QREAM: KV CACHE COMPRESSION VIA PIVOTED QR AND EXPLICIT ATTENTION MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large key-value (KV) caches are a major bottleneck for long-context large language model (LLM) inference, motivating methods that reduce their size while preserving information needed for future queries. Attention matching (AM) has recently emerged as a practical and theoretically grounded framework for controlling attention behavior, with successful applications to prefix tuning and KV-cache compression. However, existing AM-based methods for KV-cache compression separately match the numerator and normalization factor of attention, requiring auxiliary per-key biases that are incompatible with standard KV-cache representations. In this work, we propose QREAM, a bias-free formulation of attention matching that directly matches the normalized attention output rather than its numerator and normalization factor separately. Once the compressed keys are selected, this formulation reduces attention matching to an efficient linear solve over the compressed values, requiring neither training nor modifications to the standard KV-cache representation. To select informative keys, QREAM further introduces a pivoted-QR-based procedure that favors linearly independent attention contributions, providing an efficient alternative to matching-pursuit methods. We evaluate QREAM across multiple Qwen model scales, long-context QA benchmarks, and compression ratios from 5× to 100×. QREAM is competitive with existing attention-matching methods while substantially outperforming token-scoring baselines under aggressive compression, despite its simpler and more deployment-friendly formulation.

1 INTRODUCTION

Long-context inference has made key-value (KV) cache management increasingly important: the KV cache grows linearly with context length and is accessed throughout autoregressive generation. Despite context windows now exceeding one million tokens, sufficiently long inputs still require the existing context to be compressed while preserving information relevant to future queries. KV cache compression (Xiao et al., 2024; Li et al., 2024; Devoto et al., 2025; Kim et al., 2025; Zweiger et al., 2026) addresses this problem by numerically compressing[1] the cache into a substantially shorter representation, typically by factors of $2\times$–$100\times$, while retaining information needed for downstream generation.

In this work, we focus on the **query-agnostic** setting, where future queries are unknown at compression time, so the compressed cache must preserve broadly useful information rather than optimize for specific queries. Inevitably, aggressive compression discards some information: for example, a highly compressed patient record may no longer support a query such as "how many t's are there in the report?" This trade-off becomes increasingly pronounced as the compression ratio grows.

Attention matching (AM) has recently emerged as a powerful and versatile framework for studying the properties of attention mechanisms in Transformers (Vaswani et al., 2017). The central idea is to match the attention output of a second inference to that of a reference inference, where the second inference is typically subject to additional constraints and therefore has less freedom to reproduce the reference output. AM has been applied to studying the expressiveness of prefix-tuning (Lam et al., 2026) as well as KV cache compression (Liberty et al., 2026; Zweiger et al., 2026). In the

[1] As opposed to simply using a language model to summarize the context.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

prefix-tuning setting, one solves for a prefix whose attention output matches that of the reference inference. In KV cache compression, one instead solves for, or optimizes, a shorter KV cache whose attention outputs match those produced by the original cache for a set of reference queries. Such matching provides a direct criterion for assessing whether the compressed cache preserves information relevant to those queries.

However, the existing realization of attention matching for KV cache compression introduces an additional per-key bias term, which is not part of the standard key-value representation used by many mainstream attention implementations. The bias is added to the softmax logits and therefore cannot be absorbed into the cached keys or values: the compressed cache is no longer simply a pair of key and value tensors, but a triple consisting of keys, values, and per-key biases. The authors note that PyTorch SDPA and FlexAttention support additive attention terms, allowing these biases to be supplied explicitly (Zweiger et al., 2026). To the best of our knowledge, however, FlashAttention does not expose an arbitrary per-key bias through its standard interface. In our reproduction using NVIDIA's KVPress library (Devoto et al., 2025), a standardized compression evaluation library containing more than 30 implemented baselines, we therefore had to modify the attention path to support an additional per-key logit term. This modification is specific to the attention matching baseline and is not required by the other compression methods in the library.

Our goal is to realize the attention matching principle without introducing this additional bias term. This formulation is motivated by the attention matching framework developed in the study of prefix-tuning (Lam et al., 2026), where the matching equations do not require an additional per-key bias. To distinguish our formulation from the attention matching used in prior KV cache compression, we refer to the bias-free formulation as **explicit attention matching**, which we formalize in Section 3. Throughout the paper, we use **AM** to denote the existing attention matching formulation of Zweiger et al. (2026).

