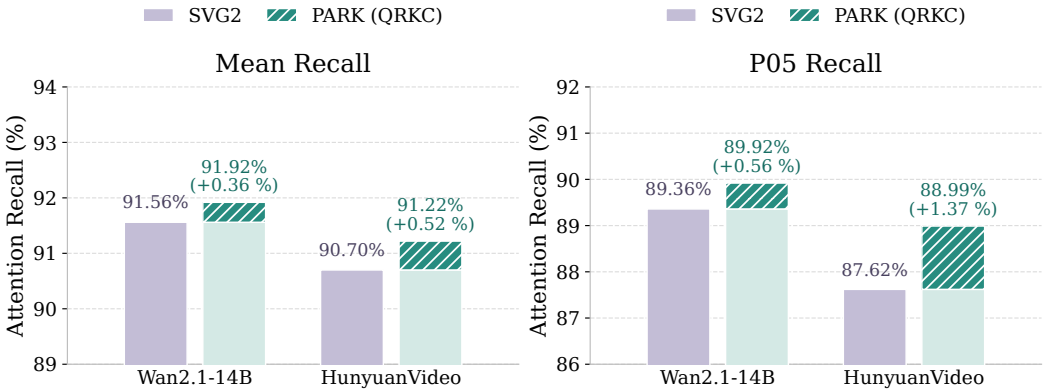


Figure 5: Effect of QRKC on mean and P05 attention recall.

D Detailed Derivations for Centroid-Based Block Retrieval

This appendix provides the complete expansions underlying the analyses in Sections 4.1 and 4.2. We retain all original keys when analyzing query-centroid approximation, and retain all original queries when analyzing key-centroid approximation.

The query and key partitions are held unchanged within each comparison. All sums over blocks below concern nonempty blocks. The quantities Δ^Q_{uv}, ϵ^K_{iv}, and Δ^K_{iv} have the same meanings as in the main text: the exact block importance minus its query-centroid estimate, the leading key-side error term, and the full key-side log-mass error, respectively.

D.1 Derivation of the Query-Side Aggregation Mismatch

Replacing queries with their centroid can introduce error even when all keys are retained exactly. Following Section 4.1, we analyze this error without any key-side compression. The exact attention weight assigned to a key block is

$$\phi_v(\mathbf{q}) = \sum_{j \in \mathcal{K}_v} p_j(\mathbf{q}), \qquad p_j(\mathbf{q}) = \frac{\exp(\mathbf{q}^\top \mathbf{k}_j/\sqrt{d})}{\sum_{\ell=1}^{L} \exp(\mathbf{q}^\top \mathbf{k}_\ell/\sqrt{d})}. \tag{17}$$

Here, $p_j(\mathbf{q})$ is the exact token-level attention weight for key $\mathbf{k}_j$, and $\phi_v(\mathbf{q})$ sums these weights over key block $\mathcal{K}_v$. The denominator includes all L original keys, and ℓ indexes individual keys. The query and key vectors belong to $\mathbb{R}^d$. Thus, averaging the exact block weights over the original queries gives the ground truth, whereas evaluating them at the query centroid gives an approximation:

$$W_{uv} = \frac{1}{n^Q_u} \sum_{i \in \mathcal{Q}_u} \phi_v(\mathbf{q}_i), \qquad \widehat{W}^{\text{centroid}}_{uv} = \phi_v(\overline{\mathbf{q}}_u). \tag{18}$$

Here, W_{uv} is the exact block importance in Eq. (3), and $\widehat{W}^{\text{centroid}}_{uv}$ is its query-centroid estimate. The set $\mathcal{Q}_u$ contains n^Q_u queries, and $\overline{\mathbf{q}}_u$ is their centroid. Both expressions use every original key. The only approximation in this comparison is replacing the queries with their centroid before Softmax normalization. Per-query computation itself is exact.

The query-block statistics used in the expansion are

$$\overline{\mathbf{q}}_u = \frac{1}{n^Q_u} \sum_{i \in \mathcal{Q}_u} \mathbf{q}_i, \qquad \boldsymbol{\epsilon}_i = \mathbf{q}_i - \overline{\mathbf{q}}_u. \tag{19}$$

Here, $\overline{\mathbf{q}}_u$ is the arithmetic mean of the queries and $\boldsymbol{\epsilon}_i$ is the vector deviation of query i from that mean. This vector is distinct from the scalar key-side error term ϵ^K_{iv}. The deviations have zero mean. Indeed,

$$\frac{1}{n^Q_u} \sum_{i \in \mathcal{Q}_u} \boldsymbol{\epsilon}_i = \frac{1}{n^Q_u} \sum_{i \in \mathcal{Q}_u} \mathbf{q}_i - \overline{\mathbf{q}}_u = \overline{\mathbf{q}}_u - \overline{\mathbf{q}}_u = \mathbf{0}. \tag{20}$$

Their second moment is

$$\boldsymbol{\Sigma}_{Q,u} = \frac{1}{n^Q_u} \sum_{i \in \mathcal{Q}_u} \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \in \mathbb{R}^{d \times d}. \tag{21}$$

Here, $\boldsymbol{\Sigma}_{Q,u}$ is the within-block query covariance, normalized by n^Q_u rather than the sample-covariance denominator $n^Q_u - 1$.

Taylor expansion of the exact block importance. For finite key vectors, ϕ_v in Eq. (17) is smooth because its denominator is strictly positive. We expand it along the segments connecting the query centroid to each query in $\mathcal{Q}_u$. The derivatives at the centroid are

$$\mathbf{g}_v = \nabla_{\mathbf{q}} \phi_v(\overline{\mathbf{q}}_u), \qquad \mathbf{H}_v = \nabla^2_{\mathbf{q}} \phi_v(\overline{\mathbf{q}}_u). \tag{22}$$

Here, $\mathbf{g}_v$ is the gradient and $\mathbf{H}_v = \mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u)$ is exactly the Hessian appearing in Eq. (5). Their dependence on the query block u is omitted from the notation for simplicity. A second-order Taylor expansion of each $\phi_v(\mathbf{q}_i)$ about the common point $\overline{\mathbf{q}}_u$ gives

$$\phi_v(\mathbf{q}_i) = \phi_v(\overline{\mathbf{q}}_u) + \mathbf{g}_v^\top \boldsymbol{\epsilon}_i + \frac{1}{2} \boldsymbol{\epsilon}_i^\top \mathbf{H}_v \boldsymbol{\epsilon}_i + \mathcal{R}^{Q,(3)}_{iv}. \tag{23}$$

Here, $\mathcal{R}_{iv}^{Q,(3)}$ is the higher-order remainder. Taylor's theorem gives

$$\left|\mathcal{R}_{iv}^{Q,(3)}\right| \leq \frac{M_v}{6}\|\boldsymbol{\epsilon}_i\|_2^3. \tag{24}$$

Here, M_v is a uniform upper bound on the operator norm of the third derivative $\nabla^3\phi_v$ along all segments connecting $\overline{\mathbf{q}}_u$ to the queries in $\mathcal{Q}_u$. The same bound M_v applies to all queries in $\mathcal{Q}_u$, so it also bounds the remainder after averaging within the block.

Substituting Eq. (23) into the definition of the exact block importance W_{uv} yields

$$\begin{aligned} W_{uv} = \phi_v(\overline{\mathbf{q}}_u) + \mathbf{g}_v^\top \left(\frac{1}{n_u^Q}\sum_{i\in\mathcal{Q}_u}\boldsymbol{\epsilon}_i\right) \\ + \frac{1}{2n_u^Q}\sum_{i\in\mathcal{Q}_u}\boldsymbol{\epsilon}_i^\top\mathbf{H}_v\boldsymbol{\epsilon}_i + \frac{1}{n_u^Q}\sum_{i\in\mathcal{Q}_u}\mathcal{R}_{iv}^{Q,(3)}. \end{aligned} \tag{25}$$

Equation (20) proves explicitly that the gradient term vanishes:

$$\mathbf{g}_v^\top\left(\frac{1}{n_u^Q}\sum_{i\in\mathcal{Q}_u}\boldsymbol{\epsilon}_i\right) = \mathbf{g}_v^\top\mathbf{0} = 0. \tag{26}$$

For the quadratic term, the identity $\boldsymbol{\epsilon}_i^\top\mathbf{H}_v\boldsymbol{\epsilon}_i = \mathrm{tr}(\mathbf{H}_v\boldsymbol{\epsilon}_i\boldsymbol{\epsilon}_i^\top)$ gives

$$\frac{1}{n_u^Q}\sum_{i\in\mathcal{Q}_u}\boldsymbol{\epsilon}_i^\top\mathbf{H}_v\boldsymbol{\epsilon}_i = \mathrm{tr}(\mathbf{H}_v\boldsymbol{\Sigma}_{Q,u}) = \langle\mathbf{H}_v, \boldsymbol{\Sigma}_{Q,u}\rangle_F. \tag{27}$$

Here, $\mathrm{tr}(\cdot)$ is the matrix trace and $\langle\mathbf{A},\mathbf{B}\rangle_F = \mathrm{tr}(\mathbf{A}^\top\mathbf{B})$ is the Frobenius inner product. Because a Hessian is symmetric, $\mathrm{tr}(\mathbf{H}_v\boldsymbol{\Sigma}_{Q,u}) = \langle\mathbf{H}_v, \boldsymbol{\Sigma}_{Q,u}\rangle_F$.

