PiKV: KV Cache Management System for MoE Architecture

Dong Liu [1 2 3] **Yanxuan Yu** [4] **Ben Lengerich** [5] **Ying Nian Wu** [3] **Xuhong Wang*** [2]

Abstract

As large-scale language models continue to scale up in both size and context length, the memory and communication cost of key-value (KV) cache storage has become a major bottleneck in multi-GPU and multi-node inference. While MoE-based architectures sparsify computation across experts, the corresponding KV caches remain dense and globally synchronized, resulting in significant overhead.

We introduce **PiKV**, a parallel and distributed KV cache serving framework tailored for MoE architecture. PiKV leverages *expert-sharded KV storage* to partition caches across GPUs, *PiKV routing* to reduce token-to-KV access, and a *PiKV Scheduling* to adaptively retain query-relevant entries. To further reduce memory usage, PiKV integrates *PiKV Compression* modules the caching pipeline for acceleration.

PiKV is recently publicly available as an open-source software library: https://github.com/NoakLiu/PiKV. Experiments details is recorded at: Experimental Results. We also have PiKV integrated with Nvidia kvpress for acceleration, details see PiKVpress. PiKV is still a living project, aiming to become a comprehesive KV Cache management system for MoE Architectures.

1. Introduction

Large Language Models (LLMs) have become the foundation of modern AI applications, powering virtual assistants, code generation, document analysis, and multi-turn reasoning. With increasing demand for longer sequences and sparse expert models (Bai et al., 2025; Rajbhandari et al., 2020; Achiam et al., 2023), there is huge demand to deploy sparsely-gated Mixture-of-Experts (MoE) structures (Lepikhin et al., 2020; Du et al., 2022) to reduce computation costs at scale.

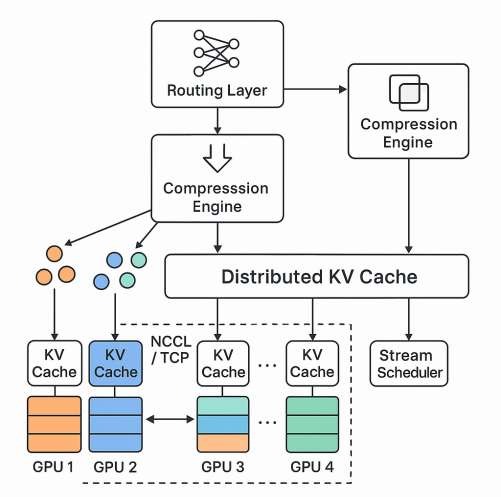

Figure 1. PiKV Framework

However, serving such models introduces significant system-level challenges. During inference, each token generation requires attending to the entire KV cache from prior tokens. For a 7B-scale MoE model with 128K context and 16 experts, the full KV cache can occupy >24GB of memory and incur excessive communication latency across GPUs and nodes. Even with FlashAttention-style optimizations (Dao et al., 2022), the need to load and attend to dense KV structures becomes the dominant bottleneck, especially in autoregressive decoding.

Prior works (Zhang et al., 2023; Gao et al., 2024) have shown that a small fraction of tokens contribute disproportionately to the final attention output, motivating selective cache access. Yet most methods either use static heuristics or ignore the underlying system cost of accessing KV entries across distributed compute nodes. In this work, we ask a deeper question: *can we design a KV caching system that is both sparsity-aware and system-optimized for distributed MoE inference?*

We propose **PiKV**, a parallel distributed KV caching system tailored for sparse mixture of expert models training

[1]Yale University [2]Shanghai AI Lab [3]University of California - Los Angeles [4]Columbia University [5]University of Wisconsin-Madison. Correspondence to: Xuhong Wang <wangxuhong@pjlab.org.cn>.

Work present at ICML 2025 ES-FoMo III.

and inference. As shown in 1, PiKV includes three synergistic components: (1) an *expert-sharded distributed KV cache* layout across multi-GPU or multi-node compute, (2) a *sparse expert routing layer* that dynamically selects top-k experts per query, and (3) an *adaptive stream scheduler* that uses activity-based eviction to retain only high-utility KV entries.

To further reduce memory and bandwidth cost, PiKV compresses KV representations using modular schemes such as LoRA (Hu et al., 2022), PyramidKV (Cai et al., 2024), and Duo (Chen et al., 2024). We track metadata and usage patterns of each KV shard to guide eviction and cache streaming policies, enabling efficient inference under both static and streaming contexts.