We present **QREAM** (Pivoted **QR** and **E**xplicit **A**ttention **M**atching), a method that matches the attention outputs of a set of reference queries by solving a system of linear equations for the value cache. For key selection, we apply pivoted QR to value-weighted attention columns, selecting keys whose attention contributions are linearly independent rather than simply those with the largest attention mass. This places our selection strategy between inexpensive attention-magnitude ranking and the substantially more expensive orthogonal matching pursuit. Finally, we construct the matching equations on-policy and account for each reference token's attention to its predecessors, so that the resulting system directly represents the readout performed by the model under compression. Whenever this system is solvable, the resulting compressed cache reproduces the reference continuation token-by-token.

Empirically, QREAM is competitive with attention matching while being simpler to deploy. Across three Qwen3 (Yang et al., 2025) model scales, three long-context QA benchmarks, and five compression ratios, with both methods using the same reference queries and per-head budget schedules, QREAM outperforms attention matching in a clear majority of experimental settings. The advantage is most pronounced at aggressive compression: at $100\times$ compression, QREAM performs better in eight of the nine (model, benchmark) combinations, whereas attention matching is more competitive at milder compression ratios. Both attention-matching methods substantially outperform the token-scoring baselines considered in our experiments, particularly on generative QA. At $100\times$ compression, for example, all eviction-based methods exhibit near-zero F1, while attention-matched caches retain most of their performance. This highlights a key limitation of token eviction under extreme compression: retaining high-scoring tokens does not necessarily preserve the information required to answer future queries.

Our contributions are summarized as follows:

- We introduce **explicit attention matching** for KV cache compression, yielding a compression method that preserves the attention-matching principle while remaining compatible with standard key-value cache representations and inference implementations.
- We develop a key selection method based on pivoted QR applied to value-weighted attention columns, selecting keys according to the linear independence of their attention contributions rather than attention magnitude alone.
- We conduct extensive experiments across three Qwen3 model scales, three long-context QA benchmarks, and five compression ratios, demonstrating that QREAM is competitive

with attention matching and substantially outperforms token-scoring baselines, particularly under aggressive compression.

2 Related Work

We group prior work according to the mechanism used to reduce the KV cache. Within each group, we highlight representative methods, most of which are also included as baselines in our experiments.

Attention-score eviction. A dominant approach is to retain a subset of cached entries according to their attention scores. H2O (Zhang et al., 2023) accumulates attention over previously generated tokens and retains heavy hitters together with a recent window. SnapKV (Li et al., 2024) estimates future importance from an observation window at the end of the prompt, exploiting the observation that attention patterns in this window predict which positions will be accessed during generation. ExpectedAttention (Devoto et al., 2025) instead estimates the expected attention a key will receive from future queries using the distributional statistics of observed queries, providing a query-agnostic importance score without relying on a particular future-query proxy. TOVA (Oren et al., 2024) uses the attention of the most recently processed token to score cached entries and greedily evicts those with the lowest scores. These methods differ in how they construct their importance scores, but share the premise that attention observed from past or proxy queries provides a useful signal for future utility.

Structural and position-based eviction. Another group selects entries using their positions or intrinsic properties of the cached representations rather than attention scores. StreamingLLM (Xiao et al., 2024) retains a fixed set of initial tokens together with a sliding window of recent tokens, motivated by the observation that initial tokens can act as attention sinks independently of their semantic content. K-Norm (Devoto et al., 2024) ranks cached keys according to their ℓ_2 norms. LagKV (Liang et al., 2025) partitions the sequence and estimates the importance of each partition by comparing it with the subsequent partition, avoiding explicit attention computation.

Budget allocation across layers and heads. Rather than determining which entries to retain, another line of work determines how a fixed cache budget should be distributed across the model. These strategies are largely orthogonal to the token-selection rule and can in principle be combined with different eviction methods. PyramidKV (Cai et al., 2025) allocates more cache to lower layers and less to higher layers based on its observation of progressively concentrated attention patterns across depth. Ada-KV (Feng et al., 2025) instead allocates the cache budget adaptively across attention heads, using an attention-output loss bound to guide the allocation.

Reconstruction and optimization-based methods. A separate line of work seeks to preserve the behavior of the original cache more directly rather than ranking entries solely by an eviction score. KVzip (Kim et al., 2025) performs context reconstruction to identify KV entries whose removal least affects the reconstruction objective, yielding a query-agnostic compression procedure. CurDKV (Sengupta et al., 2025) uses an approximated CUR decomposition to select keys and values based on leverage scores of the value-guided attention structure, explicitly accounting for the contribution of value vectors rather than attention scores alone. AM (Zweiger et al., 2026) takes a more direct functional approach: it constructs a shorter KV cache whose attention outputs match those of the original cache for a set of reference queries. Our method belongs to this latter line, but replaces the auxiliary bias used by AM with explicit attention matching and introduces a QR-based key-selection procedure.

