HubKV: Efficient KV Cache Compression via Submodular Marginal Discounting

Anonymous ACL submission

Abstract

Key-Value (KV) caching enables efficient autoregressive inference, but its memory cost grows linearly with context length. Most score-based KV compression methods retain tokens via greedy Top-K selection, implicitly treating token utility as modular. We show that high-scoring tokens are often spatially clustered and that attention heads differ in their score selectivity, which can cause Top-K retention to waste cache budget on locally redundant context. Motivated by this observation, we formulate an ideal score-weighted local coverage objective with monotone submodular structure to model diminishing returns in locally redundant cache entries. We then propose **HubKV**, a single-pass marginal-gain proxy that detects local score hubs, softly discounts neighboring tokens, and applies compression-ratio-gated head-wise calibration while preserving the parallel Top-K pruning interface of existing KV compression systems. We validate HubKV on four backbones across 16 diverse tasks. On Qwen3-8B, HubKV improves FastKVZip and KVZip by 3.23 and 6.53 average score points at 95% prefill compression, and by +1.71/+1.87 points on average across budgets. Decoding-stage experiments further show that the same score correction remains effective on reasoning datasets AIME25 and MATH.

1 Introduction

The transformative success of Transformer-based Large Language Models (LLMs) is fundamentally anchored in the self-attention mechanism, which enables the model to contextualize information across vast sequences (Vaswani et al., 2017). To maintain high throughput during autoregressive generation, modern inference systems rely on Key-Value (KV) caching to store historical context (Dai et al., 2019). However, the memory footprint of the KV cache scales with sequence length, becoming the primary bottleneck for deploying LLMs in long-context scenarios (Kwon et al., 2023).

To mitigate this burden, KV cache compression has emerged as a critical research direction. The goal of KV cache compression is to find effective ways to choose critical tokens to retain, from a massive sequence of past tokens, in order to maximize long-context inference performance. Although predicting the exact future attention of LLMs is highly complex, there exist many heuristics, such as attention sparsity (Li et al., 2024), adaptive head budgets (Feng et al., 2026), and query-agnostic reconstruction (Kim et al., 2026b), which have been shown to be effective in practice. KV cache compression is typically a ranking process in which token importance scores are computed during the prefill stage, and a subset of tokens is chosen to be preserved using the aforementioned heuristics.

While these methods have demonstrated strong performance in accelerating LLM inference, they rely on a greedy Top-K selection strategy. In their standard formulation, these algorithms implicitly optimize a *modular* objective, where the utility of each token is evaluated in complete isolation. However, this independence assumption ignores the semantic reality of language, as adjacent tokens (e.g., subwords of a single term or entities within a local phrase) often share highly overlapping information. Consequently, retaining clustered tokens provides diminishing marginal returns to the global context. Because of this, a pure Top-K approach tends to allocate the cache budget heavily on local high-scoring clusters, resulting in locally redundant samples that dilute the overall context coverage.

In order to tailor a compression method for highly correlated text sequences, we first conduct a systematic empirical study to characterize the spatial distribution of token importance indicators. Our analysis reveals two critical phenomena: (1) a sharp decline in the efficiency of standard Top-K selection in covering diverse context segments, and (2) high spatial autocorrelation in importance scores, indicating local concentration of utility.

This naturally suggests viewing cache retention as a coverage problem: a selected token should be valuable by itself, but its value should decrease when nearby tokens have already been retained. Motivated by these findings, we re-examine KV cache eviction through the lens of *Submodular Maximization*. Specifically, we model the cache utility as a monotone submodular function to naturally incorporate the principle of diminishing returns: the marginal gain of a token decreases if its semantic neighbors are already present in the cache (Lin and Bilmes, 2011). While sequential greedy solvers provide theoretical guarantees for such objectives, they introduce unacceptable sequential bottlenecks for real-time LLM inference.

To bridge this gap, we propose **HubKV**, a hardware-parallel first-order ranking proxy based on Submodular Marginal Discounting (SMD). Operating directly on the temporal sequence, HubKV uses 1D max-pooling to identify local "hub tokens" and induces a local repulsion field by applying a soft penalty to their neighbors. Furthermore, HubKV incorporates a compression-ratio-gated head-wise selectivity mechanism that calibrates the strength of this redundancy correction according to both cache pressure and the discriminative power of individual attention heads. The resulting calibrated scores can be passed directly to existing Top-K-style pruning operators, making HubKV a plug-in score refinement layer rather than a replacement cache policy.

Our main Qwen3-8B evaluation spans 16 long-context and reasoning tasks; appendix LongBench results cover three additional backbones. At 95% prefill compression, HubKV improves the corresponding FastKVZip and KVZip baselines by 3.23 and 6.53 average score points on Qwen3-8B, respectively. Averaged across evaluated prefill budgets, the paired gains are +1.71/+1.87 points. Decoding-stage tests show consistent gains over FastKVZip on AIME25/MATH.

Our contributions are summarized as follows:

- We empirically expose the modularity flaw in current Top-K eviction strategies through spatial profiling and autocorrelation analysis, and reveal head-wise functional diversity via raw attention visualization.
- We model KV cache compression as a score-weighted local coverage problem using a monotone submodular facility-location objective, and use sequential greedy selection as a conceptual oracle rather than as the deployed inference algorithm.
- We design HubKV, a hardware-parallel marginal-gain proxy that combines local hub detection, soft neighbor discounting, compression-ratio gating, and bounded head-wise selectivity calibration.
- We evaluate HubKV on four models across 16 tasks and decoding-stage reasoning tests. The results show the strongest gains in the aggressive-compression regime while preserving the same pruning interface as modular baselines.

2 Preliminaries

2.1 Notation and Problem Formulation

An autoregressive Transformer LLM f_{LM} caches historical Key and Value states during inference. For an input of N tokens, let $V = \{1, \dots, N\}$ denote the cached token positions. KV cache compression selects a retained subset $S \subset V$ under budget $|S| \leq B$, aiming to preserve the behavior of the full cache:

$$f_{LM}(\cdot \mid S) \approx f_{LM}(\cdot \mid V) \tag{1}$$

which we evaluate through downstream long-context performance.

2.2 Submodular Set Functions

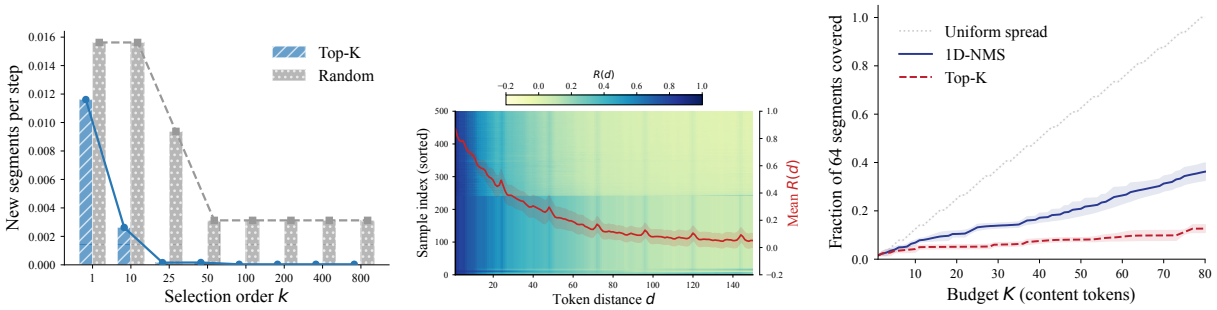
For a set utility $F : 2^V \to \mathbb{R}$, submodularity formalizes diminishing returns (Krause and Golovin, 2014): for $A \subseteq B \subseteq V$ and $v \in V \setminus B$,

$$F(A \cup \{v\}) - F(A) \geq F(B \cup \{v\}) - F(B) \tag{2}$$

Monotonicity means $F(A) \leq F(B)$ whenever $A \subseteq B$. Cardinality-constrained monotone submodular maximization is NP-hard, but sequential greedy selection achieves a $(1 - 1/e)$ approximation guarantee (Nemhauser et al., 1978). We use this result only as theoretical motivation for redundancy-aware cache selection.

2.3 Empirical Analysis and Observations

Current score-based KV cache eviction methods predominantly rely on a Top-K selection strategy, implicitly assuming that the marginal information gain of each token is independent (Li et al., 2024; Feng et al., 2026; Kim et al., 2026b,a). In this



(a) Marginal Spatial Gain (b) Locality of Importance (c) Cumulative Context Coverage

Figure 1: **Empirical Motivation for Redundancy-Penalized Eviction. (a)** The marginal spatial coverage of Top-K declines due to score clustering. **(b)** Autocorrelation analysis indicates that proxy importance scores are statistically localized. **(c)** By suppressing local score clusters, 1D-NMS improves global segment coverage.

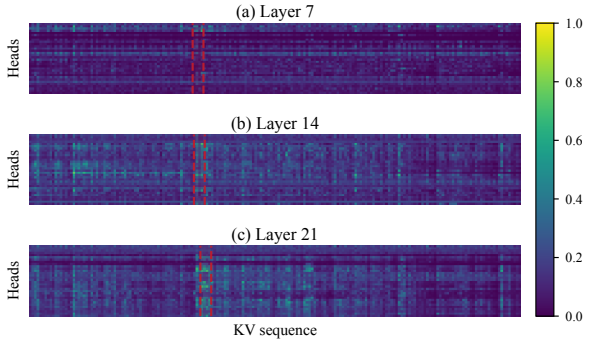


Figure 2: **Visualization of Localized Attention mass.** Raw attention weights across heads reveal clustered high-attention anchors (vertical stripes, highlighted by red dashed boxes) versus diffuse background attention (horizontal variance), motivating head-aware scoring.

section, we reveal the breakdown of this modular assumption through a proxy-based spatial analysis.

To isolate the structural properties of token importance from heuristic biases, we utilize context reconstruction (Kim et al., 2026b) as a proxy metric. We evaluate Qwen3-8B on the RULER benchmark, measuring each KV pair's contribution to the reconstruction of the original prompt. We extract importance scores across 500 samples as the target profile for spatial profiling. Appendix I describes the score-capture and analysis setup.

