AQUA: Revisiting Attention Approximation through Spectral Alignment and Query-Adaptive Feature Selection

Anonymous authors
Paper under double-blind review

Abstract

As Large Language Models (LLMs) scale to longer contexts, the high dimensionality of the attention mechanism imposes increasing computational and representational overhead. Prior works have explored approximating attention scores to enable efficient token selection, typically using orthogonal spectral transformations or magnitude-based heuristics, followed by recomputation of exact attention on a reduced set of tokens. In this work, we take a complementary perspective and investigate whether such approximations can be used directly for attention computation. We introduce **AQUA** (**A**ttention via **Q**uery-Adaptive Spectral S**U**bspace **A**pproximation), a framework that combines offline spectral projection with online, query-dependent magnitude-based dimension selection, drawing on complementary insights from prior approximation approaches. By rotating representations into a calibrated basis and dynamically selecting salient dimensions, AQUA approximates attention scores using only a dynamically selected subset of rotated features. Our analysis shows that attention representations exhibit substantial spectral redundancy: retaining only 75% of dimensions preserves near-baseline accuracy across a wide range of benchmarks on models such as Llama-3.1-8B. These results suggest that attention representations exhibit substantial compressibility under calibrated spectral projections, and that accurate attention scores can be recovered from a reduced subspace through a combination of spectral alignment and dynamic feature selection.

1 Introduction

The capabilities of Large Language Models (LLMs) have scaled rapidly, enabling them to process and synthesize increasingly long contexts (Touvron et al., 2023; OpenAI et al., 2024). However, this capability comes at a significant computational and memory cost. The auto-regressive nature of the Transformer's attention mechanism (Vaswani et al., 2023) requires attending to the entire history of past tokens at every step, leading to a bottleneck that scales linearly with sequence length. This not only increases the number of computations but also incurs substantial memory overhead due to the growing KV cache, and places heavy pressure on memory bandwidth as past activations must be repeatedly fetched during inference (Pope et al., 2022; Keles et al., 2022).

To mitigate these challenges, a wide range of optimization strategies have been proposed, each targeting different aspects of the attention bottleneck. Some approaches focus on reducing the memory footprint of the KV cache, including quantization-based methods (Hooper et al., 2025; Dong et al., 2024; Liu et al., 2024), token eviction strategies (Zhang et al., 2023) and low-dimensional caching techniques (Saxena et al., 2024; Xu et al., 2025b; S et al., 2025), while others aim to improve memory bandwidth efficiency by reducing the amount of data transferred during attention computation (Singhania et al., 2024; Ribar et al., 2024; Xu et al., 2025a).

Among methods that aim to optimize the memory bandwidth, we highlight two representative approaches that our work builds upon: **LoKi** (Singhania et al., 2024) and **SparQ** (Ribar et al., 2024). LoKi approximates attention score by computing an offline SVD over only key activations, projecting attention representations into this spectral basis, and retaining the leading principal components, for efficient attention score estimation.

SparQ approximates attention scores by exploiting the heavy-tailed distribution of query activations, dynamically selecting the largest-magnitude query dimensions, and computing attention using only the corresponding subset of key dimensions. In both cases, these reduced representations are primarily used as an intermediate step to estimate attention scores for token selection, followed by recomputation of exact attention on a reduced subset.

Despite their differences, both approximation methods use compressed attention scores only as intermediate signals for token selection before reverting to exact full-dimensional attention computation. Consequently, it remains unclear whether the approximated representations themselves are sufficiently expressive to support direct attention computation. Like these approaches, AQUA specifically targets the attention score computation stage. The softmax normalization and value aggregation operations remain unchanged.

In this work, we build upon this class of representation-level approximations for attention score computation and investigate a complementary question:

> *To what extent can attention be computed directly in an approximated subspace, without reverting to full-dimensional recomputation?*

Studying this setting provides a principled way to probe the redundancy and sparsity inherent in attention representations. We observe empirically that effective approximation requires both **spectral alignment**, which concentrates information into a structured basis, and **magnitude-based selection** of projected features, which adaptively retrieves the most relevant components for each query. While LoKi captures the former through projection and SparQ captures the latter through dynamic selection, we study their combination for direct attention computation, forming the basis of our approach.

Our contributions are summarized as follows:

- We introduce **AQUA**, a framework that combines spectral alignment with query-dependent magnitude selection to enable *direct* attention score approximation in a reduced subspace, without requiring full-dimensional recomputation.
- We show that neither spectral alignment nor magnitude-based selection alone is sufficient for high-fidelity attention score approximation. Their combination produces attention scores that are sufficiently faithful to support direct attention computation.
- We develop an efficient implementation of AQUA using custom **Triton** kernels (Tillet et al., 2019), preserving sparse computation throughout the decode path and enabling practical deployment on modern GPU architectures.
- We provide empirical evidence that attention representations exhibit a structured combination of global spectral regularity and token-specific sparsity. Leveraging this structure, AQUA maintains strong performance across diverse benchmarks on models such as `Llama-3.1-8B` and `OLMoE-1B-7B`.

2 Preliminaries: Standard Attention & Notations

Transformer inference operates in two distinct phases: **pre-filling**, where the entire prompt is processed in parallel to populate the initial cache, and **auto-regressive generation**, where tokens are generated sequentially. While the pre-filling phase processes the initial prompt in parallel, the generation phase faces the escalating challenge of attending to a continuously growing context history. This work specifically focuses on optimizing this generation regime.

2.1 Core Components and Notations

For clarity, all formulations in this section are presented for a single attention head within a single Transformer layer unless otherwise stated. We define the primary dimensions and learnable weights as follows:

- d_{model}: The model embedding dimension.
- d_{head}: The dimensionality of each query, key, and value head.

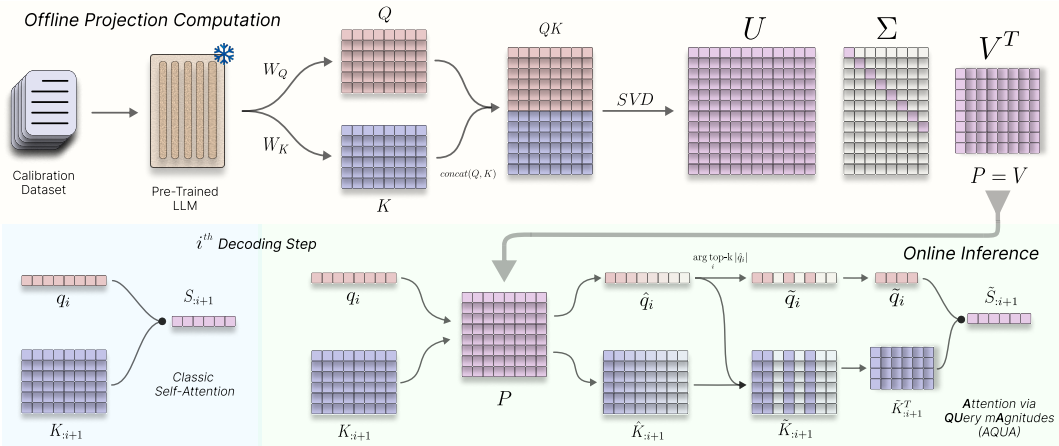

Figure 1: **Schematic Overview of the AQUA Framework.** The method operates in two phases: **(Top) Offline Calibration:** Query and Key activations from a calibration dataset are concatenated and decomposed via SVD to derive a shared projection matrix P. **(Bottom) Online Inference:** Standard attention is optimized by rotating inputs into this calibrated basis. Dimensions are dynamically selected based on the query's magnitude ($|\hat{q}_i|$), allowing attention scores to be computed on a sparse subset of salient features.

- H_q: The number of query attention heads.
- H_{kv}: The number of key-value heads. In standard multi-head attention (MHA), each query head has a dedicated key-value head, so $H_q = H_{kv}$. In Grouped-Query Attention (GQA), multiple query heads share the same key-value head, resulting in $H_{kv} < H_q$. Specifically, the H_q query heads are partitioned into groups, where all query heads within a group attend using a shared set of key and value representations.
- $W_Q \in \mathbb{R}^{d_{model} \times (H_q \cdot d_{head})}$: The query projection matrix, producing H_q query heads.
- $W_K, W_V \in \mathbb{R}^{d_{model} \times (H_{kv} \cdot d_{head})}$: The key and value projection matrices, producing H_{kv} shared key-value heads.

At decoding step i (where $0 \dots i-1$ are past tokens), the input embedding is $x_i \in \mathbb{R}^{1 \times d_{model}}$. This input is projected into the current vectors as $q_i = x_i W_Q$, $k_i = x_i W_K$, and $v_i = x_i W_V$. To enable efficient generation, the model maintains a Key-Value (KV) Cache containing vectors from all previous steps, denoted as $K_{:i}, V_{:i} \in \mathbb{R}^{i \times d_{head}}$.

2.2 The Auto-Regressive Attention Step

For the current token i, the attention output is computed via the following sequence:

1. **KV-Cache Update:** The current key and value are appended to the cache such that $K_{:i+1} = \text{concat}(K_{:i}, k_i)$ and $V_{:i+1} = \text{concat}(V_{:i}, v_i)$. Both resulting matrices are in $\mathbb{R}^{(i+1) \times d_{head}}$.

2. **Attention Score Computation:** The query q_i is compared against all keys in the updated cache to compute scores $S = q_i K_{:i+1}^T \in \mathbb{R}^{1 \times (i+1)}$. This operation has a time complexity of $O((i+1) \cdot d_{head})$.

3. **Scaling and Normalization:** The raw scores are scaled and normalized to produce the attention weights $A = \text{softmax}\left(S/\sqrt{d_{head}}\right) \in \mathbb{R}^{1 \times (i+1)}$.

4. **Context Aggregation:** The weights A aggregate values from the cache to produce the final context vector $c_i = A V_{:i+1} \in \mathbb{R}^{1 \times d_{head}}$.

3 Methodology: The AQUA Framework

The core hypothesis of AQUA is that while the attention mechanism operates in a high-dimensional space, the salient information required for accurate token generation is sparse and concentrated. However, this

Algorithm 1 AQUA Inference Step (for token i+1)

Input: Current query $q_i \in \mathbb{R}^{1\times d_{\text{head}}}$, key $k_i \in \mathbb{R}^{1\times d_{\text{head}}}$, projected key cache $\hat{K}_{:i} \in \mathbb{R}^{i\times d_{\text{head}}}$
Parameters: Retention ratio $k_{ratio} \in (0, 1]$, Projection matrix $P \in \mathbb{R}^{d_{\text{head}}\times d_{\text{head}}}$
Output: Approximate attention scores $\tilde{S} \in \mathbb{R}^{1\times(i+1)}$

$r \leftarrow \lfloor k_{ratio} \cdot d_{head} \rfloor$ ▷ Determine active dimensions to retain
$\hat{q}_i \leftarrow q_i P$ ▷ Project query (Rotation)
$\hat{k}_i \leftarrow k_i P$ ▷ Project key (Rotation)
$\hat{K}_{:i+1} \leftarrow \text{concat}(\hat{K}_{:i}, \hat{k}_i)$ ▷ Update projected key cache
$v_{\text{mag}} \leftarrow |\hat{q}_i|$ ▷ Compute element-wise magnitude
$I_{\text{topk}} \leftarrow \arg\text{TopK}(v_{\text{mag}}, r)$ ▷ Identify indices of top-k magnitudes
$\tilde{q}_i \leftarrow \hat{q}_i[:, I_{\text{topk}}]$ ▷ Slice query to sparse subset
$\tilde{K}_{:i+1} \leftarrow \hat{K}_{:i+1}[:, I_{\text{topk}}]$ ▷ Slice key cache to sparse subset
$\tilde{S} \leftarrow \tilde{q}_i \tilde{K}_{:i+1}^{\top}$ ▷ Compute sparse attention scores
return $\tilde{S}$

sparsity is not aligned with the standard coordinate axes. To exploit this, AQUA introduces a two-phase framework, illustrated in Figure 1:

1. **Phase I (Offline Calibration):** We compute a fixed offline calibrated projection matrix P that rotates the query and key vectors into a new coordinate system where dimensions are ordered according to decreasing singular value magnitude, corresponding to directions of maximal variance.

2. **Phase II (Online Inference):** During decoding, we project incoming vectors into this new basis and dynamically select a sparse subset of dimensions based on the query's magnitude, computing attention scores only on this "heavy hitter" subset.