3 Method

3.1 Explicit Attention Matching

Compression is successful if the cache can achieve the exact same thing as the context. We make this precise before introducing any machinery. Let $\mathrm{Attn}(\boldsymbol{q}; \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}\left(\boldsymbol{q}^\top \boldsymbol{K}^\top / \sqrt{d}\right) \boldsymbol{V}$ denote the attention read-out of a single query against a cache. Writing $(\boldsymbol{K}, \boldsymbol{V})$ for the full context

cache and $(\widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}})$ for the compressed one, we ask that

$$\boxed{\mathrm{Attn}(\boldsymbol{q};\, \boldsymbol{K}, \boldsymbol{V}) \;=\; \mathrm{Attn}\Big(\boldsymbol{q};\, \widetilde{\boldsymbol{K}}, \widetilde{\boldsymbol{V}}\Big) \qquad \text{for every probe query } \boldsymbol{q}.} \tag{1}$$

Let p denote the length of the original cache and $m \ll p$ the target compressed length, typically specified by the user.The core idea of attention matching, when applied to the problem of KV cache compression, is to satisfy equation 1 for as many relevant future queries as possible under this budget.

To lay out the equations in more details, we follow the notation of the explicit attention matching literature (Lam et al., 2026). Let H be the number of attention heads and h a head index; let $\boldsymbol{W}_Q^h, \boldsymbol{W}_K^h \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hidden}}}$ and $\boldsymbol{W}_V^h$ be the query, key and value matrices of head h, and let $T > 0$ be the inverse temperature. We write the exponentiated score

$$E_h(\boldsymbol{x}, \boldsymbol{y}) \;=\; \exp\!\left(\frac{T}{\sqrt{d_{\text{hidden}}}} \left(\boldsymbol{W}_Q^h \boldsymbol{x}\right)^\top \left(\boldsymbol{W}_K^h \boldsymbol{y}\right) \right), \tag{2}$$

so that the attention read-out of a query over a set of cached positions is the E_h-weighted mean of the corresponding values.

Fix a layer and a key-value head. The context occupies p cached positions with keys $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_p$ and values $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p$. Let $\boldsymbol{q} = (\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_n)$ be a **reference query**[2]. By a slight abuse of notation, we also use $\boldsymbol{q}$ to represent their hidden states at an attention head. The attention output of $\boldsymbol{q}$ at the last token $\boldsymbol{q}_n$ with the original context is

$$\boldsymbol{a}(\boldsymbol{q}_n) \;=\; \frac{\sum_{j=1}^{p} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j)\, \boldsymbol{v}_j + \sum_{j=1}^{n} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j^{\boldsymbol{q}}) \boldsymbol{v}_j^{\boldsymbol{q}}}{\sum_{j=1}^{p} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j) + \sum_{j=1}^{n} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j^{\boldsymbol{q}})}. \tag{3}$$

where $\boldsymbol{k}^{\boldsymbol{q}}$ and $\boldsymbol{v}^{\boldsymbol{q}}$ represent the keys and values of $\boldsymbol{q}$. The goal of **explicit attention matching** is to solve for keys $\widetilde{\boldsymbol{K}} = (\widetilde{\boldsymbol{k}}_1, \ldots, \widetilde{\boldsymbol{k}}_m)$ and values $\widetilde{\boldsymbol{V}} = (\widetilde{\boldsymbol{v}}_1, \ldots, \widetilde{\boldsymbol{v}}_m)$ such that

$$\frac{\sum_{j=1}^{p} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j)\, \boldsymbol{v}_j + \sum_{j=1}^{n} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j^{\boldsymbol{q}}) \boldsymbol{v}_j^{\boldsymbol{q}}}{\sum_{j=1}^{p} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j) + \sum_{j=1}^{n} E_h(\boldsymbol{q}_n, \boldsymbol{k}_j^{\boldsymbol{q}})} \;=\; \frac{\sum_{j=1}^{m} E_h(\widetilde{\boldsymbol{q}}_n, \widetilde{\boldsymbol{k}}_j)\, \widetilde{\boldsymbol{v}}_j + \sum_{j=1}^{n} E_h(\widetilde{\boldsymbol{q}}_n, \widetilde{\boldsymbol{k}^{\boldsymbol{q}}}_j) \widetilde{\boldsymbol{v}^{\boldsymbol{q}}}_j}{\sum_{j=1}^{m} E_h(\widetilde{\boldsymbol{q}}_n, \widetilde{\boldsymbol{k}}_j) + \sum_{j=1}^{n} E_h(\widetilde{\boldsymbol{q}}_n, \widetilde{\boldsymbol{k}^{\boldsymbol{q}}}_j)}. \tag{4}$$