Observation 1: Declining Marginal Spatial Coverage of Top-K. We first examine the efficiency of the standard Top-K strategy by measuring its *marginal spatial gain*—the number of new document segments (from a total of 64) covered per selected token. As illustrated in Figure 1(a), the marginal spatial gain of Top-K exhibits a rapid decline, dropping significantly after only a few tokens ($K \approx 4$). This observation motivates **query-agnostic** compression, where the specific information requested by a future query is unknown. Under such uncertainty, an ideal strategy should act as a representative summary that preserves diverse semantic information across the context (Lin and Bilmes, 2011). While random selection trivially achieves high spatial coverage, it often lacks the necessary *information density* by allocating the strict cache budget to low-utility tokens. Conversely, Top-K tends to spend the budget on isolated semantic clusters, leaving many document segments under-covered.

Observation 2: Statistical Locality and Functional Diversity. To investigate the cause of this clustering, we compute the normalized autocorrelation $R(d)$ of the proxy importance scores. Figure 1(b) shows that token importance is highly correlated at short distances ($R(1) \approx 0.87$), suggesting that high-importance indicators are statistically localized. Under the modeling assumption that nearby tokens tend to share contextual information, this locality motivates a local-overlap view of cache utility. Furthermore, Figure 2 visualizes raw attention weights, revealing shared high-attention columns and head-specific concentration patterns with sharper selectivity in some heads.

Observation 3: Balancing Importance and Coverage via Local Suppression. The identified locality suggests that we can improve global coverage while maintaining information density by penalizing local score clustering. We apply a 1D Non-Maximum Suppression (1D-NMS) filter to the proxy scores. This mechanism encourages the selection process to skip nearby neighbors of a selected "hub" and move to the next high-scoring region. As demonstrated in Figure 1(c), 1D-NMS achieves a $\sim 2.6\times$ higher cumulative segment coverage than Top-K. Unlike Random

sampling, 1D-NMS retains local peaks (highest-importance tokens within their neighborhoods), thereby improving spatial diversity while still favoring high-scoring representatives. We use this hard-suppression experiment only as a diagnostic pilot; the final HubKV method softens it through SMD, while the calibration step handles per-head score changes.

3 Methodology

In this section, we formalize KV cache compression as a combinatorial optimization problem. We first demonstrate how existing heuristic strategies implicitly optimize a modular function, and then introduce a monotone submodular objective to model semantic redundancy. Finally, we present a GPU-parallel score correction that uses this objective as a theoretical guide while avoiding the sequential bottleneck of submodular optimization.

3.1 KV Cache Eviction with Capacity Constraints

Let the ground set $V = \{1, 2, \ldots, N\}$ represent all tokens in the historical context during LLM inference. The task of KV cache compression is to select a representative subset $S \subseteq V$ to be retained in GPU memory. Given hardware constraints, we require $|S| \leq B$, where B is the KV cache budget. If we define a set function $F : 2^V \to \mathbb{R}$ to measure the utility of the retained cache S, the eviction problem can be formalized as:

$$S^* = \arg\max_{S \subseteq V} F(S) \quad \text{subject to:} \quad |S| \leq B. \tag{3}$$

In the general paradigm of score-based KV cache compression, an evaluation metric, such as accumulated attention weights (Li et al., 2024) or predictions from a lightweight neural module (Kim et al., 2026b), assigns an importance score $s_i \in [0, 1]$ to each token. Standard Top-K eviction then selects tokens with the highest scores. Mathematically, this is equivalent to defining the utility function as $F_{mod}(S) = \sum_{i \in S} s_i$. This objective is *modular*, assuming that the marginal gain of adding a token k to the cache is strictly s_k, independent of the existing elements in S.

However, in natural language, semantic information is intrinsically clustered. If a set S already contains several tokens from a specific semantic phrase, adding another adjacent token yields negligible new information. Consequently, a purely modular $F_{mod}(S)$ often leads to highly redundant context summaries. To capture this semantic overlap, we argue that the ideal utility $F(S)$ is better formulated as a *monotone submodular* function, which naturally incorporates the principle of diminishing returns. Although maximizing a submodular function subject to a cardinality constraint is NP-hard, sequential greedy selection provides a useful theoretical oracle with a $(1 - 1/e)$ approximation factor (Nemhauser et al., 1978). HubKV uses this oracle to motivate the form of its score correction, but does not claim to inherit this global guarantee.

3.2 Score-Weighted Local Coverage Objective

To properly evaluate the quality of a KV cache subset, the objective function should encourage preserving critical tokens while simultaneously penalizing local semantic redundancy. Drawing inspiration from facility location problems in operations research (Lin and Bilmes, 2011), we model the total utility of the retained KV set S as a score-weighted local coverage function:

$$F_{sub}(S) = \sum_{i \in V} \max_{j \in S} w_{i,j} \tag{4}$$

where $w_{i,j}$ represents the semantic coverage provided by token j to token i. To bridge the gap between abstract optimization and our empirical importance scores, we define $w_{i,j}$ using a normalized importance-weighted kernel:

$$w_{i,j} = s_j \cdot \frac{\mathcal{K}(|i-j|)}{Z_j}, \quad Z_j = \sum_{k \in V} \mathcal{K}(|j-k|) \tag{5}$$

where s_j is the importance score of token j, and $\mathcal{K}$ is a non-negative distance-based kernel (e.g., a window or Gaussian kernel). In the absence of an explicit semantic distance metric, we utilize spatial proximity within the sequence as a robust proxy for semantic overlap. The normalization factor Z_j ensures that the singleton utility of a token remains stable, satisfying $F_{sub}(\{j\}) = s_j$.

Crucially, as a facility location function, $F_{sub}(S)$ is inherently monotone submodular (Cornuéjols et al., 1983), satisfying the property of diminishing returns. The marginal gain of adding a token k to a set S, defined as $\Delta(k \mid S) = \sum_{i \in V} \max(0, w_{i,k} - \max_{j \in S} w_{i,j})$, decreases as S grows, particularly when S already contains tokens that are spatially close to k.

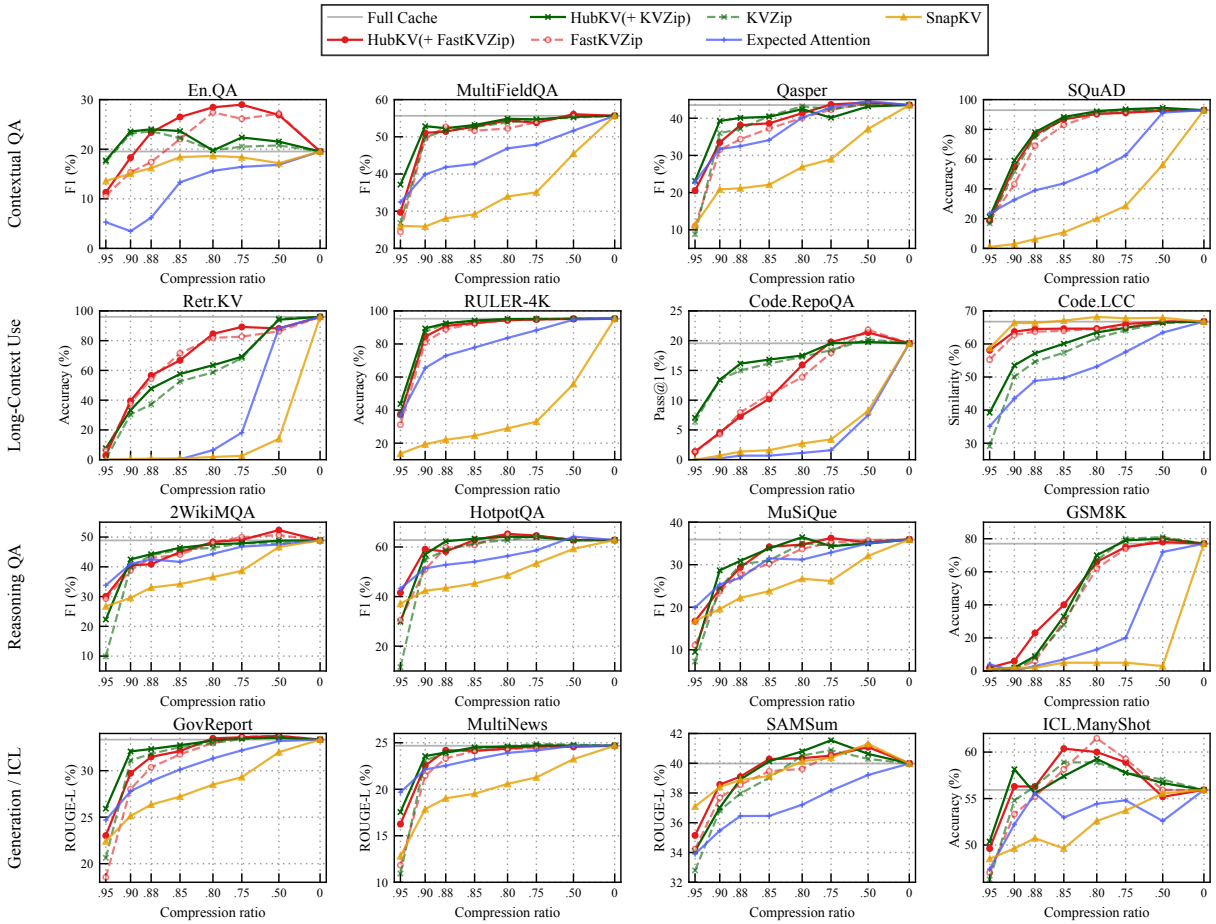


Figure 3: **Main prefill results on Qwen3-8B.** We compare HubKV-refined KVZip/FastKVZip scores against the corresponding base scorers, SnapKV, Expected Attention, and the full-cache reference across 16 tasks. The x-axis reports the true KV compression ratio, ordered from aggressive compression on the left to full cache on the right.