- We present a novel system architecture that combines sparse expert routing and distributed KV cache layout with query-aware streaming scheduling.
- We propose compression-aware KV caching, integrating multiple compression schemes and eviction policies into a unified system-level framework.
- We validate efficiency of PiKV in KV Cache Management of MoE Architectures, achieving significant improvements in memory, latency, and end-to-end generation efficiency.

2. Related Work

2.1. Long-context LLMs

With the rapid growth in model size and sequence length, long-context LLMs have gained significant attention. Architectures such as Transformer-XL (Dai et al., 2019), Longformer (Beltagy et al., 2020), and BigBird (Zaheer et al., 2020) explored architectural sparsity to improve scalability. In practice, most LLMs adopt Rotary Position Embedding (RoPE) (Su et al., 2024), which has been extended to longer contexts through rescaling (Chen et al.). Notably, Yarn-LLaMA (Peng et al., 2023) expanded the 4K-token LLaMA-2 model to 32K and 128K tokens, respectively. Context length scaling techniques have pushed limits beyond 1M tokens (An et al., 2024). Serving these long-context models introduces substantial KV cache pressure and decoding latency. PiKV addresses this by combining distributed KV cache design with token-level adaptive routing and scheduling.

2.2. Sparse Expert Routing and KV Lookup

MoE-based models (Lepikhin et al., 2020; Fedus et al., 2022; Du et al., 2022) reduce compute cost by activating a small subset of experts per token. However, inference-time memory and KV access patterns remain dense in most implementations. DeepSeek-V2 (Liu et al., 2024) and M6-MoE (Zheng et al., 2023) explored dynamic expert selection and sparsity-aware routing. PiOne (Lu et al., 2024) and RingAttention (Liu et al., 2023) explored multi-GPU KV alignment but did not optimize for token-expert selectivity. Our work differs by tightly coupling token-level sparse routing with expert-sharded KV cache layouts and a lightweight backend gating network, reducing lookup and communication cost simultaneously.

2.3. KV Compression and Stream-aware Scheduling

To alleviate memory and bandwidth bottlenecks, several methods have been proposed to compress and truncate KV caches. H2O (Zhang et al., 2023) prioritizes tokens with high cumulative attention weights. FastGen (Sheng et al., 2023) selects KV entries based on token type sensitivity. StreamingLLM (Xiao et al., 2024) introduce streaming sinks and proxy scoring methods to handle infinite-length texts. Quest (Tang et al., 2024) and TOVA (He et al., 2025) further propose query-aware KV selection by evaluating relevance to the current query.

While effective, most methods either discard unused tokens too early or require full cache for scoring. In contrast, PiKV retains a unified KV cache layout with on-demand query-aware page loading, hierarchical compression (e.g., LoRA (Hu et al., 2022), PyramidKV (Cai et al., 2024)), and streaming-aware eviction based on token activity scores. This provides a practical tradeoff between throughput, memory, and attention fidelity.

3. Methodology

The PiKV system is designed to rethink Key–Value (KV) cache management as a query-driven, memory–latency optimized process, tailored for sparse MoE inference at scale. In contrast to conventional cache systems that statically retain all past tokens, PiKV makes two fundamental shifts:

- **Sparsity-aware serving**: Only a small set of experts and KV pages are relevant per query;
- **Resource-constrained scheduling**: The memory and bandwidth budget must be dynamically partitioned across queries, experts, and streams.

To this end, we decompose PiKV into four co-designed modules: *(i) distributed expert-sharded KV storage*, *(ii) adaptive routing (PiKVRouting)*, *(iii) modular compression (PiKVCompression)*, and *(iv) query-aware stream scheduling (PiKVScheduling)*.

All components are executed in an asynchronous pipeline orchestrated by a general decoding loop, as shown in Algorithm 1. Each submodule operates independently but passes metadata to adjacent stages to inform decisions.

We now describe each module and its underlying theoretical and system-level formulation.