Here, we use $\widetilde{\boldsymbol{q}}$ on the right hand side to represent the hidden vector of $\boldsymbol{q}$ that is computed **using the compressed context from the previous layer.** If the equality is achieved at all heads and all layers, then the next token prediction with $\boldsymbol{q}$ using either the original context or compressed context are guaranteed to be identical.

The left-hand side of equation 4 is fully determined once the reference query and model parameters are fixed. As in prior work, we restrict the compressed keys to a subset of the original keys,

$$\widetilde{\boldsymbol{K}} = \boldsymbol{K}_{\mathcal{I}}, \qquad \mathcal{I} \subseteq \{1, \ldots, p\}, \qquad |\mathcal{I}| = m.$$

Once $\mathcal{I}$ and the corresponding compressed query states are fixed, equation 4 becomes a linear system in the compressed values $\widetilde{\boldsymbol{V}}$. This allows the values to be solved efficiently.

The formulation differs from the attention matching (AM) method of Zweiger et al. (2026). AM introduces an additional per-key bias and separately matches the attention numerator and denominator. In contrast, our explicit attention matching directly matches the normalized attention output in equation 4, following the formulation used in prior work on attention matching for prefix-tuning (Lam et al., 2026). This eliminates the need for an auxiliary per-key bias.

Section 3.2 describes how we construct and solve the resulting linear system. Section 3.3 presents our key-selection procedure.

3.2 Solve for value via explicit attention matching

Although equation 4 is linear in the compressed values once the keys and query states are fixed, several subtleties arise when constructing the system in practice. We discuss these below.

[2]The precise meaning of the term will be elaborated in Section 3.2; it can be understood as a sequence of tokens for now.

The use of reference queries. Although it would be desirable to satisfy equation 1 for every possible query $\boldsymbol{q}$, this is infeasible in practice without retaining the original context. We therefore follow the common approach of using a small set of specially designed **reference probes** that aim to extract information likely to be important for future queries.

Before introducing the reference probes, we first distinguish three related concepts: **reference probes**, **reference continuations**, and **reference queries**. A reference probe is a predefined prompt appended to the original context to elicit potentially important information from it. Examples include "summarize the context", "repeat every token", and "list all numerical information." When the original model is prompted in this way, it produces a sequence of output tokens, which we call the **reference continuations**. We consider the concatenation of the reference probe with the reference continuation, including possible truncations at the end, and use the corresponding hidden states as the basis for constructing equation 4. We call these hidden states **reference queries**. Thus, if the concatenated sequence contains L tokens, there are L corresponding reference queries, one for each token position.

For example, suppose the context is "The sky is blue" and the reference probe is "summarize this". Suppose the original model produces the reference continuation "sky = blue." Explicit attention matching then seeks to preserve the attention computation at each step of this continuation. The five reference queries correspond to the one prefill step and four decoding/generation steps

$$\begin{aligned}
\text{compressed_context, "summarize"} &\rightarrow \text{"this"},\\
\text{compressed_context, "summarize", "this"} &\rightarrow \text{"sky"},\\
\text{compressed_context, "summarize", "this", "sky"} &\rightarrow \text{"="},\\
\text{compressed_context, "summarize", "this", "sky", "="} &\rightarrow \text{"blue"},\\
\text{compressed_context, "summarize", "this", "sky", "=", blue"} &\rightarrow \text{"EOS"}.
\end{aligned}$$

Although the first reference query corresponds to a prefill position rather than a generation step, we also include it in the matching equations because it provides useful information about how the compressed context should behave. More generally, each token position gives rise to one vector-valued equation of the form equation 4.

For the source of reference probes, we adopt the `ss-plus-repeat` set from AM (Zweiger et al., 2026): a *repeat* component, in which the context is replayed behind an instruction to reproduce it and the queries the replayed tokens generate are collected, together with a *self-study* component, in which the model is prompted to summarize the context, to aggregate its key facts, to emit a structured digest, and to pose and then answer its own questions about it.