3.3 One-Pass Marginal-Gain Proxy via HubKV

The objective in Eq. 4 defines an ideal redundancy-aware utility, but optimizing it directly requires repeatedly evaluating marginal gains against the current selected set. This sequential dependency is undesirable for KV-cache inference, where practical methods must preserve a simple parallel scoring and Top-K-style pruning path. We therefore propose **HubKV** as a one-pass marginal-gain proxy rather than a global submodular optimizer. HubKV keeps the original importance scorer unchanged, applies a bounded redundancy-aware correction to its scores, and then uses the same compression-ratio-based pruning interface as the base method.

For clarity, the submodular objective above is written at the token level. In implementation, importance scores are maintained for every layer and attention head. Let $\mathbf{s} \in [0,1]^{L\times H\times N}$ denote the raw score tensor produced by a base scorer such as FastKVZip, and let $P_{\ell,h,i}$ indicate protected tokens, including attention sinks and the recent local window. HubKV transforms only the non-protected content scores and assigns protected tokens the maximum retention score; Appendix A.1 details the protected-token mask, budget accounting, hub detection, and head weights.

1. **Local Hub Detection:** For each layer ℓ and head h, we define a local window $\mathcal{W}_i = [i-k, i+k]$ and identify whether token i is a local hub:

$$M_{\ell,h,i} = \mathbb{I}\left(i = \arg\max_{j\in\mathcal{W}_i} s_{\ell,h,j}\right), \quad (6)$$

   where deterministic index-based tie-breaking is used for score plateaus. This operation is implemented with a 1D max-pooling primitive over the sequence dimension.

2. **Submodular Marginal Discounting:** For non-hub tokens, we apply a discount factor $\gamma \in (0,1)$ to approximate the residual

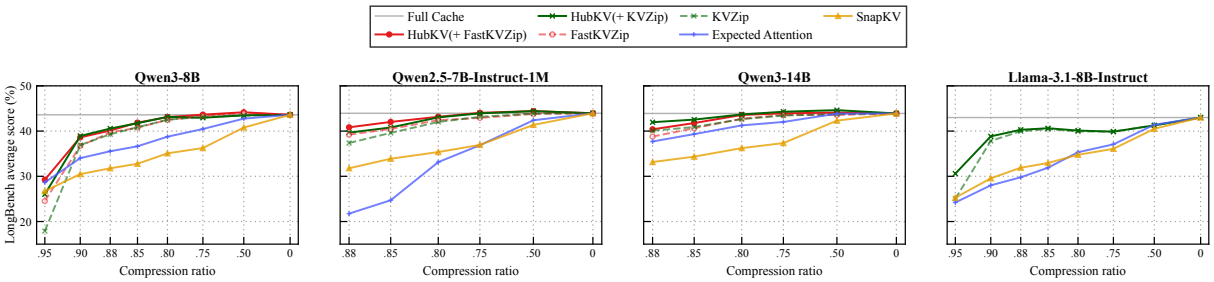


Figure 4: **LongBench average across model backbones.** We report LongBench average results for Qwen3-8B, Qwen2.5-7B-Instruct-1M, Qwen3-14B, and Llama-3.1-8B-Instruct. Each panel reports the compression ratios available for that backbone and includes the full-cache reference at $r = 0.00$.

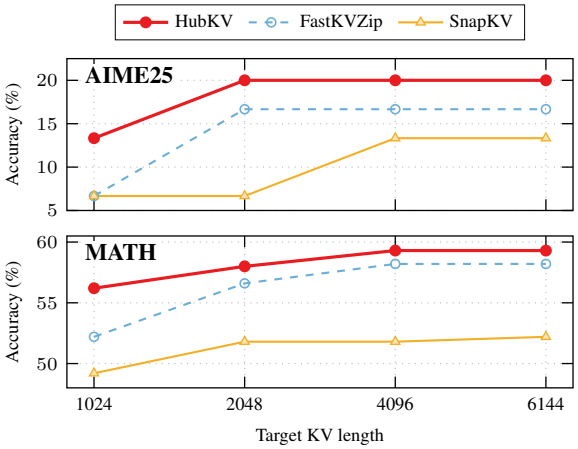


Figure 5: **Decoding-stage results on Qwen3-8B.** We evaluate AIME25 and MATH with a cache update interval of 128 decoding steps and target KV lengths of 1024, 2048, 4096, and 6144. HubKV denotes HubKV(+FastKVZip), which refines FastKVZip scores before the same target-length pruning step is applied.

marginal utility after a nearby hub has already covered the local region:

$$\tilde{s}_{\ell,h,i} = M_{\ell,h,i} s_{\ell,h,i} + (1 - M_{\ell,h,i}) \gamma s_{\ell,h,i}. \quad (7)$$

This step is the local analogue of submodular diminishing returns: tokens near a stronger local representative are not discarded outright, but their scores are pushed downward before global pruning.

3. **Bounded Head-Wise Selectivity Calibration:** Some attention heads exhibit sharp, spiky score distributions, while others remain flatter and less discriminative. We measure head selectivity on the non-protected content region using the coefficient of variation

$$c_{\ell,h} = \frac{\sigma_{\ell,h}}{\mu_{\ell,h} + \epsilon}, \quad (8)$$

where $\mu_{\ell,h}$ and $\sigma_{\ell,h}$ are computed over unprotected tokens. The relative head weight is then clipped to a bounded interval:

$$\beta_{\ell,h} = \mathrm{clip}\left(\left(\frac{c_{\ell,h}}{\bar{c}_\ell} \right)^{\tau}, \beta_{\min}, \beta_{\max} \right), \quad (9)$$
$$u_{\ell,h,i} = \beta_{\ell,h} \tilde{s}_{\ell,h,i},$$

where $\bar{c}_\ell$ is the mean selectivity across heads at layer ℓ, τ controls calibration strength, and clipping prevents head calibration from overwhelming the base scorer.

4. **Compression-Ratio Gating:** The redundancy correction should be weak when the cache is only lightly compressed and stronger when cache pressure is high. Given compression ratio r and gate exponent p, HubKV mixes the raw and corrected scores as

$$\lambda = r^p, \qquad z_{\ell,h,i} = (1 - \lambda) s_{\ell,h,i} + \lambda u_{\ell,h,i}. \quad (10)$$

For protected tokens, we set $z_{\ell,h,i} = 1$ so that sink and recent-window tokens remain available to the underlying pruning rule.

Finally, HubKV passes the calibrated scores $\mathbf{z}$ to the same compression operator used by the base scorer: KV pairs with the lowest calibrated scores are evicted until the target compression ratio is reached, either globally or layer-wise depending on the base configuration. Equivalently, if B KV entries are retained under a given budget, HubKV keeps the B highest entries under $\mathbf{z}$. This keeps HubKV compatible with existing Top-K-style pruning implementations while changing the ranking criterion from purely modular scores to a redundancy-aware, ratio-gated correction. The additional score-refinement computation consists of local max-pooling, content-region head

statistics, and element-wise score mixing, yielding $\mathcal{O}(LHNk)$ work with a small kernel size k and no sequential marginal-gain loop. Because the cache interface and retained budget are unchanged, Section 4 reports downstream evaluations under matched cache budgets.

We formally analyze the SMD core of HubKV in Appendix J. Specifically, we establish the *Local Hub Dominance* of the SMD-induced ranking and prove a *Local Marginal Sandwich* (Prop. 2), showing that the refined score $\tilde{s}_{\ell,h,i}$ serves as a local proxy for true marginal gains under localized overlap. We then characterize the ratio-gated head-wise calibration as a bounded perturbation of the base scores. This analysis intentionally separates three claims: the ideal coverage objective is submodular, sequential greedy is the theoretical oracle for that objective, and HubKV is a parallel local proxy designed for efficient inference rather than a sequential submodular optimizer.

4 Experiments

In this section, we evaluate whether the redundancy-aware score correction introduced by **HubKV** improves practical KV cache compression. Since HubKV is designed as a plug-in refinement rather than a standalone scorer, the central comparison is paired: we apply the same HubKV correction on top of KVZip and FastKVZip scores, and compare each refined variant against its own base scorer under the same compression budget. We first report the main prefill results on Qwen3-8B across a diverse suite of long-context and reasoning tasks, and then evaluate whether the same score correction remains useful in a decoding-stage setting.

4.1 Experimental Setup

Models, tasks, and baselines. The main evaluation uses Qwen3-8B (Yang et al., 2025) and a 16-task suite spanning contextual QA, retrieval and code use, reasoning, summarization, and in-context learning. We evaluate HubKV as a score-refinement layer on top of KVZip and FastKVZip, and compare each refined variant with its own base scorer under the same cache budget. We also include SnapKV and Expected Attention (Devoto et al., 2025) as attention-based baselines. Additional cross-backbone LongBench evaluations on Qwen2.5-7B-Instruct-1M, Llama-3.1-8B-Instruct, and Qwen3-14B are reported in Appendices F, G, and H. Detailed task groupings, metrics, and baseline descriptions are provided in Appendix A.

Family	0.95	0.90	0.88	0.85	0.80	0.75	0.50	Avg.
HubKV(+FastKVZip) − FastKVZip								
Contextual QA	+3.91	+4.56	+4.00	+2.65	+1.14	+0.90	+0.06	+2.46
Long-Context Use	+1.32	+1.60	+1.08	-1.09	+1.22	+2.39	+0.33	+0.98
Reasoning QA	+4.60	+3.80	+3.71	+4.22	+1.32	+0.35	-0.24	+2.54
Generation / ICL	+3.10	+1.66	+0.91	+0.85	-0.05	-0.06	-0.24	+0.88
Overall	+3.23	+2.90	+2.42	+1.66	+0.91	+0.89	-0.02	+1.71
HubKV(+KVZip) − KVZip								
Contextual QA	+7.29	+3.47	+1.81	+0.70	+0.21	+0.06	+0.48	+2.00
Long-Context Use	+6.02	+2.12	+3.59	+2.17	+1.70	+0.75	-0.23	+2.30
Reasoning QA	+8.48	+2.46	+1.45	+2.54	+1.78	-0.43	-0.06	+2.32
Generation / ICL	+4.32	+1.24	+0.19	-0.02	+0.21	+0.15	-0.06	+0.86
Overall	+6.53	+2.32	+1.76	+1.35	+0.98	+0.13	+0.03	+1.87

Table 1: **Paired average gain by task family and compression ratio.** Each value is the mean score-point difference between HubKV-refined scores and the corresponding base scorer, averaged over all tasks in the family at the specified compression ratio in Figure 3.