Averaging the pointwise remainders gives

$$\mathcal{R}_{uv}^Q = \frac{1}{n_u^Q}\sum_{i\in\mathcal{Q}_u}\mathcal{R}_{iv}^{Q,(3)}. \tag{28}$$

Here, $\mathcal{R}_{uv}^Q$ is the higher-order remainder used in the main text. Combining Eqs. (25)–(27), the exact block importance expands as

$$W_{uv} = \phi_v(\overline{\mathbf{q}}_u) + \frac{1}{2}\langle\mathbf{H}_v, \boldsymbol{\Sigma}_{Q,u}\rangle_F + \mathcal{R}_{uv}^Q. \tag{29}$$

In contrast, the query-centroid estimator evaluates ϕ_v exactly at the expansion point and is simply

$$\widehat{W}_{uv}^{\mathrm{centroid}} = \phi_v(\overline{\mathbf{q}}_u). \tag{30}$$

Subtracting Eq. (30) from Eq. (29) now gives

$$\Delta_{uv}^Q = W_{uv} - \widehat{W}_{uv}^{\mathrm{centroid}} = \frac{1}{2}\langle\mathbf{H}_v, \boldsymbol{\Sigma}_{Q,u}\rangle_F + \mathcal{R}_{uv}^Q. \tag{31}$$

Using the uniform bound in Eq. (24) and the triangle inequality, the averaged remainder satisfies

$$|\mathcal{R}_{uv}^Q| \leq \frac{M_v}{6n_u^Q}\sum_{i\in\mathcal{Q}_u}\|\boldsymbol{\epsilon}_i\|_2^3. \tag{32}$$

Thus, the first-order effect cancels exactly, giving the expression for Δ_{uv}^Q in Eq. (5). Its leading term couples query covariance with the curvature of the exact normalized block weight. The signed error can be positive or negative: a query centroid can either underestimate or overestimate the importance of a particular key block. In this analysis, Δ_{uv}^Q is the error relative to ground truth, not the difference between two approximate estimators, because no key-side approximation is used.

Dependence of the Hessian on all key blocks. To make the curvature term explicit without introducing key centroids, we write the exact token scores as

$$z_j(\mathbf{q}) = \mathbf{a}_j^\top \mathbf{q}, \qquad \mathbf{a}_j = \frac{\mathbf{k}_j}{\sqrt{d}}. \tag{33}$$

Here, $\mathbf{a}_j$ is the scaled original key, and $z_j(\mathbf{q})$ is its scaled QK score for query $\mathbf{q}$. The corresponding attention-weighted statistics over all original keys are

$$\bar{\mathbf{a}}(\mathbf{q}) = \sum_{j=1}^{L} p_j(\mathbf{q})\mathbf{a}_j, \qquad \mathbf{C}_a(\mathbf{q}) = \sum_{j=1}^{L} p_j(\mathbf{q})(\mathbf{a}_j - \bar{\mathbf{a}}(\mathbf{q}))(\mathbf{a}_j - \bar{\mathbf{a}}(\mathbf{q}))^\top. \tag{34}$$

Here, $\bar{\mathbf{a}}(\mathbf{q})$ and $\mathbf{C}_a(\mathbf{q})$ are the attention-weighted mean and covariance of the scaled keys, respectively. Both are computed using the exact token attention weights $p_j(\mathbf{q})$ defined in Eq. (17). Differentiating token-level Softmax gives

$$\nabla_{\mathbf{q}} p_j(\mathbf{q}) = p_j(\mathbf{q})(\mathbf{a}_j - \bar{\mathbf{a}}(\mathbf{q})), \qquad \nabla_{\mathbf{q}} \bar{\mathbf{a}}(\mathbf{q}) = \mathbf{C}_a(\mathbf{q}). \tag{35}$$

The second identity follows by substituting the first into the derivative of $\sum_j p_j(\mathbf{q})\mathbf{a}_j$. Here, $\nabla_{\mathbf{q}}\bar{\mathbf{a}}$ denotes its Jacobian. Differentiating the token weight once more yields

$$\nabla^2_{\mathbf{q}} p_j(\mathbf{q}) = p_j(\mathbf{q}) \left[(\mathbf{a}_j - \bar{\mathbf{a}}(\mathbf{q}))(\mathbf{a}_j - \bar{\mathbf{a}}(\mathbf{q}))^\top - \mathbf{C}_a(\mathbf{q})\right]. \tag{36}$$

Since ϕ_v sums the token weights in key block $\mathcal{K}_v$, its gradient and Hessian at the query centroid are

$$\begin{aligned} \mathbf{g}_v &= \sum_{j \in \mathcal{K}_v} p_{uj}(\mathbf{a}_j - \bar{\mathbf{a}}_u), \\ \mathbf{H}_v &= \sum_{j \in \mathcal{K}_v} p_{uj} \left[(\mathbf{a}_j - \bar{\mathbf{a}}_u)(\mathbf{a}_j - \bar{\mathbf{a}}_u)^\top - \mathbf{C}_{a,u}\right]. \end{aligned} \tag{37}$$

Here, $p_{uj} = p_j(\overline{\mathbf{q}}_u)$ is the attention weight assigned to key j by the query centroid. The quantities $\bar{\mathbf{a}}_u = \bar{\mathbf{a}}(\overline{\mathbf{q}}_u)$ and $\mathbf{C}_{a,u} = \mathbf{C}_a(\overline{\mathbf{q}}_u)$ are the mean and covariance of the scaled keys, respectively, weighted by these attention weights. These expressions use individual keys, not key centroids. Both $\bar{\mathbf{a}}_u$ and $\mathbf{C}_{a,u}$ depend on all original keys and their Softmax weights, including keys outside block v. Consequently, the Hessian captures how the exact attention weight of block v changes as the query changes, accounting for competition with all other keys under Softmax normalization.

Implications for query-centroid estimation. Accurate query-centroid estimation requires a small $|\Delta^Q_{uv}|$. Since the covariance matrix $\mathbf{\Sigma}_{Q,u}$ is positive semidefinite, the leading error term satisfies

$$\left|\frac{1}{2}\langle \mathbf{H}_v, \mathbf{\Sigma}_{Q,u}\rangle_F\right| \le \frac{1}{2}\|\mathbf{H}_v\|_{\mathrm{op}} \operatorname{tr}(\mathbf{\Sigma}_{Q,u}), \qquad \operatorname{tr}(\mathbf{\Sigma}_{Q,u}) = \frac{1}{n_u^Q}\sum_{i \in \mathcal{Q}_u} \|\boldsymbol{\epsilon}_i\|_2^2. \tag{38}$$

Here, $\mathbf{H}_v = \mathbf{H}_{\phi_v}(\overline{\mathbf{q}}_u)$ is the Hessian at the query centroid, and $\|\mathbf{H}_v\|_{\mathrm{op}}$ is its operator norm, equal to its largest absolute eigenvalue. The trace $\operatorname{tr}(\mathbf{\Sigma}_{Q,u})$ is the total within-block query variance, or the average squared Euclidean distance from the queries to their centroid. The inequality follows by applying $|\boldsymbol{\epsilon}_i^\top \mathbf{H}_v \boldsymbol{\epsilon}_i| \le \|\mathbf{H}_v\|_{\mathrm{op}}\|\boldsymbol{\epsilon}_i\|_2^2$ to each query and averaging. Thus, a small product of the Hessian norm and the total query variance is sufficient for a small leading error. Combining this inequality with Eq. (32) also bounds the full approximation error:

$$|\Delta^Q_{uv}| \le \frac{1}{2}\|\mathbf{H}_v\|_{\mathrm{op}} \operatorname{tr}(\mathbf{\Sigma}_{Q,u}) + \frac{M_v}{6n_u^Q}\sum_{i \in \mathcal{Q}_u} \|\boldsymbol{\epsilon}_i\|_2^3. \tag{39}$$

Controlling both terms on the right-hand side is sufficient to make the full error small.