Algorithm 1 General PiKV Execution Framework

Input: query stream $\{q_t\}_{t=1}^{T}$, expert set $\mathcal{E}$, shard size S
Initialize: distributed cache $\mathcal{C}$, routing policy $\mathcal{R}$, scheduler $\mathcal{S}$, compressor $\mathcal{C}_{\text{cmp}}$
for $t = 1$ to T **do**
  $g_t \leftarrow \mathcal{R}(q_t)$ // PiKV Routing
  $K_t, V_t \leftarrow f_{\text{enc}}(q_t)$
  for expert $e \in g_t$ **do**
    $s \leftarrow \text{Shard}(t, e)$
    $(\hat{K}, \hat{V}) \leftarrow \mathcal{C}_{\text{cmp}}(K_t, V_t)$ // PiKV Compression
    $\mathcal{C}[e][s] \leftarrow \text{Insert}((\hat{K}, \hat{V}), \text{metadata})$
  end for
  $\mathcal{C} \leftarrow \mathcal{S}(\mathcal{C}, q_t)$ // PiKV Scheduling
  $y_t \leftarrow f_{\text{attn}}(q_t, \mathcal{C}[g_t])$
end for

3.1. PiKV Expert-Sharded Storage

Given a KV tensor pair $(K_t, V_t) \in \mathbb{R}^{d\times 2}$ at time t, the goal is to store these vectors in a distributed cache that minimizes redundancy and maximizes parallel retrieval. Unlike traditional schemes that replicate the full KV across G GPUs, we assign tokens to shards via a hash function $h(t, e)$ and assign each shard to one GPU:

$$s(t, e) = (t \bmod N_{\text{tok}}) \oplus (e \bmod N_{\text{exp}}).$$

Each GPU stores only $\mathcal{O}(L/G + L/E)$ tokens, reducing per-device memory cost from $\mathcal{O}(EL)$.

Storage invariants. Each shard s maintains a circular buffer of capacity S, so that insertions cost $\mathcal{O}(1)$ time and reallocation is avoided. If (K_t, V_t) is compressed to $(\hat{K}_t, \hat{V}_t)$ of dimension d', the per-shard memory is:

$$\mathcal{M}_s = 2d'S = \frac{2dS}{\rho}, \quad \text{with } \rho = d/d'.$$

Total memory per GPU is then:

$$\mathcal{M}_{\text{kv}} = \frac{2d}{\rho}\left(\frac{L}{GS} + KS\right),$$

where K is the number of retained pages in PiKV scheduling.

3.2. PiKV Routing

PiKV Routing decides which experts $g_t \subseteq \mathcal{E}$ to activate for each query q_t. Formally, we define a routing function $\mathcal{R} : \mathbb{R}^d \to \{0, 1\}^E$ satisfying $\|g_t\|_0 = k$. PiKV supports multiple routing methods as in the following table 1.

ID	Mechanism	Penalty Term	Cost
$\mathcal{R}_{\mathbf{B}}$	Base hash / round-robin	—	$\mathcal{O}(1)$
$\mathcal{R}_{\mathbf{T}}$	TopK softmax	—	$\mathcal{O}(E \log k)$
$\mathcal{R}_{\mathbf{LB}}$	TopK + load balance	$-\alpha(\mu_e - \bar{\mu})$	$\mathcal{O}(E)$
$\mathcal{R}_{\mathbf{P}}$	Cache-aware (PiKVRouter)	$-\lambda \log(1 + \text{miss}_e)$	$\mathcal{O}(E)$
$\mathcal{R}_{\mathbf{E}}$	Entropy-penalised LB (EPLB)	$-\beta\, H(p_e)$	$\mathcal{O}(E)$
$\mathcal{R}_{\mathbf{A}}$	RL-adaptive gating	learned	$\mathcal{O}(k^2)$
$\mathcal{R}_{\mathbf{H}}$	Hierarchical coarse→fine	two-stage TopK	$\mathcal{O}(E + k \log k)$

Table 1. PiKV routing methods and their computational profiles (E = experts).

Attention Complexity Reduction of PiKV Routing. Let B be batch size, L sequence length, h head width, E experts, $k \ll E$ routed per token.

$$\boxed{C_{\text{dense}} = B\,L\,h\,E}, \qquad \boxed{C_{\text{sparse}} = B\,L\,h\,k}, \quad \Longrightarrow \quad \text{speedup} = \frac{E}{k}.$$

Memory traffic (Key–Value fetch, bytes):

$$M_{\text{dense}} = 2B\,L\,h\,E, \qquad M_{\text{sparse}} = 2B\,L\,h\,k, \quad \text{relief} = \frac{E}{k}.$$

Reuse-distance:

$$\text{RD}_{\text{dense}} = \frac{L}{E}, \qquad \text{RD}_{\text{sparse}} = \frac{L}{k} \implies \text{hit-rate} \approx \frac{k}{E}.$$

3.3. PiKV Compression

PiKV compression controls the space–fidelity trade-off for KV storage. Given $(K, V) \in \mathbb{R}^d \times \mathbb{R}^d$, a compressor $\mathcal{C}$ maps:

$$\mathcal{C}(K, V) = (\hat{K}, \hat{V}) \in \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \quad d' < d.$$

We define the reconstruction error as:

$$\epsilon = \frac{\|K - \mathcal{D}(\hat{K})\|_2}{\|K\|_2}, \quad \text{with decoder } \mathcal{D}.$$

PiKV supports multiple compression methods as in the following table 2.