Compression and implementation. For prefill compression, the ratio r denotes the fraction of KV entries removed; we evaluate seven ratios between 0.50 and 0.95 and focus on paired differences at the same budget. Unless otherwise stated, HubKV uses a local kernel of size 5, discount factor $\gamma = 0.5$, head-selectivity exponent $\tau = 0.5$, clipping range $[0.8, 1.2]$, and gate exponent $p = 2$. HubKV only refines scores before the same Top-K-style pruning step used by the base method; full implementation and decoding-stage settings are deferred to Appendix A.

4.2 Main Results

Figure 3 shows that HubKV most consistently helps under aggressive compression, where the retained budget is small and redundant local clusters are most damaging. This trend matches the design of the compression-ratio gate: when cache pressure is high, HubKV applies a stronger local marginal-gain correction; when the budget is looser, the refined score stays closer to the base scorer.

Figure 4 isolates LongBench performance across model backbones. The averaged curves show the same pattern as the full Qwen3-8B suite: HubKV tends to track or improve the corresponding base scorer at moderate compression and separates more clearly under stronger compression. The paired summary in Table 1 quantifies the task-family gains; Appendix C discusses these trends.

4.3 Decoding-Stage Results

We further evaluate HubKV in a decoding-stage setting, where the cache is compressed repeatedly during generation. Figure 5 compares HubKV(+FastKVZip) with FastKVZip and

SnapKV on AIME25 and MATH (Hendrycks et al., 2021) using target KV lengths of 1024, 2048, 4096, and 6144. HubKV improves over FastKVZip at every target length: on AIME25, accuracy rises from 6.67% to 13.33% at length 1024 and from 16.67% to 20.00% at larger lengths; on MATH, HubKV gains 4.0 points at length 1024 and 1.1–1.4 points elsewhere. These results suggest that the same local redundancy correction remains useful beyond one-shot prefill pruning.

5 Related Work

KV Cache Eviction and Sparsification. Efficient LLM inference is increasingly bottlenecked by the memory footprint of the Key-Value (KV) cache (Kwon et al., 2023). StreamingLLM (Xiao et al., 2024) and LM-Infinite (Han et al., 2024) preserve attention sinks and local context for stable long generation, while H2O (Zhang et al., 2023), Scissorhands (Liu et al., 2023), and FastGen (Ge et al., 2024) evict tokens using accumulated attention or observed attention patterns. These methods are effective, but they primarily operate during decoding and rely on past attention as a proxy for future utility.

Prefill-based and Query-Agnostic Compression. Prefill-time methods move compression earlier. SnapKV (Li et al., 2024) and PyramidKV (Cai et al., 2024) profile prefill attention to choose persistent KV pairs, and DuoAttention (Xiao et al., 2025) applies sliding-window attention selectively across heads. Expected Attention (Devoto et al., 2025) estimates token utility by approximating how a distribution of future queries will attend to cached keys, while KVZip (Kim et al., 2026b) estimates query-agnostic importance through context reconstruction. HubKV follows this score-based prefill interface, but changes the ranking criterion by discounting locally redundant high-score neighbors before pruning.

Recent work also studies structure inside the KV cache. ChunkKV (Liu et al., 2026) retains semantic chunks rather than isolated tokens, while FastKVZip (Kim et al., 2026a) predicts KV utility with a hardware-friendly gating unit. HubKV is complementary to such scorers: it does not replace their importance estimates, but refines them with a submodularly motivated local redundancy penalty before the same Top-K-style eviction step. Across these methods, token utility may be derived from past attention, future-query estimation, reconstruction, or learned prediction, yet the final retention rule is usually still an independent Top-K-style ranking. HubKV targets this shared final step, keeping the score source and cache layout unchanged while adding local redundancy awareness to the ranking.

This distinction also separates HubKV from methods that change the unit of allocation or the attention pattern itself. Rather than assigning a new budget to each layer, head, or semantic chunk, HubKV operates after the base utility scores have already been computed and before the standard pruning operator is invoked. This makes HubKV easy to compose with query-agnostic score sources in practice.

6 Conclusion

We presented HubKV, a redundancy-aware score refinement layer for KV cache compression. HubKV uses an ideal submodular local coverage objective as a theoretical guide, but deploys only a hardware-parallel proxy: it detects local score hubs, softly discounts neighboring non-hubs, calibrates scores by bounded head selectivity, and gates the correction by compression ratio. On Qwen3-8B, HubKV improves FastKVZip by 3.23 points and KVZip by 6.53 points on average at 95% prefill compression, and by +1.71/+1.87 points on average across budgets. Decoding-stage experiments show consistent improvements on AIME25 and MATH.

7 Limitations

HubKV is most useful under aggressive KV compression, where retaining locally redundant tokens is particularly costly. Its benefit depends on the quality of the base importance scorer and on the assumption that nearby high-scoring tokens often carry overlapping information. When adjacent tokens encode complementary evidence, such as formulas, code fragments, or tightly coupled reasoning chains, local discounting may occasionally perturb useful evidence. HubKV should therefore be viewed as a bounded parallel ranking proxy rather than a globally guaranteed submodular optimizer.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd*

annual meeting of the association for computational linguistics (volume 1: Long papers), pages 3119–3137.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. 1983. The uncapacitated facility location problem. Technical report, Cornell University Operations Research and Industrial Engineering.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 2978–2988.

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

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. 2026. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *Advances in Neural Information Processing Systems*, 38:113152–113188.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive kv cache compression for llms. In *International Conference on Learning Representations*, volume 2024, pages 22975–22988.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. 2026a. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. 2026b. Kvzip: Query-agnostic kv cache compression with context reconstruction. *Advances in Neural Information Processing Systems*, 38:167563–167591.

Andreas Krause and Daniel Golovin. 2014. Submodular function maximization. *Tractability*, 3(71-104):3.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and 1 others. 2025. Scbench: A kv cache-centric analysis of long-context methods. In *International Conference on Learning Representations*, volume 2025, pages 66063–66093.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Bo Li, Xuming Hu, and Xiaowen Chu. 2026. Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference. *Advances in Neural Information Processing Systems*, 38:28728–28778.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *International Conference on Machine Learning*.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 2383–2392.

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

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *International Conference on Learning Representations*, volume 2025, pages 37228–37253.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pages 21875–21895.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Experimental Details

The reported main prefill evaluation uses Qwen3-8B as the backbone model. The benchmark suite contains 16 tasks drawn from LongBench (Bai et al., 2024), SCBench (Li et al., 2025), RULER (Hsieh et al., 2024), SQuAD (Rajpurkar et al., 2016), and GSM8K (Cobbe et al., 2021). The selected tasks cover contextual question answering (En.QA, MultiFieldQA, Qasper, SQuAD), long-context retrieval and code use (Retr.KV, RULER-4K, Code.RepoQA, Code.LCC), multi-hop and mathematical reasoning (2WikiMQA, HotpotQA, MuSiQue, GSM8K), and generation or in-context learning (GovReport, MultiNews, SAMSum, ICL.ManyShot). We report the standard task metric used by each benchmark, including accuracy, F1, ROUGE-L, Pass@1, and code similarity. Because these metrics have different natural scales, we primarily interpret paired differences between a base scorer and its HubKV-refined counterpart on the same task and compression ratio.

Following KVZip and FastKVZip, the main experiment uses the prefill-stage compression setting, where a score is assigned to historical KV entries before downstream generation. The compression ratio r denotes the fraction of KV entries removed, so $r = 0$ corresponds to the full-cache reference and larger values correspond to more aggressive eviction. We evaluate $r \in \{0.50, 0.75, 0.80, 0.85, 0.88, 0.90, 0.95\}$, with particular attention to the high-compression regime where local redundancy is most likely to waste the remaining cache budget. The full-cache score is included as an uncompressed reference, but it should not be interpreted as an oracle upper bound: on some tasks, removing distracting context can improve the final task score.

We compare two deployed HubKV variants, HubKV(+FastKVZip) and HubKV(+KVZip), against their corresponding base scorers FastKVZip (Kim et al., 2026a) and KVZip (Kim et al., 2026b). We additionally include SnapKV (Li et al., 2024) and Expected Attention as representative attention-based baselines. This evaluation is query-agnostic for KVZip, FastKVZip, and their HubKV-refined variants, matching the intended use case where a compressed shared cache may be reused across downstream queries.

Unless otherwise stated, HubKV uses a 1D local max-pooling kernel of size 5, a non-hub discount factor $\gamma = 0.5$, a bounded head-wise selectivity exponent $\tau = 0.5$, clipping range $[0.8, 1.2]$, and compression-ratio gate exponent $p = 2$. Protected sink tokens and the recent local window follow the underlying base scorer and bypass the HubKV correction. HubKV does not alter the cache data structure or the final Top-K-style pruning interface; it only refines the layer-head token scores before the same budgeted eviction step is applied.

A.1 Protected-Token Handling

HubKV follows the protected-token convention of the underlying KV compression method. The protected set contains attention sink tokens and the recent local window, which are retained to preserve stable long-context generation and short-range continuity. Let T denote the current sequence length. We define the protected mask for each layer ℓ, head h, and token position i as

$$P_{\ell,h,i} = \mathbb{I}[i \in \mathcal{P}_{\text{sink}} \cup \mathcal{P}_{\text{recent}}], \tag{11}$$

where $\mathcal{P}_{\text{sink}}$ denotes the prefix attention-sink positions and $\mathcal{P}_{\text{recent}}$ denotes the trailing recency window. In our default FastKVZip/KVZip-based configuration, we inherit the base scorer's protection rule; for the score-trace analysis in Appendix I, this corresponds to four sink tokens and a recent window of $\lfloor 0.02T \rfloor$ tokens.

