MUSTAFAR: Promoting Unstructured Sparsity for KV Cache Pruning in LLM Inference

Donghyeon Joo[1], **Helya Hosseini**[1], **Ramyad Hadidi**[2], **Bahar Asgari**[1]
[1]Department of Computer Science, University of Maryland, [2]d-Matrix
{dhjoo98,helia,bahar}@umd.edu, rhadidi@d-matrix.ai

Abstract

We demonstrate that unstructured sparsity significantly improves KV cache compression for LLMs, enabling sparsity levels up to 70% without compromising accuracy or requiring fine-tuning. We conduct a systematic exploration of pruning strategies and find per-token magnitude-based pruning as highly effective for both Key and Value caches under unstructured sparsity, surpassing prior structured pruning schemes. The Key cache benefits from prominent outlier elements, while the Value cache surprisingly benefits from a simple magnitude-based pruning despite its uniform distribution. KV cache size is the major bottleneck in decode performance due to high memory overhead for large context lengths. To address this, we use a bitmap-based sparse format and a custom attention kernel capable of compressing and directly computing over compressed caches pruned to arbitrary sparsity patterns, significantly accelerating memory-bound operations in decode computations and thereby compensating for the overhead of runtime pruning and compression. Our custom attention kernel coupled with the bitmap-based format delivers substantial compression of KV cache up to 45% of dense inference and thereby enables longer context lengths and increased tokens/sec throughput of up to 2.23$\times$ compared to dense inference. Our pruning mechanism and sparse attention kernel is available at https://github.com/dhjoo98/mustafar.

1 Introduction

In the age of Large Language Models (LLMs), advances in the machine learning domain [41, 2, 6] and the fast and efficient computing systems [21, 35] have led to the emergence of highly capable LLMs that can summarize a book [22], write a compelling story [18], code a library [53], and generally reason over longer contexts than ever before [7]. As LLMs are increasingly tasked with processing longer sequences, the memory overhead associated with key-value (KV) caching has emerged as a critical bottleneck to scaling context length.

Prior work has approached the challenge of KV cache memory overhead through techniques such as quantization [30, 15, 48, 52], low-rank approximation [47, 4, 37, 50, 26], token-wise eviction [51, 29, 25, 8, 1, 11], and structured pruning (e.g., channel-wise removal [44, 31]). The need to improve individual compression techniques has become increasingly important, especially as joint applications of multiple methods, such as pruning combined with token eviction [44], quantization with token-wise eviction [52], and low-rank approximation with quantization [4], gain popularity. However, previous work on KV cache pruning have been limited to structured pruning, primarily due to the difficulty of efficiently leveraging finer-grained (i.e., unstructured) sparsity during execution. Effective pruning of the KV cache entails two core challenges: (1) achieving substantial reduction in KV cache size while preserving model accuracy, and (2) ensuring that the runtime pruning and compression processes are sufficiently efficient (i.e., the associated overhead must not outweigh the latency gains introduced by the resulting sparsity).

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

In this paper, we find that removing any constraint on the sparsity pattern, effectively unstructured sparsity can ensure that compressed KV cache perform with minimal model accuracy degradation while being pruned to a higher sparsity. In Section 2 (green region of Figure 1), we first present our journey to find the optimal pruning algorithm for the key and value cache, based on the element magnitude distributions of the KV cache. We explore the feasibility of various pruning algorithms on both KV cache to conclude that applying a simple per-token magnitude-based pruning on both Key and Value caches is capable of preserving the model accuracy at a high sparsity, while also demonstrating strong compatibility with orthogonal compression techniques.

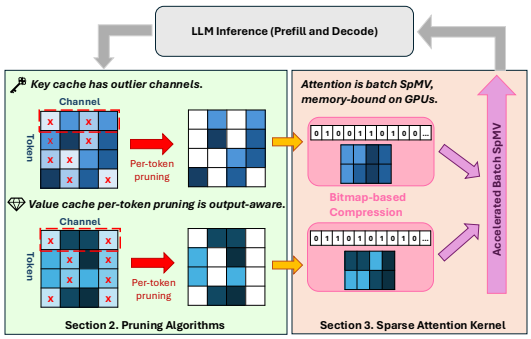


Figure 1: High-level overview of Mustafar. Green region describes the pruning algorithm of Section 2, pink region describes the custom sparse attention kernel of Section 3.

Section 3 (pink region of Figure 1) discusses the next step: having induced sparsity in the KV cache, the challenge becomes leveraging the unstructured sparsity to reduce memory footprint and accelerate computation. To this end, we adopt a bitmap-based sparse format that serves two purposes. First, the bitmap enables maximal compression of matrices with arbitrary sparsity patterns. Second, this maximal compression of matrix operands translates into computational speedup of the attention operation, which is severely memory-bound on GPUs. Alongside the sparse format, we introduce the custom attention kernel tailored to operate on the bitmap-based sparse format. We see that the speedup of our attention kernel overshadows the latency introduced by runtime pruning and compression, meanwhile effectively compressing the KV cache to high sparsity with minimal accuracy degradation.

In summary, we demonstrate that adopting unstructured sparsity in the KV cache without imposing constraints on the pruning pattern enables higher degrees of sparsity while preserving model accuracy. Furthermore, we introduce the necessary computational tools to support unstructured sparsity efficiently, ensuring that the derived high sparsity leads to gains in memory compression and end-to-end inference throughput.

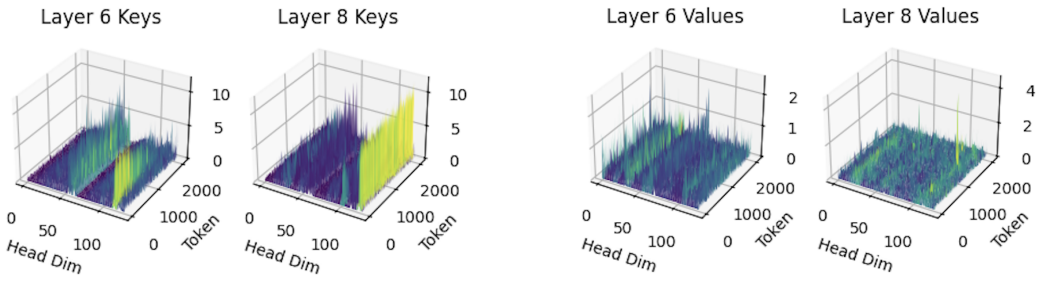
2 Pruning Algorithm for Unstructured Sparsity

Question: Does removing structural constraints in KV cache pruning allow for higher sparsity while preserving model accuracy more effectively than structured pruning methods?

We explore the potential unstructured sparsity on KV cache pruning by considering the two factors for Key and Value cache pruning: pruning direction and output-awareness. **Pruning Direction** refers to the axis along which sparsity is induced when selecting elements for removal. Since both the Key and Value caches are represented as matrices with dimensions $[tokens \times channels]$, we consider two primary pruning directions: per-channel pruning, which determines target sparsity across each channel (i.e., across tokens for each channel), and per-token pruning, which determines target sparsity across each token's cache (i.e., across model dimensions for each token). **Output-Awareness** refers to the use of a scoring metric that serves as a proxy for estimating each element's contribution to the operation's output. Commonly employed in LLM weight pruning [38] and structured KV cache pruning [44], this technique involves computing a score for each pruning unit such as a channel or an element by taking the product of the corresponding element with its associated input. This approach effectively captures the element's influence on the final output, guiding more informed pruning decisions. For a fair and effective comparison between pruning strategies, we uniformly employ a **local dense window**, where the recent 32 tokens remain untouched during the decode phase. Previous works [51, 44] have shown that this is effective in preserving model accuracy, meanwhile being small enough in size to not severely impact the compression.

2.1 Pruning Key Cache

In deciding the pruning direction, we build on top of the observation of KIVI [30], that Key cache exhibits distinct channel-wise outliers, where "channel" refers to the head dimension (Figure 2a). This leads us to focus on per-token pruning for key cache, as it can effectively capture the elements in the outlier channel.



(a) Magnitude distribution of Key cache

(b) Magnitude distribution of Value cache

Figure 2: Visualization of the KV cache in LLaMA-2 7B. Color intensity indicates element magnitude. The figure was generated using the visualization code from KIVI [30].

Based on the same observation to perform structured pruning of individual channels, ThinK [44] incorporates output-awareness by using a per-channel score of the accumulation of last 32 query, multiplied by each channel. To this end we compare the accuracy of ThinK [44], per-token magnitude-based unstructured pruning, and output-aware unstructured pruning of our design.

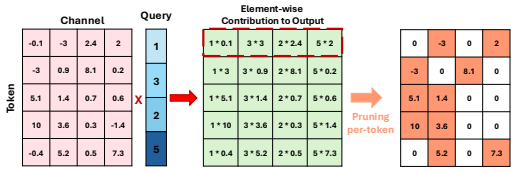


Figure 3: Per-token, output-aware pruning of Key cache

Figure 3 elaborates the per-token output-aware unstructured pruning score of Key cache. The element-wise L_1 accumulation of the current and next 31 Query vector (blue) is multiplied element-wise across each token's key vector (pink) to derive the pruning score (green). The absolute value of the score element in the corresponding position of each Key cache element is used to decide the elements to be pruned within a token's Key vector. In other words, we formulate the per-token output-aware unstructured pruning score S of a Key cache K to be:

$$S = |K| \odot broadcast\left(\sum_{t=T}^{T+31} |Q_t|\right), \quad \text{where } Q_t \text{ is the query at time } t$$

For Group Query Attention (GQA) [2], where multiple queries correspond to a KV cache pair, we sum the pruning score of all queries mapped to each KV cache.