An important issue raised by AM is the **on-policy** concern. Strictly speaking, equation 4 aims to reproduce the model's response to the reference queries under the original context. Once the context is compressed, however, subsequent hidden states are computed with respect to the compressed context rather than the original one. This creates a potential distribution mismatch between the reference queries used for matching and the queries that would actually arise during inference with the compressed context. To address this issue, we recompute the left-hand side of equation 4 using the compressed context constructed up to the current layer.

The overall procedure therefore consists of two phases:

1. **Phase 1: Reference generation.** We run the model with the reference probes and record the resulting reference continuation token IDs.
2. **Phase 2: Layer-wise compression.** We run the model again using the full reference-query sequences, which can be evaluated in a single forward pass layer by layer. After completing the inference of layer ℓ, we solve for the compressed context at layer ℓ. The resulting compressed representations are then used when evaluating layer $\ell + 1$.

The use of GQA. Under grouped-query attention (GQA), multiple query heads share a single key-value head. Consequently, a key-value head must satisfy the matching equations induced by each query head that attends to it. If a total of M_0 equations are generated across all reference probes (and their continuations) and each key-value head is shared by G query heads, then each key-value head has to satisfy $M = GM_0$ attention-matching equations.

Ridge regularization. With $\widetilde{\boldsymbol{K}}$ fixed, equation 4 is an overdetermined linear system in $\widetilde{\boldsymbol{V}}$. Solving it exactly is likely to be infeasible; even if a solution exists, it might be overfitting to the reference probe queries. We therefore solve

$$\widetilde{\boldsymbol{V}} \;=\; \arg\min_{\boldsymbol{X}} \|\boldsymbol{A}\boldsymbol{X}-\boldsymbol{B}\|_F^2 \;+\; \lambda\,\|\boldsymbol{X}-\boldsymbol{V}_{\mathcal{I}}\|_F^2 \;=\; \left(\boldsymbol{A}^\top\boldsymbol{A}+\lambda\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\top\boldsymbol{B}+\lambda\boldsymbol{V}_{\mathcal{I}}\right), \tag{5}$$

where $\boldsymbol{A}\boldsymbol{X}-\boldsymbol{B}=\boldsymbol{0}$ represents the original explicit attention matching system. We are deliberately shrinking not toward zero but toward the retained keys' *own* values, where $\mathcal{I}$ denotes the set of indices of the chosen keys $\widetilde{\boldsymbol{K}}$. As $\lambda\to\infty$, the solution approaches $\boldsymbol{V}_{\mathcal{I}}$, recovering ordinary eviction. We use $\lambda=10$ throughout and solve in double precision.

3.3 Key Selection by pivoted QR

Section 3.2 takes the index set $\mathcal{I}$ as given and we now describe how it is chosen. Once $\mathcal{I}$ is fixed, the solve of equation 5 can only produce vectors lying in the row space spanned by the retained keys' attention columns; keys whose columns are linearly dependent on those already selected therefore add nothing the solve can use, however much attention they individually receive. This suggests selecting for *span* rather than for magnitude.

The score matrix. Fix a layer and a key-value head, and let $\boldsymbol{q}_1,\ldots,\boldsymbol{q}_M$ be the reference query states of §3.2 where M is also the total number of matching equations. Collect the exponentiated scores of equation 2 into

$$\boldsymbol{E}\in\mathbb{R}^{M\times p}, \qquad E_{sj} \;=\; E_h(\boldsymbol{q}_s,\boldsymbol{k}_j), \tag{6}$$

so that column j records how every reference query responds to key j.

Value weighting. As a heuristic, we prioritize index who possess a relatively unique vector. We therefore scale each column by how far its value departs from the mean,

$$\boldsymbol{E}^{\mathrm{sel}}_{:,j} \;=\; w_j\,\boldsymbol{E}_{:,j}, \qquad w_j \;=\; \frac{\|\boldsymbol{v}_j-\bar{\boldsymbol{v}}\|_2}{\frac{1}{p}\sum_{i=1}^{p}\|\boldsymbol{v}_i-\bar{\boldsymbol{v}}\|_2}, \qquad \bar{\boldsymbol{v}}=\tfrac{1}{p}\sum_{i=1}^{p}\boldsymbol{v}_i, \tag{7}$$

which aligns the quantity selection optimizes with the quantity the solve minimizes.