If all queries in a block are identical, every deviation $\boldsymbol{\epsilon}_i$ is zero and query-centroid estimation is exact. More generally, with bounded derivatives, both the second-order term and the remainder vanish as the query deviations approach zero. In practice, clustered queries may still have different attention preferences. Standard euclidean query clustering reduces but does not generally eliminate within-block query variance and does not account for the Hessian, so it does not by itself ensure that the product in Eq. (38) is sufficiently small. We next evaluate the approximation error empirically under practical clustering settings. We therefore evaluate the error empirically. Table 4 shows that query-centroid estimation with full keys incurs mean attention-weight TV errors of 0.1335 and 0.1667 on Wan2.1-14B and HunyuanVideo, respectively, exceeding the errors obtained by retaining original queries with key centroids.

Relation to estimation with key centroids. The analysis above retains all original keys to examine the error introduced by query centroids. PAMA additionally uses key centroids, which introduce a separate approximation error. Figure 1(c) shows that retaining original queries reduces attention-weight approximation error when both estimates use the same key centroids. Details of this experiment are provided in Appendix A. Table 1 compares different query and key compression settings and shows that retaining full queries while using key centroids achieves the highest recall among the compressed representations. Together, these empirical results support preserving original queries for practical block retrieval and motivate PAMA.

D.2 Derivation of the Query-Response Key Metric

For each key block, we decompose its keys around their mean:

$$\mathbf{c}_v = \frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} \mathbf{k}_j, \qquad \boldsymbol{\delta}_{vj} = \mathbf{k}_j - \mathbf{c}_v. \tag{40}$$

Here, $\mathbf{c}_v$ is the centroid of the n_v^K keys in $\mathcal{K}_v$, and $\boldsymbol{\delta}_{vj}$ is the deviation of key j from this centroid. As on the query side, the deviations have zero mean:

$$\frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} \boldsymbol{\delta}_{vj} = \frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} \mathbf{k}_j - \mathbf{c}_v = \mathbf{0}. \tag{41}$$

The corresponding scaled score deviation is

$$x_{ivj} = \frac{\mathbf{q}_i^\top \boldsymbol{\delta}_{vj}}{\sqrt{d}}. \tag{42}$$

Here, x_{ivj} is the difference between the scaled QK score of query $\mathbf{q}_i$ and key $\mathbf{k}_j$ and the score obtained by replacing $\mathbf{k}_j$ with its block centroid $\mathbf{c}_v$. Equation (41) immediately implies

$$\frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} x_{ivj} = \frac{\mathbf{q}_i^\top}{\sqrt{d}} \left(\frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} \boldsymbol{\delta}_{vj} \right) = 0. \tag{43}$$

Exact decomposition. Using $\mathbf{k}_j = \mathbf{c}_v + \boldsymbol{\delta}_{vj}$, the exact block log-mass decomposes as

$$\begin{aligned} S_{iv}^\star &= \log \sum_{j \in \mathcal{K}_v} \exp\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}} \right) \\ &= \log n_v^K + \frac{\mathbf{q}_i^\top \mathbf{c}_v}{\sqrt{d}} + \underbrace{\log \left[\frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} \exp(x_{ivj}) \right]}_{\Delta_{iv}^K}. \end{aligned} \tag{44}$$

Here, $S_{iv}^\star$ is the exact log-mass in Eq. (7). The first two terms form $\widehat{S}_{iv} = \log n_v^K + \mathbf{q}_i^\top \mathbf{c}_v / \sqrt{d}$, the estimate in Eq. (8). Hence, $\Delta_{iv}^K = S_{iv}^\star - \widehat{S}_{iv}$ is the full log-mass approximation error. Their relation to attention is

$$P_{iv}^\star = \frac{e^{S_{iv}^\star}}{\sum_r e^{S_{ir}^\star}}, \qquad \widehat{P}_{iv} = \frac{e^{\widehat{S}_{iv}}}{\sum_r e^{\widehat{S}_{ir}}}. \tag{45}$$

Here, $P_{iv}^\star$ and $\widehat{P}_{iv}$ are the exact and estimated attention weights for query i, respectively, and the sums run over all nonempty key blocks. These expressions show how key-centroid approximation affects attention weights through the block log-masses. The term $\log n_v^K$ accounts for the number of keys represented by each centroid and is necessary when key blocks have different sizes.

Second-order expansion. To derive the second-order term and bound the remainder, we compute

$$m_{2,iv} = \frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} x_{ivj}^2, \qquad m_{3,iv} = \frac{1}{n_v^K} \sum_{j \in \mathcal{K}_v} |x_{ivj}|^3. \tag{46}$$

Here, $m_{2,iv}$ is the second moment and $m_{3,iv}$ is the absolute third moment of the centered scaled scores. For each deviation, Taylor's theorem gives $\exp(x) = 1 + x + x^2/2 + r_3(x)$. Here, $r_3(x)$ is the remainder after retaining the terms 1, x, and $x^2/2$. If $|x_{ivj}| \leq \eta$ within the block, then $|r_3(x_{ivj})| \leq e^{\eta}|x_{ivj}|^3/6$. Here, η is an upper bound on the absolute scaled-score deviations for query i within key block $\mathcal{K}_v$, satisfying $\eta \geq \max_{j\in\mathcal{K}_v} |x_{ivj}|$. Averaging this expansion and using Eq. (43) therefore gives

$$\frac{1}{n_v^K}\sum_{j\in\mathcal{K}_v}\exp(x_{ivj}) = 1 + \frac{1}{2}m_{2,iv} + \rho_{iv}^{\exp}, \qquad |\rho_{iv}^{\exp}| \leq \frac{e^{\eta}}{6}m_{3,iv}. \tag{47}$$

Here, $\rho_{iv}^{\exp} = (1/n_v^K)\sum_{j\in\mathcal{K}_v} r_3(x_{ivj})$ is the average of the Taylor remainders within key block $\mathcal{K}_v$ for query i. The key deviations also give

$$\mathbf{\Sigma}_{K,v} = \frac{1}{n_v^K}\sum_{j\in\mathcal{K}_v}\boldsymbol{\delta}_{vj}\boldsymbol{\delta}_{vj}^{\top} \in \mathbb{R}^{d\times d}. \tag{48}$$

Here, $\mathbf{\Sigma}_{K,v}$ is the within-block key covariance, normalized by the number of keys. The second moment can now be written as a quadratic form:

$$m_{2,iv} = \frac{1}{dn_v^K}\sum_{j\in\mathcal{K}_v}(\mathbf{q}_i^{\top}\boldsymbol{\delta}_{vj})^2 = \frac{1}{d}\mathbf{q}_i^{\top}\mathbf{\Sigma}_{K,v}\mathbf{q}_i. \tag{49}$$

Substituting Eq. (47) into the exact error expression in Eq. (44) gives

$$\Delta_{iv}^K = \log\left(1 + \frac{1}{2}m_{2,iv} + \rho_{iv}^{\exp}\right). \tag{50}$$

For sufficiently small score deviations, $y = m_{2,iv}/2 + \rho_{iv}^{\exp}$ is close to zero. Using the Taylor expansion $\log(1+y) = y + \mathcal{O}(y^2)$ and collecting the higher-order terms into $\mathcal{R}_{iv}^K$, we obtain

$$\Delta_{iv}^K = \frac{1}{2}m_{2,iv} + \mathcal{R}_{iv}^K = \frac{1}{2d}\mathbf{q}_i^{\top}\mathbf{\Sigma}_{K,v}\mathbf{q}_i + \mathcal{R}_{iv}^K. \tag{51}$$

To match the main-text notation, the leading term and full error are

$$\epsilon_{iv}^K = \frac{m_{2,iv}}{2} = \frac{1}{2d}\mathbf{q}_i^{\top}\mathbf{\Sigma}_{K,v}\mathbf{q}_i, \qquad \Delta_{iv}^K = \epsilon_{iv}^K + \mathcal{R}_{iv}^K. \tag{52}$$

Here, ϵ_{iv}^K is the leading error term, while $\mathcal{R}_{iv}^K$ collects higher-order contributions from the exponential and logarithmic expansions. In particular, ϵ_{iv}^K is one half of the within-block variance of scaled QK scores, or $1/(2d)$ times the variance of unscaled scores. It is not the full approximation error. For bounded, sufficiently small score deviations, there is a constant $C_\eta > 0$ such that

$$|\mathcal{R}_{iv}^K| \leq C_\eta\left(m_{3,iv} + m_{2,iv}^2\right). \tag{53}$$

The first term in the bound comes from the exponential remainder, while the second arises from the quadratic remainder of $\log(1+y)$. Independently of the small-deviation approximation, Jensen's inequality gives the exact result

$$\Delta_{iv}^K = \log\left(\frac{1}{n_v^K}\sum_{j\in\mathcal{K}_v} e^{x_{ivj}}\right) \geq \frac{1}{n_v^K}\sum_{j\in\mathcal{K}_v} x_{ivj} = 0. \tag{54}$$

Thus, the centroid estimate is always a lower bound on the exact block log-mass, and the leading error term ϵ_{iv}^K is proportional to the variance of the keys after projection onto the current query.