3.1 Phase I: Offline Projection Calibration

The objective of this phase is to find a linear transformation that aligns the vector space such that information is maximally concentrated. While an ideal projection would be computed via Singular Value Decomposition (SVD) on the live activation history ($Q_{:i}/q_i$, $K_{:i}$) at every step, this is computationally prohibitive ($O(d_{head}^3)$). Instead, we compute a single, fixed projection matrix P offline on a different calibration dataset.

3.1.1 Calibration Procedure and GQA Extension

For each attention head in the model, we perform the following one-time calibration procedure:

1. **Data Collection:** We run the pre-trained model on a representative calibration dataset (e.g., `BookCorpus` (Zhu et al., 2015)) and collect query and key activations.

2. **Global SVD:** We construct a calibration matrix D_{calib} from the collected activations and compute its singular value decomposition: $D_{\text{calib}} = U\Sigma V^T$(Klema & Laub, 1980).

3. **Projection Extraction:** The matrix $V \in \mathbb{R}^{d_{\text{head}}\times d_{\text{head}}}$ contains the principal directions of the attention subspace. We use this orthogonal basis as the fixed projection matrix $P = V$.

Extension to Grouped-Query Attention (GQA). Modern architectures such as `Llama-3.1-8B-Instruct` employ Grouped-Query Attention (GQA) (Grattafiori et al., 2024), where each key-value head is shared by H/H_{kv} query heads. To derive a shared projection matrix for each GQA group, we aggregate activations from all associated query heads and the shared key head into a single calibration matrix: $D_{\text{calib}}^{\text{GQA}} = \begin{pmatrix} D_{q_1} & D_{q_2} & \cdots & D_{q_{H/H_{kv}}} & D_k \end{pmatrix}^T \in \mathbb{R}^{\left(\left(\frac{H}{H_{kv}}+1\right)M\right)\times d_{\text{head}}}$, where M denotes the number of collected activation samples per head. Performing SVD on this combined matrix yields a single shared projection matrix P for the entire GQA group.

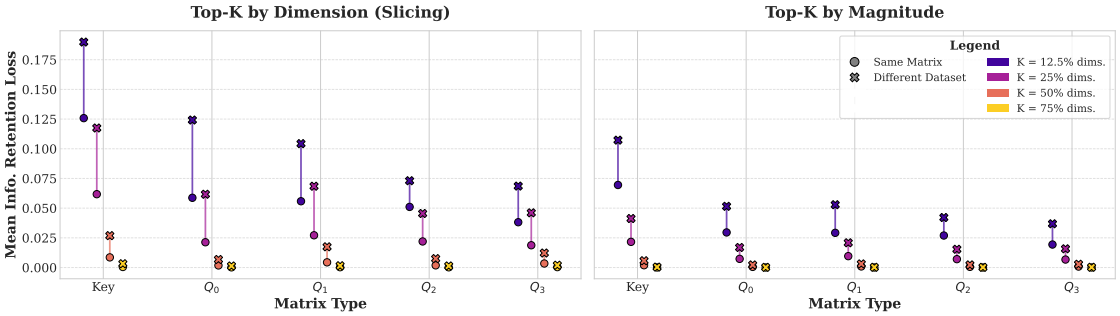

Figure 2: **Validation of AQUA's Design Choices**. We compare the Information Retention Loss (lower is better) on Layer 0, Head 0 of Llama-3.1-8B under different conditions. The horizontal axis is the retention ratio expressed as a percentage, denoted by K (i.e., $K = 100 \times k_{ratio}$).**Dynamic Magnitude vs. Static Slicing (Right vs. Left Panel):** The right panel uses our magnitude-based selection, while the left uses naive slicing (retaining the first r dimensions). The substantially lower loss on the right demonstrates that selecting dimensions by magnitude preserves significantly more information.**Offline vs. Online Calibration (Crosses vs. Circles):** We compare the ideal "Online" projection computed on the test data ($\circ$) against our proposed "Offline" projection calibrated on a different dataset ($\times$). The small gap between the two curves, particularly at larger values of K, indicates that the offline calibration captures much of the dominant spectral structure and serves as an effective approximation to the computationally expensive online decomposition.

3.2 Phase II: Online Inference Algorithm

During inference, AQUA replaces the standard attention score computation stage with an approximation operating in a rotated and sparsified basis. We introduce a controllable hyperparameter, the **Retention Ratio** ($k_{ratio} \in (0, 1]$), which determines the aggressiveness of the approximation. The exact number of dimensions retained is calculated as $r = \lfloor k_{ratio} \cdot d_{head} \rfloor$. The bottom panel of Figure 1 contrasts our method with classic self-attention. Instead of computing attention on full vectors, AQUA projects inputs into the calibrated space and computes scores using only the most salient dimensions defined by r. Algorithm 1 outlines the exact flow. First, the integer budget r is derived from k_{ratio} (Line 1). Query and key vectors are projected via matrix P (Lines 2-3), a **lossless rotation** that to an extent separates "noise" from "signal". Crucially, we dynamically identify active features by computing the element-wise magnitude of the projected query, $|\hat{q}_i|$ (Line 5). We select the indices I_{topk} with the largest values with respect to $|\hat{q}_i|$ (Line 6) and slice both the query and key cache to retain only these dimensions (Lines 7-8). Attention scores are then computed in this reduced space (Line 9), lowering dot product complexity from $O(L \cdot d_{head})$ to $O(L \cdot r)$ and improving scaling efficiency. The resulting approximate attention scores are then passed through the standard softmax normalization and value aggregation steps without modification.

Rotational Invariance: A property of our method is that the projection P is orthogonal. Thus, the projection step is a **lossless rotation** that preserves dot products: $\hat{q} \cdot \hat{k}^T = q \cdot k^T$. The approximation error arises solely from the subsequent pruning step (Lines 6-7). (See Appendix A.6 for the formal lemma).

3.2.1 Justification for Magnitude-Based Selection

A critical design choice in AQUA is *how* to select the subset of dimensions to retain. A naive approach, used in prior works like LoKi, is **Static Slicing**: simply keeping the first r dimensions, as these correspond to the largest singular values (global variance). We demonstrate that consistently selecting the same lower-index dimensions for all tokens is suboptimal because informative attention features vary across tokens and contexts.

The Flaw in Naive Slicing. PCA identifies dimensions that are important *globally* by capturing the most variance across an entire dataset. However, for any individual query, the most important dimensions are those that are most "active" for that specific token, which is best measured by their magnitude (Ribar et al., 2024). Our overlap analysis (Appendix A.5) confirms this mismatch: the dimensions with the highest magnitude for a specific query often do not align with the top principal components.

Reducing Information Loss. As shown in Figure 2, selecting dimensions dynamically based on $|\hat{q}_i|$ reduces **Information Retention Loss** by approximately 50% compared to static slicing. By adapting to the specific semantic context of the token, AQUA preserves the vector's information far more effectively than static strategies, providing a decisive advantage in preserving model accuracy.

3.3 Empirical Validation of Design Choices

The architectural decisions behind AQUA, specifically the use of an offline, shared projection matrix, are grounded in two critical empirical observations: the stability of principal components across datasets and their robustness across linguistically diverse inputs.

3.3.1 Validation of the Offline Approach

To validate that a fixed, offline matrix P is a sufficient proxy for the ideal online decomposition, we introduce the **Information Retention Loss** metric, $\mathcal{L}_{info}$ (Jolliffe & Cadima, 2016; Greenacre et al., 2022), $\mathcal{L}_{info}(v, \hat{v}, I_k) = \frac{\left|\|v\|_2 - \|\hat{v}[I_k]\|_2\right|}{\|v\|_2}$, where $\hat{v}[I_k]$ denotes the vector projected and truncated to the top-k indices. A lower $\mathcal{L}_{info}$ indicates better preservation of the original vector's information. Because attention scores are computed through inner products, preserving vector energy after projection and truncation provides a coarse proxy for preserving the magnitude structure that contributes to score computation.

3.3.2 The Stability of Offline Calibration

A central premise of AQUA is that attention representations exhibit a stable global spectral structure across diverse inputs. If this structure is sufficiently consistent, a projection matrix calibrated offline on a representative dataset should provide a useful approximation to the dominant variance directions encountered during inference. To evaluate this assumption, Figure 2 compares our proposed "Offline" projection (calibrated on `BookCorpus`) against an ideal "Oracle" projection computed by performing PCA online on the test data during inference.

As observed in Figure 2, there is a visible performance gap between the offline ($\times$) and online ($\circ$) methods, particularly at aggressive pruning ratios (e.g., $k_{ratio} = 12.5\%$). This indicates that the spectral structure exhibits some input dependence; input specificity does matter. However, the degradation incurred by using the frozen offline matrix remains limited and follows the same loss trajectory as the oracle. This suggests that a generic calibration successfully captures the dominant variance directions of the attention representations, even if it does not perfectly recover the input-specific optimum. This demonstrates that the offline projection provides a stable and effective basis in which the relevant structure is concentrated. AQUA's subsequent magnitude-based selection remains essential for identifying the dimensions that are most active for a particular query. Together, these observations support the use of a fixed offline projection as a computationally efficient substitute for expensive online decomposition while also motivating the need for dynamic feature selection during inference.

3.3.3 Cross-Lingual Generalizability

To further test whether our projection matrix overfits to specific linguistic patterns, we examined its performance on Hindi. `Llama-3.1-8B` officially supports eight languages. Among these, Hindi is unique because it uses the Devanagari script, whereas the other supported languages (e.g., English, German, French, Italian, and Spanish) primarily use the Latin script. This structural divergence makes Hindi an effective stress test for cross-lingual generalization. We applied the same projection matrix, calibrated exclusively on English text, to activations generated during inference from both English and Hindi inputs, using the `Wikipedia-Hindi` dataset for the Hindi evaluation[1]. As shown in Figure 3, the information retention loss profiles for English and Hindi are nearly indistinguishable across the full range of retention ratios. This suggests that the dominant variance directions captured by P generalize beyond surface-level lexical statistics (such as specific token embeddings), instead reflecting structural properties of the attention representations that transfer across languages.

[1] https://huggingface.co/datasets/zicsx/Wikipedia-Hindi

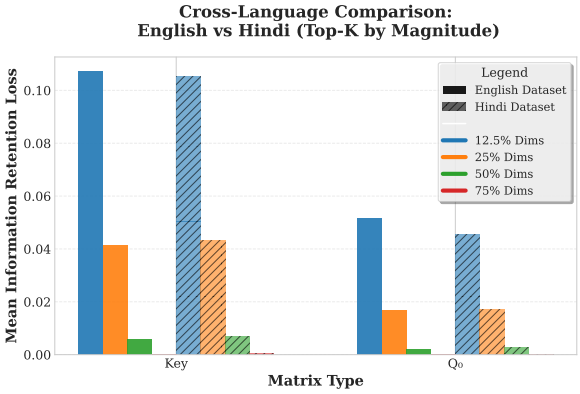

Figure 3: Cross-lingual comparison of mean information retention loss for the Key and first Query (Q_0) matrices. The similar loss profiles demonstrate the robustness of the English-calibrated projection matrix when applied to a different language and script.

3.3.4 Domain Specificity and Calibration Robustness

While our projection is language-agnostic, we distinguish this from *task specificity*. Language governs the surface form of the input, whereas the task influences which features and representations become important during inference. We hypothesized that for highly specialized domains, such as mathematical reasoning, a task-specific projection might yield better compression at lower dimensions. To investigate this, we conducted an experiment on the `GSM8K` benchmark. We calibrated a specific projection matrix using only the `GSM8K` training set and compared it against our generic `BookCorpus` matrix.

Table 1: Comparison of Generic (BookCorpus) vs. Domain-Specific (GSM8K) Projection on the GSM8K benchmark.

$\mathbf{k_{ratio}}$	**BookCorpus (Generic)**	**GSM8K (Specific)**
0.75	**0.804**	**0.803**
0.50	0.705	**0.760**
0.30	0.135	**0.371**

As detailed in Table 1, the generic calibration remains competitive with the domain-specific calibration at conservative ratios ($k_{ratio} = 0.75$) but significantly lags at aggressive ratios ($k_{ratio} = 0.30$: 13.5% vs 37.1%). This observation is consistent with the intuition that dominant high-variance directions encode broadly shared structural features, while lower-energy dimensions increasingly capture domain- or task-specific information. This result highlights that while high-variance dimensions are universal, the lower-variance "tail" dimensions may be task-specific.

3.4 Theoretical Efficiency Analysis

We formally analyze the computational complexity of AQUA's approximate attention score computation, focusing on the unnormalized attention score calculation (the dot product). Detailed derivations and discussions are provided in Appendix A.3.