Greedy pivoting. We then run column-pivoted QR on $\boldsymbol{E}^{\mathrm{sel}}$. Writing $\boldsymbol{R}$ for the running residual, initialized to $\boldsymbol{E}^{\mathrm{sel}}$, each step selects

$$j^\star \;=\; \arg\max_{j\notin\mathcal{I}\cup\mathcal{F}}\left\|\boldsymbol{R}_{:,j}\right\|_2, \tag{8}$$

appends it to $\mathcal{I}$, and deflates its span from the residual, $\boldsymbol{R}\leftarrow\boldsymbol{R}-\boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{R}$ with $\boldsymbol{Q}$ an orthonormal basis of the selected columns. Here $\mathcal{F}$ is the set of the first $n_{\mathrm{sink}}=4$ positions, which are retained unconditionally as attention sinks.

4 Experiments

4.1 Models

We evaluate on three scales of the Qwen3 family (Yang et al., 2025): Qwen3-0.6B, Qwen3-4B and Qwen3-8B.

4.2 Benchmarks

We use three long-context QA benchmarks: QuALITY (Pang et al., 2022), LongHealth (Adams et al., 2024), and QASPER (Dasigi et al., 2021). Compression is applied once per context, and every question associated with that context is then answered from the single compressed cache; this is the one-shot, question-agnostic protocol used by prior work. We subsample QuALITY to 50 contexts. LongHealth and QASPER are used in full.

We report five compression ratios – 100×, 50×, 20×, 10× and 5× – corresponding to retaining 1%, 2%, 5%, 10% and 20% of the cache.

Model	Method	100×	50×	20×	10×	5×
Qwen3-0.6B	QREAM (OURS)	**41.22**	**41.67**	**42.34**	**42.12**	40.99
	Attention Matching	25.23	26.35	25.45	25.00	25.00
	CurDKV	33.56	31.98	34.68	37.16	37.84
	KVzip	33.56	33.78	37.16	40.09	**42.34**
	ExpectedAttention	30.86	32.66	34.23	34.01	36.04
	LagKV	33.33	36.94	36.04	36.94	41.67
	StreamingLLM	36.04	35.81	35.81	37.16	38.51
	TOVA	35.36	35.36	36.71	38.51	40.09
	Ada-KV	21.17	26.13	35.59	37.61	37.61
	SnapKV	21.17	27.03	34.91	37.39	40.99
Qwen3-4B	QREAM (OURS)	54.28	58.78	**62.39**	**63.96**	64.64
	Attention Matching	**56.08**	**59.01**	**62.39**	63.74	**65.09**
	CurDKV	44.82	47.30	52.25	58.11	63.51
	KVzip	40.54	40.54	46.17	57.88	64.19
	ExpectedAttention	41.22	43.69	47.97	52.25	60.81
	LagKV	41.89	43.69	47.52	49.10	55.86
	StreamingLLM	43.02	43.92	45.05	45.72	50.00
	TOVA	44.37	44.82	44.59	47.52	52.48
	Ada-KV	27.25	33.78	46.17	48.65	54.73
	SnapKV	29.50	35.14	46.40	47.97	51.13
Qwen3-8B	QREAM (OURS)	**63.51**	**65.32**	**66.89**	**67.57**	68.02
	Attention Matching	57.88	62.16	**66.89**	67.34	68.69
	CurDKV	49.77	54.95	62.39	66.67	67.34
	KVzip	45.05	44.82	51.35	66.22	**70.27**
	ExpectedAttention	47.30	48.65	50.45	59.91	65.77
	LagKV	46.40	47.75	52.03	55.18	63.06
	StreamingLLM	47.75	50.45	53.60	55.18	58.78
	TOVA	50.90	51.35	52.25	56.53	59.91
	Ada-KV	31.31	44.82	53.38	56.08	60.59
	SnapKV	31.53	42.57	54.05	56.31	60.14

Table 1: QuALITY (Acc.). **Bold** is best per model and ratio.

4.3 BASELINES

We compare against a large group of training-free baselines available in 'kvpress': KVzip Kim et al. (2025), SnapKV (Li et al., 2024), ExpectedAttention (Devoto et al., 2025), CurDKV (Sengupta et al., 2025), StreamingLLM (Xiao et al., 2024), TOVA (Oren et al., 2024), Ada-KV (Feng et al., 2025), and LagKV (Liang et al., 2025). We also compare with AM (Zweiger et al., 2026), which is the method we most want to compare against. Precisely, we compare against the AM-HighestAttnKeys variant for Qwen3-4B and AM-HighestAttnKeys-fast variant for 0.6B and 8B, which selects keys by root-mean-square attention received over the reference queries and fits β by non-negative least squares.

