LATCHQ: LATCHING ONTO HOT-CACHE QUERY INTERACTIONS FOR KV CACHE QUANTIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

KV cache quantization reduces the memory footprint of large language model (LLM) inference, but minimizing reconstruction error alone may fail to preserve attention behavior. Although future queries are unavailable at quantization time, keys can interact with subsequent queries while residing in the full-precision hot cache. Motivated by this observation, we introduce ***LatchQ***, which uses these observed interactions to guide key clipping while preserving the baseline bit allocation. *QTrace* accumulates attention-weighted query energy for each key and channel, preserving differences in observed key usage and channel sensitivity. *QClip* leverages this evidence to select shared clipping ranges by minimizing interaction-weighted reconstruction error. Our method preserves the baseline storage format and cache schedule without extending hot-cache residency. Our analysis reveals 63% agreement between clipping range selections based on hot-cache observations and those based on both observed and future queries. Furthermore, ***LatchQ*** achieves the highest average LongBench scores among evaluated KV quantizers at 3.0 and 2.25 bits per dimension (bpd). It also outperforms the baseline quantizer on reasoning and code generation. Code is included in the supplementary material, and the project page is available at Project Page.

1 INTRODUCTION

As autoregressive inference extends to longer contexts and more demanding reasoning workloads, KV cache memory becomes a major bottleneck. KV cache quantization has therefore emerged as an important approach to efficient LLM inference. Recent methods have further refined compression decisions by exploiting runtime attention and query statistics (He et al., 2024; Zhang et al., 2026; Yang et al., 2024b; Ramachandran et al., 2026), quantization sensitivity (Liu et al., 2026; Hooper et al., 2024), and the structural properties of Rotary Positional Embeddings (RoPE) (Wang et al., 2026). Many query-aware methods use importance signals to allocate precision across KV entries or channels (Yang et al., 2024b; He et al., 2024; Zhang et al., 2026). For example, MixKVQ (Zhang et al., 2026) combines query-channel magnitudes with key quantization scales to determine channel precision. Other methods incorporate query statistics into quantization objectives or representations (He et al., 2025; Wang et al., 2025).

In streaming quantizers that retain recent KV entries in a full-precision (FP) window, each entry remains in this hot cache before being quantized and moved to a compressed cold cache. During this interval, a key interacts with subsequent queries, providing evidence of its observed use within the existing cache schedule.

We focus on key clipping while keeping the baseline bit allocation and storage format fixed. At a given precision, range selection determines how reconstruction error is distributed across keys within each group and channel. Incorporating key-specific query interactions can help weight these errors according to observed key usage. However, observations collected before quantization reflect only a key's past use, whereas the compressed key will serve future queries. To determine whether these past observations remain informative after compression, we pose the following question:

Can a key's hot-cache interactions guide its quantization for future queries?

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.

We first examine whether the interactions observed in the hot cache provide useful evidence for quantization. In our empirical analysis, clipping guided by hot-cache interactions matches the all-query oracle in 63% of evaluated decisions and achieves NLL close to that of the oracle. Based on this observation, we propose ***LatchQ***, a two-component framework consisting of *QTrace* and *QClip*. *QTrace* accumulates query-interaction evidence for each key and channel during hot-cache residency, and *QClip* uses this evidence to select shared clipping ranges based on weighted reconstruction error. This preserves the baseline bit allocation, storage format, and cache schedule. Our contributions are summarized as follows.

- **Hot-Cache Evidence for Future Queries:** We demonstrate that query interactions observed during a key's hot-cache residency provide useful evidence for guiding clipping decisions for subsequent queries.
- **Key-Specific Interaction Summaries:** We develop *QTrace* to accumulate attention-weighted query energy for each key and channel, preserving differences in observed key usage and channel sensitivity. The resulting evidence can be reused across clipping candidates.
- **Interaction-Weighted Clipping:** We propose ***LatchQ***, which uses *QClip* to select shared clipping ranges by weighting reconstruction errors with *QTrace* evidence. Our framework preserves the baseline bit allocation, storage format, and cache schedule.
- **State-of-the-Art Low-Bit Performance:** To the best of our knowledge, ***LatchQ*** achieves the highest average LongBench scores among the evaluated KV quantization methods at 3.0 and 2.25 bits per dimension (bpd), alongside strong performance on reasoning and code generation benchmarks.

2 Rethinking Query-Guided KV Quantization

In this section, we focus on observed query interactions while KV entries remain in the hot cache before quantization. We first review how prior work uses query information. We then examine whether these interactions provide useful evidence for quantization decisions aligned with future queries.

2.1 How Query Information Is Used

Existing query-aware KV compression methods utilize query information in three broad ways. First, some allocate precision using importance signals derived from runtime attention or query relevance (Yang et al., 2024b; He et al., 2024; Ramachandran et al., 2026; Zhang et al., 2026). Second, others incorporate query statistics into the quantization objective or representation (Hooper et al., 2024; He et al., 2025; Wang et al., 2025). Third, several approaches use newly observed query information during generation to revise compression decisions, such as which KV entries to retain (Zhang et al., 2023; Yang et al., 2024a; Ghadia et al., 2025; Cai et al., 2026; He et al., 2024; Ramachandran et al., 2026). These paradigms are not mutually exclusive, and some frameworks combine multiple strategies. MixKVQ (Zhang et al., 2026), for example, combines query-channel magnitudes with key quantization scales to allocate channel precision during inference. AATC (Laus et al., 2026) derives an attention-aware distortion objective and uses calibration statistics to allocate bits across transformed channels. These examples illustrate how query information guides precision allocation through online observations or offline calibration. Beyond precision allocation, SQuat (Wang et al., 2025) uses a shared subspace constructed from prompt queries to guide key quantization.

Our focus lies in utilizing observed query interactions to evaluate clipping candidates at a fixed precision. While applying a uniform positive weight to every key's reconstruction error leaves the ranking of clipping candidates unchanged, key-specific weights can change this ranking by assigning heterogeneous importance to individual errors. ***LatchQ*** therefore accumulates query-interaction evidence for each key and channel while keys reside in the hot cache. At quantization time, this evidence weights the reconstruction errors to select a shared clipping range for each KV head, group, and channel. This effectively adapts range selection to observed key usage while strictly preserving the baseline bit allocation, storage format, and cache schedule.

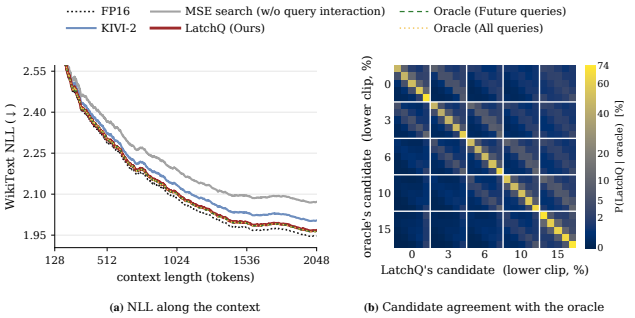
In the next subsection, we examine whether the interactions observed during hot-cache residence provide reliable evidence for clipping decisions evaluated under subsequent queries.

2.2 Query Evidence in the Recent Cache

In streaming inference, a key interacts with subsequent queries while it remains in the hot cache before quantization. These interactions provide evidence of how the key is used, including the attention it receives and the associated query-channel activations. At quantization time, this evidence can guide the evaluation of reconstruction errors beyond what is available from the key vector alone.

To examine whether this evidence remains useful for future queries, we compare ***LatchQ*** with MSE-Clip, labeled "MSE search (w/o query interaction)" in Figure 1. Both methods use the same clipping candidates.