Table 1: Comparison of ThinK [44] structured pruning, per-token magnitude-based unstructured pruning, and per-token output-aware unstructured pruning on LongBench [3] with Llama-3-8B-Instruct Key cache. K_s denotes Key cache sparsity.

Task	Dense	$K_s = 0.5$			$K_s = 0.7$		
		ThinK (Structured)	Unstructured Output-aware	Unstructured Magnitude	ThinK (Structured)	Unstructured Output-aware	Unstructured Magnitude
Average	43.19	38.53	**43.23**	42.84	26.55	**42.13**	41.55
SingleDoc QA	36.66	35.61	36.57	**36.90**	25.26	**35.78**	35.53
MultiDoc QA	36.09	34.99	**35.92**	35.77	29.75	**35.55**	35.40
Summarization	26.75	24.96	**26.87**	26.45	17.70	25.16	**25.18**
Few-shot	68.96	66.54	**68.82**	68.75	44.88	67.22	**67.84**
Synthetic	37.25	35.50	**37.00**	36.75	16.86	**35.25**	35.00
Code	55.58	29.56	**56.61**	54.14	19.15	**56.19**	51.47

In Table 1, we compare Llama-3-8B-Instruct accuracy of different pruning methods on LongBench [3]. For structured pruning, we see that even at a moderate sparsity, model accuracy retention is dismal compared to pruning to an unstructured sparsity pattern. Notably, unstructured pruning is capable of outperforming structured pruning even without the memory footprint of pruning scores involved with output-awareness. Applying output-awareness to unstructured pruning results in a slight improvement in the LongBench total average score, while individual task performance is mixed with each method outperforming the other on different tasks.

Key Cache Verdict: While the existence of outlier channels with exceptionally high magnitudes show promise for per-channel structured pruning, unstructured sparsity achieves higher accuracy at greater sparsity levels, even without output-awareness.

2.2 Pruning Value Cache

As shown in Figure 2b, Value cache exhibits more uniform distribution of activations, making it challenging to apply the same channel-wise pruning without incurring substantial degradation in model accuracy. This difficulty has led recent Value cache pruning approaches to be more susceptible to accuracy degradation.

With no discernible outliers in certain direction, we explore all possible combinations of (pruning direction, magnitude/output-aware) pairs. However, we are able to rule out per-token output-aware pruning, as the attention formulation $AttentionScore \times Value$ involves a multiply-and-accumulate operation along the token dimension. As seen in Figure 4, every element of a token's Value cache is multiplied by the same element of the attention score, with each element's impact on the output proportionate to the magnitude of each value. **That is, for Value cache pruning, per-token magnitude-based pruning is already output-aware**. For per-channel pruning, we prune each channel to the target sparsity in groups of 32 tokens, for compatibility with the local window size. For per-channel output-aware pruning, we accumulate the current and subsequent 31 attention score α of each token, which is then element-wise multiplied to the corresponding Value Cache (V) element. The following formula describes the pruning score S of per-channel output-aware pruning:

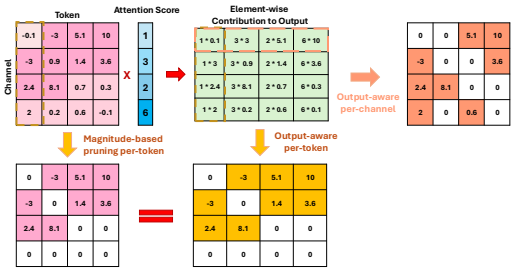


Figure 4: Output-aware per-channel (red) and magnitude-based per-token (pink) pruning of Value cache. Magnitude-based per-token pruning is equal to output-aware per-token pruning (yellow).

$$S = |V| \odot broadcast\left(\sum_{t=T}^{T+31} |\alpha_t|\right), \quad \text{where } \alpha_t \text{ is the attention score at time } t$$

Table 2: Comparison of ThinK [44] structured pruning, per-channel magnitude-based unstructured pruning, per-channel output-aware unstructured pruning, and per-token magnitude-based pruning on LongBench [3] with Llama-3-8B-Instruct Value Cache. V_s denotes Value cache sparsity.

Task	Dense	V_s = 0.5				V_s = 0.7			
		ThinK (Structured)	Magnitude (Per-channel)	Output-aware (Per-channel)	Magnitude (Per-token)	ThinK (Structured)	Magnitude (Per-channel)	Output-aware (Per-channel)	Magnitude (Per-token)
Average	43.19	38.45	42.50	42.84	**43.04**	30.60	41.69	42.67	**42.78**
SingleDoc QA	36.66	34.92	36.56	36.24	**36.75**	25.05	36.11	36.05	**36.96**
MultiDoc QA	36.09	34.74	35.45	36.07	**36.22**	23.90	35.11	**36.20**	35.82
Summarization	26.75	23.31	24.74	25.79	**26.34**	20.41	22.72	24.75	**25.19**
Few-shot	68.96	67.18	67.66	68.65	**68.91**	60.16	67.39	**68.23**	68.08
Synthetic	37.25	35.43	**38.31**	37.00	36.25	29.63	**38.75**	37.25	35.50
Code	55.58	31.97	55.07	55.57	**55.77**	20.85	52.65	56.17	**57.62**

As shown in the Table 2, we first see that applying structured pattern to Value cache pruning incurs significant accuracy degradation even in 50% sparsity. This is concurrent with ThinK [44] findings, which points to 30% sparsity as the upper-bound on acceptable accuracy. In contrast, per-token

magnitude pruning is capable of preserving model accuracy even at 70% sparsity. For per-channel pruning, we see that incorporating output-awareness boasts model accuracy retention almost to the level of per-token pruning. However, we prefer per-token magnitude-based pruning for the following two reasons. First, output-aware per-channel value cache pruning requires access to the attention score which requires additional recomputation when used alongside FlashAttention [6], where the full attention score matrix does not materialize in the global memory. Second, per-token magnitude-based pruning allows smooth compatibility with orthogonal compression method token-wise eviction [24, 51], where the retained token's KV cache can be pruned individually. We examine the accuracy of joint application in Section 4.2.

Value Cache Verdict: All unstructured pruning methods explored outperform structured pruning. Among unstructured pruning methods, token-wise pruning, which is inherently output-aware by matrix multiplication formulation, best preserves model accuracy even at high sparsity levels. While channel-wise pruning with output-awareness can achieve comparable accuracy, token-wise pruning offers advantages in both efficiency and modularity.

With the two verdicts in Key and Value caches, on Table 3 we finally validate the model accuracy retention of per-token magnitude-based pruning with both Key and Value caches pruned. Not only can Value cache be pruned to high sparsity with unstructured sparsity, but both KV cache can be pruned to 70% sparsity while showing similar or better accuracy than Key-only 50% structured pruning of ThinK [44]. In Appendix A.1, methodology of this section is applied on Llama-2 7B to reinforce the effectiveness of per-token magnitude-based KV cache pruning.

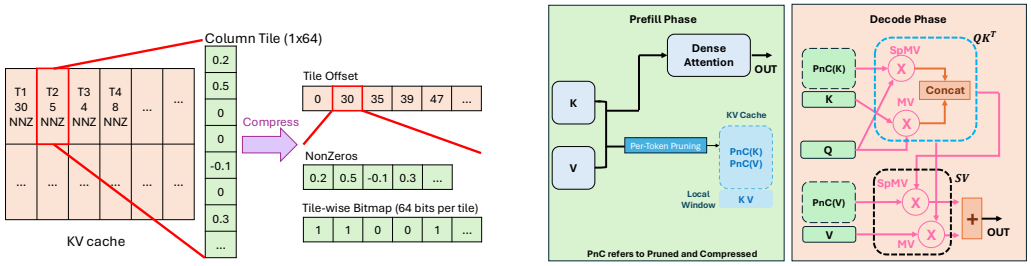
Table 3: Longbench Score of Llama-3-8B-Instruct and Mistral-7B-Instruct-v0.2 with KV Cache Per-Token Magnitude-based Pruning.

Task	Llama-3-8B-Instruct			Mistral-7B-Instruct-v0.2		
	Dense	$K_s = 0.5$ $V_s = 0.5$	$K_s = 0.7$ $V_s = 0.7$	**Dense**	$K_s = 0.5$ $V_s = 0.5$	$K_s = 0.7$ $V_s = 0.7$
Average	43.19	42.65	40.96	42.65	42.30	40.95
SingleDoc QA	36.66	36.67	35.28	36.21	36.22	36.08
MultiDoc QA	36.09	36.23	35.11	29.93	30.42	29.40
Summarization	26.75	26.05	23.57	28.10	27.77	26.72
Few-shot	68.96	68.18	66.10	66.68	66.70	66.24
Synthetic	37.25	36.00	34.13	44.85	41.92	36.13
Code	55.58	54.50	53.49	54.98	54.83	53.84

3 Sparse Attention Kernel

Our findings establish that unstructured sparsity offers superior sparsity ratios over structured sparsity while preserving accuracy. In turn, a crucial contribution of Mustafar is to leverage this advantage to enable high compression efficiency while minimizing the latency overhead of runtime pruning and compression. Prior compression methods such as quantization, structured pruning, and token eviction reduce matrix dimensions or element bitwidths. In terms of efficiency, speedup from the reduced size of dense matrix operands compensates for the additional latency introduced by compression (i.e. pruning score computation, quantization). In contrast, unstructured sparsity with no regular reduction in dimensions or element bitwidth demands a different approach.

