Marlin: I/O-Efficient Prefix KV Cache Retrieval for Long-Prefix LLM Serving

Guifeng Wang[1], Shengan Zheng[2*], Ji Fang[1], Yucheng Li[1], Shi Shu[1], Weihan Kong[1], Cong Zhou[1], Kaijiang Deng[1] and Linpeng Huang[1*]
[1]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University
[2]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
{wangguifeng,shengan,foggy_cat,golden_wind,shu-shi,weihankong,cong258258,2020-sjtu-dkj,lphuang}@sjtu.edu.cn

Abstract

As large language models (LLMs) are often deployed with long, context-rich prefixes, the prefix KV cache frequently exceeds GPU memory capacity. Although offloading the prefix KV cache to host memory or storage alleviates capacity pressure, it introduces severe I/O stalls that increase Time-to-First-Token (TTFT) latency. To address this challenge, we present Marlin, an I/O-efficient prefix KV cache retrieval system for long-prefix LLM inference. Marlin employs a dispersion-based token selector to precompute a compact, query-agnostic subset of important prefix tokens, and a sensitivity-guided head classifier that assigns different KV retrieval policies to classify prefix-sensitive and query-sensitive heads. An overlap-optimized attention pipeline further hides offload latency by overlapping head-specific KV transfers with attention computation. Experimental results demonstrate that Marlin significantly reduces TTFT compared to state-of-the-art methods while maintaining comparable model accuracy.

Keywords

LLM, Prefix KV cache, Sparse attention.

ACM Reference Format:
Guifeng Wang[1], Shengan Zheng[2*], Ji Fang[1], Yucheng Li[1], Shi Shu[1], Weihan Kong[1], Cong Zhou[1], Kaijiang Deng[1] and Linpeng Huang[1*]. 2026. Marlin: I/O-Efficient Prefix KV Cache Retrieval for Long-Prefix LLM Serving. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804140

1 Introduction

Large language models (LLMs) exhibit remarkable capabilities across various real-world applications, including text generation [2, 28], language translation [13, 20], and document summarization [10, 12]. To steer model behavior and improve response quality, production systems typically prepend the user prompt with context-rich prefixes. Typical prefixes include system prompts [11] that specify roles and rules, dialogue history [14] that preserves conversational coherence, few-shot question-answer pairs [18] that specify structure and style, and external documents [25] that ground outputs in verifiable facts.

While prefixes provide essential context that improves response accuracy and coherence, they also lengthen the input sequence and inflate prefill latency (the time to first token, TTFT). The dominant culprit is self-attention, whose computation scales quadratically with sequence length [23]. As prompts grow, attention forms larger score matrices and processes more Key-Value projections, consuming additional compute and memory bandwidth. These overheads accumulate into noticeable delays, undermining responsiveness for real-time interactive applications.

Despite higher prefill cost, a silver lining lies in the widespread reuse of prefixes across sessions [15, 25, 29]. Caching and reusing the Keys (K) and Values (V) of these prefix tokens avoids recomputation and accelerates inference. However, the prefix KV cache often outgrows GPU memory in practice, forcing offload to host DRAM or SSD [4, 6, 21]. The resulting I/O stalls and limited throughput can offset, or even outweigh, the benefits of caching, yielding even higher end-to-end latency.

Sparse attention [4, 19, 24, 27] offers a promising way to prune the KV working set by exploiting the skewness of token importance. By confining computation to the most informative prefix tokens, sparse attention significantly reduces the number of prefix KVs to compute and transfer, thus alleviating I/O stalls. Existing approaches typically partition attention heads into groups, and then identify important tokens within each group. With prefix KV cache offloading, however, both token selection and head partition face significant challenges.

First, attention-driven token selection precludes prefix KV cache prefetching during prefill. Existing token-selection methods derive token importance from attention weights computed during inference, which forces KV selection and loading to wait until attention completes. Prefilling is therefore unable to overlap I/O with computation, exposing SSD and DRAM latency on the critical path and expanding GPU pipeline bubbles. Moreover, attention-based selection is inherently query-dependent, making it unsuitable for offline token identification before runtime.

Second, static head partitioning misaligns with the variability of request semantics. Existing head partitioning schemes fix head groups using global, static classification derived from offline profiling [9] or dataset-specific pretraining [24]. This static view presumes stable head behavior and, in turn, freezes token selection across sessions. In practice, requests that share the same prefix exhibit substantial semantic variance. Consequently, static partitioning fails to reflect query-dependent attention and thus degrades.

In this paper, we present Marlin, a KV cache retrieval system that optimizes prefix KV cache offloading through dispersion-based token selection and sensitivity-guided head classification. To enable

*Shengan Zheng and Linpeng Huang are corresponding authors.