AM depends crucially on its per-model precomputed schedule, but only the schedules for Qwen3-4B are publicly available. We therefore generated native schedules for Qwen3-0.6B and Qwen3-8B using the authors' own optimizer. The published Qwen3-4B schedules use 10 articles for schedule optimization; we use 3 articles to derive the schedules for 0.6B and 8B for the sake of efficiency. Both QREAM and AM use the same schedule on each model, so the comparison is symmetric.

The entire evaluation is completed using NVIDIA's KVPress framework (Devoto et al., 2025). Most of the baselines we compared with are already implemented in the library. We implement both QREAM and AM into their framework for evaluation.

4.4 RESULTS

Several consistent trends emerge across the nine model-benchmark combinations.

Model	Method	100×	50×	20×	10×	5×
Qwen3-0.6B	QREAM (OURS)	**36.25**	**39.25**	**41.25**	**45.00**	**48.50**
	Attention Matching	19.50	19.50	22.50	24.50	24.50
	CurDKV	28.25	26.50	24.00	30.50	37.75
	KVzip	33.50	34.00	35.50	42.50	48.00
	ExpectedAttention	27.00	27.25	30.50	32.75	35.25
	LagKV	30.50	30.25	30.00	34.00	37.75
	StreamingLLM	31.00	31.50	32.75	30.50	35.00
	TOVA	30.50	27.75	29.00	30.75	30.00
	Ada-KV	30.50	30.75	30.50	30.00	34.75
	SnapKV	30.25	30.75	31.50	31.50	32.50
Qwen3-4B	QREAM (OURS)	**57.50**	61.75	65.25	71.00	74.75
	Attention Matching	54.00	**65.25**	**70.00**	**72.75**	74.00
	CurDKV	39.25	42.50	44.25	53.25	67.25
	KVzip	35.50	43.00	60.50	68.00	**77.25**
	ExpectedAttention	36.00	37.75	40.25	46.75	63.75
	LagKV	37.25	39.25	45.00	48.75	58.50
	StreamingLLM	36.00	37.50	37.25	37.25	47.75
	TOVA	36.25	36.50	38.75	40.00	44.75
	Ada-KV	36.50	38.00	40.75	44.00	46.00
	SnapKV	37.00	39.50	38.75	44.00	45.25
Qwen3-8B	QREAM (OURS)	**62.50**	64.50	72.00	76.75	77.75
	Attention Matching	56.25	**69.50**	**79.75**	**80.00**	79.75
	CurDKV	42.25	43.50	53.25	63.25	72.00
	KVzip	40.75	47.25	68.00	78.50	**81.25**
	ExpectedAttention	42.50	41.50	45.25	59.00	70.25
	LagKV	39.50	39.75	43.00	49.75	63.25
	StreamingLLM	38.00	39.00	36.75	41.75	48.75
	TOVA	40.75	38.00	38.00	41.50	43.00
	Ada-KV	38.75	39.75	40.25	43.00	49.75
	SnapKV	38.00	39.75	39.00	41.25	48.50

Table 2: LongHealth (Acc.). **Bold** is best per model and ratio.

At extreme compression QREAM leads. At 100× QREAM achieves the highest score among all evaluated methods in eight of nine model-benchmark combinations, the exception being Qwen3-4B QuALITY, where AM leads by 1.80. The margins over AM at 100× on Qwen3-8B are 6.25 (LongHealth), 5.63 (QuALITY) and 9.71 (QASPER).

At mild compression the methods converge, and eviction becomes competitive. By 5× KVzip wins or ties five of nine combinations, at a fraction of the compute – on Qwen3-8B LongHealth it reaches 81.25 against 77.75 for QREAM and 79.75 for AM. Value re-fitting earns its cost where the budget is scarce and stops earning it where it is not.

The experimental results demonstrate that QREAM is competitive with a range of existing baselines, including AM. While QREAM does not outperform AM in all settings, it provides a bias-free framework that is more compatible with standard attention implementations. Its particularly strong performance at high compression ratios also suggests the potential for combining the two approaches in domain-specific applications. More broadly, many components of the two frameworks can be independently selected and combined, allowing users to choose or adapt the components that best suit their specific use cases.