Mustafar is motivated by the observation that attention operations in the autoregressive decode stage, the *Query* $\times$ *Key*^{T} and *Attention Score* $\times$ *Value* computations are batch (different heads) of matrix-vector products (MVs) that are significantly memory-bound on GPUs compared to the prefill stage. To exploit this property, we extend the bitmap-based sparse format of Coruscant [20] as shown in Figure 5a to maximally compress the pruned KV cache. It consists of compressed tiles corresponding to a 1×64 column of the pruned cache. Per-tile bitmap of 64 bits is used to represent the position of non-zeros, and tile offset is used to address the correct position of each tile's starting non-zero. Pruning and compression are performed on-the-fly, with compression accelerated on GPU with a Triton kernel, and attention is computed directly on the compressed representation with a custom CUDA kernel that performs batch SpMV on the bitmap-based sparse format. Memory-bound decode-phase attention is accelerated by reducing the data movement from global memory to GPU Streaming Multiprocessors.



(a) Coruscant [20] bitmap-based sparse format

(b) Mustafar attention kernel formulation

Figure 5: Overview of Mustafar sparse attention kernel. In (b), multi-head, softmax, and normalization are omitted for simplicity.

Figure 5b and Algorithm 1 presents the Mustafar sparse attention kernel. KV cache generated in prefill stage is pruned and compressed before the start of decode stage, therefore compatible with prefill FlashAttention [6]. KV cache generated in decode stage is kept as-is (dense) while it is within the local window, then pruned and compressed afterwards. This entails the attention computations in the decode stage to be reformulated into two parts: SpMV for compressed KV cache (line 2 and 5 of Algorithm 1) and dense MV for the KV cache within the local window (line 1 and 5 of Algorithm 1).

Algorithm 1 Decode Phase Attention with Dense Local and Compressed KV Caches

Input: Query $\mathbf{Q}_t \in \mathbb{R}^d$; Local KV cache $\mathbf{K}_L, \mathbf{V}_L \in \mathbb{R}^{d \times N_d}$, where N_d is size of local window in tokens; Compressed KV cache $\mathbf{K}_C, \mathbf{V}_C \in \mathbb{R}^{d \times N_s}$, where N_s is number of compressed tokens.

Attention Score Computation

1: $\mathbf{S}_L \in \mathbb{R}^{1 \times N_d} \leftarrow \mathbf{Q}_t \mathbf{K}_L$ *Dense local window attention score*

2: $\mathbf{S}_C \in \mathbb{R}^{1 \times N_s} \leftarrow \mathbf{Q}_t \mathbf{K}_C$ *Sparse attention score over compressed KV cache*

3: $\mathbf{S}_t \in \mathbb{R}^{1 \times (N_s + N_d)} \leftarrow \mathrm{softmax}\big(\mathrm{concat}(\mathbf{S}_C, \mathbf{S}_L)\big)$ *Full attention score*

Output Computation

4: $[\mathbf{S}_C, \mathbf{S}_L] \leftarrow \mathrm{split}(\mathbf{S}_t; N_s, N_d)$ *Partition attention score*

5: $\mathbf{O}_t \in \mathbb{R}^d \leftarrow \mathbf{V}_C \mathbf{S}_C^\top + \mathbf{V}_L \mathbf{S}_L^\top$ *Final output vector*

Return $\mathbf{O}_t$

Mustafar SpMV kernel follows the load-as-compressed, compute-as-dense paradigm adopted by FlashLLM [43], SpInfer [9], and Coruscant [20], which target sparse matrix–dense matrix multiplication in LLM weight projection layers. The compressed KV cache is loaded from GPU global memory into registers in its compressed form, decompressed into shared memory, and then used for tile-wise dense computation. We evaluate the performance of the Mustafar attention kernel and quantify the runtime overhead of pruning and compression in Section 4.3. We further detail the formulation of the SpMV kernel, as well as the management of the compressed KV cache in Appendix C.

4 Evaluation

Methodology: We evaluate Mustafar on two aspects: Accuracy and Efficiency. For accuracy evaluation, we use tasks from LongBench [3] to test the accuracy retention of per-token magnitude-based pruning of KV cache. We evaluate on three models: Llama-2-7B [40], Llama-3-8B-Instruct [12], and Mistral-7B-Instruct-v0.2 [19]. We also explore the impact of Mustafar when jointly used with orthogonal compression techniques, KV cache quantization KIVI [30] and token-wise eviction H2O [51]. For efficiency evaluation, we evaluate the impact on KV cache compression and computational latency with Llama-2-7B and Llama-3-8B-Instruct. Efficiency evaluation is tested on NVIDIA RTX 6000ADA GPU and measured with NVIDIA Nsight Profiling Tool. Additionally, we provide accuracy evaluation on RULER [17] benchmark in Appendix A.3, accuracy comparison of Mustafar's unstructured sparsity with 2:4 semi-structured sparsity in Appendix B, and additional kernel throughput evaluation in Appendix C.3.

4.1 LongBench Results

Table 4 shows the extended LongBench evaluation of Mustafar per-token magnitude-based pruning with comparison to dense model and ThinK [44]. Under the same Key cache sparsity, unstructured nature of Mustafar constantly achieves higher accuracy than structured sparsity on ThinK [44] across all tasks. A key advantage of unstructured pruning is its ability to effectively prune the Value cache with minimal accuracy degradation, which structured pruning has struggled to achieve. Even under high sparsity 70% for both the Key and Value caches, unstructured pruning (yellow) consistently outperforms ThinK's Key-only 50% structured pruning (pink) on LLaMA-3 8B and Mistral 7B, and achieves comparable accuracy on LLaMA-2-7B.

Table 4: Mustafar accuracy with Llama and Mistral on LongBench

KV Sparsity	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RBP	
Llama-3 8B Instruct																	
Dense	23.39	43.38	43.22	46.39	38.66	23.22	29.91	22.56	27.77	74.50	90.28	42.11	4.50	70.00	57.11	54.05	**43.19**
ThinK0.5	22.38	40.96	43.48	44.01	38.37	22.59	26.61	22.20	26.08	74.00	88.83	36.79	6.00	65.00	27.95	31.17	**38.53**
K0.5 V0.0	23.40	43.68	43.63	46.00	38.60	22.72	29.39	22.33	27.64	74.50	90.66	41.09	5.00	68.50	55.89	52.39	**42.84**
ThinK0.7	17.58	27.40	30.80	40.59	29.50	19.16	18.13	17.28	17.70	34.00	83.09	17.56	4.71	29.00	17.88	20.42	**26.55**
K0.7 V0.0	22.91	42.36	41.33	45.53	38.50	22.16	26.63	21.90	27.00	73.00	90.83	39.68	4.50	65.50	51.94	50.99	**41.55**
K0.0 V0.5	23.80	43.14	43.32	46.28	39.42	22.97	29.18	22.70	27.13	74.50	90.50	41.74	5.00	67.50	57.23	54.30	**43.04**
K0.0 V0.7	24.19	42.78	43.92	45.82	39.11	22.53	26.92	22.52	26.12	74.00	90.36	39.88	5.50	65.50	59.18	56.05	**42.77**
K0.5 V0.5	23.40	46.63	42.98	46.28	39.27	23.13	28.29	22.78	27.07	74.00	90.58	39.97	5.00	67.00	55.54	53.46	**42.65**
K0.7 V0.7	24.10	40.85	40.88	44.93	38.03	22.36	24.02	21.90	24.78	70.50	90.04	37.77	5.25	63.00	54.12	52.86	**40.96**
Mistral-7B-Instruct-v0.2																	
Dense	26.76	32.51	49.36	43.49	27.48	18.81	32.95	24.36	27.00	71.00	86.23	42.80	2.89	86.81	55.89	54.07	**42.65**
ThinK0.5	24.03	26.79	46.42	38.70	24.93	15.73	32.72	24.65	27.14	71.00	85.80	41.68	2.20	73.67	48.83	47.09	**39.46**
K0.5 V0.0	26.38	33.08	49.20	43.90	28.57	18.65	32.47	24.21	27.05	71.00	86.28	42.66	3.00	84.23	55.72	54.16	**42.56**
ThinK0.7	19.25	21.33	36.48	27.96	20.34	14.08	29.32	22.23	25.64	70.50	78.99	29.66	2.92	54.42	34.28	31.68	**32.44**
K0.7 V0.0	27.02	34.37	49.26	43.77	26.37	17.45	32.05	24.09	27.43	71.00	87.19	42.30	4.65	77.24	54.26	53.06	**41.97**
K0.0 V0.5	26.29	32.54	49.01	43.99	28.02	19.28	32.07	23.74	26.98	71.00	86.56	42.79	2.71	81.77	55.14	54.16	**42.25**
K0.0 V0.7	26.83	31.66	49.24	44.15	27.40	18.36	30.58	23.80	26.63	71.00	86.82	42.02	3.77	76.32	55.58	54.16	**41.77**
K0.5 V0.5	26.90	32.99	48.76	43.90	28.90	18.45	32.24	24.09	26.99	71.00	86.68	42.41	3.20	80.64	55.51	54.15	**42.30**
K0.7 V0.7	27.11	32.23	48.90	43.63	27.12	17.43	29.38	23.99	26.79	71.00	86.59	41.14	4.69	67.57	54.86	52.82	**40.95**
Llama-2 7B																	
Dense	15.04	9.66	21.88	7.69	9.95	3.66	17.26	21.29	3.5	66.00	87.72	41.66	1.70	6.64	66.66	59.82	**27.51**
ThinK0.5	15.57	9.96	23.31	6.50	9.62	2.77	1.84	20.16	0.38	66.00	85.53	41.48	2.04	2.79	64.77	58.36	**25.69**
K0.5 V0.0	14.79	9.65	21.67	7.48	10.10	4.11	17.24	20.84	3.64	66.00	87.72	41.26	1.38	6.42	67.15	59.89	**27.46**
ThinK0.7	13.76	8.16	20.59	4.53	6.24	2.23	12.96	14.88	0.01	66.00	80.48	26.95	1.77	6.93	40.73	38.97	**21.57**
K0.7 V0.0	14.57	8.18	20.55	6.64	9.95	3.28	13.80	20.25	0.88	66.00	86.64	38.32	2.12	4.04	64.86	58.59	**26.17**
K0.0 V0.5	15.71	10.02	21.12	7.38	9.64	3.75	16.86	21.37	2.38	66.00	87.72	41.04	1.65	6.75	66.79	60.09	**27.40**
K0.0 V0.7	15.57	8.98	20.97	7.33	10.14	3.82	15.40	20.77	1.83	66.00	87.72	40.69	1.40	6.50	66.12	59.57	**27.05**
K0.5 V0.5	15.49	9.17	20.97	7.51	10.04	3.78	16.46	21.02	3.36	66.00	87.72	40.81	1.22	5.88	66.78	59.53	**27.23**
K0.7 V0.7	13.76	7.83	19.27	6.57	10.26	3.51	8.70	20.04	0.47	64.50	86.89	36.37	1.64	3.62	63.95	56.75	**25.26**