MSE-Clip evaluates candidates using unweighted reconstruction error, whereas ***LatchQ*** weights these errors using observed query interactions. We also evaluate two oracles. The future-query oracle selects clipping ranges using queries at positions after each key is quantized, while the all-query oracle uses both observed and future queries.



(a) NLL along the context **(b)** Candidate agreement with the oracle

Figure 1: Impact of query interactions on streaming KV quantization. (a) WikiText NLL vs. context length. (b) Conditional probabilities of ***LatchQ***'s clipping choices given the all-query oracle's choices. Axes enumerate lower/upper clipping pairs in lexicographical order.

Figure 1(a) shows that MSE-Clip yields a higher negative log-likelihood (NLL) than KIVI-2, indicating that minimizing unweighted reconstruction error does not necessarily improve language modeling performance in this setting. By incorporating observed interactions, ***LatchQ*** reduces KIVI-2's NLL gap relative to FP16 by approximately 64% at 2,048 tokens. This relative reduction remains between 63% and 66% over context lengths of 512–2,048 tokens. Its NLL is also close to that of both oracles, despite using only interactions observed before quantization.

Figure 1(b) compares observed-query clipping selections with those of the all-query oracle using a shared FP16 reference trace (Appendix B.1). The concentration along the diagonal indicates that they frequently select the same clipping candidate. Across the evaluated KV head, group, and channel decisions, we measure an exact agreement rate of 63%. Together with the NLL results, this agreement suggests that interactions observed during hot-cache residence provide useful evidence for clipping decisions under subsequent queries.

These findings motivate treating hot-cache residency as an observation stage before quantization. A key's interactions during this interval provide evidence for selecting the representation used after compression. The existing cache schedule thus offers an opportunity to inform range selection through observed key usage, without extending residency or delaying quantization. This leads to the design question of how to summarize these interactions efficiently and incorporate the resulting evidence into clipping decisions. Experimental details are provided in Appendix B.1.

3 LatchQ

Building on the observations in Section 2.2, we propose ***LatchQ***, a novel approach that uses query interactions observed in the hot cache to guide key quantization. Specifically, we adapt clipping ranges within the baseline 2-bit per-channel group key quantizer, preserving the storage format and bit rate while leaving value quantization unchanged.

Our method consists of two core components. First, *QTrace* extracts interaction evidence from subsequent queries while each key is in the hot cache. At quantization time, *QClip* uses this evidence to select clipping ranges. Figure 2 summarizes the overall pipeline, and the following subsections describe each component in detail.

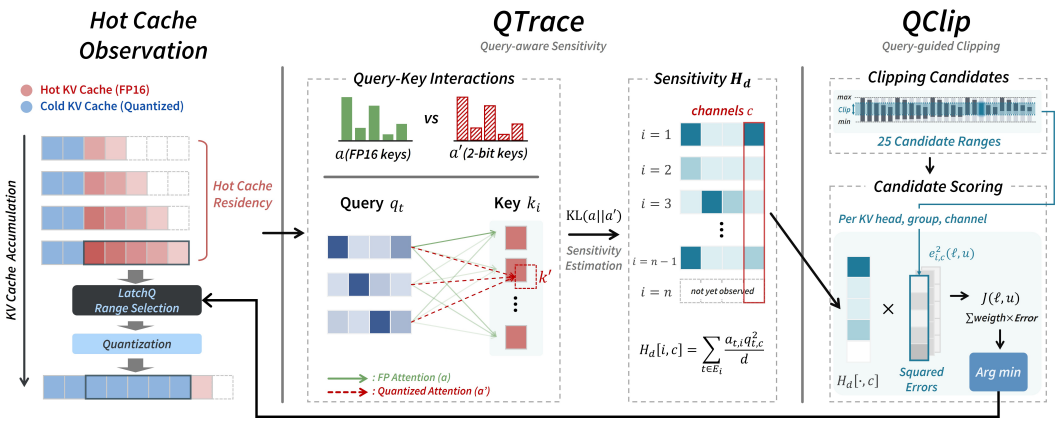


Figure 2: Overview of ***LatchQ***. (Left) Keys interact with subsequent queries during hot-cache residency. (Middle) *QTrace* accumulates attention-weighted query energy into key- and channel-specific statistics H_d. (Right) *QClip* uses this evidence to select clipping ranges that minimize interaction-weighted reconstruction error, then quantizes the keys for cold-cache storage.

3.1 *QTrace*: Capturing Query Interactions

QTrace constructs per-key and per-channel interaction evidence from the subsequent queries observed while each key resides in the hot cache. We use this evidence to evaluate each key's reconstruction error based on its observed query interactions. The core intuition is that even for reconstruction errors of the same magnitude, their impact on the attention distribution varies depending on which queries attend to the key and the attention weights they assign. To demonstrate this, we first consider the effect of altering a single key. For simplicity of notation, we omit the layer and head indices.

Consider the query and key vectors $\boldsymbol{q}_t, \boldsymbol{k}_i \in \mathbb{R}^d$ with Rotary Position Embedding (RoPE) applied. Let $\boldsymbol{a}_t$ represent the attention distribution of query t over its entire causal prefix including its own key, with $a_{t,i}$ being the specific attention probability assigned to key i. If we define the reconstruction error for key i as $\boldsymbol{e}_i = \hat{\boldsymbol{k}}_i - \boldsymbol{k}_i$, the resulting change in the attention logit is given by $\delta_{t,i} = \frac{\boldsymbol{q}_t^\top \boldsymbol{e}_i}{\sqrt{d}}$. Lemma 1 formalizes how this small logit shift affects the attention distribution.

Lemma 1 (Local impact of single-key reconstruction error)**.** *Assuming query t and the remaining keys are held constant, let $\boldsymbol{a}'_t$ denote the new attention distribution when only key i is perturbed to $\hat{\boldsymbol{k}}_i$. The second-order Taylor expansion of the KL divergence around $\delta_{t,i} = 0$ yields the following relationship, where $\mathcal{O}(|\delta_{t,i}|^3)$ denotes the remainder term of the expansion:*

$$\mathrm{KL}(\boldsymbol{a}_t \| \boldsymbol{a}'_t) = \frac{1}{2} a_{t,i}(1 - a_{t,i})\delta_{t,i}^2 + \mathcal{O}\big(|\delta_{t,i}|^3\big) = \frac{a_{t,i}(1 - a_{t,i})}{2d} \left(\boldsymbol{q}_t^\top \boldsymbol{e}_i\right)^2 + \mathcal{O}\big(|\delta_{t,i}|^3\big)\,. \quad (1)$$

As Lemma 1 indicates, the magnitude of a key's reconstruction error alone does not determine its local impact on the attention distribution. Rather, this impact varies depending on the inner product between the query and the error, as well as the assigned attention probability. This insight justifies integrating the interactions a key establishes with subsequent queries into its error assessment. Building upon this, we formulate a reconstruction objective that captures the interactions observed prior to the quantization of each key.

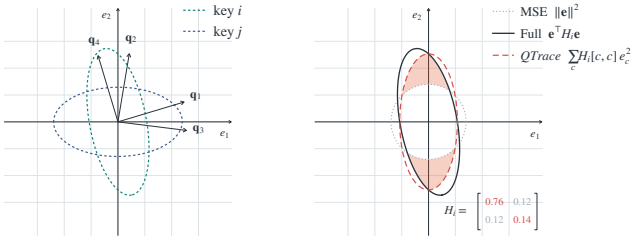
To evaluate the error of each channel independently, we apply a diagonal approximation to the quadratic term in Lemma 1, dropping the cross-channel terms. Furthermore, to weight the error proportionally to the attention, we replace $a_{t,i}(1 - a_{t,i})$ with $a_{t,i}$ and omit the common constant $1/2$, as it does not affect candidate comparisons (Appendix A.1.1).