- **AQUA Cost:** AQUA introduces a fixed projection overhead together with a reduced variable attention cost that scales with the retained dimension r: $C_{\text{AQUA}} = c_1 d_{head}^2 + c_2(i+1)r$, where c_1 and c_2 denote implementation-dependent constants associated with projection and approximate attention operations, respectively. Under the current AQUA formulation, this projection overhead remains explicit because the post-RoPE projection cannot generally be fused into the base query/key weights.
- **Standard Attention Cost:** The complexity of computing the dot product between the current query and all keys in the cache scales linearly with sequence length: $C_{\text{std}} = c_3(i+1)d_{head}$, where c_3 captures implementation-dependent constant factors associated with standard attention computation.

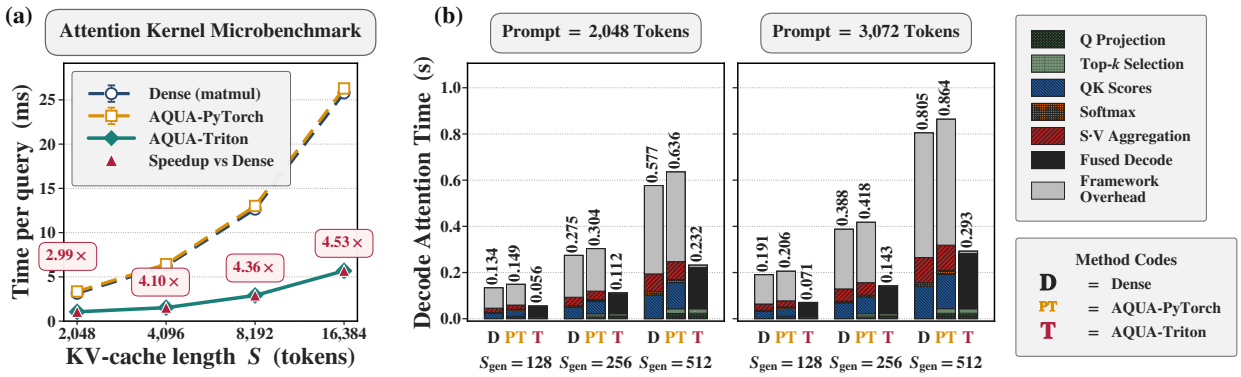

Figure 4: Decode-time view of the AQUA attention pipeline: **(a)** Kernel-level decode microbenchmark showing attention time per query as the KV-cache length S scales from 2,048 to 16,384 tokens, where the fused AQUA-Triton kernel achieves a 2.99× to 4.53× speedup over the Dense matmul baseline while the unfused AQUA-PyTorch path closely tracks Dense; **(b)** Cumulative decode-attention time decomposed into constituent operations for prompt lengths of 2,048 and 3,072 tokens at generation lengths $S_{\text{gen}} \in \{128, 256, 512\}$. AQUA-Triton (**T**) is dominated by a single fused decode kernel, which removes the per-component dispatch and framework overhead present in Dense (**D**) and AQUA-PyTorch (**PT**). As a result, AQUA-Triton achieves consistent decode-attention runtime reductions of approximately 2.4×–2.8× relative to the Dense baseline and 2.7×–3× relative to AQUA-PyTorch across all configurations.

The Break-Even Point: Comparing these asymptotic costs yields the condition under which AQUA becomes computationally favorable:

$$i + 1 > \frac{c_1 d_{head}^2}{c_3 d_{head} - c_2 k}. \tag{1}$$

This condition highlights the central trade-off in AQUA. As r decreases, the arithmetic cost of attention score computation is reduced more aggressively, allowing the fixed projection overhead to be amortized over shorter sequence lengths. Conversely, as r approaches d_{head}, the attainable savings diminish and the break-even point shifts toward longer contexts. Although the derivation incorporates explicit constant factors and therefore provides a more refined estimate than asymptotic analysis alone, it remains an arithmetic cost model. In practice, the realized performance depends not only on operation counts but also on how effectively the reduced-dimensional computation is preserved throughout the decode pipeline. Memory traffic, intermediate tensor materialization, and kernel-launch overhead can substantially impact runtime, motivating the implementation considerations discussed in the following section.

3.5 From Algorithmic Sparsity to Efficient Decode-Time Execution

The complexity analysis in Section 3.4 shows that AQUA reduces decode-time score computation from $O(L \cdot d_{\text{head}})$ to $O(d_{\text{head}}^2 + L \cdot k)$. Realizing these savings efficiently in practice, however, requires preserving the reduced-dimensional computation throughout the decode path. A naive implementation that projects queries, selects top-k dimensions, and subsequently invokes standard dense attention operators can still incur dense reductions and unnecessary memory movement, limiting the practical benefit of sparsity.

To address this, we implement AQUA using fused Triton kernels that maintain the sparse computation structure during decoding while avoiding dense intermediate representations. The projected key cache is stored in a transposed layout, $\hat{K}^T \in \mathbb{R}^{B \times H_{kv} \times d_{\text{head}} \times S}$, enabling efficient access to selected feature dimensions across long token histories. The implementation consists of three fused stages: (1) key projection and projected-cache update, (2) query projection and top-k selection, and (3) sparse score computation, softmax normalization, and weighted value aggregation. This design minimizes intermediate memory traffic and kernel-launch overhead while preserving the intended sparse execution pattern. Additional implementation details and kernel pseudocode are provided in Appendix A.7. The empirical behavior of the final AQUA

Table 2: A summary of performance for `Llama-3.1-8B-Instruct` and `OLMoE-1B-7B-Instruct` under key levels of AQUA pruning. 'B' denotes the baseline ($k_{ratio} = 1.0$), with results shown in **bold**. Results are reported as mean ± SE. See Appendix A.11 for the detailed formulation of Standard Error and a breakdown of error margins.

Model	k_{ratio}	MMLU (acc ↑)	GSM8K (acc ↑)	HellaSwag (acc ↑)	WinoGrande (acc ↑)	TruthfulQA MC2 (acc ↑)	ARC Challenge (acc ↑)	WikiText (ppl ↓)
Llama-3.1-8B Instruct	**B**	**0.687 ± 0.004**	**0.816 ± 0.011**	**0.785 ± 0.004**	**0.755 ± 0.012**	**0.551 ± 0.016**	**0.647 ± 0.014**	**8.910**
	0.75	0.685 ± 0.004	0.805 ± 0.011	0.785 ± 0.004	0.757 ± 0.012	0.551 ± 0.016	0.645 ± 0.014	8.930
	0.50	0.666 ± 0.004	0.720 ± 0.012	0.780 ± 0.004	0.738 ± 0.012	0.551 ± 0.016	0.620 ± 0.014	9.200
	0.30	0.507 ± 0.004	0.146 ± 0.010	0.732 ± 0.004	0.598 ± 0.014	0.502 ± 0.016	0.530 ± 0.015	12.550
OLMoE-1B-7B Instruct	**B**	**0.530 ± 0.004**	**0.451 ± 0.014**	**0.783 ± 0.004**	**0.673 ± 0.013**	**0.491 ± 0.016**	**0.532 ± 0.015**	**11.340**
	0.75	0.529 ± 0.004	0.453 ± 0.014	0.783 ± 0.004	0.669 ± 0.013	0.485 ± 0.016	0.542 ± 0.015	11.330
	0.50	0.526 ± 0.004	0.426 ± 0.014	0.778 ± 0.004	0.658 ± 0.013	0.488 ± 0.016	0.540 ± 0.015	11.340
	0.30	0.485 ± 0.004	0.252 ± 0.012	0.747 ± 0.004	0.615 ± 0.014	0.481 ± 0.016	0.513 ± 0.015	12.140

decode pipeline is summarized in Figure 4. Unless otherwise noted, experiments on this section are conducted on an H100 NVL in fp16 using the Llama-3.1-8B attention configuration ($H = 32$, $H_{kv} = 8$, $d_{\text{head}} = 128$) with the default retention ratio $k/d_{\text{head}} = 0.75$.

Figure 4 (a) reports the decode-time kernel microbenchmark as the KV-cache length S increases. The dense matmul baseline and the AQUA-PyTorch implementation closely track each other, indicating that a naive implementation does not effectively translate algorithmic sparsity into runtime savings. In contrast, the fused AQUA-Triton implementation consistently reduces latency across all sequence lengths by preserving sparse execution throughout the decode path. The speedup grows from 2.99× at $S = 2{,}048$ to 4.53× at $S = 16{,}384$, demonstrating that the benefits become more pronounced at longer contexts.

Figure 4 (b) breaks down cumulative decode-attention time into its constituent operations for different prompt and generation lengths. The Dense (**D**) and AQUA-PyTorch (**PT**) implementations spend a substantial fraction of runtime in fragmented framework overhead and separate attention stages. But, the AQUA-Triton (**T**) pipeline concentrates most computation into a single fused decode kernel, substantially reducing dispatch and intermediate memory overhead. Across all evaluated settings, the fused Triton path consistently achieves lower decode-attention time, typically reducing runtime by approximately 2.4× to 2.8× relative to the Dense and PyTorch implementations. Together, these results show that sparse computation alone is insufficient; the primary latency gains arise from preserving sparsity through kernel fusion and minimizing intermediate memory movement.

4 Empirical Evaluation and Results

To validate the effectiveness of AQUA, we conduct a comprehensive evaluation focusing on three key questions: (1) Does AQUA maintain model fidelity at reduced dimension? (2) Is our integrated projection and magnitude-based selection mechanism superior to prior primitives? (3) Does the method scale to long-context regimes?

4.1 Experimental Setup

We evaluate on `Llama-3.1-8B-Instruct` (Grattafiori et al., 2024) and `OLMoE-1B-7B-Instruct`. Our benchmark suite includes standard reasoning tasks (MMLU, GSM8K, ARC, HellaSwag) and the **LongBench** suite (Bai et al., 2024) for contexts up to 32k tokens. All evaluations use the `lm-evaluation-harness`[2]. Additional details regarding the specific configuration of these benchmarks and the full experimental parameters can be found in Appendix A.8 and A.9.

[2] https://github.com/EleutherAI/lm-evaluation-harness

Table 3: Comparison of approximation primitives on `Llama-3.1-8B` across multiple benchmarks. TopK Ratio indicates the retained attention proportion, with Baseline corresponding to $k_{ratio} = 1.0$. Higher is better for all metrics except WikiText (lower perplexity is better). Standard errors are omitted here for brevity but are consistent in magnitude with those reported in Table 2.

$\mathbf{k_{ratio}}$	Attention Type	MMLU (acc ↑)	GSM8K (acc ↑)	HellaSwag (acc ↑)	WinoGrande (acc ↑)	TruthfulQA MC2 (acc ↑)	ARC Challenge (acc ↑)	WikiText (ppl ↓)
1.0	Baseline	0.686	0.805	0.784	0.756	0.550	0.647	8.91
0.75	LoKi (Slicing)	0.548	0.398	0.720	0.723	0.491	0.549	11.82
	SparQ (No Proj)	0.680	0.789	0.783	0.764	0.546	0.645	8.94
	AQUA (Ours)	**0.684**	**0.803**	**0.784**	**0.754**	**0.550**	**0.646**	**8.93**
0.50	LoKi (Slicing)	0.279	0.022	0.559	0.557	0.477	0.328	20.43
	SparQ (No Proj)	0.640	0.666	0.776	0.730	0.551	0.622	9.60
	AQUA (Ours)	**0.664**	**0.705**	**0.780**	**0.738**	**0.550**	**0.616**	**9.20**
0.30	LoKi (Slicing)	0.229	0.001	0.304	0.505	0.504	0.239	144.31
	SparQ (No Proj)	0.280	0.021	0.460	0.522	0.470	0.303	37.73
	AQUA (Ours)	**0.507**	**0.135**	**0.731**	**0.607**	**0.500**	**0.529**	**12.55**

4.2 Standalone Performance: The Accuracy-Efficiency Trade-off

We first evaluate AQUA as a replacement for the standard attention score computation stage. Table 2 summarizes the performance across key benchmarks at varying retention ratios (k_{ratio}). The results reveal a clear "sweet spot". For Llama-3.1-8B, retaining 75% of projected dimensions ($k_{ratio} = 0.75$) results in statistically insignificant performance degradation across all tasks (e.g., MMLU drops only from 0.687 to 0.685). This confirms that roughly 25% of the attention computation is redundant and can be pruned based on magnitude without loss of fidelity. The OLMoE model exhibits even greater resilience, maintaining baseline performance even at 50% pruning, likely due to the natural sparsity of its independent key heads compared to GQA.