Query-induced clustering metric. To construct a key-clustering objective that accounts for all current queries, we use

$$\mathbf{G}_Q = \frac{1}{L}\sum_{i=1}^{L}\mathbf{q}_i\mathbf{q}_i^{\top} = \frac{1}{L}\mathbf{Q}^{\top}\mathbf{Q} \in \mathbb{R}^{d\times d}. \tag{55}$$

Here, $\mathbf{G}_Q$ is the uncentered second-moment matrix of the L current queries, stored as rows of $\mathbf{Q}$. Unlike a centered covariance, it also retains their mean direction. Using $\mathbf{x}^\top \mathbf{A}\mathbf{x} = \mathrm{tr}(\mathbf{A}\mathbf{x}\mathbf{x}^\top)$ and the cyclic property of the trace, we have

$$\begin{aligned}\bar{\epsilon}_v^K &= \frac{1}{L}\sum_{i=1}^{L}\frac{1}{2d}\mathbf{q}_i^\top \mathbf{\Sigma}_{K,v}\mathbf{q}_i \\ &= \frac{1}{2d}\,\mathrm{tr}(\mathbf{G}_Q\mathbf{\Sigma}_{K,v}) \\ &= \frac{1}{2dn_v^K}\sum_{j\in\mathcal{K}_v}\boldsymbol{\delta}_{vj}^\top\mathbf{G}_Q\boldsymbol{\delta}_{vj}.\end{aligned} \tag{56}$$

Here, $\bar{\epsilon}_v^K = (1/L)\sum_i \epsilon_{iv}^K$ is the leading error term averaged over all current queries. Equation (56) connects $\bar{\epsilon}_v^K$ to a query-dependent clustering objective. Multiplying by the block size and summing over key blocks gives the total token-weighted distortion

$$\begin{aligned}\mathcal{J}_{\mathrm{QRKC}} &= \sum_{v=1}^{N_K}\sum_{j\in\mathcal{K}_v}\boldsymbol{\delta}_{vj}^\top\mathbf{G}_Q\boldsymbol{\delta}_{vj} = 2d\sum_{v=1}^{N_K}n_v^K\bar{\epsilon}_v^K \\ &= \frac{1}{L}\sum_{i=1}^{L}\sum_{v=1}^{N_K}\sum_{j\in\mathcal{K}_v}\left(\mathbf{q}_i^\top\boldsymbol{\delta}_{vj}\right)^2.\end{aligned} \tag{57}$$

The last equality follows by substituting the definition of $\mathbf{G}_Q$ into each quadratic form. $\mathcal{J}_{\mathrm{QRKC}}/d$ measures the squared error introduced by replacing each key with its block centroid in the scaled attention logits. This error is summed over all keys and averaged over all queries. This objective weights each key block's query-averaged leading error term $\bar{\epsilon}_v^K$ by its number of keys n_v^K.

Ordinary Euclidean clustering instead minimizes

$$\mathcal{J}_{\mathrm{Euc}} = \sum_{v=1}^{N_K}\sum_{j\in\mathcal{K}_v}\boldsymbol{\delta}_{vj}^\top\mathbf{I}\boldsymbol{\delta}_{vj}, \tag{58}$$

where $\mathbf{I}$ is the identity matrix. Comparing Eqs. (57) and (58) shows that the mismatch is precisely the use of $\mathbf{I}$ instead of the query-induced metric $\mathbf{G}_Q$.

Euclidean realization. To optimize the query-dependent objective with a standard Euclidean K-means implementation, we need a key transformation whose squared Euclidean distances equal the quadratic forms defined by $\mathbf{G}_Q$. Directly mapping each key to $\mathbf{Q}\mathbf{k}_j/\sqrt{L}$ satisfies this requirement, but produces an L-dimensional feature vector instead of a d-dimensional one. For the long video sequences considered here, $L \gg d$, so clustering these vectors would increase both storage and distance-computation costs. We instead factorize the $d \times d$ matrix $\mathbf{G}_Q$ to construct a transformation that retains the feature dimension d while realizing the same distances. We first establish this property using a symmetric square root and then show how Cholesky decomposition provides the factor used in our implementation. The matrix $\mathbf{G}_Q$ is positive semidefinite because, for any $\mathbf{x} \in \mathbb{R}^d$,

$$\mathbf{x}^\top\mathbf{G}_Q\mathbf{x} = \frac{1}{L}\sum_{i=1}^{L}(\mathbf{q}_i^\top\mathbf{x})^2 = \frac{1}{L}\|\mathbf{Q}\mathbf{x}\|_2^2 \geq 0. \tag{59}$$

It therefore admits the eigendecomposition and principal square root

$$\mathbf{G}_Q = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \qquad \mathbf{G}_Q^{1/2} = \mathbf{U}\mathbf{\Lambda}^{1/2}\mathbf{U}^\top, \tag{60}$$

where $\mathbf{U} \in \mathbb{R}^{d\times d}$ contains orthonormal eigenvectors and $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$ is diagonal with nonnegative eigenvalues. The symmetric-square-root transformation is

$$\widetilde{\mathbf{k}}_j^{\mathrm{sqrt}} = \mathbf{G}_Q^{1/2}\mathbf{k}_j, \qquad \widetilde{\mathbf{c}}_v^{\mathrm{sqrt}} = \mathbf{G}_Q^{1/2}\mathbf{c}_v. \tag{61}$$

Here, the superscript sqrt identifies the square-root realization of the transformed key and centroid. For the residual $\boldsymbol{\delta}_{vj} = \mathbf{k}_j - \mathbf{c}_v$, we have

$$\|\widetilde{\mathbf{k}}_j^{\mathrm{sqrt}} - \widetilde{\mathbf{c}}_v^{\mathrm{sqrt}}\|_2^2 = \|\mathbf{G}_Q^{1/2}\boldsymbol{\delta}_{vj}\|_2^2 = (\mathbf{k}_j - \mathbf{c}_v)^\top\mathbf{G}_Q(\mathbf{k}_j - \mathbf{c}_v). \tag{62}$$

When $\mathbf{G}_Q$ is positive definite, it also admits the Cholesky decomposition

$$\mathbf{G}_Q = \mathbf{R}\mathbf{R}^\top. \tag{63}$$

The corresponding transformation is

$$\widetilde{\mathbf{k}}_j^{\mathrm{chol}} = \mathbf{R}^\top \mathbf{k}_j, \qquad \widetilde{\mathbf{c}}_v^{\mathrm{chol}} = \mathbf{R}^\top \mathbf{c}_v. \tag{64}$$

Here, the superscript chol identifies the Cholesky realization, whose transformed key is $\widetilde{\mathbf{k}}_j$ in Eq. (12). Their squared Euclidean distance is

$$\begin{aligned} \|\widetilde{\mathbf{k}}_j^{\mathrm{chol}} - \widetilde{\mathbf{c}}_v^{\mathrm{chol}}\|_2^2 &= \|\mathbf{R}^\top \boldsymbol{\delta}_{vj}\|_2^2 \\ &= \boldsymbol{\delta}_{vj}^\top \mathbf{R}\mathbf{R}^\top \boldsymbol{\delta}_{vj} \\ &= \boldsymbol{\delta}_{vj}^\top \mathbf{G}_Q \boldsymbol{\delta}_{vj}. \end{aligned} \tag{65}$$