For each observed query, summing the weighted squared errors across channels yields $\sum_c \frac{a_{t,i} q_{t,c}^2}{d} e_{i,c}^2$, which evaluates the key's reconstruction error. By accumulating this term over observed queries before quantization, we define the query–key interaction-weighted error $\widetilde{D}_{h,i}$:

$$\widetilde{D}_{h,i} := \sum_{r\in\mathcal{G}(h)}\sum_{t\in\mathcal{E}_i}\sum_{c=1}^{d}\frac{a_{t,i}^{(r)}(q_{t,c}^{(r)})^2}{d}e_{h,i,c}^2 = \sum_{c=1}^{d}\underbrace{\left(\sum_{r\in\mathcal{G}(h)}\sum_{t\in\mathcal{E}_i}\frac{a_{t,i}^{(r)}(q_{t,c}^{(r)})^2}{d}\right)}_{H_d[h,i,c]}e_{h,i,c}^2. \tag{2}$$

Here, $\mathcal{G}(h)$ is the set of query heads sharing KV head h. $\mathcal{E}_i$ denotes the subsequent query positions used as evidence while key i is in the hot cache. The indices r and t range over these sets, respectively. Using the diagonal approximation $((q_t^\top e_{h,i})^2 \approx \sum_c q_{t,c}^2 e_{h,i,c}^2)$, $e_{h,i,c}^2$ can be factored out in Equation 2 as it is query-independent for a fixed candidate. Thus, the expression within the parentheses is determined entirely by the observed queries q and attention weights a. *QTrace* defines this accumulated weight as the interaction evidence $H_d[h,i,c]$, combining attention-based key usage with channel-wise sensitivity captured by squared query activations. These key-specific weights are independent of clipping candidates and can therefore be computed once and reused to evaluate shared clipping ranges.

The observation set $\mathcal{E}_i$ depends on the inference phase. During prefill phase, it consists of subsequent query positions within the prompt. During decoding phase, it includes queries collected from the block targeted for quantization. Detailed observation ranges and computation timings are provided in Appendix A.1.2.



(a) Key-specific error geometry (b) *QTrace* (Ours)

Figure 3: Geometric motivation for *QTrace*. (a) Key-specific error geometry. (b) *QTrace* retains only the diagonal terms of the full quadratic surrogate, whereas standard MSE uses an isotropic error metric.

Figure 3 illustrates key-specific error sensitivity and the diagonal approximation used in *QTrace*. A shared error metric may fail to capture the distinct error geometry of each key (Figure 3a). *QTrace* derives diagonal weights from each key's observed query interactions to capture channel-wise sensitivity overlooked by standard MSE (Figure 3b).

In the subsequent *QClip* section, we utilize H_d as a weighting factor to evaluate clipping candidates and select a clipping range for each quantization group and channel within each KV head.

3.2 *QClip*: Selecting Clipping Ranges

At quantization time, *QClip* uses the accumulated *QTrace* evidence to select a clipping range for each group and channel within each KV head. Each candidate range produces a quantized reconstruction of the keys in the group. We evaluate these reconstructions using the key-specific weights in H_d. To incorporate observed query interactions into the quantized representation, *QClip* constructs candidate clipping ranges, selects the range with the lowest *QTrace*-weighted reconstruction error, and quantizes the keys for storage.

Let $\mathcal{B}_g$ denote the G key positions in quantization group g. For each KV head h and channel c, let m and M denote the minimum and maximum key values within the group. We use α and β to specify the fractions of the original range $M-m$ removed from the lower and upper ends, respectively. We compute the resulting lower and upper clipping bounds, denoted by ℓ and u, as follows:

$$\ell(\alpha) = m + \alpha(M-m), \qquad u(\beta) = M - \beta(M-m). \tag{3}$$

We independently choose α and β from $\{0, 0.03, 0.06, 0.10, 0.15\}$, yielding 25 candidates. The pair $(\alpha,\beta) = (0,0)$ preserves the original min/max range. Each candidate range is shared by all keys in the group for the given head and channel. We obtain candidate reconstructions using 2-bit asymmetric quantization with the scale and minimum offset rounded to FP16. The reconstructed value for key i is denoted by $\widehat{k}_{h,i,c}(\ell,u)$.

Algorithm 1 *LatchQ* step (*QTrace* + scheduling)

Require: Query/key batch at positions $\mathcal{T}$; cache policy $\mathcal{P}$; group size G

Append new keys to $\mathcal{K}_{\text{hot}}$
$H_d[h, i, c] \leftarrow 0$ for new keys
Compute causal attention weights a
for each hot key i and (h, c) **do** ▷ *QTrace*
  $\mathcal{E}_i^{\text{new}} \leftarrow \{t \in \mathcal{T} \mid t > i\}$
  $H_d[h, i, c] \leftarrow H_d[h, i, c] + \sum_{r \in \mathcal{G}(h)} \sum_{t \in \mathcal{E}_i^{\text{new}}} \frac{a_{t,i}^{(r)} (q_{t,c}^{(r)})^2}{d}$
end for
$\mathcal{B} \leftarrow \text{ScheduledKeys}(\mathcal{K}_{\text{hot}}, \mathcal{P})$
if $\mathcal{B} \neq \emptyset$ **then**
  $\mathcal{Z} \leftarrow$ *QClip*$(K_{\mathcal{B}}, H_{d,\mathcal{B}}, G)$
  Store $\mathcal{Z}$ in $\mathcal{K}_{\text{cold}}$
  $\mathcal{K}_{\text{hot}} \leftarrow \mathcal{K}_{\text{hot}} \setminus \mathcal{B}$
end if
if prefill step **then**
  Reset H_d for remaining hot keys
end if

Algorithm 2 *QClip*

Require: Scheduled keys K, evidence H_d, group size G
Ensure: Encoded block $\mathcal{Z}$ with metadata

$\mathcal{R} \leftarrow (0, 0.03, 0.06, 0.10, 0.15)$
$\mathcal{Z} \leftarrow \emptyset$
$\{\mathcal{B}_g\} \leftarrow \text{Group}(K, G)$
$w \leftarrow \text{AddFloor}(H_d, \{\mathcal{B}_g\})$
for each (h, g, c) **do** ▷ *QClip*
  $\mathbf{x} \leftarrow K[h, \mathcal{B}_g, c]$
  $\mathcal{C} \leftarrow \text{Candidates}(\mathbf{x}, \mathcal{R})$
  $J(\ell, u) \leftarrow \sum_{i \in \mathcal{B}_g} w_{h,i,c} \big(k_{h,i,c} - \widehat{k}_{h,i,c}(\ell, u)\big)^2$
  $(\ell^\star, u^\star) \leftarrow \arg\min_{(\ell,u) \in \mathcal{C}} J(\ell, u)$
  $(\mathbf{z}, s, b) \leftarrow \text{Quantize}_{2,\text{FP16}}(\mathbf{x}, \ell^\star, u^\star)$
  APPEND$(\mathcal{Z}, (h, g, c, \mathbf{z}, s, b))$
end for
return $\mathcal{Z}$

Let $w_{h,i,c}$ denote the *QTrace* weight $H_d[h, i, c]$ after adding a small positive floor. The candidate score $J_{h,g,c}(\ell, u)$ is computed as follows:

$$J_{h,g,c}(\ell, u) = \sum_{i \in \mathcal{B}_g} w_{h,i,c} \left(k_{h,i,c} - \widehat{k}_{h,i,c}(\ell, u)\right)^2. \tag{4}$$

We independently select the lowest-scoring candidate per KV head, group, and channel. The original min/max range is evaluated first and retained in the event of equivalent scores.