4.3 Comparative Analysis of Approximation Primitives

To isolate the efficacy of our *projected selection*, we compare it against the core approximation mechanisms of two related approaches:

- **PCA-Slicing (LoKi-style):** Uses the same SVD-based projection and statically retains the top-k principal components ranked by global variance. Unlike AQUA, the selected dimensions are fixed across all tokens and do not adapt to token-specific feature salience.
- **SparQ-style Selection:** Retains the original basis but selects dimensions dynamically based on query magnitude without projection.

Note: We evaluate these strictly as *attention score approximators* within our framework to ensure a fair, apples-to-apples comparison of the mathematical method, decoupling them from their respective token eviction policies.

As shown in Table 3, AQUA is superior to both baselines. PCA-Slicing degrades rapidly, confirming that global variance is a sub-optimal proxy for token-specific importance. SparQ collapses at aggressive ratios ($k_{ratio} = 0.3$), indicating the original basis is not sufficiently sparse. AQUA's combination of spectral alignment and dynamic selection retains significantly higher accuracy.

4.4 Long-Context Capabilities

To verify AQUA's robustness in long-context regimes, we evaluated it on the LongBench suite (Bai et al., 2024). Table 4 demonstrates exceptional stability. On retrieval-heavy tasks like PassageRetrieval, AQUA

Table 4: Performance on LongBench tasks (`Llama-3.1-8B`) under AQUA pruning. Tasks include Multi-News (MN), TREC Question Classification (TREC), Long-Context Classification (LCC), Passage Retrieval (PR), and SAMSum Dialogue Summarization (SS). k_{ratio} denotes the AQUA ratio, with baseline B corresponding to $k_{ratio} = 1.0$. Higher values indicate better performance. Representative Standard Errors (SE) across pruning ratios, for these tasks, are detailed in Appendix A.12

$\mathbf{k_{ratio}}$	MN	TREC	LCC	PR	SS
B	**0.269**	**0.725**	**0.626**	**0.967**	**0.427**
0.90	0.268	0.730	0.630	0.970	0.427
0.75	0.269	0.725	0.624	0.958	0.426
0.50	0.267	0.715	0.578	0.957	0.412
0.40	0.263	0.690	0.543	0.936	0.399

($k_{ratio} = 0.75$) scores 0.958 against the baseline's 0.967, confirming that the pruning does not discard the sparse, critical information for long contexts.

5 Discussion

The empirical results in Section 4 establish that a substantial fraction of the attention representation is redundant when viewed in a calibrated rotated basis. We close by discussing what this finding implies about attention representations themselves, why a single offline projection generalizes so broadly, and what the asymmetry between GQA and MHA models in our results suggests for future attention design.

5.1 Implications for attention representations

AQUA's effectiveness rests on two distinct properties of attention representations operating in tandem: *structural redundancy* captured by spectral concentration, and *per-token sparsity* captured by activation magnitude. The fact that both are required (Section 3.2.1) reveals that these are not equivalent notions of importance. The rotated basis identified by global SVD concentrates total variance, but Figure 6 from appendix shows that the dimensions which carry the most variance globally are not the dimensions most active for any particular query. Attention representations therefore exhibit a two-layer structure: a low-effective-rank global frame, within which per-token activity is heavy-tailed.

This decomposition connects to several recent observations about transformer internals. The heavy-tailed character of per-token activations parallels the "massive activations" phenomenon documented in feed-forward streams (Sun et al., 2024), suggesting that a small number of dimensions disproportionately carry information at any given step. The concentration of global variance into a low-dimensional subspace echoes findings on *privileged bases* in transformer activations (Elhage et al., 2022), where the standard coordinate axes are not arbitrary but are aligned with structurally meaningful directions. What AQUA establishes is that these two phenomena are simultaneously exploitable: a static rotation aligns the basis with the structural directions, and per-token magnitude selects within them. Neither alone is sufficient; their composition is what robustly enables direct attention computation in a reduced subspace.

5.2 Why offline calibration suffices

A natural concern with a fixed offline projection is that it cannot adapt to the statistical distribution of any particular input. Our cross-lingual results in Section 3.3 provide direct evidence that this concern is misplaced for the dominant variance directions. Applying a projection calibrated exclusively on English text (`BookCorpus`) to Hindi activations, written in a structurally unrelated script (Devanagari), yields information retention curves that are virtually indistinguishable from those on English (Figure 3). If the projection were tracking surface-level features of the calibration distribution, this cross-script transfer should fail.

We interpret this as evidence that the principal directions of attention heads encode language-agnostic structural regularities of the model rather than properties of the calibration data. The same does not extend, however, to the lower-variance tail: the GSM8K calibration experiment in Table 1 shows that task-specific projections outperform generic ones at aggressive retention ratios, indicating that the tail dimensions encode increasingly task-sensitive structure. The practical implication is a clear hierarchy. A single generic calibration matrix is sufficient for typical deployment at moderate ratios ($k_{ratio} \geq 0.5$). Aggressive compression regimes ($k_{ratio} \leq 0.3$) benefit from task-specific calibration, which remains a one-time offline cost. This makes AQUA's calibration overhead asymptotically negligible relative to inference, while still providing a tunable lever for specialized deployments.

5.3 Architectural asymmetry: GQA versus MHA

A striking pattern in our results is the differential resilience of the two models to dimensional pruning. `Llama-3.1-8B-Instruct` (GQA, group size 4) exhibits substantial degradation at $k_{ratio} = 0.5$, with GSM8K accuracy dropping from 0.816 to 0.720, while `OLMoE-1B-7B-Instruct` (MHA) maintains near-baseline performance at the same ratio. We attribute this asymmetry to the representational demands placed on the key vectors by each architecture.

In GQA, a single key-value head is shared across multiple query heads, requiring its representation to simultaneously support every query in its group. This forces a denser, more multiplexed encoding of information across dimensions, leaving less room for sparsity that magnitude-based selection can exploit. In MHA, each query head is paired with a dedicated key head, allowing each key vector to specialize toward a narrower subset of features and exhibit naturally higher per-token sparsity. AQUA's mechanism, which preserves only the largest-magnitude dimensions, therefore extracts more compressible signal from MHA representations than from GQA's information-dense shared keys. This observation suggests that the parameter savings of GQA come with a corresponding cost: the same compression that enables smaller KV-caches also reduces the headroom for downstream representation-level sparsity. Emerging attention variants that further share or compress key-value state such as multi-head latent attention (Ji et al., 2025) or partial weight sharing schemes may interact with dimension-wise pruning in ways that differ from both GQA and MHA. We view the interaction between attention sharing schemes and representation-level sparsity as a productive direction for further investigation, particularly as future architectures continue to trade off parameter efficiency against representational density.

6 Conclusion

In this work, we studied the extent to which attention scores can be approximated using dynamically selected features within a rotated basis. Our findings suggest that attention representations exhibit both spectral concentration and token-specific sparsity. AQUA combines these properties through orthogonal projection and magnitude-based feature selection, enabling reduced-dimensional attention computation while maintaining strong empirical performance across diverse benchmarks. We further showed that dynamic feature selection is substantially more effective than fixed variance-based retention, highlighting the importance of token-adaptive salience in attention approximation. Finally, our Triton-based implementation demonstrates that practical efficiency gains depend on preserving sparsity throughout the decode pipeline via hardware-aware execution. Overall, AQUA provides evidence that attention computation can be effectively approximated within dynamically selected low-dimensional subspaces, offering a useful perspective for future work on efficient long-context inference.

References

Yushi Bai, Xin Lv, Jiajie Zhang, et al. Longbench: A bilingual, multitask benchmark for long context understanding, 2024. URL https://arxiv.org/abs/2308.14508.

Peter Clark, Isaac Cowhey, Oren Etzioni, et al. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, et al. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Shichen Dong, Wen Cheng, Jiayu Qin, et al. Qaq: Quality adaptive quantization for llm kv cache, 2024. URL https://arxiv.org/abs/2403.04643.

Nelson Elhage, Tristan Hume, Catherine Olsson, et al. Toy models of superposition, 2022. URL https://arxiv.org/abs/2209.10652.

Blanca Calvo Figueras, Eneko Sagarzazu, Julen Etxaniz, et al. Truth knows no language: Evaluating truthfulness beyond english, 2025. URL https://arxiv.org/abs/2502.09387.

Leo Gao, Jonathan Tow, Baber Abbasi, et al. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 4th edition, 2013.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Michael Greenacre, Patrick J. F. Groenen, Trevor Hastie, et al. Principal component analysis. *Nature Reviews Methods Primers*, 2(1), dec 2022. doi: 10.1038/s43586-022-00184-w.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, et al. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2025. URL https://arxiv.org/abs/2401.18079.

Tao Ji, Bin Guo, Yuanbin Wu, et al. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms, 2025. URL https://arxiv.org/abs/2502.14837.

Ian Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374:20150202, 04 2016. doi: 10.1098/rsta.2015.0202.

Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention, 2022. URL https://arxiv.org/abs/2209.04881.

V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980. doi: 10.1109/TAC.1980.1102314.

Zirui Liu, Jiayi Yuan, Hongye Jin, et al. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, et al. Pointer sentinel mixture models, 2016.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, et al. Olmoe: Open mixture-of-experts language models, 2025. URL https://arxiv.org/abs/2409.02060.

David R. Musser. Introspective sorting and selection algorithms. *Software: Practice and Experience*, 27(8):983–993, 1997. doi: 10.1002/(SICI)1097-024X(199708)27:8<983::AID-SPE117>3.0.CO;2-#. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-024X%28199708%2927%3A8%3C983%3A%3AAID-SPE117%3E3.0.CO%3B2-%23.

OpenAI, Josh Achiam, Steven Adler, et al. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, et al. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, et al. Sparq attention: Bandwidth-efficient llm inference, 2024. URL https://arxiv.org/abs/2312.04985.

Santhosh G S, Saurav Prakash, and Balaraman Ravindran. Swan: Sparse winnowed attention for reduced inference memory via decompression-free kv-cache compression, 2025. URL https://arxiv.org/abs/2511.18936.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, et al. Winogrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, et al. Eigen attention: Attention in low-rank space for kv cache compression, 2024. URL https://arxiv.org/abs/2408.05646.

Prajwal Singhania, Siddharth Singh, Shwai He, et al. Loki: Low-rank keys for efficient sparse attention, 2024. URL https://arxiv.org/abs/2406.02542.

Mingjie Sun, Xinlei Chen, J. Zico Kolter, et al. Massive activations in large language models, 2024. URL https://arxiv.org/abs/2402.17762.

Yi Tay, Mostafa Dehghani, Dara Bahri, et al. Efficient transformers: A survey, 2022. URL https://arxiv.org/abs/2009.06732.

Philippe Tillet, H. T. 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*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Yubo Wang, Xueguang Ma, Ge Zhang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

Jiale Xu, Rui Zhang, Yi Xiong, et al. ellm: Elastic memory management framework for efficient llm serving, 2025a. URL https://arxiv.org/abs/2506.15155.

Yuhui Xu, Zhanming Jie, Hanze Dong, et al. Think: Thinner key cache by query-driven pruning, 2025b. URL https://arxiv.org/abs/2407.21018.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, et al. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, et al. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Yukun Zhu, Ryan Kiros, Rich Zemel, et al. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

A Appendix

A.1 A Primer on SVD for PCA

Singular Value Decomposition factorizes any matrix $A \in \mathbb{R}^{m \times n}$ into three matrices:

$$A = U \Sigma V^T \tag{2}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing the singular values. The columns of V are the principal components (or principal directions) of the row-space of A. Projecting the rows of A onto the first few columns of V concentrates the maximum possible variance (i.e., "energy" or "information") into those new dimensions.

A.2 Derivation of SVD Computational Complexity

In our analysis, we state that the computational complexity of computing the principal directions of the key cache matrix $K_{:i+1}$ is $O(\min((i+1)d_{head}^2, (i+1)^2 d_{head}))$. Since AQUA only requires the right singular vectors V to construct the projection matrix P, we consider the complexity of a thin (economy) SVD rather than a full SVD[3]. This appendix provides a brief derivation of this standard result from numerical linear algebra (Golub & Van Loan, 2013).

Let the matrix in question be $A \in \mathbb{R}^{m \times n}$, where, in our context, $m = i + 1$ (the sequence length) and $n = d_{head}$ (the head dimension). The SVD of A is given by $A = U \Sigma V^T$. A common approach for computing the principal directions required by AQUA is to perform an eigendecomposition of either $A^T A$ or AA^T. The choice between these two paths depends on which matrix is smaller, as this determines the more efficient computation.