4.2 Joint Application with Orthogonal KV Cache Compression Techniques

Mustafar's per-token pruning enables seamless integration with orthogonal KV cache compression techniques. We evaluate its effectiveness when combined with token eviction from H2O [51] and KV cache quantization from KIVI [30], using a representative subset of LongBench tasks from each category. H2O application is conducted with Llama-2 7B and KIVI application is conducted with LLaMA-3-8B-Instruct.

4.2.1 Joint Application with Token Eviction

H2O [51] retains a fixed budget of recent tokens and critical heavy-hitter tokens. Applying Mustafar to H2O, we retain the same scheme of pruning the KV cache of tokens that exit the local dense window. We configure 10% of KV cache budget each to recent tokens and heavy-hitter tokens. Jointly applied, all heavy-hitter tokens and a part of recent tokens is kept as pruned and compressed. In Table 5, we validate the efficacy of Mustafar's accuracy retention when jointly applied with token eviction, as we see that 50% sparsity in both KV cache retains the dense accuracy with some degradation when pruned to 70% sparsity.

Table 5: LongBench evaluation of Mustafar-H2O joint application on Llama-2-7B

	Single-Doc QA NtrvQA	Multi-Doc QA HotpotQA	Summarization GovReport	Few-shot Learning TREC	Synthetic Pcount	Code Lcc
Full KV cache	15.04	7.69	17.26	66.00	1.7	66.66
H2O 20% KV Budget						
Dense	12.26	8.35	7.76	64.00	1.43	64.20
K0.5 V0.0	11.83	8.47	7.65	64.00	2.08	64.72
K0.7 V0.0	11.39	8.46	6.34	64.00	1.69	63.92
K0.0 V0.5	12.17	8.39	6.84	64.00	1.38	64.83
K0.0 V0.7	12.39	7.79	5.81	64.00	0.76	64.88
K0.5 V0.5	12.07	8.16	7.61	64.00	2.05	65.15
K0.7 V0.7	12.20	8.18	5.22	64.00	1.65	63.73

4.2.2 Joint Application with Quantization

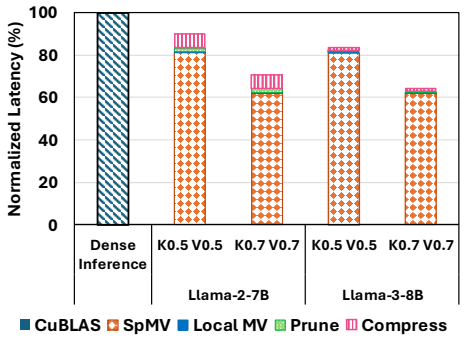
KIVI [30] applies a per-channel quantization of Key cache and per-token quantization of Value cache. Following findings of Harma et al. [14], we first prune each token's KV cache before quantization is performed. However, we note that current Mustafar sparse attention kernel implementation does not support low-bit precision. Therefore, the accuracy measurement was performed on a sparse quantized KV cache. Table 6 shows the performance of Mustafar and KIVI applied together. Similar to joint application with H2O, we see that model accuracy is retained across the tasks for 50% on Key cache, Value cache, as well as both Key and Value caches. We observe a decrease in accuracy at 70% pruning, with Summarization task seeing the most significant drop. However, other tasks, such as Single-Document QA maintain the same performance as naive 16-bit model, suggesting the potential for applying varying degrees of compression tailored to specific tasks.

Table 6: LongBench evaluation of Mustafar-KIVI joint application on Llama-3-8B-Instruct

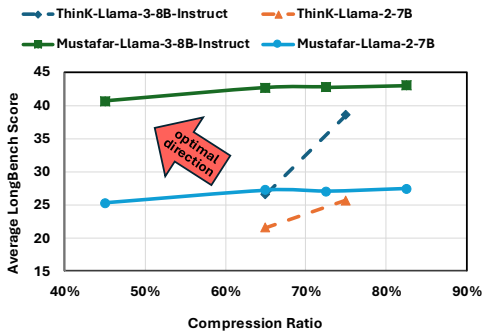
	Single-Doc QA NtrvQA	Multi-Doc QA HotpotQA	Summarization GovReport	Few-shot Learning TREC	Synthetic Pcount	Code Lcc
Naive 16-bit	23.39	46.39	29.91	74.50	4.50	57.11
KIVI 4-bit						
Dense	23.60	46.39	29.84	74.50	5.00	57.35
K0.5 V0.0	23.46	46.21	28.90	74.50	5.50	56.05
K0.7 V0.0	23.35	45.40	26.46	73.50	4.83	52.41
K0.0 V0.5	23.68	46.39	29.10	74.50	5.50	58.30
K0.0 V0.7	24.10	45.66	27.21	74.00	5.50	59.30
K0.5 V0.5	23.22	46.06	28.18	74.00	6.00	56.04
K0.7 V0.7	23.74	45.50	23.57	70.50	6.25	54.12
KIVI 2-bit						
Dense	23.33	45.47	29.69	74.50	6.50	50.38
K0.5 V0.0	22.86	45.29	29.39	74.00	5.50	49.92
K0.7 V0.0	22.88	44.60	26.91	73.00	4.50	43.84
K0.0 V0.5	23.65	45.67	29.05	74.00	5.50	51.94
K0.0 V0.7	23.68	45.47	27.57	74.00	5.50	52.90
K0.5 V0.5	22.46	45.47	28.61	74.00	4.50	48.76
K0.7 V0.7	22.72	45.18	23.84	71.00	5.12	45.68

4.3 Efficiency Evaluation

A crucial aspect of Mustafar is to ensure that the exploitation of sparsity for compressing the KV cache does not deter the inference latency. Mustafar compensates the overhead of runtime pruning and compression by achieving speedup in the memory-bound SpMV. Figure 6a compares the normalized latency of dense batched MV of cuBLAS with the components of Mustafar sparse attention kernel (Figure 5b): batched SpMV, dense batched MV of local window, runtime pruning, and compression, for input sequence length 2048 for Llama-2 and 4096 for Llama-3 and generation length 1024. In the multi-head attention of Llama-2-7B, pruning introduces 1.84%, compression introduces 6.25%, and MV of local window introduces 0.62% of the cuBLAS execution time in dense inference. In both 50% and 70%, the speedup gained from SpMV kernel more than compensates for the introduced overheads. In 50% sparsity, SpMV takes 81.07% of cuBLAS execution time and for 70% sparsity, SpMV takes 61.87% of cuBLAS execution time. In Grouped-Query Attention of Llama-3-8B, where there is reduced set of KV cache, compression and pruning overhead reduce down to 1.47% and 0.47% of cuBLAS execution time respectively.



(a) Normalized kernel latency breakdown.