HubKV applies its redundancy-aware correction only to the unprotected content region

$$\mathcal{C}_{\ell,h} = \{i : P_{\ell,h,i} = 0\}. \tag{12}$$

Protected tokens are excluded from the content-region head statistics used for head-wise calibration, including the mean, standard deviation, and coefficient of variation. They are also masked out of local hub competition: for a content token i, local max-pooling is evaluated over unprotected positions in the local window, so protected sink or recency tokens do not suppress nearby content tokens.

For $i \in \mathcal{C}_{\ell,h}$, HubKV computes the local-hub mask, SMD-discounted score, head-calibrated score, and ratio-gated score as described in Section 3.3. For protected positions, the final score is overwritten after score refinement:

$$z_{\ell,h,i} = \begin{cases} 1, & P_{\ell,h,i} = 1, \\ (1-\lambda)s_{\ell,h,i} + \lambda u_{\ell,h,i}, & P_{\ell,h,i} = 0. \end{cases} \tag{13}$$

This overwrite lets protected tokens bypass local discounting, head-wise calibration, and compression-ratio gating.

Protected tokens are not an additional cache allocation introduced by HubKV. They are retained under the same budget accounting as the underlying compression method. Equivalently, if the base method retains $B_{\ell,h}$ KV entries for a given layer and head, protected tokens occupy part of this retained budget, and the remaining budget is assigned to the highest-scoring unprotected content tokens under $z_{\ell,h,i}$. When the base method uses a global or layer-wise pruning rule rather than a per-head rule, HubKV follows the same pruning granularity and only changes the scores passed to that pruning operator. The reported compression ratio is computed after final retention: the final retained cache includes both protected positions and selected content positions and is matched to the target budget whenever the target budget exceeds the number of protected positions, which holds for our evaluated settings.

This design preserves the stability safeguards already used by the base compressor while preventing the local redundancy correction from being driven by tokens retained for architectural or generation-stability reasons. As a result, the paired comparisons in the main experiments isolate the effect of HubKV on the rank ordering of non-protected content tokens while keeping the protected-token policy fixed.

For the decoding-stage experiment, the cache is repeatedly compressed during generation rather than compressed only once after prefill. We use Qwen3-8B, set the cache update interval to 128 decoding steps, and evaluate fixed target KV lengths of 1024, 2048, 4096, and 6144. HubKV refines FastKVZip scores before applying the same target-length pruning operation, allowing the decoding-stage comparison to isolate the effect of the score correction under an identical cache budget.

The current evaluation isolates HubKV through paired comparisons against the same base scorer under the same cache budget. Table 2 complements the main evaluation with Qwen3-8B A100 ablations that contrast the final ratio-gated correction with an ungated variant and with several head-budget allocation alternatives.

B Ablation Results

Table 2 summarizes Qwen3-8B A100 ablation results. The component block shows that the final ratio-gated HubKV variant performs best for both base scorers in this setting. The head-budget block tests a different design choice: reallocating per-head budgets by waterfilling, entropy, selectivity, or a hybrid score is not a reliable substitute for bounded score correction, whereas HubKV is the only alternative with clear positive gains for both FastKVZip and KVZip under the same task subset.

Branch	Base	Ungated	HubKV	Δ Ungated	Δ HubKV	n
Component ablation at $r = 0.75$						
FastKVZip	58.62	59.99	60.50	+1.37	+1.88	6
KVZip	56.28	56.07	56.41	-0.21	+0.13	6
Branch	Base	HubKV	Waterfill	Entropy	Select.	Hybrid
Head-budget alternatives at $r = 0.88$						
FastKVZip	34.99	+5.51	-3.50	-7.57	-16.77	-7.46
KVZip	35.82	+2.11	+0.18	-7.33	-19.32	-9.07

Table 2: **Ablation of HubKV design choices on Qwen3-8B.** The component block reports average scores and gains over the paired base scorer on six shared ablation tasks; "Ungated" applies the local hub discounting and head-wise calibration without the compression-ratio gate. The head-budget block reports gains over the paired base scorer on six shared ablation tasks.

C Detailed Results Analysis

The paired summary in Table 1 confirms that the improvement in Figure 3 is not merely a visual artifact of overlapping curves. Across all 112 task-ratio pairs, HubKV(+FastKVZip) improves over FastKVZip by an average of 1.71 points, while HubKV(+KVZip) improves over KVZip by an average of 1.87 points. The gains are strongest under aggressive compression: at $r = 0.95$, the average improvements are +3.23 and +6.53 points, respectively. In absolute terms, the average score at $r = 0.95$ rises from 21.42 to 24.65 for the FastKVZip branch, and from 17.69 to 24.22 for the KVZip branch.

The task-family breakdown shows that the improvement is broadly distributed rather than concentrated in a single benchmark group. For the FastKVZip branch, contextual QA and reasoning QA show the largest average gains, while the KVZip branch benefits most on long-context use and reasoning QA. This pattern is consistent with the motivation of HubKV: tasks that require retaining evidence from several separated parts of the context are more likely to benefit when the cache budget shifts away from locally redundant high-score neighbors toward broader evidence.

The averaged LongBench comparison in Figure 4 provides an additional model-level view. The Qwen backbones retain both FastKVZip and KVZip branches, and HubKV generally improves

or closely tracks the paired base scorer. On Llama-3.1-8B-Instruct, the available data contain the KVZip branch, where the HubKV-refined curve separates most clearly at high compression. The exact set of compression ratios differs across the exported runs, so the figure should be read as a backbone-level robustness check rather than as a strict model ranking.

The results also expose the intended limitation of the method. At moderate compression, especially $r = 0.50$, the paired gains largely vanish and can become slightly negative. This is consistent with HubKV's role as a bounded correction rather than a replacement scorer: when the cache budget is already sufficient, suppressing neighboring high-score tokens is less necessary and may occasionally perturb a useful local evidence chain. The main benefit therefore lies in the aggressive compression regime, where retaining multiple near-duplicate tokens is most expensive and a local marginal-gain proxy can redirect the budget toward broader context coverage.

D Code and Math Locality Stress Test

HubKV uses positional locality as a redundancy prior: nearby high-scoring tokens are often overlapping representatives of the same local evidence. This prior is useful on average, but it is not a semantic-equivalence oracle. Code and mathematical expressions provide a natural stress test because adjacent symbols can be complementary rather than redundant, and useful dependencies may also be non-contiguous.

We therefore isolate code and math tasks from the Qwen3-8B A100 snapshot and report paired gains against the corresponding base scorer under the same cache budget. The prefill tasks are Code.RepoQA, Code.LCC, and GSM8K. We also include the decoding-stage MATH result from the fixed-target-KV experiment in Section 4.3. Table 3 reports both positive and negative entries instead of only aggregate averages.

The aggregate results remain positive on average for all six prefill branches, especially on Code.LCC and GSM8K. At the same time, the negative entries are informative. For example, HubKV(+FastKVZip) loses 0.68 points on Code.RepoQA at $r = 0.88$ and $r = 0.85$, and loses 2.00 points on GSM8K at $r = 0.50$. These pockets support the limitation that local discounting can occasionally demote adjacent tokens that jointly encode a code fragment, formula, or reasoning chain.

The decoding-stage MATH result is positive at all tested target KV lengths. Relative to FastKVZip, HubKV(+FastKVZip) improves MATH accuracy by +4.00, +1.40, +1.10, and +1.10 points at target KV lengths 1024, 2048, 4096, and 6144, respectively, for an average gain of +1.90 points. This suggests that the locality correction remains useful in math reasoning overall, despite the failure modes exposed by individual prefill budgets.

We also ran a retained-token chain diagnostic on 32 short examples, split evenly between code snippets and math snippets. For each token inside the code or formula span, we measured the fraction of layer/head positions in which that token was retained by the base FastKVZip scorer and by HubKV(+FastKVZip). Table 4 summarizes the mean retention fraction over the annotated spans.

Taken together, these results support a conservative interpretation. HubKV is generally beneficial on code and math groups in the tested setting, but the locality prior should be described as a redundancy heuristic rather than a guarantee that adjacent tokens are interchangeable. This is why the deployed method uses soft discounting, bounded head-wise calibration, and compression-ratio gating instead of hard non-maximum suppression.

E Efficiency Analysis

We evaluate the computational cost of HubKV in two ways. First, a microbenchmark isolates the score-refinement path using synthetic score tensors with the Qwen3-8B score shape $[L, B, H_{KV}, N]$, where $L = 36$ and $H_{KV} = 8$. Second, an end-to-end benchmark loads Qwen3-8B in `bfloat16` and compares FastKVZip with HubKV(+FastKVZip) at compression ratio $r = 0.95$. The HubKV configuration is the same mild ratio-gated local-hub variant used in the main experiments.

Table 5 reports the score-stage microbenchmark. The measured operation "HubKV refine+select" applies local hub detection, bounded head-wise calibration, ratio-gated score mixing, and then the same bottom-score selection plan as the base method. The relative overhead is visible because the base selection stage is already small; however, the absolute latency remains in the low-millisecond range in our setup.

Table 6 reports the end-to-end timing result for Qwen3-8B with batch size 1 and 32 decoded tokens.

