PolarQuant: Leveraging Polar Transformation for Key Cache Quantization and Decoding Acceleration

Songhao Wu[1*] **Ang Lv**[1*] **Xiao Feng**[2]
Yufei Zhang[2] **Xun Zhang**[2] **Guojun Yin**[2†] **Wei Lin**[2] **Rui Yan**[1†]
[1] Gaoling School of Artificial Intelligence, Renmin University of China
[2] ShanghaiTech University [3] Meituan
{songhaowu, anglv, ruiyan}@ruc.edu.cn
fengxiao2023@shanghaitech.edu.cn
{zhangyufei08, zhangxun12, yinguojun02, linwei31}@meituan.com

Abstract

The increasing demand for long-context generation has made the KV cache in large language models a bottleneck in memory consumption. Quantizing the cache to lower bit widths is an effective way to reduce memory costs; however, previous methods struggle with key cache quantization due to outliers, resulting in suboptimal performance. We propose a novel quantization approach PolarQuant, which provides a new perspective for key cache quantization and efficiently addresses the outlier dilemma. We observe that the distribution of the key states reveals well-structured patterns under polar transformation. Outliers generally appear in only one of the two dimensions, which are rotated together by a specific angle when rotary position embeddings are applied. When represented as two-dimensional vectors, these dimensions exhibit well-organized patterns, with radii and angles smoothly distributed in polar space. This alleviates the channel-wise outliers, making them well-suited for key cache quantization. PolarQuant divides key vectors into groups of two-dimensional sub-vectors, encoding them as the quantized radius and the polar angle, rather than quantizing original key vectors directly. PolarQuant achieves the superior efficiency in KV cache quantization and accelerates the decoding process by turning the query-key inner product into a table lookup, all while maintaining the downstream performance of full-precision models. Our code is available at `https://github.com/ericshwu/PolarQuant`.

1 Introduction

Large language models (LLMs) have achieved remarkable success across a wide range of real-world applications. As these models evolve, the demand for enhanced long-context capabilities also increases, especially in tasks like contextual retrieval in question answering [17] and reasoning chains generation for complex reflection and decision-making [21, 20]. However, a significant challenge in developing long-context LLMs is the rising memory cost associated with increasing context lengths, which hinders both their practical deployment and further research.

The attention mechanism [4] in LLMs[3]is a major source of computational overhead and memory consumption, which increase rapidly with context length. To reduce this cost, Key-Value cache (KV cache) is a common strategy, which stores and reuses keys and values for generation to avoid the

[*]Equal contribution. Work done during Songhao Wu's internship at Meituan.
[†]Corresponding authors: Rui Yan (`ruiyan@ruc.edu.cn`) and Guojun Yin (`yinguojun02@meituan.com`).