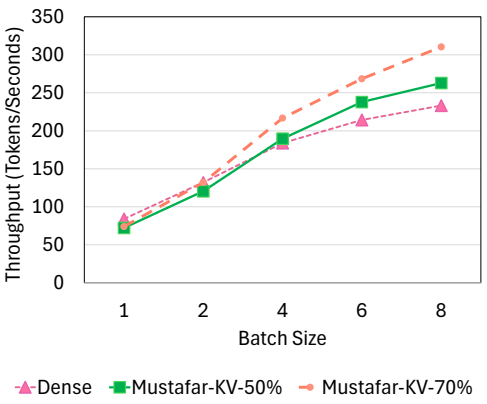


(b) Compression ratio-accuracy comparison of Mustafar and ThinK.

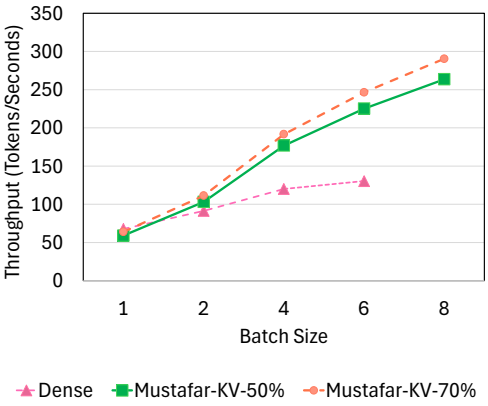
Figure 6: Efficiency evaluation of Mustafar. In (b), compression ratio refers to percentage of compressed size compared to dense KV cache.

Figure 6b compares the KV cache compression ratio (% of size in memory compared to dense KV cache) of Mustafar and ThinK along with the LongBench average score achieved with Llama-2-7B and Llama-3-8B-Instruct. In this plot, the red arrow points to the optimal direction, where a model achieves higher LongBench score while achieving high compression of the KV cache. For ThinK [44] which prune only Key cache, 50% sparsity leads to 75% compression ratio to dense KV cache, and 70% Key cache sparsity leads to 65% compression ratio. In the case of Mustafar where both Key and Value Cache can be pruned, KV cache 50% sparsity leads to 65% compression ratio. The reason behind 15% additional memory footprint is due to the tile offset overhead as shown in Figure 5a and the multiples-of-8 padding enforced to coalesce memory access in GPU. KV cache 70% sparsity leads to 45% compression ratio, 50% sparsity to either Key or Value cache leads to 83% compression ratio, and single-cache 70% sparsity leads to 72.5% compression ratio. Overall, we see that Mustafar is able to achieve better accuracy given the compression ratio, with the compression ratio-accuracy curve closer to the optimal direction than ThinK.

Figure 7 shows the throughput comparison to inference with dense models. For Llama-2 7B, we used input sequence length of 2048 and generated 2048 tokens. For Llama-3 8B, we use input sequence length of 4096 and generated 4096 tokens. For dense baseline, FlashAttention [6] was used on prefill and decode phase. Overall, we see that Mustafar is able to achieve higher throughput as well as support larger batch size owing to the reduced memory footprint of KV cache. In Llama-3, we see that enabling batch size of 8 leads to 2.23× tokens/sec throughput compared the dense inference of



(a) Llama-2 7B Throughput.



(b) Llama-3 8B Instruct Throughput.

Figure 7: Throughput comparison of Mustafar to dense inference.

batch size 6. Even within the same batch size, we see an increased throughput upto 1.89×. This is due to the pruning and compression overhead amortized by the speedup of Mustafar sparse attention kernel, leading to faster inference latency. However in batch size 1, we see that throughput is lower than dense inference. This is due to the underutilization of GPU in Mustafar sparse attention kernel with small batch size, where the number of threadblocks is smaller than the number of SMs. We provide additional throughput comparison with different input:output token ratios on Appendix C.3.

5 Related Work

KV cache compression Alongside aforementioned work in KV cache pruning [44, 31], quantization [30, 15, 48, 52], token-wise eviction [51, 29, 25, 8, 1, 11], and low-rank approximation [47, 4, 37, 50, 26], KV cache offloading [24, 28, 13, 5] evicts KV cache to CPU memory and speculatively prefetchs critical tokens' KV cache. Layer-centric compression [49, 27] applies different level of compression to different layers, adhering to layer-wise importance. Head-level compression [10, 39] applies different level of compression to each heads, from the observation that not all heads contribute equally. Phase-specific compression [42] applies different strategy for prefill and decode phase, with information retention prioritized in prefill and heavy hitter selection applied on decode phase.

System/Kernel for Attention While Mustafar attention kernel focuses on operating directly on the bitmap-compressed sparse KV cache, there exists various contributions from the system and kernel-levels to optimize for attention. PagedAttention [23] introduces an paging-inspired attention algorithm that partitions KV cache into memory blocks to reduce memory fragmentation and efficient sharing across sequences. FlashDecoding[16] introduces double-buffering to accelerate memory-bound GeMM of decode phase. FlashInfer [45] unifies KV cache format using a block-sparse representation for an efficient management of KV cache that leads to increased throughput. Loki [36] uses a sparse attention method that leverages the low-dimensionality of key vectors to perform an approximate attention in a reduced PCA space.

6 Conclusion and Limitations

In this work, we demonstrate that unstructured sparsity presents a powerful and novel solution for KV cache pruning. By removing constraints on the pruning pattern, we show that per-token magnitude-based pruning achieves high sparsity while maintaining model accuracy. To unlock the practical benefits of unstructured sparsity, we introduce a bitmap-based sparse format and a custom attention kernel that directly operates on compressed KV cache. Together, our pruning strategy, sparse format, and custom kernel form an end-to-end system that substantially reduces KV cache memory usage and improves throughput, making it possible to support longer contexts and more efficient inference. Mustafar establishes a foundation for future efforts to integrate unstructured sparsity into practical LLM deployment pipelines and opens new directions for memory-efficient LLM inference at scale. In future work, we plan to explore the joint effect of leveraging KV sparsity of Mustafar with sparsity in weights derived by works such as output-aware weight pruning [38], pruning with low-rank adapters for accuracy retention [34, 33], and activation-aware calibration and efficiency enhancement [32, 46]. Additionally, this paper focuses on showing that unstructure sparsity can prune both Key and Value caches to a higher sparsity with better accuracy than structured sparsity, leaving our method's ability to map arbitrary sparsity degree untouched. While we explore higher sparsity uniformly applied to the entire KV cache in Appendix A.4, a future work involves deriving the optimal target sparsity to a smaller granularity (e.g. per-head or per-layer) to maximize sparsity and accuracy retention.

Acknowledgments and Disclosure of Funding

We gratefully acknowledge the support of National Science Foundation (NSF) under program PPoSS, Award Number 2316177.

References

[1] Muhammad Adnan et al. "Keyformer: KV Cache reduction through key tokens selection for Efficient Generative Inference". In: *Proceedings of Machine Learning and Systems*. Ed. by P. Gibbons, G. Pekhimenko, and C. De Sa. Vol. 6. 2024, pp. 114–127. URL: https://proceedings.mlsys.org/paper_files/paper/2024/file/48fecef47b19fe501d27d338b6d52582-Paper-Conference.pdf.

[2] Joshua Ainslie et al. "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints". In: *The 2023 Conference on Empirical Methods in Natural Language Processing*. 2023. URL: https://openreview.net/forum?id=hmOwOZWzYE.

[3] Yushi Bai et al. "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)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 3119–3137. DOI: 10.18653/v1/2024.acl-long.172. URL: https://aclanthology.org/2024.acl-long.172/.

[4] Chi-Chih Chang et al. "Palu: KV-Cache Compression with Low-Rank Projection". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=LWMS4pk2vK.

[5] Zhuoming Chen et al. "MagicPIG: LSH Sampling for Efficient LLM Generation". In: *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. 2025. URL: https://arxiv.org/abs/2410.16179.

[6] Tri Dao. "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning". In: *Proceedings of the 12th International Conference on Learning Representations*. 2024.

[7] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. URL: https://doi.org/10.48550/arXiv.2501.12948.

[8] Harry Dong et al. "Get more with LESS: synthesizing recurrence with KV cache compression for efficient LLM inference". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2024.

[9] Ruibo Fan et al. "SpInfer: Leveraging Low-Level Sparsity for Efficient Large Language Model Inference on GPUs". In: *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys '25)*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 243–260. DOI: 10.1145/3689031.3717481. URL: https://doi.org/10.1145/3689031.3717481.

[10] Yu Fu et al. "Not All Heads Matter: A Head-Level KV Cache Compression Method with Integrated Retrieval and Reasoning". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=FJFVmeXusW.

[11] Suyu Ge et al. "Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2024.

[12] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. 2024. URL: https://arxiv.org/abs/2407.21783.

[13] Jitai Hao et al. "OmniKV: Dynamic Context Selection for Efficient Long-Context LLMs". In: *Proceedings of the 13th International Conference on Learning Representations (ICLR)*. 2025.

[14] Simla Burcu Harma et al. "Effective Interplay between Sparsity and Quantization: From Theory to Practice". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=wJv4AIt4sK.

[15] Yefei He et al. "ZipCache: Accurate and Efficient KV Cache Quantization with Salient Token Identification". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=5t4ZAkPiJs.