In particular, the direct query-based transformation and the Cholesky transformation satisfy

$$\left\| \frac{\mathbf{Q}\boldsymbol{\delta}_{vj}}{\sqrt{L}} \right\|_2^2 = \boldsymbol{\delta}_{vj}^\top \mathbf{G}_Q \boldsymbol{\delta}_{vj} = \|\mathbf{R}^\top \boldsymbol{\delta}_{vj}\|_2^2. \tag{66}$$

The vector on the left has L components, whereas the transformed residual on the right has only d components. Thus, Cholesky decomposition lets us realize the query-dependent distance in Eq. (11) without constructing an L-dimensional representation for every key or modifying the distance computation in K-means. For each attention head, we compute $\mathbf{R}$ once before K-means and use it to transform all keys. For the same numbers of keys, clusters, and iterations, the K-means stage retains the computational complexity of clustering the original d-dimensional keys. The additional work is computing $\mathbf{G}_Q$, factorizing it, and transforming the keys. Figure 6 shows the time breakdown of these operations, whose total measured overhead remains below 2 ms across the evaluated settings. Equations (62) and (65) show that the symmetric-square-root and Cholesky transformations produce the same squared distance under $\mathbf{G}_Q$. Because a linear transformation also maps each original centroid to the centroid of the transformed keys, summing either distance over all clusters gives

$$\sum_{v=1}^{N_K} \sum_{j \in \mathcal{K}_v} \|\widetilde{\mathbf{k}}_j - \widetilde{\mathbf{c}}_v\|_2^2 = \sum_{v=1}^{N_K} \sum_{j \in \mathcal{K}_v} \boldsymbol{\delta}_{vj}^\top \mathbf{G}_Q \boldsymbol{\delta}_{vj} = \mathcal{J}_{\mathrm{QRKC}}. \tag{67}$$

Therefore, Euclidean K-means after either transformation optimizes exactly the same QRKC objective in Eq. (57). The symmetric square root remains valid when $\mathbf{G}_Q$ is rank deficient, whereas standard Cholesky decomposition requires a positive-definite matrix.

Numerically stable Cholesky factorization. To ensure a stable Cholesky factorization, we apply scale-aware shrinkage to the query-induced metric. The scale used for stabilization is

$$s_Q = \max\left(\frac{1}{d} \operatorname{tr}(\mathbf{G}_Q),\ s_{\min} \right), \qquad s_{\min} > 0. \tag{68}$$

Here, s_Q is the average diagonal value floored at $s_{\min} > 0$, which is set to 10^{-12} in the implementation. We construct the stabilized metric

$$\overline{\mathbf{G}}_Q = (1-\alpha)\mathbf{G}_Q + \alpha s_Q \mathbf{I} + \epsilon s_Q \mathbf{I}, \tag{69}$$

where $\alpha \in [0,1]$ is the shrinkage coefficient and $\epsilon > 0$ is a small stability constant. The isotropic terms make $\overline{\mathbf{G}}_Q$ positive definite while keeping their scale consistent with $\mathbf{G}_Q$. We then compute

$$\overline{\mathbf{G}}_Q = \mathbf{R}\mathbf{R}^\top \tag{70}$$

and transform each key by $\widetilde{\mathbf{k}}_j = \mathbf{R}^\top \mathbf{k}_j$. Following the same argument as Eq. (65), Euclidean K-means on these transformed keys optimizes the stabilized objective

$$\overline{\mathcal{J}}_{\mathrm{QRKC}} = \sum_{v=1}^{N_K} \sum_{j \in \mathcal{K}_v} \boldsymbol{\delta}_{vj}^\top \overline{\mathbf{G}}_Q \boldsymbol{\delta}_{vj}. \tag{71}$$

The stabilized objective satisfies

$$\overline{\mathcal{J}}_{\mathrm{QRKC}} = (1-\alpha)\mathcal{J}_{\mathrm{QRKC}} + (\alpha+\epsilon)s_Q\mathcal{J}_{\mathrm{Euc}}. \tag{72}$$

Thus, stabilization adds an isotropic penalty, so the resulting objective differs from the unregularized objective. Even small stabilization terms can change cluster assignments when the original metric assigns very little weight to certain directions. The scalar ϵ here is a numerical stability parameter, distinct from the leading approximation error ϵ_{iv}^K.

K-means optimizes the specified objective iteratively. The shrinkage term also provides a controlled interpolation between the query-induced metric and ordinary Euclidean distance: setting $\alpha = 0$ recovers the original metric up to the small stability term, whereas $\alpha = 1$ yields an isotropic metric up to a global scaling factor. With $\alpha = 0.05$ and the numerical stabilization described above, we observed no Cholesky decomposition failures in our experiments.

E FUSED PAMA KERNEL

Algorithms 1 and 2 describe two fused implementations for one attention head. Programs for different attention heads and query blocks are executed in parallel. The queries are stored contiguously according to their query-block assignments, and o_u and o_{u+1} denote the beginning and end of query block $\mathcal{Q}_u$, respectively. The Full-K kernel retains the logits over all key centroids in registers and is used when $N_K \le 1024$. For larger N_K, retaining the complete logit row creates excessive register pressure, so we use a two-pass kernel that processes the key centroids in tiles.

In both algorithms, i indexes a query in the current tile I, and v indexes a key block in the current key-centroid tile J or in $\{1,\dots,N_K\}$ for Full-K. Tile entries S_{iv} and P_{iv} use these query and key-block indices. All indexed assignments apply in parallel over the relevant indices in these ranges, not as serial loops.

Algorithm 1 Full-K fused PAMA kernel

Require: Reordered queries $\mathbf{Q} \in \mathbb{R}^{L\times d}$; query-block offsets $\{o_u\}_{u=1}^{N_Q+1}$; key centroids $\mathbf{C} \in \mathbb{R}^{N_K\times d}$; key-block sizes $\{n_v^K\}_{v=1}^{N_K}$
Ensure: Block importance $\widehat{\mathbf{W}}^{\mathrm{PAMA}} \in \mathbb{R}^{N_Q\times N_K}$

for all query blocks $u = 1,\dots,N_Q$ in parallel **do**
  $\mathbf{a} \leftarrow \mathbf{0} \in \mathbb{R}^{N_K}$ ▷ register-resident accumulator
  for each query tile $I \subseteq \{o_u,\dots,o_{u+1}-1\}$ **do**
    $\mathbf{S} \leftarrow \mathbf{0} \in \mathbb{R}^{|I|\times N_K}$ ▷ register-resident logits
    for each feature tile $D_t \subseteq \{1,\dots,d\}$ **do**
      Load $\mathbf{Q}_{I,D_t}$ and $\mathbf{C}_{:,D_t}$
      $\mathbf{S} \leftarrow \mathbf{S} + \mathbf{Q}_{I,D_t}\mathbf{C}_{:,D_t}^\top$
    end for
    $S_{iv} \leftarrow S_{iv}/\sqrt{d} + \log n_v^K$
    $m_i \leftarrow \max_v S_{iv}$ ▷ row maximum
    $E_{iv} \leftarrow \exp(S_{iv} - m_i)$ ▷ register resident
    $z_i \leftarrow \sum_v E_{iv}$ ▷ row normalization factor
    $P_{iv} \leftarrow E_{iv}/z_i$ ▷ row-wise Softmax
    $a_v \leftarrow a_v + \sum_{i\in I} P_{iv}$ ▷ sum over queries
  end for
  $\widehat{\mathbf{W}}^{\mathrm{PAMA}}_{u,:} \leftarrow \mathbf{a}/(o_{u+1}-o_u)$ ▷ the only HBM write
end for
return $\widehat{\mathbf{W}}^{\mathrm{PAMA}}$

The intermediate tensors and attention-weight accumulator in Algorithm 1 remain in on-chip registers, without materializing the full $L \times N_K$ logit or probability matrix in HBM. It computes every QK-centroid product once, but its register footprint grows with N_K.