Once the clipping range is selected, *QClip* quantizes the keys into 2-bit codes using the corresponding FP16 scale and minimum offset. These FP16 parameters are shared within each group for a given KV head and channel and stored alongside the codes in the cold cache. This incorporates observed query interactions into the quantized representation while preserving the baseline storage format and bit rate. Values use the baseline per-token 2-bit quantizer under the same cache schedule.

Algorithms 1 and 2 summarize the ***LatchQ*** pipeline. Initialized with empty caches, Algorithm 1 executes during prefill and each decoding step to accumulate query evidence and schedule keys. Because prompt-tail evidence is reset after prefill, these keys rely exclusively on decode-time evidence. For each scheduled block, Algorithm 1 passes keys $K_{\mathcal{B}}$ and evidence $H_{d,\mathcal{B}}$ to Algorithm 2. Using weights from `AddFloor` (Appendix A.2.1), it evaluates candidates in ascending (α, β) order from $(0, 0)$, favoring earlier candidates in ties. Finally, Algorithm 2 returns the encoded 2-bit codes and their FP16 metadata ($\mathcal{Z}$) to be stored in the cold cache.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

We evaluate ***LatchQ*** on eight LongBench (Bai et al., 2024) tasks using Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Qwen et al., 2025). Additional results on Mistral-7B-Instruct-v0.3 (Jiang et al., 2023) are provided in Appendix B.2. For reasoning and code generation, we evaluate Llama-3.1-8B-Instruct on GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), GPQA Diamond (Rein et al., 2024), and LiveCodeBench (Jain et al., 2025). We compare against KIVI-2 (Liu et al., 2024), KVQuant (Hooper et al., 2024), SQuat (Wang et al., 2025), and TurboQuant (Zandieh et al., 2026), with FP16 as the reference. ***LatchQ*** uses 2-bit key quantization and a 128-token FP16 hot cache, preserving the baseline value quantization and cache schedule. We target 3.0 and 2.25 bpd, with nearby rates for KVQuant and TurboQuant due to format differences. TurboQuant uses 0.5–1.0 extra bpd for a conservative comparison. Detailed experimental settings are provided in Appendix B.

Table 1: LongBench results on Llama-3.1-8B-Instruct and Qwen2.5-7B-Instruct at 3.0 and 2.25 bpd (3.3 and 2.3 bpd for KVQuant, and 4.0/3.75 and 3.0/2.75 bpd for TurboQuant). All results are obtained from our own evaluations. **Bold** and underlined values indicate the best and second-best scores among quantized methods, respectively.

Llama-3.1-8B-Instruct

bpd	Method	Qasper	QMSum	Multi-News	TREC	TriviaQA	SAMSum	LCC	RepoBench	Average (macro)
16	FP16	44.62	25.34	26.91	72.50	92.65	43.85	65.45	58.60	53.74
3.0	KIVI-2 *(ICML 2024)*	43.06	25.50	26.49	**72.50**	92.16	43.24	63.99	57.03	53.00
	KVQuant *(NeurIPS 2024)*	43.62	25.07	26.75	71.50	90.42	43.40	63.71	56.62	52.64
	SQuat *(COLM 2025)*	**43.84**	25.48	**26.81**	**72.50**	92.10	42.94	62.03	55.17	52.61
	TurboQuant* *(ICLR 2026)*	43.40	25.32	26.52	**72.50**	91.07	**43.46**	61.41	54.16	52.23
	LatchQ (Ours)	43.73	**25.52**	26.78	**72.50**	**92.30**	43.43	**65.32**	**58.50**	**53.51**
2.25	KIVI-2	42.90	24.80	26.52	71.50	91.14	**43.68**	61.75	55.56	52.23
	KVQuant	**43.65**	24.17	26.06	71.00	89.77	40.84	56.02	49.35	50.11
	SQuat	41.16	24.94	**26.64**	**72.50**	91.78	42.72	60.81	51.87	51.55
	TurboQuant*	41.03	22.33	24.78	70.05	90.86	40.97	54.58	47.66	49.03
	LatchQ (Ours)	43.64	**25.30**	26.21	**72.50**	**92.68**	42.72	**64.31**	**56.89**	**53.03**

Qwen2.5-7B-Instruct

bpd	Method	Qasper	QMSum	Multi-News	TREC	TriviaQA	SAMSum	LCC	RepoBench	Average (macro)
16	FP16	44.16	24.37	25.79	72.50	89.84	47.93	62.98	67.61	54.40
3.0	KIVI-2	42.22	24.09	25.13	72.00	89.10	**47.63**	61.65	64.88	53.34
	KVQuant	39.33	**24.35**	25.17	70.00	89.28	45.70	62.20	65.03	52.63
	SQuat	41.52	22.01	24.15	70.50	83.31	41.62	55.10	43.24	47.68
	TurboQuant*	41.45	23.56	**25.71**	71.00	86.52	45.87	62.25	59.89	52.03
	LatchQ (Ours)	**43.06**	24.33	25.54	**72.50**	**89.30**	46.75	**62.57**	**65.98**	**53.75**
2.25	KIVI-2	40.10	23.75	24.71	**72.50**	87.84	45.51	59.04	63.16	52.08
	KVQuant	36.21	23.03	24.75	70.50	86.12	44.25	60.18	54.43	49.93
	SQuat	36.09	19.51	23.50	67.50	69.62	35.80	51.70	34.13	42.23
	TurboQuant*	39.79	20.76	22.75	64.50	62.78	35.64	51.38	35.95	41.69
	LatchQ (Ours)	**41.60**	**24.01**	25.05	**72.50**	**88.61**	**46.33**	**61.43**	**63.49**	**52.88**

* TurboQuant uses 0.5–1.0 more bits per dimension than the indicated settings, as specified in the caption.

4.2 Experimental Results

We evaluate ***LatchQ*** in terms of model performance and memory efficiency. First, we compare against KV cache quantization baselines across models and bit-rates on long-context, reasoning, and code generation tasks. Next, we ablate *QTrace* and *QClip* and analyze the diagonal approximation. Finally, we verify that ***LatchQ*** preserves the baseline KV storage footprint.

4.2.1 Long-Context Performance

Table 1 compares ***LatchQ*** with existing KV cache quantization methods on LongBench across multiple model families and target bit-rates. We evaluate three model families in total, with additional results for Mistral deferred to the Appendix B.2. Across both models, ***LatchQ*** consistently achieves the highest average scores among quantized baselines at 3.0 and 2.25 bpd. Crucially, its relative advantage becomes even more pronounced in the extreme low-bit regime, where the margin over the strongest baseline expands from 0.51 to 0.80 points on Llama and from 0.41 to 0.80 points on Qwen. These results indicate that our query-guided approach becomes particularly effective as the storage budget becomes highly constrained.

4.2.2 Reasoning and Code Generation

Table 2 compares reasoning and code generation performance on Llama-3.1-8B-Instruct. At 3.0 bpd, ***LatchQ*** achieves the highest scores on GSM8K, MATH-500, and GPQA Diamond among the reported quantized methods. At the 2.25 bpd target setting, ***LatchQ*** achieves the highest GSM8K and LiveCodeBench scores among the evaluated quantized methods. It surpasses SQuat by 5.91 points on GSM8K and 5.40 points on MATH-500, while retaining a LiveCodeBench pass@1 of 11.38, close to the FP16 score of 11.53. These results further demonstrate the effectiveness of query-guided clipping in preserving reasoning and code generation performance under severe storage constraints.