This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804140

I/O-efficient prefix KV retrieval, Marlin employs a dispersion-based token selector that precomputes a compact, query-agnostic subset of important prefix tokens during prefix KV cache construction. This selector leverages the insight that tokens whose Key vectors exhibit lower dimensional dispersion across attention heads are more likely to receive high attention. By estimating token importance before attention-weight computation, Marlin decouples token selection from the latency-critical inference path and incurs negligible runtime overhead. The resulting sparse prefix KV cache remains stable across queries and faithfully preserves the essential semantics of the original prefix.

Leveraging the inherent heterogeneity of attention heads, Marlin employs a sensitivity-guided head classifier that assigns distinct KV retrieval policies to different heads. Rather than forcing all heads to operate on the same token set, the classifier compares each head's attention pattern against the static important-token set to estimate its reliance on the prefix. Heads whose attention consistently overlaps the static set are treated as prefix-sensitive and are served solely by the sparse prefix KV cache. Heads that focus primarily on query-specific tokens are classified as query-sensitive and retain the full KV cache. By decoupling KV retrieval policy across heads, Marlin preserves accuracy for query-sensitive heads while substantially reducing I/O for prefix-sensitive computation.

To further mitigate the I/O bottlenecks introduced by prefix KV offloading, Marlin organizes KV retrieval and attention computation into an overlap-optimized attention pipeline. Leveraging head partitioning, the pipeline schedules head-specific KV loads and attention kernels so that Key and Value transfers proceed asynchronously with ongoing computation within each layer. In parallel, it prefetches KVs for subsequent layers while the current layer performs attention and feed-forward operations. The carefully-orchestrated pipeline staging shortens effective KV access latency, improves GPU utilization, and prevents offloaded KV retrieval from dominating inference time.

Our main contributions are summarized as follows:

- We present a dispersion-based token selector that leverages the dispersion of Key vectors to identify important prefix tokens, minimizing prefix KV transfer during inference. To the best of our knowledge, it is the first Key-dispersion-driven important token identification approach.
- We propose a sensitivity-guided head classifier that adaptively partitions attention heads according to their query sensitivity, enabling head-specific token selection policies that balance model accuracy and I/O overhead.
- We introduce an overlap-optimized attention pipeline that tightly interleaves KV cache loading with attention computation, thereby concealing retrieval latency and improving GPU utilization.
- We implement Marlin based on FlexGen [21] and conduct extensive experiments on various models and datasets. The results demonstrate that Marlin significantly reduces the Time-to-First-Token (TTFT) by up to 2.5× compared to state-of-the-art methods while maintaining comparable accuracy.

2 Background and Related Work

2.1 Large Language Models Basics

Procedure of LLM Inference. Large language models (LLMs) generate text autoregressively, a process divided into the prefill phase, which computes the state for the input prompt, and the decode phase, which iteratively generates subsequent tokens [1, 23]. This computation is performed by a stack of transformer blocks, each comprising a self-attention layer and a feedforward network (FFN). The self-attention layer first computes interactions between tokens by projecting the input embeddings into Query (Q), Key (K), and Value (V) tensors, which are partitioned into multiple attention heads. These heads compute scaled dot-product attention in parallel, with each head independently operating ($\frac{QK^T}{\sqrt{d_k}}$) on its respective Q, K, and V slices. The outputs from all heads are then concatenated and passed through the FFN to produce the final output of the transformer block.

KV Cache. To optimize the autoregressive decode phase, inference systems employ a KV cache to store the Key and Value tensors for all tokens in the sequence [15, 23]. This caching mechanism is critical for performance, as it eliminates the need to recompute K and V tensors for previous tokens during each decode step. When generating a new token, the model only needs to compute its own QKV vectors and perform attention operations against the K and V tensors already stored in the cache, significantly reducing redundant computation.

Shared Prefixes. Modern LLM services frequently prepend user prompts with context-rich prefixes, such as system instructions, to enhance response quality [25]. Leveraging the observation that these prefixes are often shared across requests [4, 6], systems like vLLM [15] and SGLang [29] amortize the high cost of prefill by caching and reusing the KV state of these common prefixes across multiple requests. This prefix KV reuse strategy, however, introduces a severe memory bottleneck. Since the KV cache size scales linearly with both the prefix length and model dimensions, the aggregate storage required for these prefix KVs quickly exceeds the limited and expensive GPU VRAM as model scale and input sequence lengths grow.