Task	Branch	0.95	0.90	0.88	0.85	0.80	0.75	0.50	Avg.	Worst
Code.RepoQA	H(+FastKVZip)	+0.00	+0.23	-0.68	-0.68	+2.05	+1.82	-0.46	+0.33	-0.68 @ 0.88
Code.RepoQA	H(+KVZip)	+0.69	+0.00	+1.14	+0.68	+0.23	+1.14	-0.46	+0.49	-0.46 @ 0.50
Code.LCC	H(+FastKVZip)	+2.86	+0.99	+0.81	+0.66	+0.02	+1.10	-0.07	+0.91	-0.07 @ 0.50
Code.LCC	H(+KVZip)	+10.10	+3.43	+2.55	+2.73	+1.69	+0.76	+0.23	+3.07	+0.23 @ 0.50
GSM8K	H(+FastKVZip)	+1.00	+5.00	+17.00	+10.00	+4.00	+1.00	-2.00	+5.14	-2.00 @ 0.50
GSM8K	H(+KVZip)	+1.00	+1.00	+1.00	+5.00	+3.00	-1.00	-1.00	+1.29	-1.00 @ 0.75

Table 3: **Code and math gain-loss analysis on Qwen3-8B.** Each value is the paired score difference between HubKV-refined scores and the corresponding base scorer under the same prefill compression ratio. "H" denotes HubKV. Positive values indicate improvements; negative values identify cases where the locality prior can perturb useful local evidence chains.

r	Ex.	Tok.	Base	HubKV	Δ	Changed
0.50	32	415	.4298	.4184	-.0114	82.2%
0.75	32	415	.1870	.1677	-.0193	90.4%
0.80	32	415	.1353	.1155	-.0197	91.8%
0.85	32	415	.0849	.0641	-.0207	89.6%
0.88	32	415	.0532	.0303	-.0228	91.8%
0.90	32	415	.0307	.0100	-.0207	60.7%
0.95	32	415	.0131	.0000	-.0131	26.3%

Table 4: **Retained-token chain diagnostic for code and math spans.** Values are mean per-token retention fractions over annotated code/formula spans, averaged across examples and positions. The diagnostic is qualitative: lower span retention under HubKV is consistent with the small negative pockets in Table 3, but downstream benchmark deltas remain the primary evidence.

In this full inference path, the score-refinement overhead is largely hidden by the model prefill and decode computation. The maximum measured TTFT overhead is +1.26%, and the median TTFT overhead is -0.12%. Prefill throughput stays between 0.984× and 1.012× of FastKVZip, and decode throughput is effectively unchanged.

The efficiency results qualify the lightweight claim. HubKV adds measurable work to the score-selection stage, but this work is small in absolute terms and does not produce a meaningful end-to-end TTFT regression in the tested Qwen3-8B setup. Peak memory increases with sequence length because HubKV materializes temporary score-refinement tensors, from +9.0 MB at 4k to +72.0 MB at 32k relative to FastKVZip, which is modest compared with the full model inference footprint in this setup.

F Additional LongBench Results on Qwen2.5-7B-Instruct-1M

We further evaluate the same prefill-compression setting on Qwen/Qwen2.5-7B-Instruct-1M and report LongBench average results. The full-cache reference is taken from FastKVZip at compression ratio $r = 0.00$. Averaged over LongBench and five compressed ratios, HubKV improves FastKVZip by +1.14 average points and KVZip by +1.16 average points. The gains grow with compression pressure, reaching +1.63 points for the FastKVZip branch and +2.29 points for the KVZip branch at $r = 0.88$.

G Additional LongBench Results on Llama-3.1-8B-Instruct

We also evaluate the same LongBench average setting on meta-llama/Llama-3.1-8B-Instruct, aligned with Appendix F. The exported Llama-3.1 data contains the KVZip branch, Expected Attention, SnapKV, and the full-cache reference from SnapKV at compression ratio $r = 0.00$. Averaged over LongBench and seven compressed ratios, HubKV improves KVZip by +1.03 average points. The average gain is small near moderate compression but increases in the high-compression regime, reaching +5.56 points at $r = 0.95$.

H Additional LongBench Results on Qwen3-14B

We further evaluate the same LongBench average setting on Qwen/Qwen3-14B, aligned with Appendix G. This run includes both FastKVZip and KVZip branches, together with Expected Attention, SnapKV, and the full-cache reference from SnapKV at compression ratio $r = 0.00$. Averaged over LongBench and five compressed ratios, HubKV improves FastKVZip by +0.90 average points and KVZip by +1.26 average points. The gains are again larger under stronger compression, reaching +1.60 points for the FastKVZip branch and +1.89 points for the KVZip branch at $r = 0.88$.

I Details of Observation Experiments

The observation study in Section 2.3 is a diagnostic analysis of score locality rather than a task-level evaluation. We therefore separate score capture from eviction. The score traces are produced by `insights/capture_scores.py`, which wraps the KVZip (Kim et al., 2026b) scoring module with a

Scorer	N	Batch	Base select ms	HubKV refine+select ms	Overhead	Peak MB
FastKVZip	4096	1	1.50	2.51	67.7%	47.32
FastKVZip	4096	4	6.74	10.19	51.3%	182.15
FastKVZip	8192	1	3.10	4.76	53.7%	91.98
FastKVZip	8192	4	11.86	17.40	46.8%	362.37
FastKVZip	16384	1	6.75	10.18	50.7%	182.15
FastKVZip	16384	4	15.35	20.96	36.5%	722.80
FastKVZip	32768	1	11.38	17.90	57.3%	362.37
FastKVZip	32768	4	22.15	26.73	20.7%	1445.37
KVZip	4096	1	1.48	2.49	68.0%	47.32
KVZip	4096	4	6.77	10.20	50.6%	182.15
KVZip	8192	1	3.09	4.76	54.1%	91.98
KVZip	8192	4	11.88	18.40	55.0%	362.37
KVZip	16384	1	6.76	10.08	49.1%	182.15
KVZip	16384	4	14.47	21.01	45.2%	722.80
KVZip	32768	1	11.88	17.40	46.4%	362.37
KVZip	32768	4	20.61	26.87	30.4%	1445.37

Table 5: **Score-stage efficiency microbenchmark.** Synthetic score tensors match the Qwen3-8B layer/head shape and use `bfloat16`. The table reports median latency over the measured repeats. Across all rows, the maximum relative overhead is 68.0%, but the absolute refine-plus-select latency remains below 27 ms even at $N = 32768$ and batch size 4.

Method	N	TTFT s	Prefill tok/s	Decode tok/s	Peak MB
FastKVZip	4096	0.549	8672.9	13.5	52.7
HubKV(+FastKVZip)	4096	0.552	8658.5	13.2	61.7
FastKVZip	8192	1.006	8906.6	12.0	104.7
HubKV(+FastKVZip)	8192	0.997	9013.5	11.9	123.6
FastKVZip	16384	1.967	8821.0	9.3	207.5
HubKV(+FastKVZip)	16384	1.992	8681.9	9.6	244.4
FastKVZip	32768	4.370	7754.1	7.0	415.6
HubKV(+FastKVZip)	32768	4.339	7816.6	7.0	487.6

Table 6: **End-to-end efficiency on Qwen3-8B.** We compare FastKVZip and HubKV(+FastKVZip) at compression ratio $r = 0.95$, batch size 1, and 32 generated tokens. TTFT denotes prefill plus the first decode step. The measured TTFT overheads for HubKV are +0.49%, -0.89%, +1.26%, and -0.72% for $N = 4096, 8192, 16384, 32768$, respectively.

capture-only press. The wrapper runs Qwen3-8B on 500 RULER-4096 test contexts (Hsieh et al., 2024), truncates each input to 2,048 tokens, triggers the prefill scoring phase through a one-token greedy generation call, and saves the resulting score tensor without evicting any KV entry. Each saved tensor has shape $36 \times 8 \times 2048$, corresponding to model layers, KV heads, and token positions.

All three score-based observation experiments use the same content-token preprocessing in `insights/analyze.py`. We remove the four protected sink tokens and the trailing local recency window used by the base scorer. With the 2,048-token inputs used here, the recency window is $\lfloor 0.02T \rfloor = 40$ tokens, leaving 2,004 content positions per sample. Unless otherwise stated, a token's scalar importance is the mean score over all layers and KV heads.

For Observation 1, we sort content tokens by descending importance and divide the content region into 64 equal-width segments. At each Top-K selection step, the marginal spatial gain is $1/64$ if the selected token falls in a segment not previously covered, and 0 otherwise. The curve in Figure 1(a) reports this marginal gain over the first 80 selections, averaged over 500 samples, with a fixed random-order baseline used only to contextualize the coverage saturation behavior.

For Observation 2, we measure locality through the normalized autocorrelation of the mean importance sequence. For each sample, we center the importance vector and compute $R(d) = \mathbb{E}_i[(s_i - \bar{s})(s_{i+d} - \bar{s})]/\mathbb{E}_i[(s_i - \bar{s})^2]$ for token distances $d = 1, \ldots, 150$. Figure 1(b) reports the mean and sample standard deviation across the 500 score traces.

For Observation 3, the 1D-NMS diagnostic uses a max-pooling neighborhood of size 7. Tokens that are local maxima keep their original importance score, while non-maximal neighbors are multiplied by 0.3. We then rank tokens by the NMS-adjusted score and compare its segment coverage against ordinary Top-K under budgets $K = 1, \ldots, 80$. The uniform-spread line in Figure 1(c) is an oracle-style reference that places selected tokens evenly across the content sequence; it is not a deployable eviction baseline for inference.