Path 1: Eigendecomposition of $A^T A$

This path is generally preferred when $m \geq n$ (i.e., when the sequence length is greater than or equal to the head dimension, which is the common case in LLMs).

1. **Form the covariance matrix $A^T A$:**
   - The dimensions are $(n \times m) \times (m \times n) = (n \times n)$.
   - The computational cost of this matrix multiplication is $O(n^2 m)$.
2. **Perform eigendecomposition on $A^T A$:**
   - We solve $(A^T A)V = V\Lambda$, where Λ is the diagonal matrix of eigenvalues and the columns of V are the eigenvectors.
   - The cost of eigendecomposition for an $n \times n$ matrix is typically $O(n^3)$.

The total complexity for this path is the sum of these steps, $O(n^2 m + n^3)$. Since we assume $m \geq n$, the dominant term is $O(n^2 m)$. Substituting our original variable names, this complexity is $O(d_{head}^2 (i+1))$.

Path 2: Eigendecomposition of AA^T

This path is more efficient when $n > m$ (i.e., when the head dimension is larger than the sequence length, a less common scenario).

1. **Form the covariance matrix AA^T:**
   - The dimensions are $(m \times n) \times (n \times m) = (m \times m)$.

[3] A full SVD that explicitly computes both $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ incurs additional cubic terms. Since AQUA only requires the principal directions contained in V, the thin-SVD complexity is the relevant quantity.

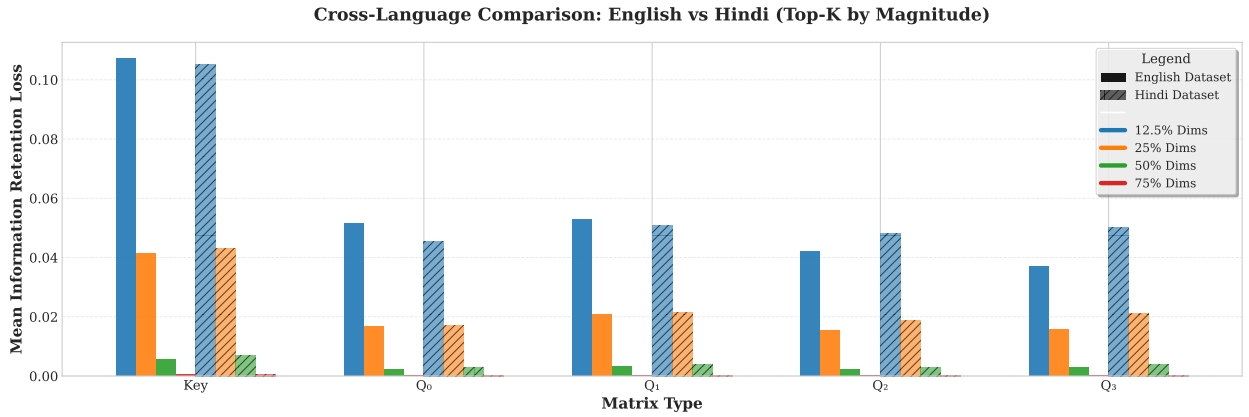

Figure 5: Cross-lingual comparison of mean information retention loss using an English-calibrated projection matrix on English (`WikiText`) and Hindi (`wikipedia-hi`) datasets for a full GQA group (1 Key and 4 corresponding Query matrices). The similar loss profiles across all query and key matrices demonstrate that the projection matrix generalizes well across languages with different scripts, indicating it captures fundamental, language-agnostic properties of the attention heads.

- The computational cost of this matrix multiplication is $O(m^2n)$.

2. **Perform eigendecomposition on AA^T:**
   - We solve $(AA^T)U = U\Lambda$.
   - The cost of eigendecomposition for an $m \times m$ matrix is typically $O(m^3)$.

The total complexity for this path is $O(m^2n + m^3)$. Since we assume $n > m$, the dominant term is $O(m^2n)$. Substituting our original variable names, this complexity is $O((i+1)^2 d_{head})$.

Overall Complexity

For computing the principal directions used by AQUA (equivalently, the right singular vectors of a thin SVD), an efficient implementation will choose the more favorable of these two paths based on the matrix dimensions. Therefore, the overall computational complexity is the minimum of the dominant costs from each path.

$$\text{Complexity} = O(\min(n^2m, m^2n))$$

Substituting $m = i + 1$ and $n = d_{head}$, we arrive at the complexity cited in the main text:

$$\text{Complexity} = O(\min(d_{head}^2(i+1), (i+1)^2 d_{head}))$$

A.3 Detailed Theoretical Analysis and Proofs

This appendix provides the detailed proofs for the complexity results presented in Section 3.4. The analysis focuses on the computational cost required to produce the unnormalized attention scores, as this is the primary stage targeted by our optimization.

Proposition A.1 (Complexity of Standard Attention)**.** *The computational complexity of the unnormalized score calculation in standard auto-regressive attention for a single head at token $i+1$ is $C_{std} = O((i+1) \cdot d_{head})$ (Vaswani et al., 2023; Tay et al., 2022).*

Proof. The cost is dominated by the matrix-vector product $q_i K_{:i+1}^T$, where the query $q_i \in \mathbb{R}^{1 \times d_{head}}$ and the transposed key cache $K_{:i+1}^T \in \mathbb{R}^{d_{head} \times (i+1)}$. This operation requires $(i+1) \cdot d_{head}$ multiplication-addition pairs, leading to a complexity of $O((i+1) \cdot d_{head})$ (Goodfellow et al., 2016). □

Theorem A.2 (Complexity of AQUA)**.** *The computational complexity of the unnormalized score calculation in the AQUA algorithm (Algorithm 1) for a single head at token $i+1$ is $C_{AQUA} = O(d_{head}^2 + (i+1) \cdot k)$.*

Proof. The total complexity is the sum of the complexities of the constituent steps of the algorithm:

1. **Projections:** Projecting the current query q_i and key k_i with matrix $P \in \mathbb{R}^{d_{head} \times d_{head}}$ requires two matrix-vector multiplications, for a total cost of $O(d_{head}^2)$ (Goodfellow et al., 2016).
2. **Magnitude Calculation & Top-k Selection:** Computing the element-wise absolute value of $\hat{q}_i$ is an $O(d_{head})$ operation. Finding the indices of the top-k elements can be done efficiently using a selection algorithm (e.g., Introsort) in $O(d_{head})$ average-case time (Musser, 1997).
3. **Dimension Slicing:** Constructing the sliced key matrix $\tilde{K}_{:i+1}$ requires accessing $(i+1) \cdot k$ elements from memory, introducing memory traffic proportional to $O((i+1) \cdot k)$. In practice, explicitly materializing this sliced tensor can reduce the practical benefits of sparsity, motivating the fused kernel implementation discussed in Section 3.4.
4. **Concatenation Operation:** The concatenation in Line 4 processes vectors of size d_{head}, resulting in complexity $O(d_{\text{head}})$.
5. **Absolute-Value Operation:** The element-wise absolute-value computation in Line 5 also operates over d_{head} elements, giving complexity $O(d_{\text{head}})$.
6. **Approximate Attention Computation:** The final matrix-vector product is between $\tilde{q}_i \in \mathbb{R}^{1 \times k}$ and $\tilde{K}_{:i+1}^T \in \mathbb{R}^{k \times (i+1)}$. The complexity of this step is $O((i+1) \cdot k)$.

Summing these complexities and retaining the dominant terms, we get:

$$\begin{aligned} C_{AQUA} &= O(d_{\text{head}}^2) + O(d_{\text{head}}) + O(d_{\text{head}}) + O((i+1) \cdot k) \\ &= O(d_{\text{head}}^2 + (i+1) \cdot k) \end{aligned}$$

This concludes the proof.

□

Corollary A.3 (Asymptotic Computational Break-Even Point)**.** *Let c_1, c_2, and c_3 denote implementation-dependent constant factors associated with the projection operation, approximate attention computation, and standard attention computation, respectively. Then, AQUA is asymptotically more efficient than standard attention for computing unnormalized scores when the sequence length $i+1$ satisfies:*

$$i + 1 > \frac{c_1 d_{head}^2}{c_3 d_{head} - c_2 k}.$$

Proof. Using implementation-dependent constant factors, we compare the computational costs:

$$C_{\text{AQUA}} = c_1 d_{\text{head}}^2 + c_2 (i+1) k$$

and

$$C_{\text{std}} = c_3 (i+1) d_{\text{head}}.$$

We seek the condition under which

$$C_{\text{AQUA}} < C_{\text{std}}.$$

Therefore,

$$\begin{aligned} c_1 d_{\text{head}}^2 + c_2 (i+1) k &< c_3 (i+1) d_{\text{head}} \\ c_1 d_{\text{head}}^2 &< (i+1)(c_3 d_{\text{head}} - c_2 k). \end{aligned}$$

Assuming

$$c_3 d_{\text{head}} > c_2 k,$$

we divide both sides by the positive quantity $(c_3 d_{\text{head}} - c_2 k)$ to obtain:

$$i + 1 > \frac{c_1 d_{\text{head}}^2}{c_3 d_{\text{head}} - c_2 k}.$$

Thus, AQUA becomes asymptotically efficient beyond this sequence length threshold.

□

Remark: The above break-even point is derived from asymptotic operation counts and explicitly incorporates implementation-dependent constant factors. In practice, the values of c_1, c_2, and c_3 depend on hardware characteristics, memory access patterns, kernel fusion, cache locality, and optimized matrix multiplication routines. Consequently, the empirical break-even point may differ from the theoretical threshold derived above, motivating the use of specialized GPU implementations such as Triton kernels as we demonstrated in Subsection 3.5.

Let us clarify this relationship between the hyperparameter r and current sequence length i by dissecting the trade-off between fixed overhead and accumulated savings.

Decomposing Computational Costs

To understand the break-even point, we must separate the costs into two components:

1. **Fixed Overhead Cost (C_{fixed}):** This is a per-step cost incurred independently of the sequence length i. It is dominated by the projection of the RoPE-transformed query and key vectors, with complexity $O(d_{\text{head}}^2)$. Unlike standard linear projections, this cost cannot in general be fused into the original query/key projection matrices because the AQUA projection is applied after the position-dependent RoPE transformation. Since RoPE and the AQUA projection matrix do not generally commute, the effective transformation cannot be rewritten as a single precomputed projection matrix. Consequently, this overhead remains an explicit component of the computation.

2. **Variable Savings (S_{var}):** This represents the computational savings achieved for each of the $i + 1$ tokens in the context. The saving for each token is proportional to the number of dimensions we prune, $(d_{head} - k)$. Thus, the total accumulated savings across the entire context is proportional to $(i + 1)(d_{head} - k)$.

The Break-Even Condition and the Role of r

Under the asymptotic cost model, AQUA becomes computationally favorable when the accumulated savings from reducing the attention dimension outweigh the fixed projection overhead. Ignoring implementation-dependent constant factors for interpretability, this condition can be approximated as:

$$(i + 1)(d_{\text{head}} - k) > d_{\text{head}}^2$$

This expression provides intuition for the role of r. When r is small (an aggressive approximation), the per-token reduction $(d_{\text{head}} - r)$ is large, meaning that fewer decoding steps are required to amortize the fixed projection overhead. Conversely, when r is large, the per-token reduction becomes smaller, requiring substantially longer sequences before the accumulated savings dominate the overhead cost.

A.4 Detailed Cross-Lingual Generalizability Analysis

This appendix provides a more detailed account of the cross-lingual experiment designed to test the robustness of our offline projection matrix.

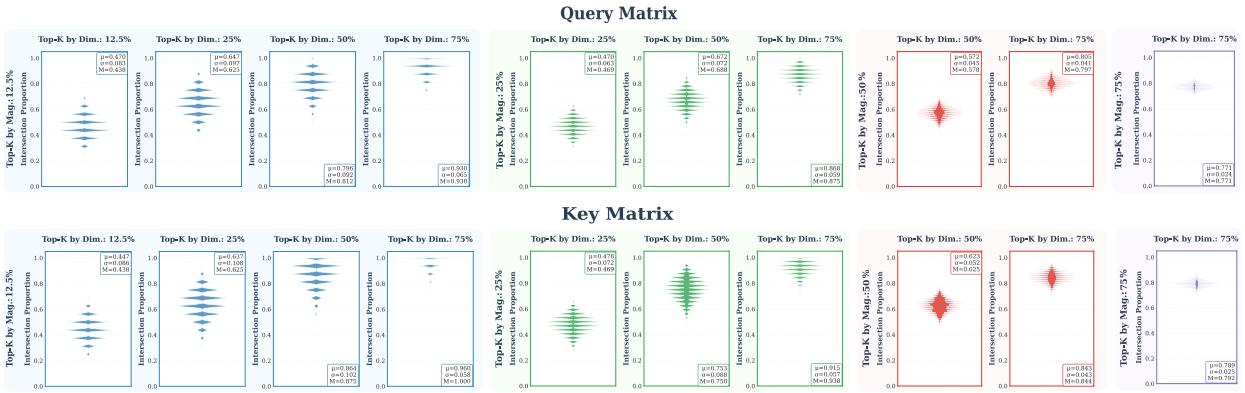

Figure 6: Overlap analysis for the Query and Key matrices (Layer 31, Head 31). The plots show the intersection proportion between the set of top dimensions selected by magnitude and the set of top dimensions selected by PCA index. The low overlap, especially for smaller K, demonstrates that global variance (PCA) and token-specific activity (magnitude) are not equivalent notions of importance.