2.2 KV Cache Offloading

To accommodate KV cache that overflows GPU VRAM [19, 21], a common technique is to offload the cache to a larger, slower memory tier like CPU DRAM or SSD [4, 6, 8, 14, 17, 19, 21]. FlexGen [21] is a pioneering system that enables offloading the KV cache to CPU memory and SSD, allowing LLMs to handle longer input sequences with limited GPU memory. AttentionStore [6], HCache [8] offload the prefix KV cache to memory and SSD, using optimizations such as prefetching or storing hidden states as a compact proxy for the full KV cache. However, these systems treat all prefix tokens uniformly and require retrieving the *entire* prefix state. Consequently, they fail to fundamentally reduce the I/O volume transferred, as the complete set of prefix data must be loaded back to the GPU.

2.3 Sparse Attention

To reduce the KV cache size and speed up attention computation, sparse attention mechanisms [3, 16] have been proposed to focus on a subset of important tokens rather than the full context. This approach can reduce the volume of prefix KV data that must be transferred from SSD and DRAM. Prior works [4, 9, 24, 27] have proposed various sparse attention algorithms. H2O [27] and IMPRESS [4] dynamically select important tokens based on the attention weight computed during inference. IMPRESS further optimizes

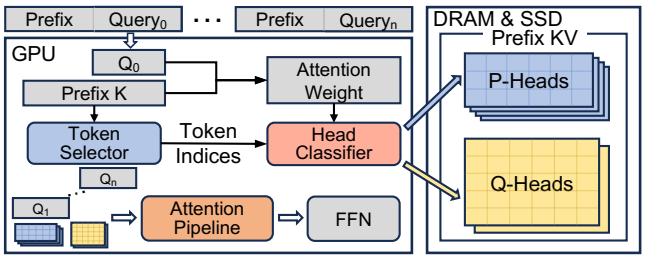


Figure 1: The overall architecture of Marlin.

data transfer by exploiting similarity between heads. However, these methods can only begin loading the required KVs after the attention weight are computed, preventing any overlap between I/O and selection.

FastGen [9] and DuoAttention [24] classify attention heads into static groups based on pre-determined attention patterns. A group of heads access the full KV cache, while the rest focus on specific tokens and only retrieve KVs for those tokens. The classification allows the system to preload KVs for the designated sparse heads, enabling I/O-compute overlap. Nevertheless, rigid classification is not adaptive and may not generalize across different prefixes or tasks, potentially leading to model accuracy degradation.

3 Design

This section presents the design of Marlin, an I/O-efficient prefix KV cache retrieval system without sacrificing model accuracy. As depicted in Figure 1, Marlin offloads the prefix KV cache from capacity-constrained GPU memory to a hierarchical storage pool spanning DRAM and SSD. To mitigate the high I/O cost of cache transfers, Marlin integrates a dispersion-based token selector to identify a compact set of important prefix tokens. Building on this selection, a dynamic head classifier partitions attention heads into groups with distinct KV retrieval policies, balancing accuracy against I/O savings. Finally, Marlin introduces an attention pipeline that tightly couples prefix KV loading with attention computation, effectively hiding I/O latency. Marlin is guided by the following design principles:

Preselect important tokens efficiently. Marlin's dispersion-based token selector (Section 3.1) performs a one-time, offline analysis of prefix Key vectors during KV cache construction. It ranks tokens by their dimensional dispersion and retains only those that exhibit broad importance across heads. This static selection amortizes computation, yields a compact prefix KV subset that preserves query-agnostic semantics, and removes token-selection overhead from the latency-critical inference path.

Classify heads by query sensitivity. The sensitivity-guided head classifier (Section 3.2) compares the static important-token set with a query-specific ground-truth set for each head, using their overlap to distinguish prefix-sensitive P-Heads from query-sensitive Q-Heads. P-Heads are served by the sparse, preselected KV cache, while Q-Heads retain their full KV cache and rely on runtime attention for token selection. This training-free hybrid policy tailors I/O reduction to head behavior and remains adaptable to various queries.

Maximize computation-I/O overlap. The overlap-optimized attention pipeline (Section 3.3) exploits the head partitioning to split attention into parallel P-Head and Q-Head streams whose

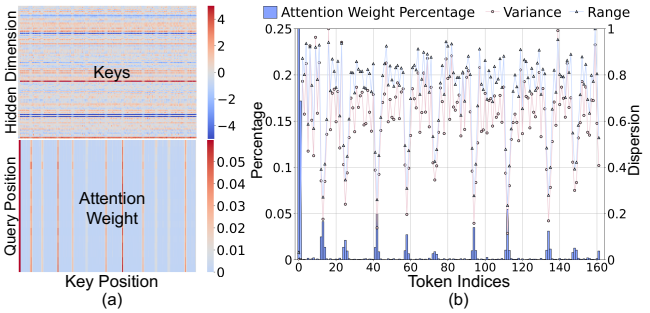


Figure 2: (a) Key and attention heatmaps extracted from OPT-13B model (Layer 14, Head 0). (b) Distribution of per-token attention weight share and Key vector variance.