Compression-Aware Latency of PiKV Compression. *Variables:* d full width, $d' = d/\rho$ compressed width ($\rho > 1$), k experts/query, B tokens/batch, β HBM bandwidth (B/s), γ core throughput (B/s), $\eta \leq 2$ decode factor.

[1]Full SVD is offline; at inference only the $O(dr)$ projection is executed.

ID	Mechanism	ϵ^2 (squared error bound)	Cost (Big-O)
C_{Lo}	LoRA (rank r)	$\sum_{i=r+1}^{d} \sigma_i^2$	$\mathcal{O}(dr)$
$C_{\text{Lo}+}$	LoRA^{++}	$\|K - W_d W_u K - b\|_2^2$	$\mathcal{O}(dr)$
C_{Py}	PyramidKV (L levels)	$\sum_{\ell=0}^{L-1} \frac{\|P^{(\ell)} K - K\|_2^2}{4^\ell}$	$\mathcal{O}(d)$
C_{Ch}	ChunkKV (block PCA)	$\sum_{\text{blk}} \sum_{i>r} \sigma_i^2$	$\mathcal{O}(dr)$
C_{SVD}	Truncated SVD (r)	$\sum_{i>r} \sigma_i^2$	$\mathcal{O}(d^2 r)$[1]
C_{F}	FastV (crop to r)	$\|K_{r:d}\|_2^2$	$\mathcal{O}(d)$
C_{Dis}	Distillation (offline)	$\text{KL}(q_{\text{teach}} \,\|\, q_{\text{stud}})$	$\mathcal{O}(d\,r)$
C_{Pr}	Structured Pruning	$\sum_{j \in \mathcal{Z}} K_j^2$	$\mathcal{O}(d)$

Table 2. Analytic reconstruction bounds and asymptotic compression cost (d = width, $r \ll d$ retained rank).

$$T_{\text{read}} = \frac{2d'kB}{\beta} = \frac{2dkB}{\rho\beta}, \tag{1}$$

$$T_{\text{decode}} = \frac{\eta d'kB}{\gamma} = \frac{\eta dkB}{\rho\gamma}, \tag{2}$$

$$T_{\text{step}} = T_{\text{read}} + T_{\text{decode}} = \frac{dkB}{\rho}\Big(\tfrac{2}{\beta} + \tfrac{\eta}{\gamma}\Big). \tag{3}$$

Speed-up. For two compression ratios $\rho_1 < \rho_2$,

$$\text{Speedup}(\rho_1 \to \rho_2) = \frac{T_{\text{step}}(\rho_1)}{T_{\text{step}}(\rho_2)} = \frac{\rho_2}{\rho_1}. \tag{4}$$

Higher ρ linearly reduces both read and decode time until $T_{\text{decode}} \approx T_{\text{read}}$, after which the gain plateaus.

3.4. PiKV Scheduling

PiKV Scheduler implements dynamic retention of cached KV pages under bounded memory. Instead of static eviction rules, PiKV formulates scheduling as a per-page scoring problem, where each entry i is assigned a scalar utility score u_i based on features such as attention intensity, recency of access, and reuse patterns. PiKV supports multiple scheduling methods as in following table 3

Memory Usage of PiKV. We analyze the total per-GPU memory consumption $\mathcal{M}_{\text{total}}$ of PiKV under compressed KV storage and bounded scheduling. Let:

- d: original hidden size of each KV vector;
- $\rho = d/d'$: compression ratio, where d' is the reduced dimensionality;
- L: number of cached tokens per expert globally;
- G: number of GPUs (i.e., KV shards);

ID	Scheduling Methods u_i	Adaptive
$\mathcal{S}_{\text{H2O}}$	$u_i = a_i$	$\times$
$\mathcal{S}_{\text{SL}}$	$u_i = \mathbb{I}[t_i > \tau]$	$\times$
$\mathcal{S}_{\text{QUEST}}$	$u_i = \text{MLP}_\theta\big([K_i, V_i]\big)$	✓
$\mathcal{S}_{\text{Flex}}$	$u_i = \mathcal{M}_{\text{plan}}(t_i)$	$\times$
$\mathcal{S}_{\text{LRU}}$	$u_i = -r_i$	$\times$
$\mathcal{S}_{\text{LRU+}}$	$u_i = -r_i + \lambda \cdot f_i$	$\times$
$\mathcal{S}_{\text{AdaKV}}$	$u_i = \sum_j \alpha_j \phi_j(i), \quad \theta \leftarrow \theta + \gamma(\eta^* - \eta)$	✓
$\mathcal{S}_{\text{Duo}}$	$u_i = \sum_{\ell=1}^{L} a_i^{(\ell)}$	✓