Algorithm 2 Two-pass fused PAMA kernel

Require: Reordered queries $\mathbf{Q} \in \mathbb{R}^{L\times d}$; query-block offsets $\{o_u\}_{u=1}^{N_Q+1}$; key centroids $\mathbf{C} \in \mathbb{R}^{N_K\times d}$; key-block sizes $\{n_v^K\}_{v=1}^{N_K}$
Ensure: Block importance $\widehat{\mathbf{W}}^{\mathrm{PAMA}} \in \mathbb{R}^{N_Q\times N_K}$

Initialize $\widehat{\mathbf{W}}^{\mathrm{PAMA}} \leftarrow \mathbf{0}$ in HBM
for all query blocks $u = 1, \ldots, N_Q$ in parallel **do**
  for each query tile $I \subseteq \{o_u, \ldots, o_{u+1} - 1\}$ **do**
    Load $\mathbf{Q}_{I,:}$
    $m_i \leftarrow -\infty$, $z_i \leftarrow 0$ ▷ Softmax states in registers
    for each key-centroid tile $J \subseteq \{1, \ldots, N_K\}$ **do** ▷ Pass 1
      $\mathbf{S} \leftarrow \mathbf{Q}_{I,:}\mathbf{C}_{J,:}^{\top}/\sqrt{d}$
      $S_{iv} \leftarrow S_{iv} + \log n_v^K$
      $m_i' \leftarrow \max(m_i, \max_{v\in J} S_{iv})$
      $z_i \leftarrow z_i \exp(m_i - m_i') + \sum_{v\in J} \exp(S_{iv} - m_i')$
      $m_i \leftarrow m_i'$
    end for
    $\ell_i \leftarrow m_i + \log z_i$ ▷ row-wise LogSumExp in registers
    for each key-centroid tile $J \subseteq \{1, \ldots, N_K\}$ **do** ▷ Pass 2
      Recompute $\mathbf{S} \leftarrow \mathbf{Q}_{I,:}\mathbf{C}_{J,:}^{\top}/\sqrt{d}$
      $S_{iv} \leftarrow S_{iv} + \log n_v^K$
      $P_{iv} \leftarrow \exp(S_{iv} - \ell_i)$ ▷ in registers
      $\widehat{W}_{uv}^{\mathrm{PAMA}} \mathrel{+}= \sum_{i\in I} P_{iv}/(o_{u+1} - o_u)$ ▷ HBM accumulation
    end for
  end for
end for
return $\widehat{\mathbf{W}}^{\mathrm{PAMA}}$

The first pass computes row-wise LogSumExp through online updates over key-centroid tiles. The second recomputes the logits, normalizes each tile, and accumulates block importance. This approximately doubles QK-centroid GEMM work but keeps register usage independent of N_K for fixed tile sizes. Logits, probabilities, and LogSumExp values remain on chip rather than in HBM, allowing larger N_K than Full-K.

Table 6 compares PyTorch with the Two-pass and Full-K kernels in Triton (Tillet et al. (2019)) and TileLang (TileLang). Both benchmarks use batch size 1, $N_Q = 256$, $N_K = 1{,}024$, and $d = 128$, with $(H, L) = (24, 115{,}200)$ for HunyuanVideo and $(40, 75{,}600)$ for Wan2.1-14B.

Additional full-model timing shows that Triton Full-K completes PAMA block-importance estimation across all layers in less than 1 s on both models. Four such block-importance recomputations during video generation account for less than 0.21% of the end-to-end latencies in Table 2.

Table 6: PAMA runtime and additional GPU memory. Speedup, memory reduction (multiplicative factor), and memory saving are relative to PyTorch.

Model	Implementation	Runtime		Additional GPU Memory		
		Latency ↓	Speedup ↑	Peak ↓	Reduction ↑	Saving ↑
HunyuanVideo	PyTorch	87.95ms	1.00×	21.09 GB	1.00×	0.00%
	Triton (Two-pass)	28.12ms	3.13×	24.12 MB	895.60×	99.89%
	Triton (Full-K)	15.82ms	5.56×	24.17 MB	893.86×	99.89%
	TileLang (Two-pass)	21.35ms	4.12×	24.12 MB	895.60×	99.89%
	TileLang (Full-K)	21.08ms	4.17×	24.17 MB	893.86×	99.89%
Wan2.1-14B	PyTorch	92.81ms	1.00×	23.07 GB	1.00×	0.00%
	Triton (Two-pass)	31.35ms	2.96×	40.20 MB	587.71×	99.83%
	Triton (Full-K)	17.72ms	5.24×	40.27 MB	586.57×	99.83%
	TileLang (Two-pass)	23.82ms	3.90×	40.20 MB	587.71×	99.83%
	TileLang (Full-K)	23.22ms	4.00×	40.27 MB	586.57×	99.83%

F QRKC PSEUDOCODE

Algorithm 3 summarizes QRKC for one attention head. The transformed keys are used only to determine the cluster assignments. The labels are then applied to the original keys and values, and PAMA uses centroids computed in the original key space.

Algorithm 3 Query-Response Key Clustering (QRKC)

Require: Queries $\mathbf{Q} \in \mathbb{R}^{L\times d}$; keys $\mathbf{K} \in \mathbb{R}^{L\times d}$; values $\mathbf{V} \in \mathbb{R}^{L\times d}$; number of key clusters N_K; K-means iterations T; shrinkage α; stability constant ϵ

Ensure: Key labels $\mathbf{z}$; permutation $\boldsymbol{\pi}_K$; reordered keys $\mathbf{K}'$ and values $\mathbf{V}'$; original-space centroids $\{\mathbf{c}_v\}_{v=1}^{N_K}$; cluster sizes $\{n_v^K\}_{v=1}^{N_K}$

$\mathbf{G}_Q \leftarrow \mathbf{Q}^\top\mathbf{Q}/L$

$\mathbf{G}_Q \leftarrow (\mathbf{G}_Q + \mathbf{G}_Q^\top)/2$ ▷ enforce numerical symmetry

$s_Q \leftarrow \max\big(\mathrm{tr}(\mathbf{G}_Q)/d, 10^{-12}\big)$

$\overline{\mathbf{G}}_Q \leftarrow (1-\alpha)\mathbf{G}_Q + \alpha s_Q\mathbf{I} + \epsilon s_Q\mathbf{I}$

$\mathbf{R} \leftarrow \mathrm{Cholesky}(\overline{\mathbf{G}}_Q)$ ▷ $\overline{\mathbf{G}}_Q = \mathbf{R}\mathbf{R}^\top$

$\widetilde{\mathbf{K}} \leftarrow \mathbf{K}\mathbf{R}$ ▷ query-conditioned key features

Initialize transformed centroids $\{\widetilde{\boldsymbol{\mu}}_v\}_{v=1}^{N_K}$ from $\widetilde{\mathbf{K}}$

for $t = 1, \dots, T$ **do**

  for all keys $j = 1, \dots, L$ in parallel **do**

    $z_j \leftarrow \arg\min_{v\in\{1,\dots,N_K\}} \|\widetilde{\mathbf{k}}_j - \widetilde{\boldsymbol{\mu}}_v\|_2^2$

  end for

  for all clusters $v = 1, \dots, N_K$ in parallel **do**

    $\widetilde{\boldsymbol{\mu}}_v \leftarrow \mathrm{mean}\{\widetilde{\mathbf{k}}_j : z_j = v\}$

  end for

end for

$\boldsymbol{\pi}_K \leftarrow \mathrm{argsort}(\mathbf{z})$

$\mathbf{K}' \leftarrow \mathbf{K}[\boldsymbol{\pi}_K, :]$, $\mathbf{V}' \leftarrow \mathbf{V}[\boldsymbol{\pi}_K, :]$

for all clusters $v = 1, \dots, N_K$ in parallel **do**

  $\mathcal{K}_v \leftarrow \{j : z_j = v\}$, $n_v^K \leftarrow |\mathcal{K}_v|$

  $\mathbf{c}_v \leftarrow (n_v^K)^{-1}\sum_{j\in\mathcal{K}_v} \mathbf{k}_j$ ▷ centroid in the original key space

end for

return $\mathbf{z}, \boldsymbol{\pi}_K, \mathbf{K}', \mathbf{V}', \{\mathbf{c}_v, n_v^K\}_{v=1}^{N_K}$

Because $\widetilde{\mathbf{K}} = \mathbf{K}\mathbf{R}$, Euclidean K-means in the transformed space minimizes the stabilized QRKC objective in Eq. (71). The transformation does not change the feature dimension, and no transformed values are required. Thus, K-means retains the same computational complexity for the same token count, cluster count, and number of iterations.

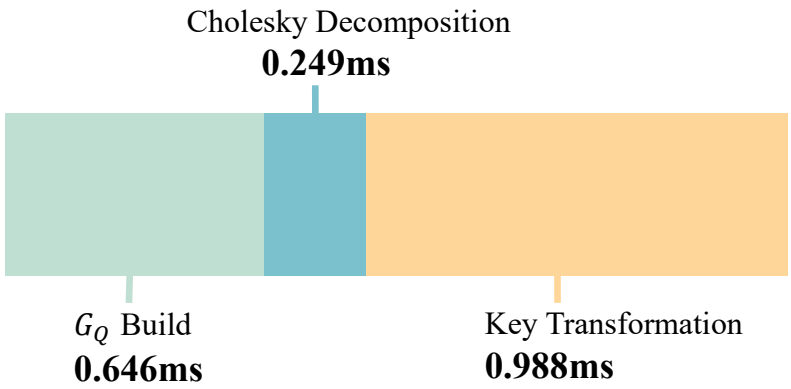


Figure 6: Time breakdown of QRKC overhead excluding K-means clustering. The total measured overhead remains below 2 ms. We use the same experimental settings as in Table 6.