[16] Ke Hong et al. "FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics". In: *Proceedings of Machine Learning and Systems*. Ed. by P. Gibbons, G. Pekhimenko, and C. De Sa. Vol. 6. 2024, pp. 148–161. URL: https://proceedings.mlsys.org/paper_files/paper/2024/file/5321b1dabcd2be188d796c21b733e8c7-Paper-Conference.pdf.

[17] Cheng-Ping Hsieh et al. "RULER: What's the Real Context Size of Your Long-Context Language Models?" In: *Proceedings of COLM 2024*. 2024.

[18] Fantine Huot et al. “Agents’ Room: Narrative Generation through Multi-step Collaboration”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=HfWcFs7XLR.

[19] Albert Q. Jiang et al. “Mistral 7B”. In: *arXiv preprint arXiv:2310.06825*. 2023. URL: https://arxiv.org/abs/2310.06825.

[20] Donghyeon Joo et al. “Coruscant: Co-Designing GPU Kernel and Sparse Tensor Core to Advocate Unstructured Sparsity in Efficient LLM Inference”. In: *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. 2025. ISBN: 9798400715730. DOI: 10.1145/3725843.3756065. URL: https://doi.org/10.1145/3725843.3756065.

[21] Norman P. Jouppi et al. “TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings”. In: *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. 2023. URL: https://arxiv.org/abs/2304.01433.

[22] Yekyung Kim et al. “FABLES: Evaluating faithfulness and content selection in book-length summarization”. In: *First Conference on Language Modeling*. 2024. URL: https://openreview.net/forum?id=YfHxQSoaWU.

[23] Woosuk Kwon et al. “Efficient Memory Management for Large Language Model Serving with PagedAttention”. In: *Proceedings of the 29th Symposium on Operating Systems Principles*. SOSP ’23. Koblenz, Germany: Association for Computing Machinery, 2023, 611–626. ISBN: 9798400702297. DOI: 10.1145/3600006.3613165. URL: https://doi.org/10.1145/3600006.3613165.

[24] Wonbeom Lee et al. “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)*. Santa Clara, CA: USENIX Association, July 2024, pp. 155–172. ISBN: 978-1-939133-40-3. URL: https://www.usenix.org/conference/osdi24/presentation/lee.

[25] Yuhong Li et al. “SnapKV: LLM Knows What You are Looking for Before Generation”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=poE54GOq2l.

[26] Bokai Lin et al. “MatryoshkaKV: Adaptive KV Compression via Trainable Orthogonal Projection”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=BQwsRy1h3U.

[27] Akide Liu et al. “MiniCache: KV Cache Compression in Depth Dimension for Large Language Models”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=sgVOjDqUMT.

[28] Di Liu et al. *RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval*. 2024. URL: https://arxiv.org/abs/2409.10516.

[29] Zichang Liu et al. “Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=JZfg6wGi6g.

[30] Zirui Liu et al. “KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache”. In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=L057s2Rq8O.

[31] Bo Lv et al. “KVPruner: Structural Pruning for Faster and Memory-Efficient Large Language Models”. In: *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2025, pp. 1–5. DOI: 10.1109/ICASSP49660.2025.10889000.

[32] Iman Mirzadeh et al. “ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models”. In: *Proceedings of the Twelfth International Conference on Learning Representations*. 2024. URL: https://arxiv.org/pdf/2310.04564.

[33] Mohammad Mozaffari, Amir Yazdanbakhsh, and Maryam Mehri Dehnavi. “SLiM: One-shot Quantization and Sparsity with Low-rank Approximation for LLM Weight Compression”. In: *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*. 2025. URL: https://arxiv.org/abs/2410.09615.

[34] Mohammad Mozaffari et al. “SLoPe: Double-Pruned Sparse Plus Lazy Low-Rank Adapter Pretraining of LLMs”. In: *Proceedings of the International Conference on Learning Representations (ICLR 2025)*. 2025. URL: https://arxiv.org/abs/2405.16325.

[35] Md Aamir Raihan, Negar Goli, and Tor Aamodt. *Modeling Deep Learning Accelerator Enabled GPUs*. 2019.

[36] Prajwal Singhania et al. "Loki: Low-rank Keys for Efficient Sparse Attention". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=raABeiV71j.

[37] Hanshi Sun et al. "ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference". In: *Proceedings of the Forty-Second International Conference on Machine Learning*. 2025. URL: https://arxiv.org/abs/2410.21465.

[38] Mingjie Sun et al. "A Simple and Effective Pruning Approach for Large Language Models". In: *Proceedings of the 12th International Conference on Learning Representations*. 2024.

[39] Hanlin Tang et al. "RazorAttention: Efficient KV Cache Compression Through Retrieval Heads". In: *arXiv preprint arXiv:2407.15891*. 2024. URL: https://arxiv.org/abs/2407.15891.

[40] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. URL: https://arxiv.org/abs/2307.09288.

[41] Ashish Vaswani et al. "Attention Is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*. 2017. URL: https://doi.org/10.48550/arXiv.1706.03762.

[42] Jialong Wu et al. "SCOPE: Optimizing Key-Value Cache Compression in Long-context Generation". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. 2025. URL: https://aclanthology.org/2025.acl-long.529.pdf.

[43] Haojun Xia et al. "Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity". In: vol. 17. 2. VLDB Endowment. DOI: 10.14778/3626292.3626303. URL: https://doi.org/10.14778/3626292.3626303.

[44] Yuhui Xu et al. "ThinK: Thinner Key Cache by Query-Driven Pruning". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=n0OtGl6VGb.

[45] Zihao Ye et al. *FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving*. 2025.

[46] Ruokai Yin et al. *DuoGPT: Training-free Dual Sparsity through Activation-aware Pruning in LLMs*. 2025. arXiv: 2506.20194 [cs.LG]. URL: https://arxiv.org/abs/2506.20194.

[47] Zhihang Yuan et al. *ASVD: Activation-Aware Singular Value Decomposition for Compressing Large Language Models*. arXiv preprint arXiv:2312.05821. DOI: 10.48550/arXiv.2312.05821. URL: https://doi.org/10.48550/arXiv.2312.05821.

[48] Tianyi Zhang et al. "KV Cache is 1 Bit Per Channel: Efficient Large Language Model Inference with Coupled Quantization". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=pNnvzQsS4P.

[49] Yanqi Zhang et al. *Unifying KV Cache Compression for Large Language Models with LeanKV*. 2024. URL: https://doi.org/10.48550/arXiv.2412.03131.

[50] Yifan Zhang et al. *Tensor Product Attention Is All You Need*. 2025. URL: https://arxiv.org/abs/2501.06425.

[51] Zhenyu Zhang et al. "H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=RkRrPp7GKO.

[52] Zhenyu Zhang et al. "Q-Hitter: A Better Token Oracle for Efficient LLM Inference via Sparse-Quantized KV Cache". In: *MLSys*. 2024. URL: https://proceedings.mlsys.org/paper_files/paper/2024/hash/bbb7506579431a85861a05fff048d3e1-Abstract-Conference.html.

[53] Wenting Zhao et al. "Commit0: Library Generation from Scratch". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=MMwaQEVsAg.

A Extended Evaluation

A.1 Section 2 Methodology Applied to LLaMA-2 7B

We follow the same methodology of exploring pruning direction and output-awareness on Llama-2-7B to further solidify our findings on a model architecture with Multi-Head Attention. In Table 7, we observe a similar trend to that of Llama-3-8B-Instruct in Section 2. Unstructured pruning outperforms structured pruning of ThinK [44], with ouput-awareness bringing a small accuracy increase to pure magnitude-based pruning.

Table 7: Comparison of ThinK [44] structured pruning, per-token magnitude-based unstructured pruning, and per-token output-aware unstructured pruning on LongBench [3] with Llama-2-7B Key cache.

Task	**Dense**	$K_s = 0.5$			$K_s = 0.7$		
		ThinK (Structured)	Unstructured Output-aware	Unstructured Magnitude	ThinK (Structured)	Unstructured Output-aware	Unstructured Magnitude
Average	27.51	25.70	**27.55**	27.46	21.57	**26.78**	26.17
SingleDoc QA	15.53	16.28	**15.52**	15.37	14.17	**15.82**	14.43
MultiDoc QA	7.10	6.30	6.90	**7.23**	4.33	6.44	**6.62**
Summarization	14.02	7.46	**14.51**	13.91	9.28	**12.99**	11.64
Few-shot	65.13	64.34	**65.20**	65.00	57.81	**63.77**	63.65
Synthetic	4.17	2.42	**3.98**	3.90	4.35	3.00	**3.08**
Code	63.24	61.57	63.22	**63.52**	39.85	**62.67**	61.73

In Table 8, a unique phenomenon is the stark contrast of model accuracy in per-channel unstructured pruning methods. Whereas per-channel magnitude-based pruning of Table 2 show good model accuracy retention for Llama-3-8B-Instruct, for Llama-2-7B we see that accuracy degradation is very severe. Nevertheless, concurrent to our previous finding, we once again see that per-channel pruning achieves the same level of accuracy retention to per-token pruning as output-awareness is applied. This highlights the importance of output-awareness in Value cache pruning. In Table 9 we see that the model accuracy of 70% unstructured sparsity on both Key and Value cache achieves similar accuracy to 50% ThinK pruning.