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

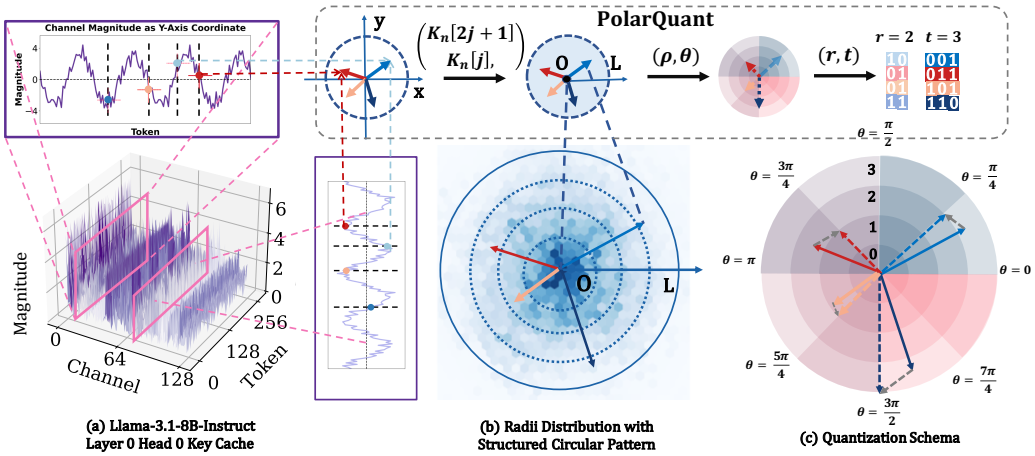

Figure 1: (a) Illustration of the activation distribution for the key cache, exemplified by Llama 3.1-8B-Instruct (Layer 0, Head 0). The key cache exhibits channel-wise outliers, where the magnitudes of a few channels significantly larger than others across tokens. We observe that these outliers generally appear in only one of the two dimensions rotated together by RoPE. (b) When viewing these two dimensions in a two-dimensional plane, although the individual x- or y-axis may contain outliers, they collectively form stable circular patterns, making quantization of the original outliers easier. Each blue dot represents a mapped two-dimensional vector, with transparency indicating frequency. (c) PolarQuant using $r = 2$ bits to quantize radii and $t = 3$ bits to quantize polar angles. The colorful arrows indicate sub-vectors formed by pairs of dimensions in the keys; the quantized results are shown with colorful dashed arrows. The quantization error is represented by the grey dashed arrow.

redundant computation. Nevertheless, as the context length increase, the memory required for KV cache can surpass that of the model weights, making it the dominant factor in overall memory usage.

Many solutions have been proposed to reduce the memory cost of KV cache. Some studies introduce memory-efficient attention modules, such as MQA [25], GQA [2] and MLA [9]. While promising, these module architectures require training LLMs from scratch, which limits their applicability. Another research line focuses on the reduction of KV cache size in a compatible manner with pre-trained LLMs. This includes techniques like KV cache eviction [7, 15, 36], which identifies and drops unimportant tokens from the cache , and KV cache quantization [12–14, 19, 37].

This paper focuses on the key cache quantization, which converts the floating-point key cache into low-bit integers to reduce memory usage. In general, key cache quantization is more challenging than value cache due to the presence of channel-wise distributed outliers. Prior studies [13, 19] have highlighted the widespread existence of such outliers in key cache. As shown in Figure 1(a), the key states exhibit larger activations along certain channel dimensions, making token-wise quantization difficult. To address this issue, KIVI [19] proposes a channel-wise quantization strategy that groups and quantizes key elements along the channel dimensions. Building upon this perspective, KVQuant [13] further identifies RoPE as the primary source responsible of the outliers observed in the key cache. The rotation operations in RoPE disturb the magnitude consistency, making accurate quantization complicated. To mitigate this, KVQuant proposes quantizing the keys before applying RoPE, which is described as pre-RoPE quantization. Promising as it is, this approach requires on-the-fly RoPE computation, which consequently introduces potential computational overhead. In this work, we aim to preserve the benefits of pre-RoPE quantization in reducing approximation errors while eliminating redundant computations at each generation step. We propose a polar transformation perspective on handling outliers in the key cache, and effectively address the dilemma in 2D polar coordinates.

KVQuant [13] observes a clear and structured pattern in pre-RoPE key activations: channel-wise magnitude are highly consistent. Recall that RoPE operates a rotation to every two-dimensional

[3]In this paper, we focus on decoder-only Transformer-based [29] LLMs using rotary position embedding (RoPE, 26), which are the predominant implementation of advanced LLMs.

sub-vector within the key using an orthogonal 2×2 rotary matrices.[4] Since rotation is a magnitude-preserving transformation, these 2D sub-vectors inherit the magnitude characteristics seen in the pre-RoPE case. As shown in Figure 1(b), they form well-structured circular patterns when analyzed in 2D polar coordinates. By encoding each sub-vector as its corresponding radius ρ and polar angle θ, the entire key vector can be represented as a collection of all radii and angles. This transformation effectively mitigates outliers, as both the radii and polar angles become smoothly distributed. Building on this, we propose a novel quantization method, PolarQuant, which significantly simplifies the quantization of the key cache. PolarQuant reduces the problem of quantizing key vectors to asymmetrically quantizing ρ and θ into an r-bit and an t-bit integer. Intuitively, PolarQuant defines 2^{r+t} distinct regions based on 2^r angles and 2^t radii. Each sub-vector is then encoded by the index of the region it belongs to. Figure 1(c) illustrates PolarQuant for $r = 2$ and $t = 3$.