G Sensitivity to Query and Key Centroid Counts

We analyze the sensitivity of PARK to the numbers of query and key centroids, denoted by N_Q and N_K, respectively.

Table 7: Sensitivity to the numbers of query and key centroids on HunyuanVideo.

N_Q	N_K	PSNR ↑	SSIM ↑	LPIPS ↓	Latency ↓	Speedup ↑
64	1,024	29.38	0.926	0.175	1817s	1.68×
128	1,024	29.85	0.931	0.168	1827s	1.67×
192	1,024	29.94	0.933	0.167	1841s	1.66×
256	1,024	30.13	0.934	0.164	1855s	1.64×
320	1,024	30.13	0.935	0.162	1875s	1.63×
128	512	29.56	0.928	0.174	1830s	1.67×
128	768	29.74	0.931	0.170	1834s	1.66×

Table 8: Sensitivity to the numbers of query and key centroids on Wan2.1-1.3B.

N_Q	N_K	PSNR ↑	SSIM ↑	LPIPS ↓	Latency ↓	Speedup ↑
64	512	27.30	0.903	0.183	410s	1.82×
128	512	27.52	0.908	0.177	423s	1.76×
192	512	27.66	0.911	0.174	427s	1.74×
256	512	27.71	0.911	0.173	430s	1.73×
320	512	27.79	0.913	0.172	440s	1.69×
128	768	27.67	0.910	0.175	416s	1.79×
128	1024	27.72	0.912	0.173	420s	1.77×

H Clustering and Sparse-Mask Reuse

We evaluate the effect of the recomputation schedule on HunyuanVideo with 50 denoising steps. Let $\mathcal{T}$ denote the set of denoising step indices at which clustering results and sparse masks are computed or recomputed. Both are reused during subsequent sparse-attention steps until the next recomputation. Table 9 compares four schedules in terms of generation quality and inference efficiency.

Table 9: Effect of clustering and sparse-mask recomputation schedules on HunyuanVideo with 50 denoising steps. Speedup is measured relative to dense attention.

$\mathcal{T}$	PSNR ↑	SSIM ↑	LPIPS ↓	Latency ↓	Speedup ↑
$\{10\}$	29.55	0.928	0.174	1829s	1.67×
$\{10, 30\}$	29.67	0.930	0.171	1819s	1.68×
$\{10, 20, 30\}$	29.84	0.932	0.168	1819s	1.68×
$\{10, 20, 30, 40\}$	29.85	0.931	0.168	1827s	1.67×

As shown in Table 9, increasing the number of recomputations generally improves similarity to dense attention, while three and four recomputations yield nearly identical results. All four schedules achieve similar end-to-end latency. These results support reusing clustering results and sparse masks across diffusion steps, with occasional recomputation to preserve generation quality.

I EFFICIENCY-QUALITY TRADE-OFF STUDY

We study the efficiency-quality trade-off of PARK under different sparsity settings on a subset of VBench prompts using HunyuanVideo, Wan2.1-1.3B, and Wan2.1-14B. As shown in Figure 7, although similarity to dense attention decreases as latency is reduced, PARK maintains high generation quality across the tested sparsity settings, as reflected by its relatively stable VBench-Q scores.

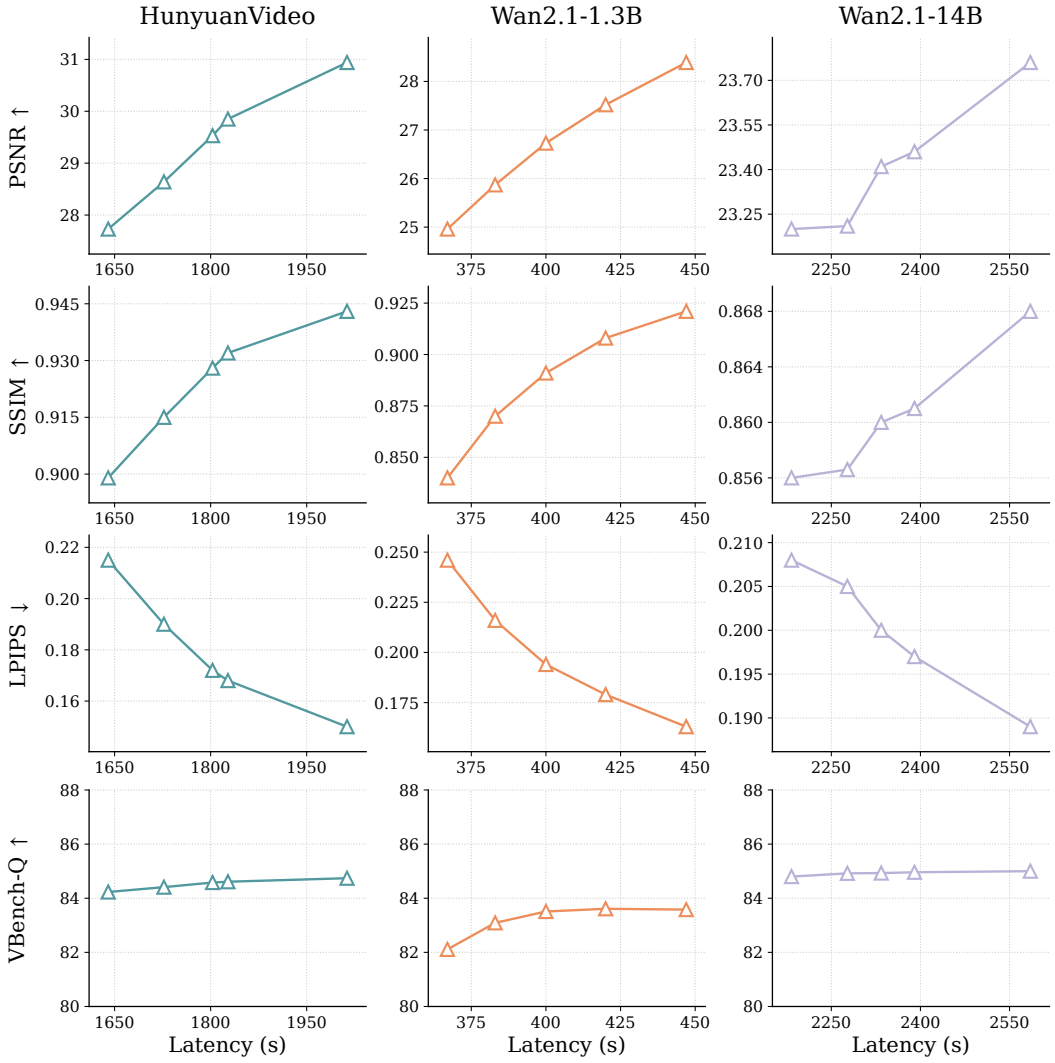


Figure 7: Efficiency-quality trade-off of PARK.

J Query and Key Compression under Top-k Retrieval

We extend the query/key compression comparison in Table 1 to Top-k retrieval with a fixed retrieval ratio ρ. For each query block, we retrieve the top ρ fraction of key blocks in descending order of estimated importance. We set $\rho = 0.1$, corresponding to approximately 10% of the key blocks. Table 10 reports attention recall under clustered and original token orderings, with full-query and full-key estimation serving as the reference.

Table 10: Top-k block-retrieval attention recall at a fixed retrieval ratio ($\rho = 0.1$) under different query/key representations. Mean, P05, and Worst denote the mean, fifth percentile, and minimum attention recall across all attention heads and layers, respectively. All values are percentages. Higher is better.