Table 3. Summary of PiKV scheduling strategies. Notation: a_i = attention, r_i = recency, f_i = frequency, t_i = age, $\phi_j(i)$ = feature scores, θ = eviction threshold, η = hit-rate. ✓ = adaptive threshold, $\times$ = fixed.

- S: circular buffer size (in tokens) per expert shard;
- K: number of active cache pages selected by the scheduler per GPU.

The total memory per GPU decomposes into two parts:

$$\mathcal{M}_{\text{token}} = \frac{2d'}{G} \cdot \frac{L}{S}, \qquad \text{(sharded token buffer)}$$
$$\mathcal{M}_{\text{page}} = 2d' \cdot K \cdot S, \qquad \text{(scheduled page buffer)}$$

Summing the two and replacing $d' = d/\rho$ yields:

$$\mathcal{M}_{\text{total}} = \mathcal{M}_{\text{token}} + \mathcal{M}_{\text{page}} = \frac{2d}{\rho}\left(\frac{L}{GS} + KS\right).$$

To minimize $\mathcal{M}_{\text{total}}$ with respect to S, we take the derivative:

$$\frac{\partial \mathcal{M}_{\text{total}}}{\partial S} = -\frac{2dL}{\rho G S^2} + \frac{2dK}{\rho}, \quad \text{set } \frac{\partial \mathcal{M}_{\text{total}}}{\partial S} = 0 \Rightarrow S^* = \sqrt{\frac{L}{KG}}.$$

Therefore, the optimal buffer size S^* trades off between sharding granularity and reuse coverage. Substituting back:

$$\mathcal{M}^*_{\text{total}} = \frac{4d}{\rho}\sqrt{\frac{KL}{G}}.$$

This closed-form provides a practical design rule for setting shard capacity S to minimize GPU memory cost under fixed compression ρ, token budget L, and scheduler retention K.

4. Conclusion

We present **PiKV**, a parallel and distributed KV cache management framework optimized for sparsely activated MoE-based large language models. PiKV introduces a KV cache management system for MoE, including sparse expert routing, cache compression, and stream-aware scheduling. This architecture rethinks KV caching not only as passive memory storage, but as a dynamic, query-driven retrieval system.

PiKV is a living project for scalable MoE serving, aiming for briding MoE sparsity and efficient system design optimization. Future work will explore online adaptation, hierarchical memory tiers, and integration with training-time sparsity strategies for end-to-end efficient large model deployment with MoE architecture.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report, 2023.

An, C., Huang, F., Zhang, J., Gong, S., Qiu, X., Zhou, C., and Kong, L. Training-free long-context scaling of large language models. *arXiv preprint arXiv:2402.17463*, 2024.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Cai, Z., Zhang, Y., Gao, B., Liu, Y., Liu, T., Lu, K., Xiong, W., Dong, Y., Chang, B., Hu, J., et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Chen, L., Zhao, H., Liu, T., Bai, S., Lin, J., Zhou, C., and Chang, B. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pp. 19–35. Springer, 2024.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation, 2023. *URL https://arxiv. org/abs/2306.15595*.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., and Zuo, P. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 111–126, 2024.

He, Z., Cao, Y., Qin, Z., Prakriya, N., Sun, Y., and Cong, J. HMT: Hierarchical memory transformer for efficient long context language processing. In Chiruzzo, L., Ritter, A., and Wang, L. (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8068–8089, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. URL https://aclanthology.org/2025.naacl-long.410/.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

Lu, X., Liu, Q., Xu, Y., Zhou, A., Huang, S., Zhang, B., Yan, J., and Li, H. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. *arXiv preprint arXiv:2402.14800*, 2024.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. 2024. URL https://arxiv.org/abs/2309.17453.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Zheng, N., Jiang, H., Zhang, Q., Han, Z., Ma, L., Yang, Y., Yang, F., Zhang, C., Qiu, L., Yang, M., et al. Pit: Optimization of dynamic sparse deep learning models via permutation invariant transformation. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 331–347, 2023.