Table 8: Comparison of ThinK [44] structured pruning, per-channel magnitude-based unstructured pruning, per-channel output-aware unstructured pruning, and per-token magnitude-based pruning on LongBench [3] with Llama-2-7B Value cache.

Task	**Dense**	$V_s = 0.5$				$V_s = 0.7$			
		ThinK (Structured)	Magnitude (Per-channel)	Output-aware (Per-channel)	Magnitude (Per-token)	ThinK (Structured)	Magnitude (Per-channel)	Output-aware (Per-channel)	Magnitude (Per-token)
Average	27.51	24.59	6.16	27.33	**27.39**	21.10	5.81	26.30	**27.05**
SingleDoc QA	15.53	12.64	1.68	**15.96**	15.62	10.05	1.60	**15.48**	15.17
MultiDoc QA	7.10	7.37	2.17	**6.97**	6.92	7.15	1.82	6.97	**7.10**
Summarization	14.02	9.18	4.51	**13.98**	13.54	9.10	3.15	**13.06**	12.67
Few-shot	65.13	61.82	9.93	64.07	**64.92**	57.12	8.83	60.09	**64.80**
Synthetic	4.17	3.86	1.82	**4.45**	4.20	1.65	2.45	**4.69**	3.95
Code	63.24	56.31	20.03	62.72	**63.44**	41.96	20.90	62.34	**62.85**

Table 9: Longbench evaluation of Llama-2 7B with KV cache per-token magnitude-based pruning

Task	Llama-2-7B		
	Dense	$K_s = 0.5$ $V_s = 0.5$	$K_s = 0.7$ $V_s = 0.7$
Average	27.51	27.23	24.71
SingleDoc QA	15.53	15.21	13.62
MultiDoc QA	7.10	7.11	6.78
Summarization	14.02	13.61	6.84
Few-shot	65.13	64.84	62.59
Synthetic	4.17	3.55	2.63
Code	63.24	63.16	60.35

A.2 Scaling to Larger Model

In Table 10, we include the accuracy evaluation of Mustafar per-token magnitude-based pruning on Llama-2-13B-chat [40], validating the effectiveness of Mustafar on model with larger size. While unstructured pruning constantly outperforms structured sparsity, we see that the Key cache of Llama-2-13B-chat is more susceptible to accuracy degradation at 70% sparsity (yellow). In this case, we leverage the modularity of Mustafar, being able to apply different target sparsity to Key and Value cache to find the best combination, to use 50% sparsity for Key Cache and 70% sparsity for Value cache (pink), thereby reaching the higher overall sparsity while maintaining the model accuracy.

Table 10: Mustafar accuracy with Llama-2-13B-chat on LongBench

KV Sparsity	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	Gov-Report	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RBP	
Llama-2-13B-Chat																	
Dense	18.54	24.09	37.01	36.43	31.40	15.81	24.48	20.25	25.74	67.50	86.90	42.07	3.00	12.00	50.12	50.53	**34.12**
ThinK0.5	16.95	22.39	37.54	34.00	29.93	14.33	24.49	20.21	24.78	67.50	87.16	40.53	2.55	13.07	45.79	46.23	**32.80**
K0.5 V0.0	18.46	23.12	37.26	37.16	31.18	15.56	23.90	20.55	25.57	67.50	87.23	41.99	3.00	11.50	50.33	48.88	**33.95**
ThinK0.7	17.86	19.93	32.37	33.03	27.22	13.99	21.19	19.47	12.04	59.0	86.67	31.26	1.54	1.87	27.79	29.35	**27.16**
K0.7 V0.0	14.63	20.97	34.05	34.70	30.69	13.72	10.60	20.01	7.63	61.00	81.91	37.76	1.00	1.00	45.29	33.54	**28.03**
K0.0 V0.5	18.75	23.68	37.34	36.83	31.36	15.50	23.97	20.83	25.46	67.50	87.20	41.45	2.50	10.00	49.32	49.37	**33.82**
K0.0 V0.7	19.29	22.90	37.65	36.57	31.24	15.35	22.44	20.52	24.75	68.00	87.49	40.55	2.50	8.10	49.33	49.14	**33.49**
K0.5 V0.5	19.08	22.66	36.97	37.25	31.38	15.46	23.70	20.66	25.39	67.50	87.23	40.59	3.00	10.10	49.39	48.06	**33.64**
K0.5 V0.7	18.60	22.57	37.18	35.40	31.55	15.25	22.30	20.43	24.81	68.00	87.23	39.91	2.50	7.70	49.02	47.38	**33.24**
K0.7 V0.7	17.86	19.93	32.37	33.03	27.22	13.99	21.19	19.47	12.04	59.00	86.67	31.26	1.54	1.87	27.79	29.35	**27.16**

A.3 Evaluation on RULER

For a more diverse evaluation, we evaluate Llama-3.1-8B-Instruct on RULER [17] benchmark for context length of 65,536 tokens.

Table 11: Accuracy comparison on RULER benchmark

Sparsity	Method	Needle-Single1	Needle-Single2	Needle-MultiKey1	Needle-MultiKey2	Needle-MultiQuery	Needle-MultiValue	QA-1	QA-2	Variable Tracking	Freq. Words Extract
Llama-3.1-8B-Instruct											
Dense	—	1.000	1.000	0.990	0.979	0.990	0.979	0.844	0.594	0.973	0.851
Key 50%	ThinK	1.000	1.000	0.990	0.979	0.995	0.969	0.833	0.594	0.919	0.854
	Mustafar	1.000	1.000	0.990	0.979	0.995	0.996	0.833	0.573	0.971	0.813
Key 70%	ThinK	0.448	0.490	0.229	0.188	0.646	0.487	0.615	0.510	0.208	0.427
	Mustafar	1.000	1.000	0.990	0.969	0.992	0.903	0.833	0.594	0.966	0.823
Value 50%	ThinK	1.000	1.000	0.990	0.969	0.914	0.958	0.823	0.573	0.910	0.792
	Mustafar	1.000	1.000	0.979	0.995	0.995	0.971	0.833	0.604	0.983	0.830
Value 70%	ThinK	0.948	0.927	0.948	0.510	0.698	0.688	0.646	0.500	0.558	0.677
	Mustafar	1.000	1.000	1.000	0.979	0.992	0.969	0.833	0.594	0.985	0.826
Key&Value 50%	ThinK	0.958	1.000	0.948	0.854	0.828	0.956	0.740	0.531	0.742	0.823
	Mustafar	1.000	1.000	0.990	0.979	0.997	0.997	0.833	0.573	0.862	0.809
Key&Value 70%	ThinK	0.000	0.073	0.000	0.000	0.000	0.000	0.219	0.250	0.000	0.035
	Mustafar	1.000	1.000	0.990	0.969	0.995	0.914	0.833	0.583	0.869	0.799

As shown in Table 11, even in the challenging Needle-in-a-Haystack scenarios with multiple keys and queries, Mustafar maintains accuracy comparable to the dense model. It also outperforms the structured pruning baseline ThinK, with particularly notable gains at 70% joint Key-Value sparsity. While structured pruning does perform well in isolated cases, such as the Needle-Single tasks for 70% Value sparsity, it exhibits significant accuracy drops in other tasks. In contrast, Mustafar's unstructured sparsity consistently preserves accuracy across all tasks. This contrast highlights the versatility of unstructured sparsity in adapting to diverse task requirements.

A.4 Higher Sparsity

While the main paper primarily focused on 50% and 70% sparsity of both Key and Value Cache, we present the performance of Mustafar per-token magnitude-based pruning of KV cache 80% and 90% sparsity in Table 12. While we see that Key cache suffers from accuracy degradation in higher sparsity, Value cache, despite the even distribution of element magnitude as in Figure 2b, retains some level of the model accuracy even at 90% sparsity on selective tasks. Model accuracy is retained for tasks such as 2WikiMultihopQA (pink), while degraded significantly in tasks such as GovReport (yellow).

Table 12: Mustafar accuracy with Llama-3-8B-Instruct on LongBench

KV Sparsity	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RBP	
Llama-3-8B-Instruct																	
Dense	23.39	43.38	43.22	46.39	38.66	23.22	29.91	22.56	27.77	74.5	90.28	42.11	4.50	70.00	57.11	54.05	**43.19**
K0.8 V0.0	22.67	39.08	39.44	44.98	38.51	21.94	21.75	21.00	23.88	69.00	90.24	36.92	7.50	64.50	49.15	45.79	**39.77**
K0.9 V0.0	19.90	28.92	35.21	41.56	30.77	18.89	11.78	18.40	14.95	39.50	81.79	29.18	2.75	61.50	40.30	33.46	**31.80**
K0.0 V0.8	24.48	42.54	43.96	45.48	38.71	22.46	24.47	21.64	25.09	73.00	90.11	39.03	5.62	64.00	56.39	56.54	**42.22**
K0.0 V0.9	24.12	37.90	42.53	44.68	38.29	21.99	20.22	21.29	21.61	69.00	90.15	36.04	3.29	62.50	55.87	53.59	**40.19**
K0.8, V0.8	21.82	36.53	38.61	44.38	36.31	21.33	19.18	20.74	20.80	59.50	88.27	32.68	5.25	64.00	51.03	48.29	**38.05**
K0.9, V0.9	17.47	24.13	30.64	38.63	29.24	17.24	13.50	19.67	15.03	35.50	75.29	27.39	5.50	63.00	41.77	34.39	**30.52**

B Comparison with Semi-structured Sparsity

Between the structured pruning of rows and columns, and unstructured pruning of element, lies the 2:4 semi-structured sparsity where 2 out of 4 consecutive elements are non-zero, enforcing a global 50% sparsity. Supported by NVIDIA Sparse Tensor Cores, 2:4 semi-structured sparsity also pursue the same objectives of Mustafar bitmap-based sparse format (Figure 5a), maximal compression and fast computation. In Table 13, we apply 2:4 semi-structured pruning to the per-token magnitude-based scheme. Comparing semi-structured sparsity to Key, Value, and both Key and Value cache to unstructured sparsity of Mustafar, we see that unstructured sparsity constantly outperforms semi-structured pattern of the same sparsity. This emphasizes the impact of fine-grained unstructured sparsity of element-wise pruning in model accuracy retention.