Experimental Design and Rationale

To create a rigorous test case, we sought a language that was maximally distant from the English used in our calibration dataset (`BookCorpus`). We selected Hindi for two primary reasons. First, it is an officially supported language in the `meta-llama/Llama-3.1-8B-Instruct` model, ensuring that the tokenizer and internal representations are well-defined. Second, and more importantly, Hindi uses the Devanagari script, which is structurally and visually unrelated to the Latin script used for English. This provides an extreme test: if the projection matrix had overfit to linguistic or orthographic features of English, we would expect its performance to degrade significantly when applied to Hindi activations.

We ensured experimental consistency by sampling an equal number of query and key vectors from both the English evaluation set (`WikiText` (Merity et al., 2016)) and the Hindi dataset (`wikipedia-hi`).

Results and Discussion

The full results for the entire GQA group are presented in Figure 5. The plot shows the mean information retention loss for the shared Key matrix and all four associated Query matrices (Q_0 through Q_3). These are the matrices corresponding to a single projection matrix. Across all matrix types, the performance on the Hindi data closely mirrors the performance on the English data. This consistency strongly supports our hypothesis that the SVD-based projection method captures fundamental, language-agnostic structural properties of the attention heads, rather than superficial linguistic patterns. This makes our approach highly generalizable and robust for multilingual applications.

A.5 Detailed Analysis of Magnitude vs. PCA-based Selection

This appendix provides the detailed empirical analysis that justifies our use of magnitude-based dimension selection over a naive slicing of principal components.

Methodology

Our study investigates the overlap between two distinct sets of "important" dimensions within the Key and Query matrices of a specific attention head (Layer 31, Head 31) from the `meta-llama/Llama-3.1-8B-Instruct` (Grattafiori et al., 2024) model. The analysis was conducted on text sequences from the WikiText-2-raw-v1 dataset. Let $v \in \mathbb{R}^{d_{head}}$ be a vector from either a Query or Key matrix. We define two sets of indices from the full set of dimensions $\mathcal{I} = \{1, \ldots, d_{head}\}$:

1. **The Set of Top-k Magnitude Dimensions**, $\mathcal{S}_{mag}(v, K)$, is the set of indices corresponding to the r largest absolute values of the components of the unprojected vector v. Formally:

$$\mathcal{S}_{\text{mag}}(v, K) = \arg\operatorname{TopK}_{j \in \mathcal{I}}\big(|v_j|\big), \quad \text{where } |\mathcal{S}_{\text{mag}}(v, K)| = K$$

2. **The Set of Top-K' PCA Dimensions**, $\mathcal{S}_{pca}(K')$, is the set of indices corresponding to the first K' principal components derived from our offline calibration. As these components are ordered by variance, this is simply the set of the first K' indices:

$$\mathcal{S}_{pca}(K') = \{1, 2, \ldots, K'\}$$

We consider values for K and K' from the set $\{0.125 \cdot d_{head}, 0.25 \cdot d_{head}, 0.5 \cdot d_{head}, 0.75 \cdot d_{head}\}$. To quantify the alignment between token-specific importance (magnitude) and global importance (PCA), we calculate the intersection proportion, ρ, for each vector v:

$$\rho(v, K, K') = \frac{|\mathcal{S}_{mag}(v, K) \cap \mathcal{S}_{pca}(K')|}{K}$$

This metric measures the fraction of the top magnitude dimensions that are also captured by the top K' principal components. The distributions of ρ across the dataset are then visualized using violin plots.

Results and Discussion

The results are presented in Figure 6. The key observations are as follows:

- **Discrepancy Between Magnitude and PCA Importance:** The central finding is that the overlap is often far from perfect. For example, when selecting the top 12.5% of dimensions by magnitude, only a fraction of these are captured by the top 12.5% of principal components. This indicates a significant discrepancy between the dimensions that are most "active" for a specific token (high magnitude) and those that capture the most variance globally (top principal components).
- **Increasing Overlap with More PCA Components:** As expected, the overlap increases as we include more principal components (moving horizontally across the columns in the figures). However, even when considering 75% of the PCA dimensions, the overlap with the top 12.5% of magnitude dimensions is not total.

This empirical analysis demonstrates that simply slicing the top k dimensions by index after projection is a suboptimal strategy. While PCA effectively identifies directions of maximal global variance, these directions do not consistently align with the dimensions that are most salient for a specific query, as indicated by their magnitude. This finding provides strong motivation for the central mechanism of the AQUA algorithm: dynamically selecting the dimensions with the highest magnitude for each query, rather than relying on a fixed, static set of principal component indices.

A.6 Rotational Invariance of Attention Scores

This appendix provides the formal proof for the claim that projecting query and key vectors with an orthogonal matrix is a lossless rotation that preserves the dot product scores.

Lemma A.4 (Rotational Invariance of Attention Scores)**.** *Let $P \in \mathbb{R}^{d_{head} \times d_{head}}$ be an orthogonal projection matrix (i.e., $PP^T = P^T P = I$) derived from offline calibration. Let the projected query and key matrices be $\hat{q}_i = q_i P$ and $\hat{K}_{:i+1} = K_{:i+1} P$. The attention scores computed using the original vectors are identical to those computed using the projected vectors.*

Proof. The original attention scores are given by the dot product $S = q_i K_{:i+1}^T$. Without any pruning, the attention scores computed with the projected vectors are $\hat{S} = \hat{q}_i \hat{K}_{:i+1}^T$.

We can show that $\hat{S}$ is equivalent to S:

$$\begin{aligned}
\hat{S} &= (\hat{q}_i)(\hat{K}_{:i+1})^T \\
&= (q_i P)(K_{:i+1} P)^T && \text{[Substituting definitions]} \\
&= (q_i P)(P^T K_{:i+1}^T) && [(AB)^T = B^T A^T] \\
&= q_i (P P^T) K_{:i+1}^T && \text{[Associativity]} \\
&= q_i I K_{:i+1}^T && [P P^T = I] \\
&= q_i K_{:i+1}^T = S
\end{aligned} \tag{3}$$

Thus, the scores are identical ($\hat{S} = S$). This proves that the projection is a lossless rotation of the coordinate space that preserves the dot product relationships between all query and key vectors. The only source of approximation error in our method, therefore, comes from the subsequent step of selecting a subset of the dimensions. □

A.7 Implementation Details of the Final AQUA Decode Path

This section describes the decode-time Triton implementation used for the latency results in the main text.

The central implementation invariant is the following: AQUA only yields latency gains when the selected low-dimensional computation is preserved throughout the decode path. It is not sufficient to select a sparse set of dimensions if those dimensions are later scattered back into dense tensors, or if intermediate logits and attention weights are materialized as separate tensors. The final implementation therefore preserves sparsity through score computation, softmax normalization, and value aggregation.

A.7.1 From Algorithmic Sparsity to an Efficient Decode Path

AQUA projects the query and keys, selects the top-k projected query dimensions by magnitude, and computes attention scores only over those retained dimensions. At the algorithmic level, this reduces the score computation from a full $D = d_{\text{head}}$-dimensional dot product to a k-dimensional dot product. However, the way this sparsity is represented in the implementation determines whether the theoretical reduction becomes an actual latency improvement.

Our initial implementation was a direct PyTorch realization of Algorithm 1. It projected the query and key, selected the top-k projected query dimensions, and then reconstructed a dense query tensor before invoking standard dense attention operators. This implementation followed the AQUA selection rule, but it did not preserve reduced-dimensional execution. Once the retained values were re-embedded into a dense tensor, the backend still performed dense reductions over all D dimensions. Thus, this path added projection and top-k overhead without removing the dense score-computation cost.

The first systems correction was to compute sparse scores directly. Instead of scattering retained query coordinates back into a dense representation, the kernel computed attention logits by iterating only over the selected dimensions and reading the corresponding slices of the projected key cache. This made the reduced-dimensional dot product real at execution time. However, this implementation still materialized intermediate logits and attention weights in memory. As a result, memory traffic and fragmented kernel launches remained significant latency bottlenecks.

The next step was to fuse sparse score computation, softmax normalization, and weighted value aggregation into a single sparse decode kernel. This removed the need to write logits to memory, read them back for softmax, write attention weights, and read them again for value aggregation. After this fusion, the dominant remaining overhead came from the surrounding decode-time preparation steps: query projection, top-k selection, key projection, and projected-cache update.

The final implementation is therefore a three-kernel decode pipeline rather than a single monolithic kernel. One kernel fuses key projection with projected-cache update, a second kernel fuses query projection with top-k selection, and a third kernel fuses sparse score computation, online softmax, and value aggregation.

These three kernels have distinct dataflow and storage requirements, so separating them gives a cleaner and more efficient decode-time implementation.

A.7.2 Runtime Conventions and Tensor Layout

We use the following notation in this appendix. Let B denote batch size, H the number of query heads, H_{kv} the number of key-value heads, $g = H/H_{kv}$ the grouped-query attention fanout, $D = d_{\text{head}}$ the head dimension, $S_{\max}$ the allocated cache capacity, and S_{valid} the active decode-time prefix length. In the decode regime, the query length is $S_Q = 1$.

After the model's standard query, key, and value projections and RoPE application, the decode-time tensors have shapes

$$Q \in \mathbb{R}^{B\times H\times 1\times D}, \qquad K_{\text{new}} \in \mathbb{R}^{B\times H_{kv}\times 1\times D}, \qquad V \in \mathbb{R}^{B\times H_{kv}\times S_{\max}\times D}.$$

AQUA uses a shared per-H_{kv}-head projection matrix

$$P \in \mathbb{R}^{H_{kv}\times D\times D}.$$

On the query side, the implementation uses a query-head-expanded view of this same projection. Therefore, each query head uses the projection associated with its grouped-query key-value head. The final design does not use separate learned projection tensors for query and key sides.

For grouped-query attention, each query head h maps to its associated key-value head by

$$h_{kv} = \left\lfloor \frac{h}{g} \right\rfloor.$$

The projected key cache is maintained in the transposed layout

$$\widehat{K}^T_{\text{cache}} \in \mathbb{R}^{B\times H_{kv}\times D\times S_{\max}}.$$

This layout is important because sparse decode reads keys in the pattern

$$\text{for selected dimension } d: \qquad \widehat{K}^T_{\text{cache}}[b, h_{kv}, d, 1 : S_{\text{valid}}].$$

Thus, for a fixed selected feature dimension, token positions are contiguous in memory. In the standard layout $[B, H_{kv}, S, D]$, the same access pattern would be strided across the head-dimension axis. The transposed layout therefore better matches the sparse selected-dimension access pattern used by AQUA.

Across all Triton kernels, accumulations are performed in fp32 even when the surrounding model runs in bf16 or fp16. This applies to projected dot products, online softmax state, and weighted value accumulation.

A.7.3 Final Decode-Time Kernel Pipeline

Algorithm 2 summarizes the final decode-time execution path. The standard model projections $q_{\text{proj}}, k_{\text{proj}}, v_{\text{proj}}$, and o_{proj} remain outside the AQUA Triton fusion. The AQUA-specific work is implemented by the three kernels described below.

The pipeline is specialized to auto-regressive decode-time execution. In the current implementation, the fused path assumes $S_Q = 1$, uses fixed-capacity `StaticCache`-style storage, and bounds attention by the runtime prefix length $S_{\text{valid}} =$ `valid_seq_len`. Although the value cache and projected key cache are passed at full preallocated capacity, the decode kernels attend only over the populated prefix.

A.7.4 Kernel 1: Fused Key Projection and Projected-Cache Update

The first kernel projects the newly generated key into AQUA space and writes the result directly into the transposed projected-key cache. It also emits the projected key in standard layout so that the ordinary cache update path can proceed.