KV transfers and computations are carefully staged across time and layers. By overlapping prefetching of sparse P-Head KVs, full Q-Head KVs, and per-head attention, Marlin hides offload latency and keeps GPU and storage resources well utilized.

3.1 Dispersion-based Token Selector

Marlin mitigates the I/O overhead of the prefix KV cache retrieval from SSD by introducing a query-agnostic dispersion-based token selector. Our key insight is that *low-dispersion Key vectors tend to yield higher attention weight*. Leveraging this correlation, Marlin identifies important tokens during prefix KV generation with negligible additional overhead. The selected tokens capture the prefix's essential semantics and remain stable across diverse queries.

The insight arises from the geometry of attention mechanisms in transformer models between Query (Q) and Key (K) vectors. The expected energy a Key k_t contributes to the attention score QK^T is proportional to $k_t^T \Sigma_Q k_t$, where Σ_Q is the Query subspace covariance matrix ($\Sigma_Q = \frac{1}{n} Q^T Q$). This quadratic form reveals that amplification is governed by k_t's alignment with the principal directions of Q (eigenvectors of Σ_Q associated with large eigenvalues). Low-variance keys inherently concentrate energy in common, low-frequency directions that overlap these principal components, producing larger dot products and, consequently, higher attention scores. In contrast, high-variance keys contain more oscillatory components that are largely orthogonal to the principal query subspace, and thus contribute less to the overall attention.

Figure 2 substantiates this insight using data extracted from the OPT-13B model. We present heatmaps of the attention weight and Key matrix in Figure 2(a). The attention heatmap highlights a small subset of tokens that consistently receive high attention across queries, indicating their general importance. Meanwhile, the Key matrix heatmap shows that these same tokens exhibit stable, low-dispersion Key patterns across hidden dimensions (appear in lighter, more uniform colors). Figure 2(b) further validates this pattern by plotting each token's attention weight distribution against its Key vector variance. The plot demonstrates a strong inverse correlation: tokens with dominant attention shares consistently exhibit the lowest Key vector variance. These observations validate that dimensional dispersion is an accurate and robust metric for token importance.

Guided by this insight, Marlin employs a dispersion-based token selector that identifies important prefix tokens through per-head filtering. For each attention head, Marlin ranks prefix tokens by

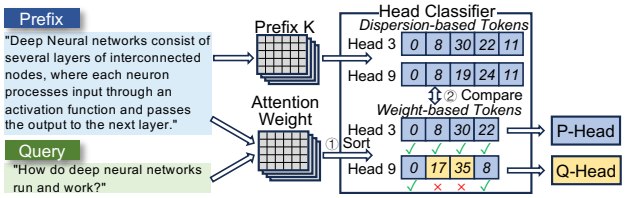


Figure 3: Procedure of the sensitivity-guided head classifier.

the range of their Key vectors (the difference between their maximum and minimum dimensions), and selects the top-K tokens with the smallest range. The range metric serves as a computationally efficient yet informative approximation for variance to capture the dispersion of Key vectors. Since layer normalization enforces near-zero means and constrains Key dimensions to a sub-Gaussian interval, the range grows essentially monotonically with the underlying variance, enabling accurate dispersion estimation at negligible cost. The per-head filtering yields a compact KV cache set composed of tokens based on each head's feature, thereby maintaining accuracy while substantially reducing the KV cache size.

Our dispersion-based token selector is designed to deliver both high efficiency and robust accuracy. Efficiency is achieved by performing selection only once, as an offline procedure during the initial prefix cache generation. Its negligible computational overhead is fully amortized and incurs no additional latency for online queries. At the same time, the selector attains query-agnostic accuracy by exploiting the intrinsic dimensional properties of the Key vectors. Instead of resorting to costly, per-query profiling, the token selector leverages dimensional dispersion to select tokens that are inherently aligned with the dominant directions of the attention mechanism. Consequently, it yields a compact yet faithful prefix KV cache that maintains model fidelity over a wide range of queries.

3.2 Sensitivity-guided Head Classifier

Marlin dynamically classifies attention heads into prefix-sensitive or query-sensitive based on their query dependency, coupling each with a tailored KV-cache retrieval method. Prior approaches treat all heads uniformly, forcing a trade-off between I/O efficiency and model accuracy. They either statically select a fixed prefix-based important-token subset for each head or dynamically select tokens per query for all heads. The former reduces data transfers but degrades accuracy on query-sensitive heads, while the latter preserves accuracy but incurs prohibitive I/O overhead. Marlin resolves this by exploiting the inherent heterogeneity in attention head behavior that *different heads attend to different aspects of the input*. The dynamic head classifier partitions the attention heads into prefix-sensitive (P-Heads) and query-sensitive (Q-Heads), and applies a hybrid KV cache retrieval strategy that achieves high efficiency without compromising accuracy.