PolarQuant achieves superior quantization effectiveness and efficiency over previous methods. On one hand, polar transformation enables smoother distributions of radii and angles, which alleviates the burden of channel-wise quantization outliers. The superior performance on downstream tasks further demonstrates PolarQuant's superiority in quantization error reduction.

On the other hand, **PolarQuant offers a brand new perspective on key cache quantization, which enables a novel decoding acceleration method.** Unlike pre-RoPE quantization like [13], PolarQuant eliminates the overhead of RoPE recomputation. In the attention mechanism, it replaces the standard query-key multiplication with inner products between two-dimensional query sub-vectors and a quantized polar coordinate representation of key sub-vectors, which have finite and deterministic states. This transforms matrix multiplication to a table lookup, greatly speeding up attention computation. Our contributions are threefold:

(1) We introduce polar transformation for key cache quantization for the first time and derive PolarQuant, a novel and efficient quantization approach;

(2) We propose a new decoding acceleration algorithm as a natural byproduct of PolarQuant. We implement custom Triton kernels to perform fused dequantization and query-key multiplication, which achieves up to 3.18$\times$ speedup in long-context generation;

(3) We conduct comprehensive experiments on tasks and models, which further demonstrate the superiority and robustness of our PolarQuant across a wide range of model families and tasks.

2 Background

Consider a specific Transformer layer where the input hidden states to the attention block are denoted as $\mathbf{X} \in \mathbb{R}^{L \times D}$, where L is the sequence length and D is the hidden state dimension. For any attention head, the d-dimensional query, key, and value vectors are obtained by applying three linear transformations to $\mathbf{X}$. Specifically, for each head h, the corresponding computations are as follows:

$$\tilde{\mathbf{Q}} = \mathbf{X}\,\mathbf{W}_Q, \quad \tilde{\mathbf{K}} = \mathbf{X}\,\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\,\mathbf{W}_V,$$

where each $\mathbf{W}_* \in \mathbb{R}^{D \times d}$, and the resulting variables have shapes of $\mathbb{R}^{L \times d}$.

The query and key vectors are then applied with RoPE [26] to incorporate positional information. For a query or key vector at position $m \in [1, L]$, the corresponding rotary matrix $\boldsymbol{R}_{m,\Phi} \in \mathbb{R}^{d \times d}$ is defined as:

$$\boldsymbol{R}_{m,\Phi} = \begin{bmatrix} \boldsymbol{r}_{m,\phi_1} & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \boldsymbol{r}_{m,\phi_2} & \cdots & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \cdots & \boldsymbol{r}_{m,\phi_{d/2}} \end{bmatrix}, \qquad \boldsymbol{r}_{m,\phi_i} = \begin{bmatrix} \cos(m\phi_i) & -\sin(m\phi_i) \\ \sin(m\phi_i) & \cos(m\phi_i) \end{bmatrix}, \quad (1)$$

where $\mathbf{O}$ is a zero matrix, and each $\boldsymbol{r}_{m,\phi_i}$ is a 2×2 orthogonal matrix. Here, ϕ_i is typically defined as $\phi_i = b^{-2i/d}$, where b is the hyperparameter for RoPE base.