Table 13: Comparison of 2:4 semi-structured and unstructured sparsity with Llama-3-8B-Instruct on LongBench

KV Sparsity	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RBP	
Llama-3-8B-Instruct																	
Dense	23.39	43.38	43.22	46.39	38.66	23.22	29.91	22.56	27.77	74.5	90.28	42.11	4.50	70.00	57.11	54.05	**43.19**
K0.5 (2:4)	21.79	39.77	42.34	45.15	38.81	21.72	24.34	22.21	25.44	69.50	90.87	39.10	7.00	62.50	54.33	50.29	**40.95**
K0.5 (Unstructured)	23.40	43.68	43.63	46.00	38.60	22.72	29.39	22.33	27.64	74.50	90.66	41.09	5.00	68.50	55.89	52.39	**42.84**
V0.5 (2:4)	23.69	42.72	43.94	45.48	39.42	22.78	28.51	22.53	26.66	73.50	90.31	40.92	4.50	68.00	58.35	55.68	**42.94**
V0.5 (Unstructured)	23.80	43.14	43.32	46.28	39.42	22.97	29.18	22.70	27.13	74.50	90.50	41.74	5.00	67.50	57.23	54.30	**43.04**
K0.5(2:4) V0.5(2:4)	22.32	39.42	42.64	45.45	38.25	21.52	23.41	21.82	24.38	69	91.04	39.59	7.5	62.5	55.02	50.41	**40.89**
K0.5 V0.5 (Unstructured)	23.40	46.63	42.98	46.28	39.27	23.13	28.29	22.78	27.07	74.00	90.58	39.97	5.00	67.00	55.54	53.46	**42.65**

C Sparse Attention Kernel Details

As a supplement to Section 3, we offer more detail onto the Mustafar sparse attention kernel, which accelerates memory-bound batch SpMV.

C.1 Load-as-Compressed, Compute-as-Dense Pipeline

Crucial insight of accelerating SpMV involves reducing the data movement between the GPU global memory and the local memory of each GPU Streaming Multiprocessor. First proposed by FlashLLM [43], load-as-compressed, compute-as-dense pipeline as shown in Figure 8 involves sending each matrix tile in the corresponding compressed form to the SM registers ('gmem2reg' in the figure), decompressing the compressed tile into the dense from to the shared memory ('extract'), then initializing computation on the next pipeline stage ('smem2tc'). Computation is mapped to tensor core to utilize the high fp16 compute throughput. To map MV, unused N dimensions are padded to zero for computation. Non-zero thread-tile of 1×64 in Figure 5a represents the granularity of non-zeros that a warp thread decompresses at a pipeline stage. Each warp thread decompresses 2 thread-tile per stage using the corresponding bitmap to determine the correct position of each non-zero. Effectively, each warp operates on a 64×64 sized matrix tile at a time.

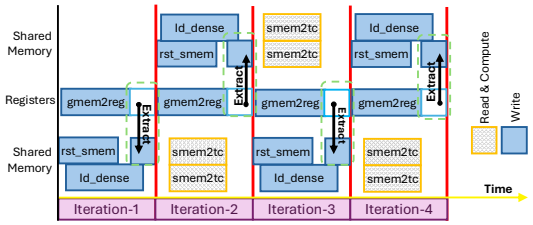
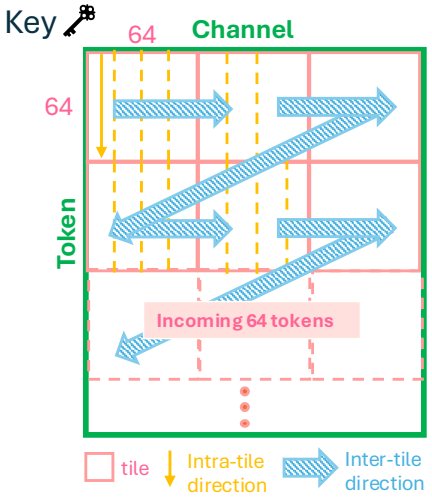


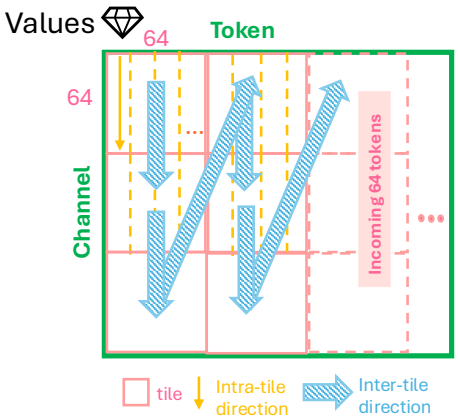
Figure 8: Load-as-compressed, compute-as-compute pipeline of FlashLLM [43]

C.2 KV Cache Management

Tile size of 64×64 of each warp-tile (pink tiles in Figure 9), requires the KV cache to be compressed and appended to the existing KV cache in token groups of 64. Due to the dynamic nature of KV cache where new entries are added during generation, a kernel-compatible management of KV cache update is necessary. That is, (1) column tiling direction of KV cache must be orthogonal to the dimension that is being multiplied with: Key cache is multiplied on the channel-dimension, thus column tiling is across token dimension (yellow arrow in Figure 9a), value cache is multiplied on the token-dimension, thus column-tiling is across the channel dimension (yellow arrow in Figure 9b).



(a) Tile Ordering of Key Cache



(b) Tile Ordering of Value Cache

Figure 9: Tile ordering scheme of Key and Value cache

(2), the layout of warp-tile must ensure that newly compressed tokens' KV cache can be appended to the existing compressed KV cache. As newly compressed KV cache are added onto the token

dimension, traversal across multiple warp-tiles is done along channel-major dimension for both Key and and Value caches so that the compressed KV cache of the new tokens can be appended at the end.

C.3 Decode Speed Evaluation

Extrapolating on Figure 7, we evaluate Mustafar decoding on various input:output token ratios with batch size 4. For Llama-2-7B, we use input sequence length of 2048. For Llama-3-8B-Instruct, we use input sequence length of 4096. We use output sequence lengths of 512, 1024, and 2048.

Table 14: Decode speed comparison with dense inference

Model	KV Format	TTFT	Decode Speed (decode 512)	Decode Speed (decode 1024)	Decode Speed (decode 2048)
Llama2	Dense	1.396 sec	88.685 tokens / sec	88.512 tokens / sec	79.185 tokens / sec
	Mustafar K0.5 V0.5	2.532 sec	89.452 tokens / sec	89.514 tokens / sec	85.687 tokens / sec
	Mustafar K0.7 V0.7	2.249 sec	96.386 tokens / sec	97.436 tokens / sec	95.120 tokens / sec
Llama3	Dense	2.769 sec	61.993 tokens / sec	61.220 tokens / sec	59.242 tokens / sec
	Mustafar K0.5 V0.5	3.269 sec	78.434 tokens / sec	83.768 tokens / sec	83.303 tokens / sec
	Mustafar K0.7 V0.7	3.151 sec	84.065 tokens / sec	88.293 tokens / sec	89.699 tokens / sec

While Figure 7 measured the token throughput by considering both input and output tokens processed, in Table 14 we derived the average decoding speed by measuring the end-to-end duration, and dividing it to the number of tokens generated to penalize Mustafar with the overhead of KV cache pruning and compression in both prefill and decode stages.

While time-to-first-token is delayed due to the overhead of pruning and compressing the KV cache during the prefill stage, the delay is offset by the accelerated attention computation during decoding, resulting in higher overall token generation throughput. Notably, Llama-3 exhibits a larger performance gain compared to Llama-2, as its GQA architecture reduces the overhead of KV cache pruning and compression.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We give thorough evaluation throughout the paper to support our claims.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We give the accuracy scores of each method, as well as the extent of the kernel's support and delay in first-time-to-token.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We justify all theoretical claims with real measurements.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer:[Yes]

   Justification: Yes, and the source code is available on Github.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Source code is available on Github to reproduce results.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify the benchmarks, models, and platform used.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We average our measurements across multiple iterations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide information of our compute platform.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We promote the democratization LLMs with KV cache compression.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to

generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work promotes efficient usage of existing models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All code, data, and model has been properly credited and referenced.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: Source code of our sparse attention kernel is provided in the supplementary material as well as our project Github.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs were not used in core methods of this research.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.