Table 2: Reasoning and code generation results on Llama-3.1-8B-Instruct at 3.0 and 2.25 bpd (3.3 and 2.3 bpd for KVQuant, and 4.0/3.75 and 3.0/2.75 bpd for TurboQuant). All results are obtained from our own evaluations. LCB denotes LiveCodeBench.

Method	3.0 bpd				2.25 bpd			
	GSM8K	MATH-500	GPQA Diamond	LCB (pass@1)	GSM8K	MATH-500	GPQA Diamond	LCB (pass@1)
FP16	78.71	46.60	23.23	11.53	78.71	46.60	23.23	11.53
KIVI-2	72.86	39.00	19.19	9.60	65.50	24.80	9.09	8.05
KVQuant	73.84	41.80	21.25	10.46	63.31	34.40	20.22	9.29
SQuat	76.50	41.60	21.21	**11.22**	68.84	29.80	20.71	8.36
TurboQuant*	75.66	44.60	20.77	9.91	63.38	**42.40**	**23.23**	3.72
LatchQ (Ours)	**78.22**	**45.00**	**22.78**	10.84	**74.75**	35.20	21.21	**11.38**

* TurboQuant uses 0.5–1.0 more bits per dimension than the bpd labels indicate.

Table 3: Ablation of *QTrace* and *QClip* on Llama-3.1-8B-Instruct at 3.0 bpd. Without *QTrace*, clipping candidates are evaluated using uniform weights instead of observed query interactions. LCB denotes LiveCodeBench.

Configuration	*QTrace*	*QClip*	GSM8K	MATH-500	GPQA Diamond	LCB (pass@1)
FP16	–	–	78.71	46.60	23.23	11.53
Baseline	×	×	72.86	39.00	19.19	9.60
LatchQ (w/o Query Interaction)	×	✓	68.17	23.06	12.60	8.43
LatchQ (Ours)	✓	✓	**78.22**	**45.00**	**22.78**	**10.84**

4.3 Ablation of *QTrace* and *QClip*

4.3.1 Importance of Observed Query–Key Interactions

Table 3 examines the individual and joint contributions of *QTrace* and *QClip*. Applying clipping with uniform weights (i.e., omitting *QTrace*) drastically degrades performance below the baseline across all four benchmarks. For instance, accuracy drops from 39.00 to 23.06 on MATH-500, and from 72.86 to 68.17 on GSM8K. This deterioration occurs because uniform weighting evaluates clipping candidates solely based on reconstruction error, ignoring how each key and channel is actually utilized by the observed queries. Consequently, minimizing this naive error fails to preserve the critical KV interactions that drive model performance.

In contrast, the joint application of *QTrace* and *QClip* significantly elevates performance, driving MATH-500 up to 45.00 and GSM8K to 78.22. This improvement highlights the synergy between the two mechanisms. *QTrace* establishes an interaction-aware metric by quantifying attention-weighted query energy per key and channel, while *QClip* exploits these weights to strictly optimize clipping ranges. This weighting allows clipping selection to account for differences in observed key usage and channel sensitivity within each quantization group. The gains over uniform weighting support the use of observed interactions in range selection, with improvements in reasoning and code generation under 2-bit key quantization and the existing cache schedule.

4.3.2 Weighting and Diagonal Approximation in *QTrace*

Prior post-training quantization methods employ diagonal curvature approximations to capture element-wise sensitivity (Li et al., 2021; Kim et al., 2024). We examine whether this approximation is effective for the key-specific interactions collected by *QTrace*, alongside the contributions of its weighting components. Table 4 compares these variants. MSE-Clip uses uniform weights, while Query-Energy and Attention-Only use squared query activations and attention weights, respectively. MSE-Clip and Query-Energy underperform the baseline, whereas ***LatchQ (diagonal)*** combines attention weights with query energy and outperforms both single-component variants on all three metrics. Relative to Attention-Only, this combination lowers NLL and achieves improvements of 1.21 points on GSM8K and 2.70 points on MATH-500. ***LatchQ full*** improves NLL and MATH-500 but requires $16.8\times$ the reported latency. The diagonal approximation therefore provides a favorable trade-off between predictive performance and clipping cost, motivating its use in ***LatchQ***.

Table 4: Ablation of *QTrace* components and the diagonal approximation on Llama-3.1-8B-Instruct at 3.0 bpd. Latency is measured per 128-key block and includes statistic construction and clipping-range selection.

Method	NLL ↓	GSM8K ↑	MATH-500 ↑	Latency (ms) ↓
Baseline	2.0038	72.86	39.00	0.07
MSE-Clip	2.0719	68.17	23.06	**2.61**
Query-Energy	2.0611	67.85	23.60	2.68
Attention-Only	1.9879	77.01	42.30	**2.61**
LatchQ *(diagonal)*	1.9673	**78.22**	45.00	2.67
LatchQ *(full)*	**1.9641**	77.41	**46.80**	44.8

Table 5: Calculated KV storage at 32,768 tokens with a 128-token FP16 residual cache. K/V denotes bpd, and % is relative to FP16.

	≈3 bpd			≈2.25 bpd		
Method	**K/V**	**GB**	**%**	**K/V**	**GB**	**%**
FP16	16/16	4.00	100	–	–	–
KIVI-2	3.00/3.00	0.76	19.1	2.25/2.25	0.58	14.4
KVQuant	3.33/3.33	0.83	20.8	2.33/2.33	0.58	14.6
SQuat	3.00/3.00	0.76	19.1	2.25/2.25	0.58	14.4
TurboQuant	4.00/3.75	0.98	24.5	3.00/2.75	0.73	18.3
LatchQ (Ours)	3.00/3.00	0.76	19.1	2.25/2.25	0.58	14.4

4.3.3 Computational and Memory Overhead

Table 5 presents the calculated KV storage at a context length of 32,768 tokens, including a 128-token FP16 residual cache. ***LatchQ*** matches the KV storage of KIVI-2 and SQuat, requiring 19.1% and 14.4% of the FP16 reference at 3.0 and 2.25 bpd, respectively. For KVQuant, the listed storage rates are estimates based on our accounting convention with a 1% outlier fraction, yielding 3.33 and 2.33 bpd. TurboQuant is evaluated at higher effective key and value bit-rates for a conservative comparison, with its storage accounting for QJL correction (Zandieh et al., 2024) and norm overhead. At matching KV storage budgets, ***LatchQ*** achieves higher average LongBench scores than KIVI-2 and SQuat (Table 1). Beyond KV storage, we measure peak memory overhead in a quantize–dequantize framework that retains KV entries in FP16. Across the evaluated settings, ***LatchQ*** increases total peak memory, including model weights, by less than 1% relative to KIVI-2. The current implementation also incurs computational overhead from attention recomputation during prefill. Detailed experimental settings and results are provided in Appendix B.3.

5 Discussion and Limitations

LatchQ demonstrates that actual query–key interactions observed in the hot cache can improve quantization decisions under a fixed storage budget. It captures these interactions to weight reconstruction errors for each key and channel, then selects clipping ranges for each group and channel. This adapts quantization to observed key usage while preserving bit allocation, storage format, and the quantization schedule. Our findings highlight the observation and summarization of query interactions before compression as an additional design dimension in KV cache quantization.