Marlin classifies each attention head by measuring the divergence between a query-agnostic static token set and a query-specific ground-truth token set. Marlin first derives the static important-token sets from the prefix via the dispersion-based token selector (Section 3.1). Then, Marlin performs a one-time calibration run that appends the first query to the prefix, and ranks tokens by attention weight to form a ground truth set. As illustrated in Figure 3, the classifier aggregates and sorts the prefix-token attention weight of each head. It then computes the cumulative weight distribution to

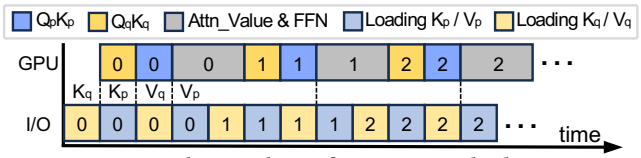


Figure 4: The pipeline of attention calculation.

select the minimal set of tokens whose combined attention share exceeds a predefined threshold (e.g., 80%). These tokens constitute the query-specific ground-truth set of the attention head.

Once the static and ground-truth sets are established, the classifier quantifies their divergence by computing their overlap ratio. Large overlap between the two sets implies that the head behaves predominantly on the prefix. The head is classified as prefix-sensitive, allowing Marlin to prune unselected KVs without harming accuracy. Conversely, small overlap indicates that the head is strongly sensitive to query semantics. The head is classified as query-sensitive, and its full KV cache is preserved to maintain accuracy. Therefore, Marlin ranks heads by overlap ratio, classifies the lowest r% as Q-Heads and the remainder as P-Heads, where r matches the KV retention ratio in P-Heads. This coordinated partitioning pairs a small set of Q-Heads with full caches and a large set of P-Heads with sparse caches, equalizing I/O load across streams.

After classification, Marlin serves the prefix-sensitive P-Heads with a sparse, pre-selected KV subset chosen by the dispersion-based token selector during prefix KV construction to reduce data transfer cost. Fewer query-sensitive Q-Heads retain their full KV cache to preserve model accuracy. To further guard against accuracy loss, Marlin exempts the first layer from pruning, keeping the full KV cache for all heads to preserve semantics in raw input embeddings that are critical for downstream reasoning. In deeper layers where features are more structured, the hybrid policy prunes P-Heads safely while leaving Q-Heads intact, yielding an adaptive KV retrieval scheme that sustains accuracy while reducing I/O.

To adapt to evolving conversational contexts, Marlin periodically re-evaluates the head partitions. This re-classification incurs negligible overhead, as the lightweight process executes asynchronously with respect to the main inference path. The primary benefit of this dynamic approach is its enhanced adaptivity. Unlike static, one-time classification methods that only optimize for the initial prefix, Marlin continually adjusts to semantic shifts that emerge during a long-running session. This ensures that the I/O-efficient cache management strategy remains effective, preserving high performance and model accuracy over long multi-turn sessions.

3.3 Overlap-Optimized Attention Pipeline

Marlin employs an attention pipeline that orchestrates attention computation and KV retrieval, leveraging the head classification to split attention processing into two parallel streams. In the P-Head stream, prefix-sensitive heads consume a sparse KV subset chosen offline by the dispersion-based token selector, and Marlin prefetches these KVs directly from storage. In the Q-Head stream, Marlin first loads the full Key cache, computes attention weight to dynamically identify the most salient tokens, and then fetches only the corresponding Value vectors. The outputs from the P-Head and Q-Head streams are finally concatenated to produce the final attention output for the layer.

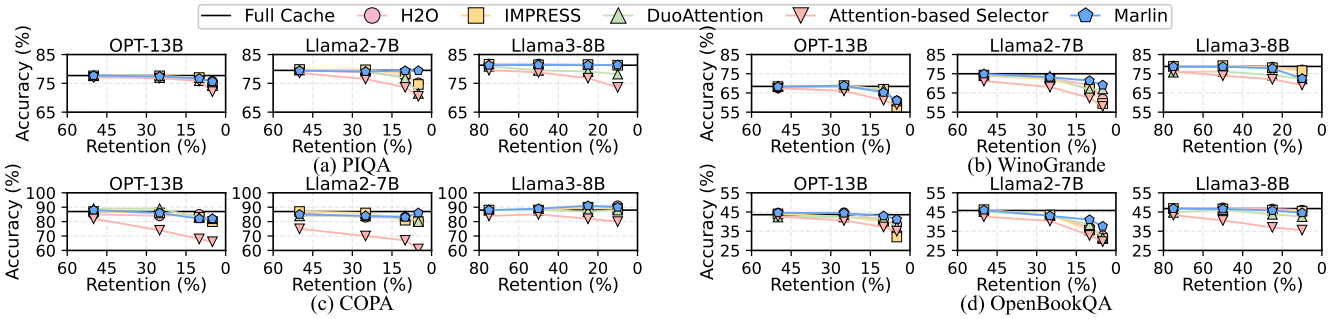


Figure 5: Accuracy of Marlin and baselines across various datasets and models with different retention ratios.