This kernel avoids a separate projected-key transpose and a separate write into the transposed cache.

Algorithm 2 Final AQUA decode-time pipeline

Require: $Q \in \mathbb{R}^{B\times H\times 1\times D}$, $K_{\text{new}} \in \mathbb{R}^{B\times H_{kv}\times 1\times D}$, $V \in \mathbb{R}^{B\times H_{kv}\times S_{\max}\times D}$, shared AQUA projection P, transposed projected-key cache $\widehat{K}^T_{\text{cache}}$, cache position s, valid prefix length S_{valid}, retained dimension count k

Ensure: Decode-time attention output $O \in \mathbb{R}^{B\times H\times 1\times D}$

1: $\widehat{K}_{\text{new}} \leftarrow \textsc{KeyProjectAndUpdate}(K_{\text{new}}, P, \widehat{K}^T_{\text{cache}}, s)$
2: update the projected key-value cache with $\widehat{K}_{\text{new}}$ and the new value
3: $(I_{\text{topk}}, Q_{\text{topk}}) \leftarrow \textsc{QueryProjectTopK}(Q, P, k)$
4: $O \leftarrow \textsc{SparseDecodeOnlineSoftmax}(I_{\text{topk}}, Q_{\text{topk}}, \widehat{K}^T_{\text{cache}}, V, S_{\text{valid}})$
5: **return** O

Algorithm 3 Kernel 1: fused key projection and projected-cache update

Require: $K_{\text{new}} \in \mathbb{R}^{B\times H_{kv}\times D}$, shared AQUA projection $P \in \mathbb{R}^{H_{kv}\times D\times D}$, transposed projected-key cache $\widehat{K}^T_{\text{cache}} \in \mathbb{R}^{B\times H_{kv}\times D\times S_{\max}}$, cache position s

Ensure: Projected key $\widehat{K}_{\text{new}} \in \mathbb{R}^{B\times H_{kv}\times D}$ and in-place update of $\widehat{K}^T_{\text{cache}}[:,:,:,s]$

1: **for all** batch index b and key-value head h_{kv} in parallel **do**
2: load $k \leftarrow K_{\text{new}}[b, h_{kv}, :]$
3: compute projected key in fp32:

$$\widehat{k}[d_{\text{out}}] \leftarrow \sum_{d_{\text{in}}=1}^{D} k[d_{\text{in}}]\, P[h_{kv}, d_{\text{in}}, d_{\text{out}}]$$

4: cast $\widehat{k}$ back to model dtype for storage
5: store $\widehat{K}_{\text{new}}[b, h_{kv}, :] \leftarrow \widehat{k}$
6: store $\widehat{K}^T_{\text{cache}}[b, h_{kv}, :, s] \leftarrow \widehat{k}$
7: **return** $\widehat{K}_{\text{new}}$

A.7.5 Kernel 2: Fused Query Projection and Top-k Selection

The second kernel projects the query into AQUA space and immediately performs top-k dimension selection. Instead of returning a dense projected query, it emits only the retained indices and retained projected values.

This kernel removes separate launches for query projection, absolute-value computation, top-k selection, and gather.

A.7.6 Kernel 3: Fused Sparse Decode with Online Softmax

The third kernel consumes the sparse query representation and the transposed projected-key cache. It computes sparse logits directly, performs numerically stable online softmax, and accumulates the value output without materializing logits or attention weights.

Although the backing storage for $\widehat{K}^T_{\text{cache}}$ and V has capacity $S_{\max}$, the loop is bounded by S_{valid}. This prevents the kernel from attending to future zero-padded cache slots. The online softmax formulation is necessary because it preserves numerical stability while avoiding materialization of the logits tensor and the attention-weight tensor.

A.7.7 Measurement Boundary

For the per-layer breakdown in the main text, we time the query-side fused preparation kernel and the fused sparse decode kernel. Key-side projection and projected-cache update are excluded from that breakdown to isolate the core decode-attention computation. This separation makes the measurement boundary explicit: the breakdown focuses on the query-dependent decode attention path, while the full decode implementation also includes the key-side projected-cache update needed to maintain the AQUA cache state.

Algorithm 4 Kernel 2: fused query projection and top-k selection

Require: $Q \in \mathbb{R}^{B\times H\times D}$, query-head-expanded projection view $P^{(q)} \in \mathbb{R}^{H\times D\times D}$, retained dimension count k
Ensure: Retained indices $I_{\text{topk}} \in \mathbb{N}^{B\times H\times k}$, retained values $Q_{\text{topk}} \in \mathbb{R}^{B\times H\times k}$

for all batch index b and query head h in parallel **do**
  load $q \leftarrow Q[b, h, :]$
  compute projected query in fp32:

$$\widehat{q}[d_{\text{out}}] \leftarrow \sum_{d_{\text{in}}=1}^{D} q[d_{\text{in}}]\, P^{(q)}[h, d_{\text{in}}, d_{\text{out}}]$$

  compute magnitudes $m[d] \leftarrow |\widehat{q}[d]|$
  find the k-th largest magnitude threshold τ
  mark all dimensions with $m[d] > \tau$
  among dimensions with $m[d] = \tau$, keep only as many as needed to obtain exactly k retained dimensions
  compact retained dimensions into index-value pairs
  store retained indices $I_{\text{topk}}[b, h, :]$
  store retained projected values $Q_{\text{topk}}[b, h, :]$
return $(I_{\text{topk}}, Q_{\text{topk}})$

The kernel microbenchmark exercises the optimized AQUA decode path directly. In both cases, the reported speedups should be interpreted as decode-time attention speedups under the operating assumptions described in this appendix.

A.7.8 Why the Final Design Works

The complete implementation story can be summarized by three findings.

First, sparse representations are not enough. If retained dimensions are re-embedded into a dense tensor before score computation, the backend still performs dense reductions over all D dimensions, and AQUA's theoretical savings are lost.

Second, sparse logits are not enough. Once score computation becomes truly sparse, the next bottleneck is memory traffic from separately materializing logits and normalized attention weights.

Third, after the sparse decode body is fused, the remaining overhead is the surrounding decode-time setup. The final design therefore fuses query-side preparation and key-side cache update into separate kernels around the fused sparse decode kernel.

This is why the final realization is a three-kernel pipeline. The decode body is one fused kernel, while the query-side preparation and key-side cache update are fused separately because they have different dataflow, output tensors, and cache-update responsibilities.

A.7.9 Scope and Limitations

The implementation described above should be interpreted as a decode-time realization of AQUA, not as a general-purpose attention kernel. The fused path assumes single-query decode with $S_Q = 1$, fixed-capacity cache storage, incremental projected-key-cache updates, and a runtime valid prefix length S_{valid}. Prefill uses a separate non-fused compatibility path.

The kernels are also tuned for the Llama-style operating regime with $D = 128$ and Hopper-class GPUs. Therefore, the exact implementation choices are specific to this regime, while the broader implementation principle is more general: AQUA's sparsity must be preserved through score computation, softmax, and value aggregation to produce real latency gains.

Algorithm 5 Kernel 3: fused sparse decode

Require: Retained indices $I_{\text{topk}} \in \mathbb{N}^{B\times H\times k}$, retained values $Q_{\text{topk}} \in \mathbb{R}^{B\times H\times k}$, transposed projected-key cache $\widehat{K}^T_{\text{cache}} \in \mathbb{R}^{B\times H_{kv}\times D\times S_{\max}}$, value cache $V \in \mathbb{R}^{B\times H_{kv}\times S_{\max}\times D}$, grouped-query factor g, valid prefix length S_{valid}, scaling factor $\alpha = 1/\sqrt{D}$

Ensure: Decode-time attention output $O \in \mathbb{R}^{B\times H\times D}$

for all batch index b and query head h in parallel **do**
  map to key-value head $h_{kv} \leftarrow \lfloor h/g \rfloor$
  initialize online softmax state:

$$m \leftarrow -\infty, \qquad \ell \leftarrow 0, \qquad a[1:D] \leftarrow 0$$

  for all token tiles $T \subseteq \{1,\dots,S_{\text{valid}}\}$ **do**
    for all $t \in T$ **do**
      compute sparse score in fp32:

$$s_t \leftarrow \alpha \sum_{r=1}^{k} Q_{\text{topk}}[b,h,r]\, \widehat{K}^T_{\text{cache}}[b,h_{kv},I_{\text{topk}}[b,h,r],t]$$

    compute tile maximum:

$$m_{\text{blk}} \leftarrow \max_{t\in T} s_t$$

    update running maximum:

$$m' \leftarrow \max(m, m_{\text{blk}})$$

    compute rescaling factor:

$$\beta \leftarrow \exp(m - m')$$

    for all $t \in T$ **do**
      compute local exponent:

$$p_t \leftarrow \exp(s_t - m')$$

    rescale accumulated output:

$$a \leftarrow \beta a$$

    accumulate tile contribution:

$$a \leftarrow a + \sum_{t\in T} p_t\, V[b,h_{kv},t,:]$$

    update normalizer:

$$\ell \leftarrow \beta\ell + \sum_{t\in T} p_t$$

    set $m \leftarrow m'$
  finalize output:

$$O[b,h,:] \leftarrow a/\ell$$

return O

A.8 Benchmark and Evaluation Details

This appendix provides a detailed account of the models and benchmark suite used in our empirical evaluation.

Table 5: Full performance results for `Llama-3.1-8B-Instruct` and `OLMoE-1B-7B-Instruct` on various benchmarks under different levels of AQUA pruning ratio (k_{ratio}). The baseline for each model, denoted by 'B' ($k_{ratio} = 1.0$), is highlighted in bold. Further details are provided in the Appendix.

Model	k_{ratio}	MMLU (acc ↑)	GSM8K (acc ↑)	HellaSwag (acc ↑)	WinoGrande (acc ↑)	TruthfulQA MC2 (acc ↑)	ARC Challenge (acc ↑)	WikiText (ppl ↓)
Llama-3.1-8B Instruct	**B**	**0.687 ± 0.004**	**0.816 ± 0.011**	**0.785 ± 0.004**	**0.755 ± 0.012**	**0.551 ± 0.016**	**0.647 ± 0.014**	**8.910**
	0.90	0.687 ± 0.004	0.792 ± 0.011	0.784 ± 0.004	0.756 ± 0.012	0.551 ± 0.016	0.647 ± 0.014	8.910
	0.75	0.685 ± 0.004	0.805 ± 0.011	0.785 ± 0.004	0.757 ± 0.012	0.551 ± 0.016	0.645 ± 0.014	8.930
	0.50	0.666 ± 0.004	0.720 ± 0.012	0.780 ± 0.004	0.738 ± 0.012	0.551 ± 0.016	0.620 ± 0.014	9.200
	0.40	0.634 ± 0.004	0.541 ± 0.014	0.773 ± 0.004	0.696 ± 0.013	0.540 ± 0.016	0.600 ± 0.014	9.810
	0.30	0.507 ± 0.004	0.146 ± 0.010	0.732 ± 0.004	0.598 ± 0.014	0.502 ± 0.016	0.530 ± 0.015	12.550
	0.20	0.242 ± 0.004	0.019 ± 0.004	0.391 ± 0.005	0.511 ± 0.014	0.471 ± 0.015	0.236 ± 0.012	44.960
	0.10	0.230 ± 0.004	0.012 ± 0.003	0.261 ± 0.004	0.496 ± 0.014	0.491 ± 0.016	0.236 ± 0.012	970.440
OLMoE-1B-7B Instruct	**B**	**0.530 ± 0.004**	**0.451 ± 0.014**	**0.783 ± 0.004**	**0.673 ± 0.013**	**0.491 ± 0.016**	**0.532 ± 0.015**	**11.340**
	0.90	0.530 ± 0.004	0.463 ± 0.014	0.782 ± 0.004	0.668 ± 0.013	0.489 ± 0.016	0.538 ± 0.015	11.340
	0.75	0.529 ± 0.004	0.453 ± 0.014	0.783 ± 0.004	0.669 ± 0.013	0.485 ± 0.016	0.542 ± 0.015	11.330
	0.50	0.526 ± 0.004	0.426 ± 0.014	0.778 ± 0.004	0.658 ± 0.013	0.488 ± 0.016	0.540 ± 0.015	11.340
	0.40	0.512 ± 0.004	0.385 ± 0.013	0.771 ± 0.004	0.640 ± 0.013	0.488 ± 0.016	0.532 ± 0.015	11.470
	0.30	0.485 ± 0.004	0.253 ± 0.012	0.747 ± 0.004	0.615 ± 0.014	0.481 ± 0.016	0.513 ± 0.015	12.140
	0.20	0.403 ± 0.004	0.042 ± 0.006	0.665 ± 0.005	0.549 ± 0.014	0.485 ± 0.016	0.378 ± 0.014	15.960
	0.10	0.243 ± 0.004	0.014 ± 0.003	0.346 ± 0.005	0.511 ± 0.014	0.486 ± 0.016	0.239 ± 0.012	74.440

Models

- **`meta-llama/Llama-3.1-8B-Instruct` (Grattafiori et al., 2024):** A powerful, state-of-the-art 8-billion parameter model that serves as our primary testbed. It features a model dimension (d_{model}) of 4096 across 32 layers. The model is built on a Grouped-Query Attention (GQA) architecture, with 32 query heads and 8 key/value heads (a group size of 4), where each head has a dimension (d_{head}) of 128.
- **`OLMoE-1B-7B-Instruct` (Muennighoff et al., 2025):** A Mixture-of-Experts (MoE) model whose inclusion allows us to test the generalizability of our method on a different architecture type. This model is built on a standard Multi-Head Attention (MHA) architecture, where every head has a unique Key and Query. It features a model dimension (d_{model}) of 4096, 64 experts, 27 layers, 16 heads, and a head dimension (d_{head}) of 128.