This formulation encodes the relative distance $m - n$ between a query at position $m > n$ and a key at position n into their inner product, as shown by:

$$(\tilde{\mathbf{Q}}_m\,\boldsymbol{R}_{m,\Phi})\,(\tilde{\mathbf{K}}_n\,\boldsymbol{R}_{n,\Phi})^\top = \tilde{\mathbf{Q}}_m\,\boldsymbol{R}_{m-n,\Phi}\,\tilde{\mathbf{K}}_n^\top,$$

[4] For readers unfamiliar with RoPE, please refer to Section 2.

the keys (after applying RoPE) and values, specifically $\tilde{\mathbf{K}}_n \boldsymbol{R}_{n,\Phi}$ (which we abbreviate as $\mathbf{K}_n$) and $\mathbf{V}_n$, are thus cached to avoid re-computation, known as the KV cache. The KV cache leads to increased memory usage when processing long-context inputs.

Quantization is a simple but effective way to reduce the KV cache size. To clarify the mechanism, we provide a brief overview of key-value quantization below. For the value at position n, we follow the token-wise paradigm in [19] and quantize $\mathbf{V}_n \in \mathbb{R}^d$ into b-bit, denoted as $Q(\mathbf{V}_n)$.

For an arbitrary dimension $0 \leq j < d$, we have:

$$Q(\mathbf{V}_n[j]) = \texttt{Clamp}\left(\left\lfloor \frac{\mathbf{V}_n[j] - z_n}{s_n} \right\rceil, 0, 2^b - 1\right),$$

where $z_n = \min(\mathbf{V}_n[:])$ is the zero-point, $s_n = \left(\max(\mathbf{V}_n[:]) - \min(\mathbf{V}_n[:])\right) \,/\, \left(2^b - 1\right)$ is the scaling factor. The colon here denotes iteration over all dimensions, following Python indexing syntax. The function $\texttt{Clamp}(x, min, max)$ restricts x to integers within the range $[min, max]$.

Outliers in key states make per-token quantization challenging, as we discussed earlier and illustrated in Figure 1(a). To address this, previous approaches [19, 13] quantize key vectors channel-wise.

For an arbitrary dimension j, a quantized key $Q(\mathbf{K}_n)$ at position n is given by:

$$Q(\mathbf{K}_n[j]) = \texttt{Clamp}\left(\left\lfloor \frac{\mathbf{K}_n[j] - z_{[:]}[j]}{s_{[:]}[j]} \right\rceil, 0, 2^b - 1\right),$$

where the zero-point and scaling factor alternate as:

$$z_{[:]}[j] = \min(\mathbf{K}_{[:]}[j]), \quad s_j = \frac{\max(\mathbf{K}_{[:]}[j]) - \min(\mathbf{K}_{[:]}[j])}{2^b - 1}.$$

Here, the colon in the subscript denotes iteration over all token positions.

3 Method

We begin by presenting the key findings of the activation patterns in the key cache (Section 3.1). These insights serve as the foundation for our proposed quantization approach, PolarQuant (Section 3.2).

3.1 Motivation

As discussed earlier, outliers in key vectors pose a dilemma for key cache quantization. Our solution to this challenge arises from a key observation:

Observation: *When mapping the paired dimensions with outliers to polar coordinates, the resulting 2D vectors show consistent magnitudes. The outliers present in one channel are compensated by the activations of the other, which significantly simplifies quantization.*

Recall that in key vectors, elements in certain dimensions are jointly rotated by the same rotary sub-matrix $\boldsymbol{r}_{n,\phi_i}$. Our analysis shows that the most prominent outliers tend to occur in only one of these dimension pairs.[5]

Figure 1 (b) maps the paired dimensions from Figure 1 (a) onto a 2D Cartesian coordinate system, where the x-axis represents the first dimension and the y-axis represents the second.

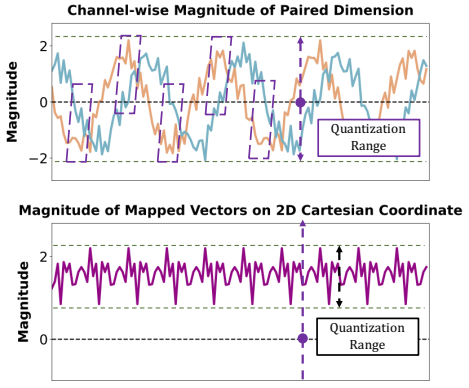

Figure 2: This figure supplements the transformation from Figure 1 (a) to Figure 1 (b), showing how PolarQuant better resolves the outliers.