Marlin hides the latency of offloaded KV retrieval by tightly interleaving KV cache loading with attention computation in the attention pipeline illustrated in Figure 4. Marlin first loads Q-Head Keys and triggers GPU computation of Q-Head attention weight. During attention computation, Marlin asynchronously prefetches the sparse, preselected P-Head Keys so that they are present in VRAM when the Q-Head computation finishes. The pipeline then retrieves the corresponding Q-Head and P-Head Values, while the GPU concurrently performs P-Head attention using the prefetched Keys. Once all Values are available, Marlin fuses the attention outputs from P-Heads and Q-Heads and forwards the result to the remaining transformer computations. At the same time, Marlin prefetches Q-Head Keys and P-Head KVs for the next layer, ensuring continuous overlap between KV transfers and GPU computation.

The overlap-optimized attention pipeline leverages the complementary KV patterns of P-Heads and Q-Heads to form well-balanced pipeline stages. The larger set of P-Heads retrieves only a sparse KV subset, while a smaller set of Q-Heads loads the complete Key cache. This division equalizes the effective I/O load across streams and enables steady, high utilization of the pipeline. Concurrently, cross-layer KV prefetching overlaps future data transfers with current computation, improving GPU and memory-bandwidth utilization and preventing KV cache retrieval from becoming the dominant bottleneck.

4 Evaluation

4.1 Experimental Setup

Models and System Configurations. We evaluate Marlin using three representative language models: OPT-13B [26], Llama2-7B [22], and Llama3-8B [5]. All experiments run on a server equipped with an NVIDIA A6000 GPU (48GB VRAM), two 2GHz Intel Xeon Gold 5416S CPUs, 128GB DDR5 DRAM, and a 4TB Samsung 9A3 SSD. The GPU is connected to the host via PCIe 4.0 ×16.

Datasets and Methodology. We use four datasets from the LM-Evaluation-Harness [7] (PIQA, WinoGrande, COPA, and OpenBookQA) to benchmark diverse tasks, including commonsense reasoning and question answering. Following previous works [4, 17], we prepend the inputs with few-shot examples as shared system prefixes. To evaluate performance with long prefixes, we measure the time-to-first-token (TTFT) after extending these prefixes to each model's maximum context length (2048 for OPT-13B; 4096 for Llama2-7B and 8192 for Llama3-8B).

Baselines. We evaluate Marlin against five representative baselines to characterize its performance and isolation efficacy. Full cache retains the entire prefix KV cache without token selection. H2O [27] selects tokens based on accumulated attention weight during inference. IMPRESS [4] further optimizes H2O by exploiting inter-head similarity to reduce data transfer during dynamic token selection. DuoAttention [24] statically classifies heads into two groups via pretraining based on their attention patterns. As DuoAttention does not support OPT models, we use Marlin's head classification (on RTE, a dataset which is also from [7]) as a proxy for static classification for OPT-13B model. An Attention-based Selector ablation baseline replaces Marlin's dispersion-based token selector with a traditional attention weight based selector (akin to H2O), while all other components of Marlin remain unchanged.

4.2 Overall Performance

Accuracy. Figure 5 illustrates the accuracy of Marlin and baseline methods across all datasets and models as a function of the prefix KV cache retention ratio. We vary this ratio from 5% to 50% for OPT-13B and Llama2-7B, and from 10% to 75% for Llama3-8B to account for its smaller KV cache size resulting from Grouped Query Attention (GQA). The results demonstrate that Marlin incurs negligible accuracy loss compared to dynamic runtime selection methods (H2O, IMPRESS). Notably, Marlin still maintains high accuracy and even exceeds H2O and IMPRESS at lower retention ratios, confirming that Marlin effectively captures essential prefix semantics through the dispersion-based token selection. The accuracy of IMPRESS declines more steeply at lower retention ratios, as its inter-head similarity optimization can inadvertently omit important tokens when the selection budget is tight.

Marlin also achieves higher accuracy than DuoAttention and the Attention-based Selector, highlighting the benefits of sensitivity-guided head classification strategy. The accuracy degradation observed in DuoAttention arises from its rigid, static partitioning, which fails to accommodate heads that exhibit query-dependent attention variability. In the case of the Attention-based Selector, the accuracy drop is even more pronounced despite utilizing the identical sensitivity-guided head classifier and Q-Head budget as Marlin. This performance gap isolates and validates the superiority of our dispersion-based token selection method over traditional attention-based importance metrics.