A.9 Experimental Setup

All evaluations were conducted on a **single NVIDIA A100 GPU**, with the model loaded in **16-bit half-precision (FP16)** to mirror standard deployment environments. We utilized the standardized EleutherAI `lm-evaluation-harness` (Gao et al., 2024) framework for all benchmarks. To ensure rigorous comparability and reproducibility, we adopted community-standard few-shot settings as described in the next paragraph A.9. For all instruction-tuned tasks, we applied the official chat template with few-shot examples formatted as multi-turn conversations, using a batch size of 1 to ensure deterministic scoring. For all instruction-tuned tasks, we applied the official chat template with few-shot examples formatted as multi-turn conversations, using a batch size of 1 to ensure deterministic scoring.

Benchmark Suite Rationale

The chosen benchmarks and few-shot settings align with common practices in LLM evaluation to ensure comparability and reproducibility. Table 6 and Table 7 provide detailed statistics on the training and evaluation splits used.

Standard Reasoning & Knowledge We utilize standard evaluation sets to measure core model competencies. As shown in Table 6, we evaluate on the full test/validation splits provided by the `lm-evaluation-harness` to ensure statistical significance.

Table 6: Dataset statistics for Standard Reasoning benchmarks. The "Evaluated" column denotes the actual number of samples processed during our inference runs.

Benchmark	Train Samples	Evaluated Samples
MMLU (5-shot)	∼14,000 (Aux)	14,368 (Test)
GSM8K (8-shot)	7,473	1,319 (Test)
HellaSwag (10-shot)	∼39,000	10,042 (Val)
WinoGrande (5-shot)	∼40,000	1,267 (Test)
ARC Challenge (25-shot)	1,119	1,172 (Test)
TruthfulQA (MC2)	N/A	817 (Test)
WikiText-103 (0-shot)	103M Tokens	∼250k Tokens

- **MMLU (Wang et al., 2024):** Evaluates massive multitask understanding across 57 subjects using the standard 5-shot setting.
- **GSM8K (Cobbe et al., 2021):** Tests multi-step mathematical reasoning using an 8-shot Chain-of-Thought (CoT) strategy.
- **HellaSwag (Zellers et al., 2019):** Evaluates commonsense inference (10-shot).
- **WinoGrande (Sakaguchi et al., 2019):** Targets pronoun resolution and commonsense reasoning (5-shot).
- **ARC Challenge (Clark et al., 2018):** Contains difficult science questions requiring retrieval and reasoning (25-shot).
- **TruthfulQA (Figueras et al., 2025):** Measures truthfulness via the multiple-choice (MC2) variant.
- **WikiText-103 (Merity et al., 2016):** Evaluates fundamental language modeling via perplexity on the full test set.

Long-Context Capabilities (LongBench) To assess performance under the **LongBench** suite (Bai et al., 2024), we selected a subset of tasks covering summarization, classification, and retrieval. Table 7 outlines the sample counts and average context lengths.

Table 7: Dataset statistics for the LongBench subset. Note that for TREC and SAMSum, training samples are embedded directly into the context window of each test instance to serve as few-shot examples.

Task	Type	Samples	Avg. Length
Multi-News (MN)	Summarization	200	2,113 words
TREC	Classification	200	5,177 words
Long-Context (LCC)	Classification	500	1,235 tokens
Passage Retrieval (PR)	Retrieval	200	9,289 words
SAMSum (SS)	Summarization	200	6,258 words

- **Multi-News (MN) & SAMSum (SS):** Summarization tasks testing synthesis capabilities over long documents and dialogue histories.
- **TREC & Long-Context Classification (LCC):** Classification tasks evaluating consistency over extended prompts. For TREC, few-shot examples are randomly sampled and embedded within the context.
- **Passage Retrieval (PR):** A synthetic "Needle-in-a-Haystack" style task measuring precision in locating specific information buried in distractor text.

A.10 Detailed Standalone AQUA Performance Results

This appendix provides the complete results and a detailed analysis for the standalone AQUA evaluation presented in the main paper. Table 5 shows the performance of both models across the full spectrum of pruning ratios.

Detailed Analysis

For `Llama-3.1-8B-Instruct`, we can observe that reducing the retention ratio to 0.90 has almost no effect. At $k_{ratio} = 0.75$, the performance remains exceptionally strong, with only a 0.02 point increase in perplexity and statistically insignificant changes in accuracy on most tasks. This confirms that a 25% dimensionality reduction is nearly "free" in terms of performance. The first significant drop occurs at $k_{ratio} = 0.50$, where the model's mathematical reasoning ability (GSM8K) begins to suffer, although its performance on commonsense tasks like HellaSwag remains robust. This suggests that complex, multi-step reasoning is more sensitive to dimensionality reduction than commonsense inference. The performance degradation accelerates significantly below a ratio of 0.40, with a near-total collapse of reasoning capabilities at 0.20 and below.

The `OLMoE-1B-7B-Instruct` model shows a similar overall trend but with a more graceful degradation curve. Even at a k_{ratio} of 0.50, the performance drop is minimal across all benchmarks. The degradation becomes more noticeable at 0.40 and 0.30, but it is less severe than what is observed with Llama-3.1 at the same ratios. As discussed in the main text, we attribute this increased resilience to its MHA architecture, where each query has a dedicated key. This allows for greater sparsity in the key vectors compared to the information-dense shared keys in GQA, making them less sensitive to pruning.

A.11 Statistical Analysis and Error Calculation

To ensure the reliability of our results, we report the Standard Error (SE) of the mean for all standard reasoning benchmarks. This metric quantifies the uncertainty in the model's performance estimate, treating the evaluation set as a sample from the population of possible questions.

A.11.1 Calculation Methodology

All statistical uncertainties are computed using the `lm-evaluation-harness` framework.

General Case. For tasks with continuous scoring or multi-part questions, the Standard Error is calculated using the Central Limit Theorem (CLT):

$$SE_{CLT} = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^{n}(s_i - \bar{s})^2}{n(n-1)}} \tag{4}$$

where n is the number of evaluation samples, s_i is the score on question i, and $\bar{s}$ is the mean score.

Binary Tasks. For tasks where the outcome is binary (correct/incorrect), such as multiple-choice accuracy, the calculation simplifies to the standard error of a Bernoulli distribution:

$$SE_{Bernoulli} = \sqrt{\frac{\bar{s}(1-\bar{s})}{n}} \tag{5}$$

A.12 Statistical Analysis of LongBench

For the LongBench suite, we calculated Standard Errors (SE) to quantify the measurement uncertainty inherent in these long-context tasks. To establish a robust upper bound on variance, we computed these statistics at the most aggressive pruning ratio ($k_{ratio} = 0.30$).

As the sample size N remains constant across all experimental runs (see Table 7), these Standard Error values serve as the representative error margins for the results reported in Table 4. As shown in Table 8, the uncertainty is generally low ($\leq 1.1\%$) for summarization and code tasks, with slightly higher variance ($\sim 2-3\%$) observed in classification and retrieval tasks due to their binary or "all-or-nothing" scoring nature.

A.13 Qualitative Analysis of Generative Coherence under AQUA Pruning

To complement our quantitative benchmark results, this section provides a qualitative analysis of how generative coherence is affected by varying levels of AQUA based pruning. We conducted a simple generative

Table 8: Representative Standard Errors (SE) for LongBench tasks, derived from the `Llama-3.1-8B` evaluation at $k_{ratio} = 0.30$. These values quantify the uncertainty of the mean scores reported in the main text.

Task	Metric	Standard Error (±)
Multi-News (MN)	ROUGE Score	0.004
TREC	Classification Score	0.033
Long-Context (LCC)	Code Sim Score	0.011
Passage Retrieval (PR)	Retrieval Score	0.026
SAMSum (SS)	ROUGE Score	0.011

task to observe the model's behavior as we decrease the k_{ratio} hyperparameter, which controls the percentage of dimensions retained during the attention computation.

Table 9: Generated summaries from `Llama-3.1-8B-Instruct` (Grattafiori et al., 2024) with varying levels of AQUA pruning.

AQUA Ratio (k_{ratio})	**Generated Summary**
1.0 (Baseline)	The text is a well-known pangram, a sentence that uses all the letters of the alphabet at least once. It is often used as a demonstration of a font or keyboard's capabilities. The sentence itself describes a scene where a quick, brown fox jumps over a lazy dog.
0.90	The text is a well-known pangram, a sentence that uses all the letters of the alphabet at least once. It is often used as a demonstration of a font or keyboard's capabilities. The sentence itself describes a scene where a quick brown fox jumps over a lazy dog.
0.75	The text is a well-known pangram, a sentence that uses all the letters of the alphabet at least once. It is often used as a demonstration of a font or keyboard's capabilities. The sentence itself describes a scene where a quick brown fox jumps over a lazy dog.
0.50	The given text is a well-known pangram, a sentence that uses all the letters of the alphabet at least once. The text is: "The quick brown fox jumps over the lazy dog." It is often used as a demonstration of a font or keyboard's capabilities.
0.40	The text is a well-known pangram, a sentence that uses all the letters of the alphabet at least once. It is often used as an example in typing and typing games. The sentence is a simple and short phrase describing a scene where a quick brown fox jumps over a lazy dog.
0.30	*The text is a well-known English idiom, often used to demonstrate a sentence that is grammatically correct but semantically nonsensical.*
0.20	*There is no text to summarize.*

Experimental Setup

We prompted the `Llama-3.1-8B-Instruct` model (Grattafiori et al., 2024), modified with our AQUA mechanism, to perform a basic summarization task.

- **Prompt:** "Give me a summary of the following text: The quick brown fox jumps over the lazy dog."
- **Decoding Strategy:** We used a deterministic decoding strategy (do_sample=False) to ensure that any variation in the output is directly attributable to the change in the k_{ratio} and not to sampling randomness.

Results and Analysis

The generated responses for different k_{ratio} values are presented in Table 9.

The results illustrate a clear and graceful degradation profile, followed by a sharp collapse in coherence:

- **Graceful Degradation (1.0 down to 0.40):** From the baseline down to a k_{ratio} of 0.40, the model correctly identifies the text as a pangram and provides a factually accurate summary. The responses are nearly identical down to a ratio of 0.75. At 0.50 and 0.40, the phrasing changes slightly, but the core semantic content remains perfectly intact. This aligns with our quantitative results, showing that a significant portion of the attention computation can be pruned with minimal impact on the model's knowledge and reasoning abilities.
- **Semantic Failure (at 0.30):** A critical failure occurs at a k_{ratio} of 0.30. The model loses its ability to correctly identify the pangram and instead misclassifies it as a "semantically nonsensical" idiom. This represents the point where the information loss from pruning becomes too great, leading to a fundamental error in reasoning.
- **Complete Collapse (at 0.20):** At a k_{ratio} of 0.20, the model's capabilities collapse entirely. It fails to even recognize the presence of the input text, indicating a catastrophic failure in the attention mechanism's ability to process information.

This qualitative analysis provides an intuitive demonstration of the trade-offs involved in our method. It confirms that AQUA offers a robust "sweet spot" where efficiency can be gained with negligible performance loss, while also clearly defining the operational limits beyond which model coherence is compromised. Also, please note that this property would differ from model to model and architecture to architecture.