To isolate the contribution of the observed interactions, our evaluation uses a fixed hot-cache schedule and 2-bit key quantization. Generalization to other cache management policies requires further validation, as different retention strategies can change both the observation period for each key and the interactions available for quantization. Future work should jointly consider the quality of observations provided by hot-cache policies and the representational capacity of the quantizer.

6 Conclusion

We present ***LatchQ***, a KV cache quantization method that captures actual query–key interactions while keys reside in the hot cache. *QTrace* aggregates these interactions into key- and channel-specific weights, which *QClip* uses to select clipping ranges for each group and channel. Extensive experiments on long-context, reasoning, and code generation tasks demonstrate performance gains over the base quantizer while preserving bit allocation, storage format, and the quantization schedule. These results show that the existing hot-cache observation window provides useful information for improving quantization decisions under a fixed storage budget. We hope these insights inspire further advances in the increasingly important field of KV cache quantization.

AI USE STATEMENT

We used generative AI tools for language editing and translation. We also used these tools to check the consistency of our theoretical analysis, including proofs, and to obtain feedback on methodology and experimental design. All AI-assisted suggestions were reviewed by the authors. We conducted the literature review, developed the research concepts and hypotheses, formulated the mathematical claims, developed the proofs, implemented the methods, designed the experiments, interpreted the results, and drew the final conclusions. AI was not used to generate synthetic datasets, clean or reformat datasets, or conduct qualitative or thematic data analysis.

REPRODUCIBILITY STATEMENT

Experimental settings are detailed in Appendix B. Code is included in the supplementary material, and we plan to release it publicly on GitHub.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Zefan Cai, Wen Xiao, Hanshi Sun, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, et al. R-kv: Redundancy-aware kv cache compression for reasoning models. *Advances in neural information processing systems*, 38:60980–61005, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ravi Ghadia, Avinash Kumar, Gaurav Jain, Prashant J. Nair, and Poulami Das. Dialogue without limits: Constant-sized KV caches for extended response in LLMs. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=SuYO70ZxZX.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Junhui He, Junna Xing, Nan Wang, Rui Xu, Shangyu Wu, Peng Zhou, Qiang Liu, Chun Jason Xue, and Qingan Li. A2ats: Retrieval-based kv cache reduction via windowed rotary position embedding and query-aware vector quantization. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 12451–12463, 2025.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification, 2024. URL https://arxiv.org/abs/2405.14256.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Sehoon Kim, Coleman Richard Charles Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 23901–23923. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/kim24f.html.

Hannah Laus, Claudio Mayrink Verdun, Hao Wang, Flavio du Pin Calmon, and Felix Krahmer. Kv cache compression through the lens of transform coding. *arXiv preprint arXiv:2608.14191*, 2026.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. Pm-kvq: Progressive mixed-precision kv cache quantization for long-cot llms. In *International Conference on Learning Representations*, volume 2026, pp. 107881–107900, 2026.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 110072–110106, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Hao Wang, Ligong Han, Kai Xu, and Akash Srivastava. SQuat: Subspace-orthogonal KV cache quantization. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=gKdhzBiHay.

Xuefei Wang, Haoyu Tang, Tianyuan Liang, Zhibin Wang, Yupeng Hu, and Weili Guan. Resonating with rope: Spectral quantization for high-fidelity key cache compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 37328–37348, 2026.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3258–3270, 2024a.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024b.

Amir Zandieh, Majid Daliri, and Insu Han. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead, 2024. URL https://arxiv.org/abs/2406.03482.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pp. 56418–56439, 2026.

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. MixKVQ: Query-aware mixed-precision KV cache quantization for long-context reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7189–7204, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.326. URL https://aclanthology.org/2026.acl-long.326/.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

A Additional Method Details

A.1 *QTrace*

A.1.1 Diagonal Approximation and Attention Weighting

This section explains how we construct the channel-wise attention-weighted surrogate from the quadratic term in Lemma 1. For clarity, we first consider a single query head and omit the head index. Let $\mathbf{e}_i = \widehat{\mathbf{k}}_i - \mathbf{k}_i$ denote the reconstruction error of key i, and let d denote the head dimension. With the query and all other keys fixed, the second-order KL term for query position t is given by:

$$D_{t,i}^{(2)} = \frac{a_{t,i}(1-a_{t,i})}{2d} \left(\mathbf{q}_t^\top \mathbf{e}_i\right)^2. \tag{5}$$

Expanding the squared inner product gives:

$$\left(\mathbf{q}_t^\top \mathbf{e}_i\right)^2 = \sum_{c=1}^{d} q_{t,c}^2 e_{i,c}^2 + 2 \sum_{1 \le c < c' \le d} q_{t,c} q_{t,c'} e_{i,c} e_{i,c'}. \tag{6}$$

The second sum couples reconstruction errors across channels. Dropping these cross-channel terms yields the diagonal approximation:

$$\widetilde{D}_{t,i}^{(2)} = \frac{a_{t,i}(1-a_{t,i})}{2d} \sum_{c=1}^{d} q_{t,c}^2 e_{i,c}^2. \tag{7}$$

This approximation enables independent channel-wise evaluation. It does not assume that the omitted terms vanish for every query or error vector. Since those terms can have either sign, the diagonal approximation is not generally an upper or lower bound on the original quadratic term.

We further choose to weight reconstruction errors proportionally to observed attention usage. The local KL curvature factor $a_{t,i}(1 - a_{t,i})$ decreases as $a_{t,i}$ approaches one, whereas an attention-proportional weight increases with $a_{t,i}$. We therefore replace this curvature factor with $a_{t,i}$ and define the surrogate as follows:

$$S_{t,i} = \frac{a_{t,i}}{d} \sum_{c=1}^{d} q_{t,c}^2 e_{i,c}^2. \tag{8}$$

This substitution changes the weighting criterion and can change the relative scores of clipping candidates. By contrast, removing the common positive factor $1/2$ only rescales the surrogate and leaves its minimizer unchanged.

For grouped-query attention, let $\mathcal{G}(h)$ denote the query heads associated with KV head h, and let $\mathcal{E}_i$ denote the observation set for key i. Accumulating the surrogate coefficients over these heads and observed queries gives:

$$H_d[h,i,c] = \sum_{r \in \mathcal{G}(h)} \sum_{t \in \mathcal{E}_i} a_{t,i}^{(r)} \frac{(q_{t,c}^{(r)})^2}{d}. \tag{9}$$

The accumulated surrogate for key i is consequently $\sum_{c=1}^{d} H_d[h,i,c] e_{h,i,c}^2$. These coefficients form the *QTrace* evidence used by *QClip*, with the positive floor described in Appendix A.2.1 applied before candidate evaluation.

A.1.2 Observation Sets and Computation Timing

Let L denote the prompt length, with positions indexed from 1 to L. After prefill attention, the first $L - (L \bmod 128)$ keys are quantized. For each such key i, the observation set includes all subsequent queries within the prompt:

$$\mathcal{E}_i = \{\, t \mid i < t \le L \,\}. \tag{10}$$

The remaining prompt-tail keys stay in the hot cache. Their prefill evidence is discarded in our implementation, so they use only decode-time evidence.

During decoding, a block is quantized when the hot cache contains 128 keys, after attention for the current token has been computed. Let $\mathcal{B}$ denote the token positions in this scheduled block. For each key $i \in \mathcal{B}$, we use:

$$\mathcal{E}_i = \{\, t \in \mathcal{B} \mid t > i,\ t > L \,\}. \tag{11}$$

The conditions $t > i$ and $t > L$ restrict observations to subsequent queries generated during decoding. These observation sets are used in the main-text definition of $H_d[h, i, c]$, preserving the baseline quantization schedule.