[5]For efficiency, the rotary matrix is typically applied in an element-wise multiplication manner [26]. To simplify implementation, dimensions i and $i + d/2$ are often rotated together, rather than i and $i + 1$. This results in non-adjacent outliers in Figure 1(a), but it does not affect our analysis, which is based on the matrix multiplication formulation (Eq. 1).

Despite large variations in individual x and y values (which would indicate outliers in isolation), the mapped vectors form a well-structured pattern. In other words, when transformed into polar coordinates, the outliers are characterized by a smoothly distributed radial coordinate r and a polar angle θ. This structure significantly alleviates the quantization challenges faced by key caches.

Figure 2 provides a supplementary illustration of how this polar transformation improves quantization. The top figure illustrates a pair of dimensions exhibiting outliers, each with a large individual value range. Quantizing this large range inevitably results in a loss of precision. In the bottom figure, by combining these dimensions into a 2D vector, the norm (i.e., the polar radius) shows a significantly narrowed value range, facilitating quantization.

3.2 PolarQuant: Polar-coordinate-based quantization of post-RoPE key cache

Building on these insights, we introduce a novel key cache quantization approach based on polar transformation. Since the benefits of adopting a polar-coordinate have been outlined in the previous subsection, here we focus on the implementation details.

For an arbitrary 2-dimensional subvector $\Big(\mathbf{K}_n[2j], \mathbf{K}_n[2j+1]\Big)$ in the key cache at position n, where $0 \leq j < d/2$, we interpret $\Big(\mathbf{K}_n[2j], \mathbf{K}_n[2j+1]\Big)$ as Cartesian coordinates in the xy-plane. This 2D vector is then converted to polar coordinates, where the radius $\rho_n[j]$ is given by:

$$\rho_n[j] = \sqrt{\mathbf{K}_n[2j]^2 + \mathbf{K}_n[2j+1]^2},$$

and the polar angle is:

$$\theta_n[j] = \texttt{atan2}\left(\mathbf{K}_n[2j+1], \mathbf{K}_n[2j]\right) + \pi, \;\; \theta_n[j] \in (0, 2\pi)\,,$$

where $\texttt{atan2}\,(y, x)$ returns the angle between the positive x-axis and the point (x, y).

We perform asymmetric group-wise quantization on $\rho_n[j]$ and $\theta_n[j]$, using a group size of g, with r-bit precision for $\rho_n[j]$ and t-bit precision for $\theta_n[j]$:

$$Q(\rho_n[j]) = \texttt{Clamp}\left(\left\lfloor \frac{\rho_n[j] - z_{\rho[:]}[j]}{s_{\rho[:]}[j]} \right\rfloor, 0, 2^r - 1\right),$$

$$Q(\theta_n[j]) = \texttt{Clamp}\left(\left\lfloor \frac{\theta_n[j] - z_{\rho[:]}[j]}{s_{\theta[:]}[j]} \right\rfloor, 0, 2^t - 1\right),$$

where $z_{\rho[:]}[j]$, $z_{\theta[:]}[j]$ are the zero-points and $s_{\rho[:]}[j]$, $s_{\theta[:]}[j]$ are the scaling factors:

$$s_{\rho[:]}[j] = \frac{\max(\rho_{[:]}[j]) - \min(\rho_{[:]}[j])}{2^r}, \quad z_{\rho[:]}[j] = \frac{\max(\rho_{[:]}[j]) - \min(\rho_{[:]}[j])}{2^r},$$

$$s_{\theta[:]}[j] = \frac{\max(\theta_{[:]}[j]) - \min(\theta_{[:]}[j])}{2^t}, \quad z_{\theta[:]}[j] = \frac{\max(\theta_{[:]}[j]) - \min(\theta_{[:]}[j])}{2^t}.$$