Average TTFT. We retain 25% of the prefix KV cache for OPT-13B and Llama2-7B, and 50% for Llama3-8B and measure the average TTFT excluding the prefix token processing time. Figure 6 depicts the results, revealing that Marlin consistently outperforms all the baselines. On average, Marlin reduces TTFT by 82.7%, 75.7%, and 52.2% relative to full cache, H2O and IMPRESS, respectively. These

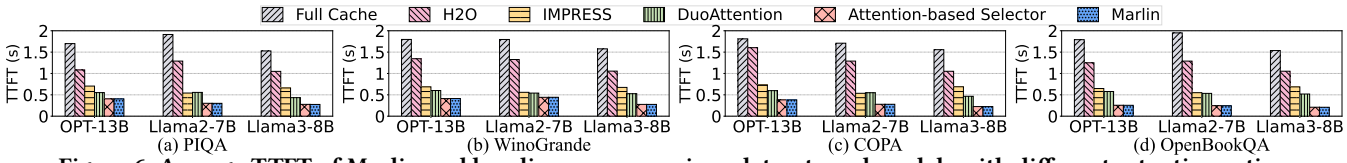


Figure 6: Average TTFT of Marlin and baselines across various datasets and models with different retention ratios.

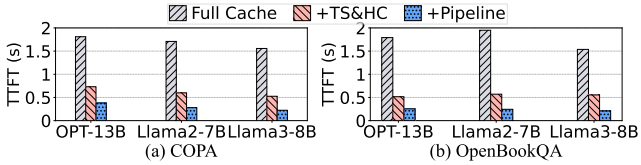


Figure 7: The impact of individual techniques on TTFT.

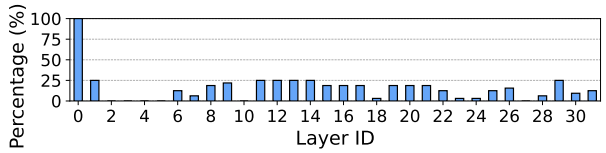


Figure 8: Percentage of Q-Heads in Marlin on Llama2-7B.

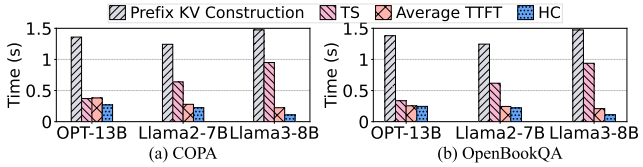


Figure 9: Overhead of token selection and head classification.

gains stem from the ability of Marlin to preload a compact token subset for P-Heads, thereby alleviating I/O bottleneck. IMPRESS improves upon H2O by loading probe heads to guide token selection, but it must complete token selection before loading the rest of the KVs, which severely limits I/O-compute overlap. Furthermore, its performance is limited for GQA-based models like Llama3-8B, where the reduced number of KV heads diminishes the benefits of probe-based token selection.

We then evaluate Marlin against established head classification baselines, demonstrating a 41.5% average reduction in TTFT relative to DuoAttention. The performance advantage stems from limitations of DuoAttention's static classification strategy, which frequently diverges from dynamic query semantics. Consequently, DuoAttention conservatively retains the full KV cache for a surplus of attention heads, leading to suboptimal head classification and exacerbated I/O overhead. In contrast, Marlin employs an adaptive mechanism to dynamically calibrate the number of Q-Heads, effectively reducing the retrieval of non-essential data. Although the Attention-based Selector offers latency reductions comparable to Marlin, it severely compromises output quality, incurring significant accuracy degradation.

4.3 Impact of Individual Techniques

Ablation Study. To quantify the contributions of Marlin's key components, we conduct an ablation study decoupling the contributions of the sparse attention mechanism from the overlap-optimized pipeline. As detailed in Figure 7, the evaluation proceeds from the full cache baseline, sequentially integrating dispersion-based token selection and sensitivity-guided head classification (*+TS&HC*), and then *+Pipeline*, which incorporates the overlap-optimized attention pipeline. Each component progressively reduces TTFT across all models and datasets. On average, *TS&HC* reduces TTFT by 60.6% compared to the full cache baseline, while the overlap-optimized pipeline contributes an additional 22.3% reduction.

Percentage of Q-Heads. Figure 8 details the layer-wise distribution of query-sensitive heads (Q-Heads) within Llama2-7B, subject to a 25% upper bound aligned with our KV retention policy. We observe that the proportion of Q-Heads falls below this threshold across the majority of layers. This validates that our dispersion-based token selection effectively captures the essential prefix tokens, allowing most heads to rely on the sparse prefix KV cache. Consequently, Marlin substantially mitigates I/O overhead by preloading the crucial subset for P-Heads, thereby optimizing retrieval latency while maintaining comparable model accuracy.