The raw-attention validation uses `insights/capture_attention_hooks.py`. It runs Qwen3-8B with eager attention and forward

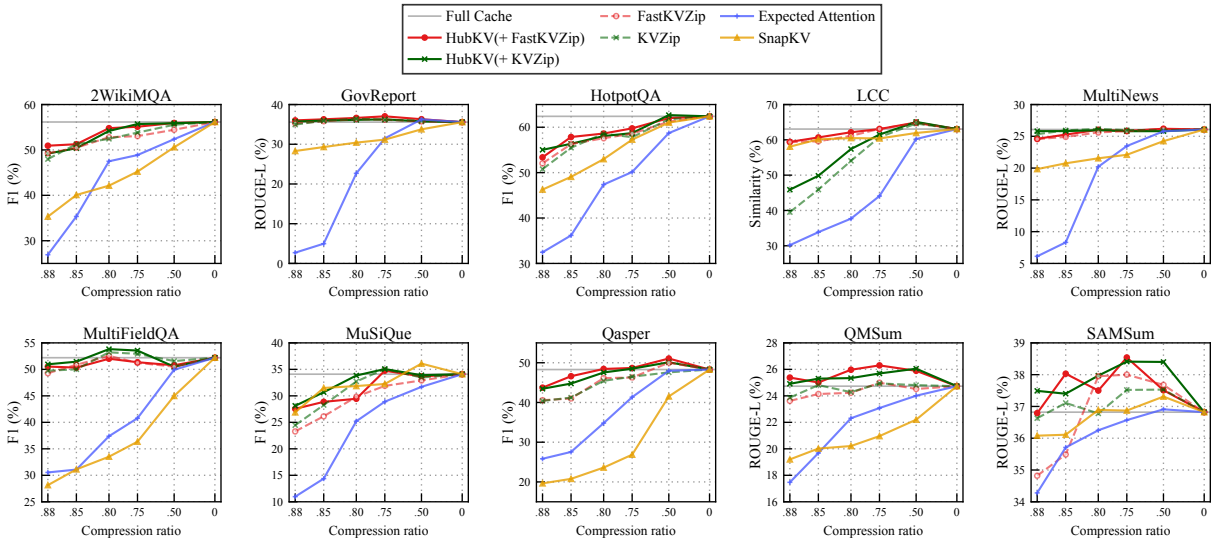


Figure 6: **Additional LongBench results on Qwen2.5-7B-Instruct-1M.** We compare LongBench average scores from HubKV-refined KVZip/FastKVZip against the corresponding base scorers, SnapKV, Expected Attention, and the full-cache reference. The x-axis reports the true KV compression ratio, ordered from aggressive compression on the left to full cache on the right.

hooks, then summarizes the content-only causal attention maps instead of materializing all full maps in Python. The saved validation trace used for the appendix figures is from LongBench 2WikiMQA, sample index 47, at 2,048 tokens. For each layer ℓ and head h, the key-side attention density is computed as $a_{\ell,h}(j) = |\mathcal{Q}|^{-1} \sum_{i \in \mathcal{Q}} A_{\ell,h}(i,j)$ over content query positions. We also store layer-wise distance profiles up to distance 128 and 192-token local crops from layers 4, 18, and 32 around the strongest key-mass region in the probe layer. The additional raw-attention autocorrelation plot trims the earliest 100 content tokens to reduce prompt-prefix bias before computing $R(d)$.

J Theoretical Analysis of the SMD Ranking Proxy

This section provides a structured analysis of the Submodular Marginal Discounting (SMD) algorithm. We analyze the SMD core of HubKV and then characterize the compression-ratio-gated head-wise calibration as a bounded perturbation of the base importance scores. We formalize the facility location objective, establish local properties of the SMD proxy, and discuss global approximation behavior for the proxy.

J.1 Properties of the Score-Weighted Coverage Objective

We first define the theoretical utility function that SMD aims to approximate.

Definition 1 (Score-Weighted Local Coverage)**.** *Let $w_{x,j} = s_j \frac{\mathcal{K}(|x-j|)}{Z_j}$ be the importance-weighted coverage kernel, where $Z_j = \sum_{x \in V} \mathcal{K}(|x-j|)$ is the normalization factor. The utility of a set S is defined as $F_{sub}(S) = \sum_{x \in V} \max_{j \in S} w_{x,j}$.*

Proposition 1 (Monotone Submodularity)**.** *The normalized objective $F_{sub}(S)$ is monotone non-decreasing and submodular, with the singleton utility property $F_{sub}(\{j\}) = s_j$ for all $j \in V$.*

Proof. Monotonicity and submodularity are standard properties of facility location functions. The singleton property follows from normalization: $F_{sub}(\{j\}) = \sum_x w_{x,j} = \frac{s_j}{Z_j} \sum_x \mathcal{K}(|x-j|) = s_j$. ■

J.2 Local Properties of SMD

Unlike iterative greedy solvers, SMD is a one-shot parallel algorithm. We first show that it strictly preserves local importance peaks.

Theorem 1 (Local Hub Dominance)**.** *For any local window $\mathcal{W}_h$ with a unique local maximum (hub) h such that $s_h > s_i$ for all $i \in \mathcal{W}_h \setminus \{h\}$, the SMD algorithm ensures that the hub remains ranked above all its discounted neighbors: $\tilde{s}_h > \tilde{s}_i$.*

Proof. By the SMD refined score definition, $\tilde{s}_h = s_h$ and $\tilde{s}_i = \gamma s_i$. Since $s_h > s_i$ and $\gamma \in (0,1)$, it follows that $s_h > \gamma s_i$. ■

We now establish the "Marginal Sandwich" which relates the refined score $\tilde{s}_i$ to the true submodular marginal gain $\Delta(i \mid \{h\})$.

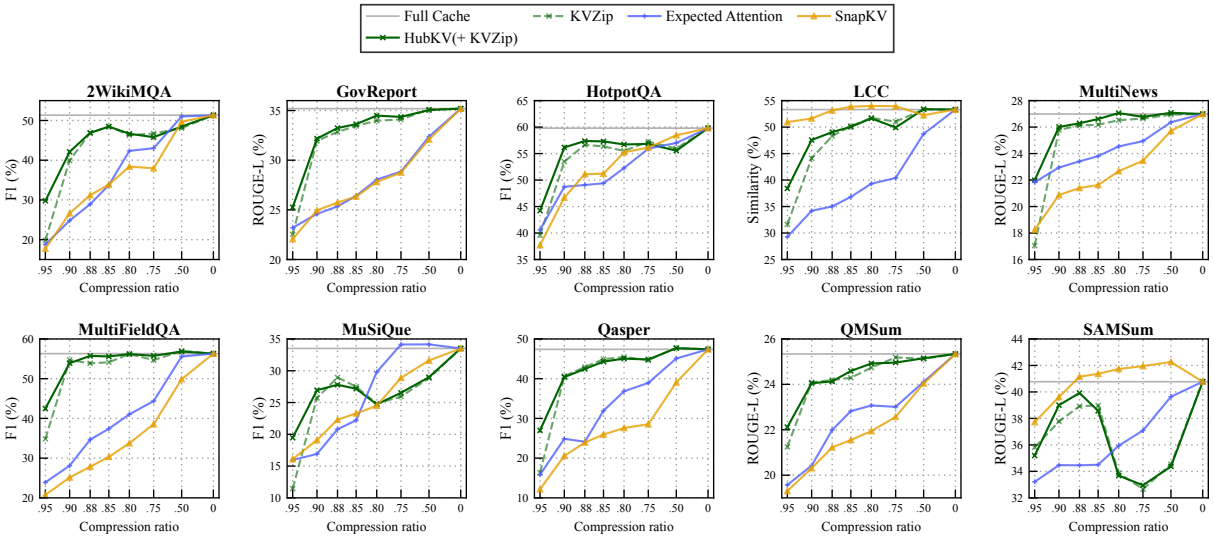


Figure 7: **Additional LongBench results on Llama-3.1-8B-Instruct.** We compare LongBench average scores from HubKV-refined KVZip against KVZip, SnapKV, Expected Attention, and the full-cache reference. The x-axis reports the true KV compression ratio, ordered from aggressive compression on the left to full cache on the right, covering compressed ratios $r \in \{0.50, 0.75, 0.80, 0.85, 0.88, 0.90, 0.95\}$ plus the full-cache point $r = 0.00$.

Proposition 2 (Local Marginal Sandwich)**.** *Let h be a selected hub and $i \in \mathcal{W}_h$ be a neighbor. Define the coverage overlap ratio as $\rho(i,h) = \frac{1}{s_i}\sum_{x\in V}\min(w_{x,i}, w_{x,h})$. The exact marginal gain is $\Delta(i \mid \{h\}) = (1-\rho(i,h))s_i$. Furthermore, if $\rho(i,h) \in [1-\gamma-\eta, 1-\gamma+\eta]$ for some error $\eta \geq 0$, then:*

$$(\gamma-\eta)s_i \leq \Delta(i \mid \{h\}) \leq (\gamma+\eta)s_i, \qquad (14)$$

and the refined score $\tilde{s}_i = \gamma s_i$ satisfies $|\Delta(i \mid \{h\}) - \tilde{s}_i| \leq \eta s_i$.

Proof. For the facility location objective, $\Delta(i \mid \{h\}) = \sum_x \max(0, w_{x,i} - w_{x,h})$. Using $\max(0, A-B) = A - \min(A,B)$, we have $\Delta(i \mid \{h\}) = s_i - \Omega(i,h) = (1-\rho(i,h))s_i$. The bounds follow directly from the assumption on $\rho(i,h)$. ■

Interpretation. This result indicates that when the local kernel overlap ρ is approximately $1-\gamma$, SMD's discounted score acts as a first-order proxy for the true marginal utility. By the submodularity of F_{sub}, we further have $\Delta(i \mid S) \leq \Delta(i \mid \{h\}) \leq (\gamma+\eta)s_i$, implying that SMD serves as an approximate upper proxy for the marginal gain in highly redundant clusters.

J.3 Bounded Effect of Ratio-Gated Head Calibration

The deployed HubKV variant mixes the raw base score with the SMD-corrected score. For a non-protected content token, write the SMD head-calibrated score as $u_i = \beta_i q_i s_i$, where $q_i \in \{\gamma, 1\}$ indicates whether token i is a non-hub or a hub, and $\beta_i \in [\beta_{\min}, \beta_{\max}]$ is the clipped head-wise selectivity weight. Given compression ratio r and gate exponent p, the final score is

$$z_i = (1-\lambda)s_i + \lambda u_i, \qquad \lambda = r^p. \qquad (15)$$

Proposition 3 (Bounded Ratio-Gated Perturbation)**.** *For every non-protected token with $s_i \geq 0$, the final HubKV score satisfies*

$$(1-\lambda+\lambda\gamma\beta_{\min})\, s_i \leq z_i \leq (1-\lambda+\lambda\beta_{\max})\, s_i. \qquad (16)$$

Consequently, HubKV is a bounded multiplicative perturbation of the base scorer. Moreover, for any two non-protected tokens a and b, if

$$\frac{s_a}{s_b} > \frac{1-\lambda+\lambda\beta_{\max}}{1-\lambda+\lambda\gamma\beta_{\min}}, \qquad (17)$$

then $z_a > z_b$ regardless of their local hub status or clipped head weights.