Intuitively, PolarQuant partitions the two-dimensional plane into 2^{r+t} regions, spanned by 2^r radii and 2^t polar angles. Each 2D sub-vector of the key vector is then represented by the center of the region in which it resides. Figure 1 provides an illustration of the quantization process. The corresponding Cartesian coordinates in the key vector at dimensions $2j$ and $2j+1$ are then calculated for the quantized representation $\Big(Q(\rho_n[j]), Q(\theta_n[j])\Big)$, which is formulated as:

$$\left[\widetilde{\mathbf{K}}_n[2j],\; \widetilde{\mathbf{K}}_n[2j+1]\right] = \left[\tilde{\rho}_n[j] \cdot \cos\left(\tilde{\theta}_n[j]\right),\; \tilde{\rho}_n[j] \cdot \sin\left(\tilde{\theta}_n[j]\right)\right],$$

where $\tilde{\rho}_n[j]$ and $\tilde{\theta}_n[j]$ are the dequantized radius and polar angle:

$$\tilde{\rho}_n[j] = \big(Q(\rho_n[j]) + \frac{1}{2}\big) \cdot s_{\rho[:]}[j] + z_{\rho[:]}[j], \quad \tilde{\theta}_n[j] = \big(Q(\theta_n[j]) + \frac{1}{2}\big) \cdot s_{\theta[:]}[j] + z_{\theta[:]}[j].$$

3.3 Efficient decoding with PolarQuant

In this section, we present PolarQuant's design for query-key multiplication, which incorporates the idea of post-multiplication dequantization to achieve acceleration of the decoding process. We begin by reviewing the conventional approach to dequantized generation. Specifically, during the generation phase, the cached keys must be dequantized before being multiplied by the current query Q_m at position m. For each dimension $0 \leq j < d$, we have:

$$\widetilde{\mathbf{K}}_n[j] = Q(\mathbf{K}_n[j]) \cdot s_{[:]}[j] + z_{[:]}[j],$$

where $\widetilde{\mathbf{K}}_n$ denotes the dequantized key, and the inner product is then computed as $\mathbf{Q}_m \cdot \widetilde{\mathbf{K}}_{[:]}$.

The standard dequantization-then-multiplication operation introduces overhead for PolarQuant, as it demands extra computation. We argue that this overhead is redundant. At any dimension j, the dequantized outcomes come from a finite set of size 2^{r+t}. Since the cache size far exceeds this set, it is more efficient to pre-compute and reuse the post-multiplication intermediate results using a lookup table (LUT). This approach of leveraging LUTs has been explored in prior studies [13, 35]. KVQuant adopts an LUT to dequantize the key cache and restores positional information via RoPE recomputation. PolarQuant, in contrast, avoids RoPE recomputation by directly constructing a LUT for QK product on the fly. This is the key insight behind how PolarQuant accelerates decoding.

Specifically, PolarQuant builds its lookup table within each channel by mapping quantized polar coordinates to Cartesian coordinates and computing the dot products with the query sub-vectors. We implement custom Triton kernels to perform fused dequantization and query-key multiplication for efficient GPU execution of PolarQuant. The breakdown time analysis presented in Section 4.2 further demonstrates the effectiveness of our PolarQuant implementation. More implementation details can be found in Appendix A and our released code.

4 Experiment

In this section, we evaluate the performance of PolarQuant to highlight its *effectiveness* and *efficiency*. Our empirical results confirm that PolarQuant can be integrated with LLMs, while maintaining near-lossless performance of generative tasks (Section 4.1). We also highlight the speedup achieved by PolarQuant to showcase the superiority of our decoding algorithm and implementation (Section 4.2).