5 CONCLUSIONS

We presented QREAM, a KV cache compression method that combines explicit attention matching with pivoted QR key selection. By directly matching attention outputs without introducing auxiliary bias terms, QREAM retains the core information-preservation objective of attention matching while using the standard key-value cache representation. Its QR-based key selection identifies a compact set of keys that captures diverse attention contributions, enabling effective reconstruction

Model	Method	100×	50×	20×	10×	5×
Qwen3-0.6B	QREAM (OURS)	**13.57**	10.41	13.02	14.66	15.55
	Attention Matching	10.51	**12.96**	13.32	12.76	12.73
	CurDKV	9.36	10.21	12.84	14.36	14.16
	KVzip	7.52	8.27	10.78	**15.38**	13.63
	ExpectedAttention	12.00	11.92	11.13	10.20	11.10
	LagKV	8.06	9.30	12.66	12.24	10.56
	StreamingLLM	12.82	12.82	**13.58**	14.26	**17.72**
	TOVA	12.33	12.01	11.05	11.33	9.83
	Ada-KV	2.04	8.32	10.71	11.71	11.35
	SnapKV	2.25	9.00	11.22	11.08	11.95
Qwen3-4B	QREAM (OURS)	**28.92**	**33.07**	36.50	39.58	40.50
	Attention Matching	22.53	27.24	**38.33**	**42.06**	**40.82**
	CurDKV	14.51	18.95	21.44	29.34	31.23
	KVzip	6.02	5.81	7.07	35.04	39.12
	ExpectedAttention	17.23	19.79	21.43	28.01	38.62
	LagKV	5.85	6.50	12.25	17.65	23.41
	StreamingLLM	11.63	12.94	13.96	16.22	21.65
	TOVA	13.64	11.36	10.22	14.14	20.34
	Ada-KV	4.21	12.23	11.91	19.26	25.93
	SnapKV	3.35	11.31	13.05	17.44	25.51
Qwen3-8B	QREAM (OURS)	**27.93**	**32.96**	**35.96**	**37.66**	41.99
	Attention Matching	18.22	25.16	29.95	35.66	38.88
	CurDKV	15.90	18.50	26.66	28.18	37.12
	KVzip	5.95	5.80	9.25	37.25	**42.18**
	ExpectedAttention	20.82	21.75	22.09	31.66	40.21
	LagKV	6.00	7.14	11.95	21.94	27.65
	StreamingLLM	10.09	9.48	13.26	16.06	20.83
	TOVA	14.86	11.31	10.11	14.49	24.73
	Ada-KV	6.80	9.52	12.99	22.83	27.12
	SnapKV	6.73	8.80	10.79	20.91	26.83

Table 3: LongBench-QASPER (F1). **Bold** is best per model and ratio.

of the corresponding values. Experiments across model scales, benchmarks, and compression ratios demonstrate that QREAM remains effective under substantial cache reduction and is particularly competitive in the highly compressed regime. Furthermore, our results highlight the potential of attention matching as a general principle for KV cache compression. We hope this work will inspire further applications of attention matching in KV cache compression and related areas.

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing and to draft sections of the paper. We also used generative AI for retrieval and discovery, for research ideation or execution. It is not used for for generating synthetic datasets and for proving mathematical claims. We have reviewed all AI-assisted work, including generated text and code. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo JWL Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. Longhealth: A question answering benchmark with long clinical documents. *arXiv preprint arXiv:2401.14490*, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. In *Conference on Language Modeling*, 2025.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the*

2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4599–4610, 2021.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for KV cache compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: Kv cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems*, 2025.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *Advances in Neural Information Processing Systems*, 2025.

Peter Cho-Ho Lam, Ziyi Wang, and Zirui Zhou. Prefix-tuning for arbitrary output sequences on pretrained transformers. In *Advances in Neural Information Processing Systems*, 2026. To appear.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024.

Manlai Liang, JiaMing Zhang, Xiong Li, and Jinlong Li. Lagkv: Lag-relative information of the kv cache tells which tokens are important. *arXiv preprint arXiv:2504.04704*, 2025.

Edo Liberty, Alexandr Andoni, and Eldar Kleiner. Nearly optimal attention coresets. *arXiv preprint arXiv:2605.05602*, 2026.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In *Empirical Methods in Natural Language Processing*, pp. 18724–18741, 2024.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. QuALITY: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022.

Ayan Sengupta, Siddhant Chaudhary, and Tanmoy Chakraborty. Value-guided KV compression for LLMs via approximated CUR decomposition. In *Advances in Neural Information Processing Systems*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via Attention Matching. *International Conference on Machine Learning*, 2026.