Ordering	Query	Key	HunyuanVideo			Wan 2.1-14B		
			Mean	P05	Worst	Mean	P05	Worst
Clustered	Full	Full	77.05	27.41	18.97	73.48	32.81	19.66
	Centroid	Centroid	75.37	27.27	18.96	72.35	32.78	19.66
	Centroid	Full	76.14	27.38	18.96	72.88	32.80	19.66
	Full	Centroid	**76.50**	27.29	18.96	**73.12**	32.79	19.66
Original	Full	Full	61.90	18.93	12.73	56.49	21.17	10.51
	Centroid	Centroid	56.78	17.56	08.46	54.78	20.52	10.51
	Centroid	Full	56.91	18.91	12.72	53.17	21.13	10.51
	Full	Centroid	**56.94**	17.49	08.59	**54.84**	20.66	10.51

K Limitations

This work focuses on accelerating video generation with diffusion models. Our evaluation is limited to this setting and does not establish the applicability of PARK to text-based large language models or other multimodal tasks. These settings may exhibit different attention patterns and inference workflows, which may require adaptations to PARK. Further research is needed to determine whether PARK can provide similar efficiency benefits in these settings while preserving output quality.

L VISUALIZATION OF THE GENERATED VIDEOS

We provide visual comparisons between PARK and Dense Attention on HunyuanVideo, Wan2.1-14B, Wan2.1-1.3B, and Wan2.2-14B. The results in Figures 8, 9, 10, and 11 show that PARK preserves high pixel-level fidelity and achieves generation quality similar to that of dense attention. More video samples are provided in the supplementary materials.

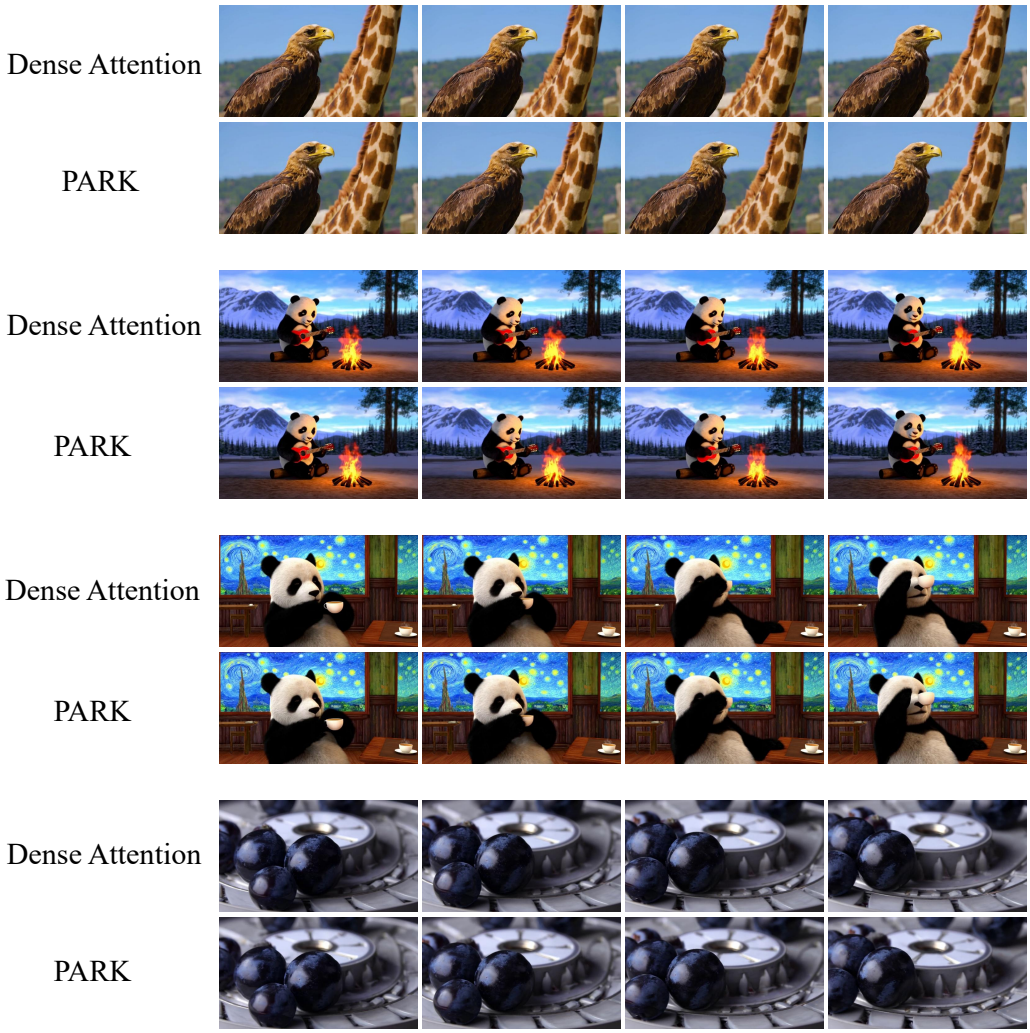


Figure 8: Comparison of Dense Attention and PARK for text-to-video generation with HunyuanVideo.



Figure 9: Comparison of Dense Attention and PARK for text-to-video generation with Wan2.1-14B.

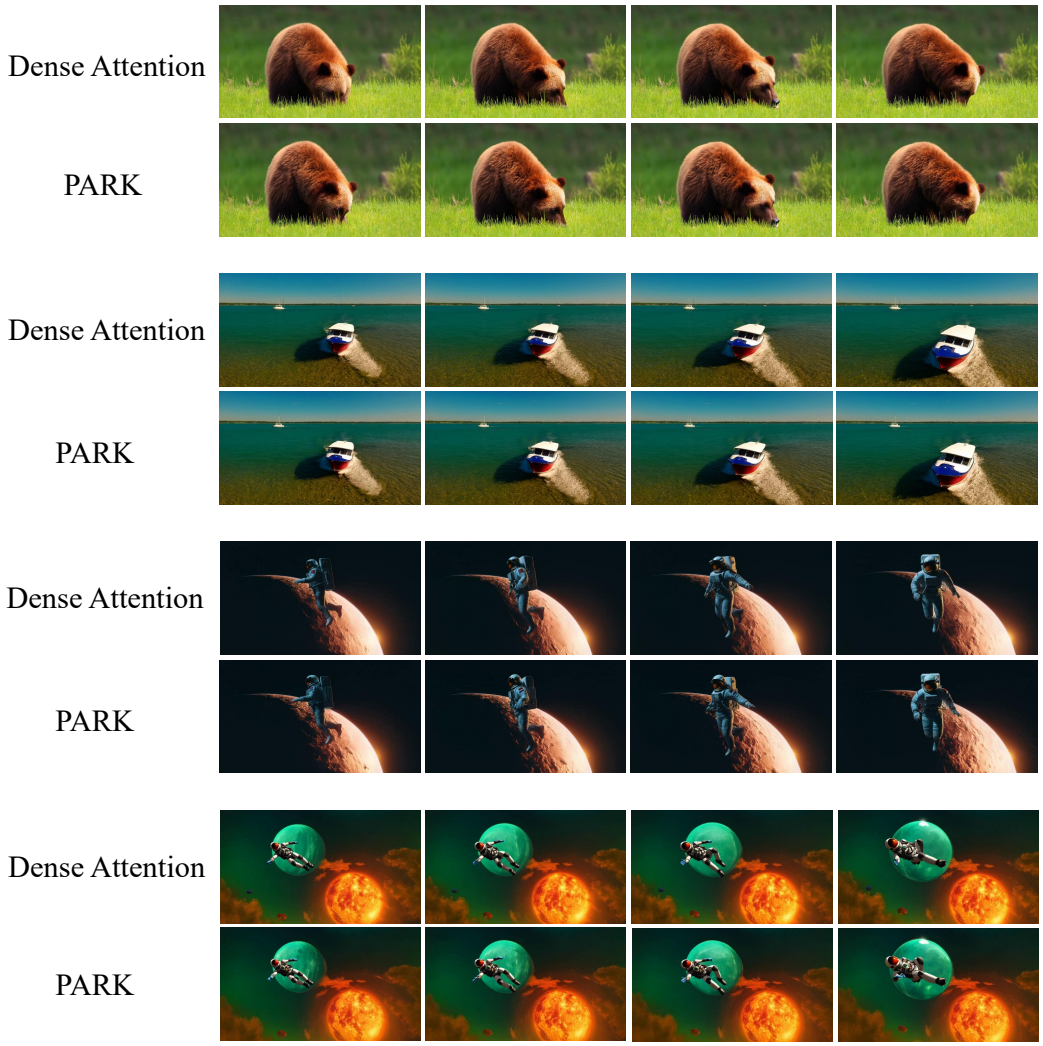


Figure 10: Comparison of Dense Attention and PARK for text-to-video generation with Wan2.1-1.3B.



Figure 11: Comparison of Dense Attention and PARK for text-to-video generation with Wan2.2-14B.