During prefill phase, the evaluated implementation processes queries in chunks of 256 and collects evidence from all subsequent prompt queries for each key, without sampling. Attention weights are recomputed for evidence collection rather than reused from the model's attention operation. Per-layer evidence tensors are stored in FP16 and released after the corresponding quantization step.

A.2 *QClip*

A.2.1 Positive Floor for QTrace Weights

The `AddFloor` operation in Algorithm 2 converts the accumulated evidence $H_d[h, i, c]$ into the evaluation weights $w_{h,i,c}$ used in Eq. 4. Let $\mathcal{B}_g$ contain the G key positions in group g, and let d denote the head dimension. For each KV head h and group g, we add 10^{-3} times the mean evidence over the group's keys and channels, together with a positive constant 10^{-12}:

$$w_{h,i,c} = H_d[h, i, c] + \frac{10^{-3}}{Gd} \sum_{j \in \mathcal{B}_g} \sum_{c'=1}^{d} H_d[h, j, c'] + 10^{-12}, \qquad i \in \mathcal{B}_g. \tag{12}$$

The same offset is added to every key and channel within the group, with the mean computed separately for each KV head. This ensures that entries with zero accumulated evidence still receive positive evaluation weights. If all evidence in the group is zero, the weights are uniformly 10^{-12}, yielding the MSE-Clip selection criterion up to a common positive factor.

B Additional Experimental Details

Models and benchmarks. For long-context evaluation, we use Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Qwen2.5-7B-Instruct (Qwen et al., 2025), and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023). We also measure negative log-likelihood (NLL) on WikiText (Merity et al., 2016) to evaluate language modeling performance. We evaluate on eight LongBench tasks, including Qasper, QMSum, Multi-News, TREC, TriviaQA, SAMSum, LCC, and RepoBench (Bai et al., 2024). We report each task's score and the unweighted arithmetic mean across the eight tasks. Our reasoning and coding evaluations cover GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), GPQA Diamond (Rein et al., 2024), and LiveCodeBench. For LiveCodeBench (Jain et al., 2025), we report pass@1. We evaluate LiveCodeBench on a 323-problem subset obtained by applying the v5 date window to the v6 release files. This subset differs from the standard v5 release.

Compared methods. The LongBench comparison includes KIVI-2 (Liu et al., 2024), KVQuant (Hooper et al., 2024), SQuat (Wang et al., 2025), TurboQuant (Zandieh et al., 2026), and ***LatchQ***, with FP16 serving as the reference. We consider target storage settings of 3.0 and 2.25 bits per dimension (bpd).

For KVQuant, we estimate storage as $b + 0.01 \times 33$ bits per dimension, where b is the nominal quantization bit width. This accounting assumes a 1% outlier fraction at both context lengths. The observed outlier fractions range from 1.44% to 1.63% across 2,200 audited records.

For SQuat, we increase the FP16 residual-cache size from 32 to 128 tokens. We apply a query pre-scaling patch only in the Llama LongBench experiments and a separate patch to the Mistral FlashAttention prefill path.

For TurboQuant, official code was unavailable at the time of our experiments, so we reimplemented the method following the original paper. Our channel-ranking rule for mixed-precision allocation is an implementation choice, as the paper does not specify this procedure.

The component ablation compares KIVI-2, MSE-Clip, and ***LatchQ***, with FP16 as the reference. MSE-Clip applies *QClip* with uniform weights.

Quantization settings. For ***LatchQ***, keys use 2-bit asymmetric per-channel group quantization, with FP16 scales and minimum offsets. Values follow the baseline per-token 2-bit quantizer. The residual block size is 128 tokens, and the baseline quantization schedule is preserved. In our quantize–dequantize evaluation, KIVI-2 and MSE-Clip use FP32 scales and offsets, whereas ***LatchQ*** rounds these parameters to FP16. Observation ranges and the handling of prompt-tail keys are described in Appendix A.1.2.

The lower and upper clipping ratios are independently selected from $\mathcal{R} = \{0, 0.03, 0.06, 0.10, 0.15\}$, yielding 25 candidate ranges for each KV head, quantization group, and channel. Candidates are evaluated in ascending order of the lower ratio, followed by the upper ratio, with $(0, 0)$ evaluated first. Ties retain the first candidate in this order. The relative and absolute weight-floor coefficients are 10^{-3} and 10^{-12}, respectively. The evaluation weights are defined in Appendix A.2.1.

B.1 Experimental Details for the Query-Evidence Analysis

The analysis presented in Figure 1 evaluates Llama-3.1-8B-Instruct (Grattafiori et al., 2024) on the WikiText-2 dataset (Merity et al., 2016) at 3.0 bpd. The setup employs 2-bit key and value quantization, a group size of 32 keys, and a 128-token FP16 hot cache. We concatenate the test corpus into a single token stream, selecting the first 12 consecutive, non-overlapping windows of 2,048 tokens. Each window is evaluated using teacher forcing in 128-token chunks. Figure 1(a) illustrates the cumulative mean next-token negative log-likelihood (NLL). To compute this metric, losses are initially averaged across the 12 windows at each relative position and subsequently averaged over all positions up to that point. No additional smoothing is applied.

All clipping-based variants evaluate a consistent set of 25 candidate pairs $(\alpha, \beta) \in \{0, 0.03, 0.06, 0.10, 0.15\}^2$ under an identical quantization schedule. MSE-Clip relies on the unweighted reconstruction error, whereas ***LatchQ*** leverages subsequent queries observed within each key's 128-token block prior to quantization. The future-query oracle incorporates all queries occurring after that block up to the end of the context window. Furthermore, the all-query oracle utilizes every subsequent query by combining both observed and future queries. For the NLL comparison, ***LatchQ*** collects evidence directly during quantized execution. The oracle baselines instead extract attention weights from an unquantized FP16 reference pass. Throughout this analysis, all clipping-based variants employ FP16 scales and offsets, whereas KIVI-2 preserves these parameters in FP32.

To isolate the effect of the query observation set in Figure 1(b), both the observed-query (***LatchQ***) and all-query selections are derived from a shared unquantized FP16 reference trace. Within the presented matrix, rows correspond to the all-query oracle selections, while columns represent the observed-query selections. We normalize each row to represent the conditional probability of the observed-query selections given an all-query oracle selection. Decisions are aggregated across 12 windows, 32 layers, 8 KV heads, 59 groups, and 128 channels. This aggregation excludes the initial group containing attention-sink keys along with the final 128-token block. Exact agreement is the fraction of decisions with identical lower/upper clipping selections, yielding $14{,}703{,}199/23{,}199{,}744 \approx 63.4\%$, which we report as 63% in the main text.

B.2 Additional Long-Context Results

Table 6 reports additional LongBench results on Mistral-7B-Instruct-v0.3 at 3.0 and 2.25 bpd. At 3.0 bpd, quantized baselines already perform close to FP16, and ***LatchQ*** shows no clear overall advantage. This can be explained by the smaller groups of 32 keys, which allow quantization ranges to better match local key values and leave less room for improvement through clipping. At 2.25 bpd, using 128 keys per group reduces metadata overhead. However, larger groups can increase within-group variation and sensitivity to outliers, making query-guided clipping more useful. Under these tighter storage constraints, ***LatchQ*** achieves the highest reported average score of 52.79 among quantized methods.