Overhead Analysis. Figure 9 characterizes the overhead associated with dispersion-based token selection (TS) and sensitivity-guided head classification (HC). Since token selection is performed only once during prefix KV cache construction, we compare its overhead against the time of prefix KV construction. The results indicate that the overhead of token selection can be overlapped with prefix KV construction, while the cost of one-time token selection can be amortized. Specifically, the time of token selection is proportional to the prefix length, increasing for Llama2-7B and Llama3-8B due to their longer prefixes compared to OPT-13B. Regarding HC, its overhead remains lower than the average TTFT of the subsequent inference with prefix KV. As we perform head classification asynchronously, we ensure HC remains off the critical path of inference. Moreover, the HC cost correlates directly with model size, decreasing as the number of attention heads reduces from OPT-13B to Llama3-8B.

5 Conclusion

This paper presents Marlin, an I/O-efficient prefix KV cache retrieval system for long-prefix LLM inference. Marlin decouples token selection from the latency-critical path via a dispersion-based token selector that precomputes a compact, query-agnostic subset of important prefix tokens. It further introduces a sensitivity-guided head classifier, which separates prefix-sensitive and query-sensitive heads and assigns them different KV retrieval policies, and an overlap-optimized attention pipeline that hides storage latency by interleaving KV movement with attention computation. Built on FlexGen, Marlin substantially reduces TTFT while preserving model accuracy across models and prefixes, and requires no model retraining or architectural changes.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and insightful suggestions. This work is supported by National Key Research and Development Program of China (Grant No. 2024YFB4505203), National Natural Science Foundation of China (NSFC) (Grant No. 62227809, 62332012, 62302290), and the Fundamental Research Funds for the Central Universities.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S Yu, and Lichao Sun. 2023. A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt. *arXiv preprint arXiv:2303.04226* (2023).

[3] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. 2021. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems* 34 (2021), 17413–17426.

[4] Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. 2025. {IMPRESS}: An {Importance-Informed}{Multi-Tier} Prefix {KV} Storage System for Large Language Model Inference. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. 187–201.

[5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv e-prints* (2024), arXiv–2407.

[6] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 111–126.

[7] Leo Gao, Jonathan Tow, Stella Biderman, Shawn Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jasmine Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept* 10 (2021), 8–9.

[8] Shiwei Gao, Youmin Chen, and Jiwu Shu. 2025. Fast state restoration in LLM serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems*. 128–143.

[9] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801* (2023).

[10] Alireza Ghadimi and Hamid Beigy. 2022. Hybrid multi-document summarization using pre-trained language models. *Expert Systems with Applications* 192 (2022), 116292.

[11] Louie Giray. 2023. Prompt engineering with ChatGPT: a guide for academic writers. *Annals of biomedical engineering* 51, 12 (2023), 2629–2633.

[12] Bharathi Mohan Gurusamy, Prasanna Kumar, Parathasarathy Srinivasan, Hanish Kb, et al. 2023. Text Summarization for Big Data Analytics: A Comprehensive Review of GPT 2 and BERT Approaches. *Springer (Accepted)* (2023).

[13] Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. 2023. How good are GPT models at machine translation? A comprehensive evaluation. arXiv. *arXiv preprint arXiv:2302.09210* (2023).

[14] Jinwoo Jeong and Jeongseob Ahn. 2025. Accelerating LLM Serving for Multi-turn Dialogues with Efficient Resource Management. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1–15.

[15] 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*. 611–626.

[16] Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. 2025. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766* (2025).

[17] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 155–172.

[18] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems* 36 (2023), 43447–43478.

[19] Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. 2025. InstAttention: In-Storage Attention Offloading for Cost-Effective Long-Context LLM Inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1510–1525.

[20] Vikas Raunak, Amr Sharaf, Yiren Wang, Hany Hassan Awadallah, and Arul Menezes. 2023. Leveraging GPT-4 for automatic translation post-editing. *arXiv preprint arXiv:2305.14878* (2023).

[21] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*. PMLR, 31094–31116.

[22] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[23] 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 (2017).

[24] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819* (2024).

[25] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*. 94–109.

[26] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

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

[28] Zoie Zhao, Sophie Song, Bridget Duah, Jamie Macbeth, Scott Carter, Monica P Van, Nayeli Suseth Bravo, Matthew Klenk, Kate Sick, and Alexandre LS Filipowicz. 2023. More human than human: LLM-generated narratives outperform human-LLM interleaved narratives. In *Proceedings of the 15th Conference on Creativity and Cognition*. 368–370.

[29] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems* 37 (2024), 62557–62583.