Proof. Since $q_i \in \{\gamma, 1\}$ and $\beta_i \in [\beta_{\min}, \beta_{\max}]$, we have $\gamma\beta_{\min}s_i \leq u_i \leq \beta_{\max}s_i$ for all non-protected tokens. Substituting this interval into $z_i = (1-\lambda)s_i + \lambda u_i$ gives the first inequality. The ranking-stability condition follows by lower-bounding z_a and upper-bounding z_b. ■

Interpretation. The ratio gate prevents HubKV from arbitrarily rewriting the base importance scores. When compression is mild, $\lambda = r^p$ is small and the method remains close to the base scorer.

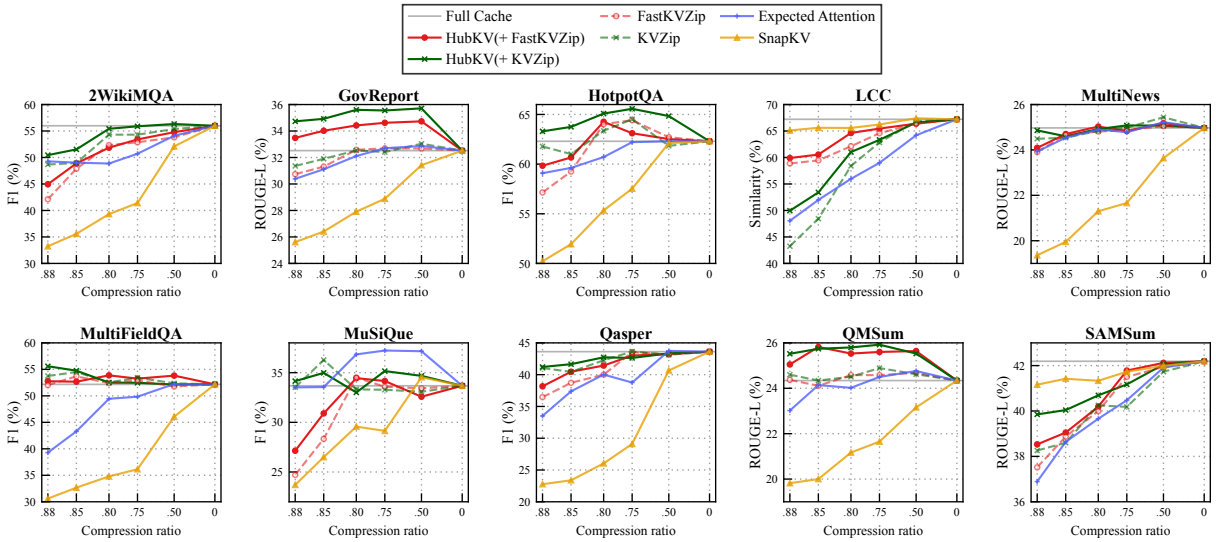


Figure 8: **Additional LongBench results on Qwen3-14B.** We compare LongBench average scores from HubKV-refined KVZip/FastKVZip against the corresponding base scorers, SnapKV, Expected Attention, and the full-cache reference. The x-axis reports the true KV compression ratio, ordered from aggressive compression on the left to full cache on the right, covering compressed ratios $r \in \{0.50, 0.75, 0.80, 0.85, 0.88\}$ plus the full-cache point $r = 0.00$.

When compression is aggressive, the local redundancy correction becomes stronger, but its effect remains bounded by the clipped head weights and the SMD discount factor. This is the theoretical guarantee attached to the deployed parallel proxy; it is distinct from the global approximation guarantee of sequential submodular greedy.

J.4 Global Robustness and Limitations

SMD does not inherit the $(1 - 1/e)$ approximation ratio of sequential greedy algorithms. The following baseline robustness bound applies before head-wise calibration to the unweighted SMD core.

Proposition 4 (Global Lower Bound)**.** *Let S_{SMD} be the set selected by SMD with budget B, and S^* be the optimal set. Then $F(S_{SMD})/F(S^*) \geq 1/B$.*

Proof. SMD always selects the global maximum s_{max} as it is a local hub. Thus $F(S_{SMD}) \geq s_{max}$. Conversely, $F(S^*) \leq \sum_{j \in S^*} s_j \leq B s_{max}$. The ratio follows. ■

Remark (Lack of Constant Approximation Guarantee)**.** *In the absence of additional assumptions on the score distribution, SMD can still fall into local redundancy traps if the discount factor γ is not small enough. Consider a cluster of B highly redundant tokens with scores ≈ 1 and $B - 1$ distant isolated hubs with scores $\gamma(1 - 2\epsilon)$. SMD may prefer the B redundant tokens over the distant hubs, leading to a performance ratio that decays as $\mathcal{O}(1/B)$. This underscores that SMD is a **hardware-parallel proxy** intended for empirical efficiency in LLM inference, rather than a guaranteed submodular optimizer.*

K Empirical Validation via Raw Attention Analysis

To further test whether the observed local redundancy also appears in model-native attention, we conducted a supplemental experiment using raw attention weights from Qwen3-8B on the LongBench 2WikiMQA task (Bai et al., 2024).

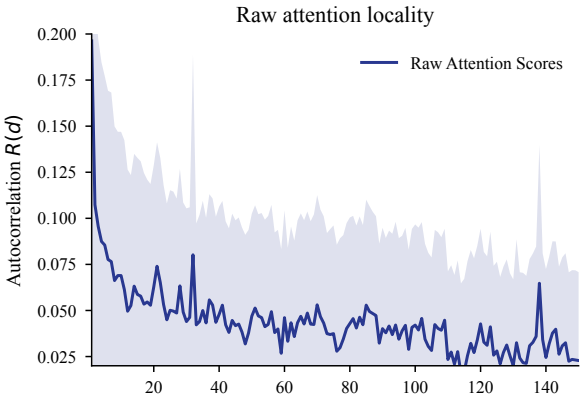


Figure 9: **Spatial correlation of raw attention weights.** The key-side attention density exhibits significant local autocorrelation at short token distances, providing additional evidence that model-native attention mass is spatially clustered.

Findings. As illustrated in Figure 9, the key-side attention density exhibits significant exponential

decay in autocorrelation $R(d)$. This structural redundancy in model-native attention mass provides additional empirical evidence for the local repulsion mechanism employed by SMD. By suppressing local non-maxima, SMD encourages the selection of diverse representatives, aligning the one-shot ranking with the observed locality of LLM attention distributions.

L Additional Related Work and Baseline Scope

Token eviction and compression baselines. The main paper focuses on token-retention methods that expose a Top-K-style pruning interface. Decoding-time eviction methods such as H2O, Scissorhands, and FastGen use past attention or learned discard signals to remove KV entries during generation (Zhang et al., 2023; Liu et al., 2023; Ge et al., 2024). Prefill-time methods such as SnapKV, PyramidKV, Expected Attention, KVZip, and FastKVZip estimate token utility before downstream generation (Li et al., 2024; Cai et al., 2024; Devoto et al., 2025; Kim et al., 2026b,a). HubKV is designed for this second interface: it does not replace the base scorer, but modifies the final ranking so that locally redundant high-score neighbors are softly discounted before the same budgeted pruning step is applied.

Semantic and chunk-level baselines. Semantic-chunk methods address a related but different unit-of-selection question. ChunkKV, for example, retains semantically coherent chunks rather than treating every token as an independent item (Liu et al., 2026). HubKV keeps the token-level cache layout of existing score-based systems, but its local discounting plays a complementary role: it discourages spending a tight cache budget on multiple neighboring tokens when a local representative already covers the region. Combining chunk-level grouping with HubKV-style local marginal discounting is a natural follow-up direction.

KV-cache quantization. KV-cache quantization reduces the bytes used by retained keys and values rather than changing which positions are retained. KIVI studies asymmetric low-bit KV-cache quantization and KVQuant develops low-precision KV-cache quantization for very long contexts (Liu et al., 2024; Hooper et al., 2024). These methods are therefore orthogonal to HubKV's token-selection objective. A quantized cache can still require an eviction policy under extreme context lengths or tight memory budgets, while HubKV can in principle be applied before or together with quantization to choose retained entries.

M Additional Discussion

HubKV is designed as a score refinement layer for cache eviction methods that already expose a Top-K-style pruning interface. If the underlying scorer assigns low importance to genuinely useful tokens, the redundancy correction cannot recover those tokens after ranking. Conversely, if a local region contains several adjacent tokens that are jointly necessary, the discounting step may move some useful non-hub tokens below the retention threshold. This behavior is most likely when the remaining budget is already large enough to preserve local evidence chains, which is consistent with the smaller or slightly negative gains observed under moderate compression.

The submodular formulation should also be interpreted as a modeling guide rather than as the deployed optimization algorithm. Sequential greedy maximization of the ideal coverage objective would introduce a dependency chain that is undesirable for LLM inference. HubKV instead uses a one-pass local proxy that preserves the parallel score-ranking interface of existing systems. The theoretical analysis therefore characterizes local marginal behavior and bounded score perturbation, but it does not imply the global approximation guarantee of greedy submodular maximization.

N Ethical Considerations

HubKV is an infrastructure method for reducing the memory cost of long-context LLM inference, rather than an application-facing system. More efficient KV cache compression can make long-context inference less expensive and more accessible, but it can also lower the cost of large-scale document analysis, surveillance, or other privacy-sensitive applications. HubKV does not detect, filter, anonymize, or protect sensitive information in the input context. If a prompt or document contains personal, confidential, copyrighted, or otherwise sensitive content, the compressed KV cache should still be treated as derived model state from that content rather than as an anonymized representation. Because HubKV changes which historical tokens remain represented in the cache, practitioners should validate compressed systems on the in-

tended data distribution before using them in high-stakes settings.

O Reproducibility Statement

The experimental sections report the backbone models, task families, compression ratios, protected-token handling, and HubKV hyperparameters used in the current evaluation. Appendix A provides additional implementation details for prefill and decoding-stage experiments. We will release code, plotting scripts, and result snapshots with the camera-ready version, subject to model and benchmark licenses and release policies.