Table 6: LongBench results on Mistral-7B-Instruct-v0.3 at 3.0 and 2.25 bpd. All results are obtained from our own evaluations. Bold and underlined values indicate the best and second-best scores among quantized methods at each bpd, respectively.

Mistral-7B-Instruct-v0.3										
bpd	**Method**	**Qasper**	**QMSum**	**Multi-News**	**TREC**	**TriviaQA**	**SAMSum**	**LCC**	**RepoBench**	**Average (macro)**
16	FP16	38.15	25.61	26.93	76.00	88.69	47.39	61.76	62.54	53.38
3.0	KIVI-2	**38.05**	25.16	26.33	**76.00**	88.37	**47.28**	60.72	61.52	52.93
	KVQuant	37.78	25.17	26.77	**76.00**	**88.66**	46.38	59.41	61.06	52.65
	SQuat	37.58	24.58	26.04	75.50	**88.66**	46.49	59.08	58.65	52.07
	TurboQuant	36.64	25.09	**26.82**	75.50	88.52	46.33	59.59	60.39	52.36
	LatchQ (Ours)	37.98	**25.26**	26.33	**76.00**	88.58	46.35	**61.73**	**62.16**	**53.05**
2.25	KIVI-2	36.58	24.68	25.79	**76.00**	**89.27**	**47.39**	59.73	60.18	52.45
	KVQuant	31.86	24.14	25.42	73.50	86.94	43.16	58.68	57.36	50.13
	SQuat	36.47	24.45	25.79	**76.00**	88.22	45.36	56.89	55.70	51.11
	TurboQuant	32.28	21.90	25.86	69.50	83.64	42.95	53.18	48.26	47.20
	LatchQ (Ours)	**37.23**	**25.03**	**25.93**	**76.00**	89.14	46.53	**61.12**	**61.37**	**52.79**

Table 7: Peak allocated GPU memory in the shared quantize–dequantize evaluation, reported in GB ($1\,\mathrm{GB} = 1024\,\mathrm{MB}$). The bpd column denotes the target storage rate.

bpd	**Method**	$L = 10{,}240$		$L = 32{,}768$	
		Prefill	**Decode**	**Prefill**	**Decode**
3.0	KIVI-2	17.38	16.51	22.63	19.86
	LatchQ (Ours)	17.40	16.55	22.69	19.97
2.25	KIVI-2	17.38	16.51	22.63	19.86
	LatchQ (Ours)	17.40	16.54	22.69	19.94

B.3 Memory Overhead

We measure the additional peak memory introduced by *QTrace* and *QClip* relative to KIVI-2 on Llama-3.1-8B-Instruct. Experiments use an A100-80GB GPU, FP16 model weights, and batch size 1. Each run processes a synthetic prompt of length $L \in \{10{,}240, 32{,}768\}$ followed by 130 decoding steps. We record `max_memory_allocated` separately for prefill and decoding, including model weights.

Both methods use a shared quantize–dequantize evaluation framework that retains KV values in FP16. The KIVI-2 baseline simulates its quantization rules and cache schedule. ***LatchQ*** shares the same cache layout, value quantization, 128-token residual-cache schedule, and SDPA backend. Both overwrite cached values in place without retaining a separate full-cache copy. The 3.0 and 2.25 bpd settings use quantization groups of 32 and 128, respectively.

Table 7 shows that ***LatchQ*** introduces modest additional peak memory in the evaluated setting. Across both bit rates, prefill memory increases by 20 MB at $L = 10{,}240$ and 64 MB at $L = 32{,}768$. Decoding overhead ranges from 24 to 110 MB, with the 2.25 bpd setting requiring less additional peak memory at both context lengths. These measurements quantify incremental execution memory within the shared simulation framework. Because cached values remain in FP16, they do not reflect the storage reduction from packed quantization. The KV storage requirements calculated from the quantization format are reported separately in Table 5. Peak-memory overhead with packed KV storage requires separate measurement, as packing and attention kernels can change temporary allocations and their overlap.

B.4 Runtime of the Optimized Implementation

Implementation and validation. Our benchmark accuracy results rely on the reference implementation detailed in Appendix A.1.2. For the runtime evaluation, we deploy an optimized version of *QTrace* featuring fused two-pass Triton kernels, strictly reusing the query and key projections from the model's forward pass. By exploiting tensor-core arithmetic and FP16 high/low decomposition, these kernels accelerate the attention-weighted query-energy product. Rigorous validation is

Table 8: Runtime at 3.0 bpd (group size 32) in the quantize–dequantize framework. ***LatchQ*** uses the optimized implementation described above. Entries illustrates the median [minimum–maximum] across three runs after warm-up. Decoding latency is averaged over 130 steps and includes scheduled quantization.

Prompt length	Method	Prefill (s)	Decode (ms/token)
10,240	KIVI-2	1.059 [1.050–1.066]	28.55 [28.52–28.57]
	LatchQ	2.607 [2.603–2.610]	29.89 [29.84–29.89]
32,768	KIVI-2	4.532 [4.529–4.536]	33.99 [33.99–34.00]
	LatchQ	15.258 [15.249–15.265]	36.33 [36.32–36.33]

ensured by keeping the observation sets, causal attention normalization, evidence definition, and quantization schedule identical to the reference. The weight floor, clipping candidates, tie-breaking heuristics, and quantization parameters are similarly preserved without modification. The entire optimized runtime environment is built on Triton 3.1.0.

We validate the optimized implementation against the reference on Llama-3.1-8B-Instruct at group sizes 32 and 128, including 32,768-token contexts. The validation compares interaction evidence, clipping selections, model outputs, and cache behavior to assess numerical consistency. The benchmark accuracy results remain those obtained with the reference implementation.

Measurement protocol. We measure runtime on an NVIDIA A100 PCIe GPU using Llama-3.1-8B-Instruct with FP16 weights, SDPA, batch size one, and a 128-token hot cache. Both methods use the same model, synthetic prompt token IDs generated with seed zero, deterministic settings, and cache schedule. Measurements use the quantize–dequantize framework that retains KV entries in FP16 and do not measure inference with packed KV storage.

We perform one warm-up run followed by three measured runs, initializing fresh KV and evidence caches for each run. Prefill time includes the forward pass, evidence collection, and all scheduled prompt KV quantization. The pending prompt flush is executed before the decoding timer starts. Decoding latency is averaged over 130 steps with EOS stopping disabled, including one key-block flush per layer. Tokens remain on the GPU, with synchronization at measurement boundaries and no per-step host synchronization.

Runtime and memory. According to Table 8, our optimized implementation imposes a minor decoding latency penalty of 1.34 and 2.34 ms per token at prompt lengths of 10,240 and 32,768 tokens. These values translate to relative overheads of only 4.7% and 6.9% compared to KIVI-2. In contrast, prefill operations take 2.46× and 3.37× as long as KIVI-2. These metrics clearly delineate the computational trade-off of integrating observed query interactions into the clipping selection process. While the decoding latency overhead remains limited in the evaluated settings, evidence collection during prefill constitutes the primary performance bottleneck. Memory profiling of this optimized setup at 32,768 tokens reveals an increase in peak memory consumption of approximately 2.0 GB during prefill and 0.62 GB during decoding compared to KIVI-2. This prefill overhead arises because the system materializes and retains per-layer FP16 evidence tensors until the prompt flush.

Despite the additional computation and working memory, ***LatchQ*** achieves higher average LongBench scores than KIVI-2 at matching quantized KV storage budgets. These findings support observed query interactions as a useful signal for improving accuracy within the baseline KV storage format and bit allocation, with the measured runtime and memory overheads quantifying the associated execution cost.