4.1 Preserving performance in quantized language and reasoning models

General setup. To ensure fair comparisons, we retain the value cache in full precision to avoid potential bias from its quantization in downstream tasks. We evaluate PolarQuant against several quantization baselines built on HuggingFace Transformers codebase [31]. For all group-wise quantization methods, the group size is fixed at $g = 128$. Additional details about the baseline methods and experimental setup are provided in Appendix B. We evaluate all baselines alongside our PolarQuant on a range of models from mainstream model families, including Llama [1, 28] and Qwen [27].

Quantizing language models. In real-world applications of language models, the key-value cache often becomes the primary memory bottleneck when processing long-context inputs. We evaluate PolarQuant and baseline methods on LongBench [5], a widely used benchmark for long-context evaluation. Table 1 presents results on Qwen and Llama. There are two advantages of PolarQuant:

1. PolarQuant performs robustly across different model backbones. Quantization is especially challenging for Qwen models, which exhibit extreme channel-wise outliers in their key cache.[6] When applied to these models, token-wise quantization methods — such as per-token Int. and ZipCache — tend to collapse. In contrast, KIVI-4 and PolarQuant constrain the accuracy drop to within 10%. For Llama-3.1-8B-Instruct, PolarQuant even improves average performance under 3-bit quantization, whereas all baseline methods, including KIVI, result in performance degradation.
2. Averaged across all evaluated settings—covering both model families and quantization precisions—PolarQuant achieves the best overall performance preservation.

[6] Qwen2.5 language models are configured with attention bias that can introduce outliers in specific channels.

Table 1: Performance comparison of quantization methods on LongBench. Cell colors reflect the degree of performance degradation compared to the backbone model. QJL results for Qwen2.5 are excluded due to incompatibility with its official kernel. Parenthetical values in the last column denote the performance drop of the quantized model relative to its backbone.

		Single Doc. QA			Multi Doc. QA			Code Completion		
Quantization	**Bits**	**NtrvQA**	**Qasper**	**MF-en**	**2Wiki**	**Hotpot**	**Musique**	**Lcc**	**RepoBench**	**Avg. ↑**
					Qwen-2.5-1.5B-Instruct (128K)					
Bf16	**16**	19.44	37.22	49.69	32.68	41.57	23.99	41.86	48.55	**38.88**
Int-4	**4.25**	5.11	8.80	10.60	11.99	5.90	32.82	24.65	22.57	**15.30 (-23.58)**
***ZipCache*$_4$**	**4.25**	4.35	8.52	12.07	14.14	17.04	7.57	23.68	21.48	**13.61 (-25.27)**
KIVI-4	**4.25**	19.89	36.52	49.83	32.18	41.51	22.89	40.69	48.72	**36.53 (-2.35)**
***PolarQuant*$_{44}$**	**4.25**	19.34	36.48	50.80	32.80	43.25	23.25	38.62	46.76	**36.41 (-2.47)**
Int-3	**3.25**	3.37	6.96	8.31	9.53	10.79	3.64	22.32	20.69	**10.70 (-28.18)**
***ZipCache*$_3$**	**3.25**	5.28	7.81	9.38	10.30	12.66	6.36	26.53	22.72	**12.63 (-26.25)**
QJL	**3.13**	**N.A**								
KIVI-2	**3.00**	18.39	36.07	47.94	32.51	42.09	23.33	39.50	45.31	**35.64 (-3.24)**
***PolarQuant*$_{33}$**	**3.25**	19.09	35.60	49.47	32.16	43.47	23.02	35.77	44.16	**35.34 (-3.54)**
					Llama-2-7B-Chat (4K)					
Bf16	**16**	18.95	21.14	37.70	30.64	27.81	6.94	58.30	52.17	**31.71**
Int-4	**4.25**	18.24	21.79	37.56	29.80	26.24	8.83	57.95	52.70	**31.64 (-0.07)**
***ZipCache*$_4$**	**4.25**	19.53	19.80	36.35	31.47	26.35	8.23	58.91	51.80	**31.55 (-0.16)**
KIVI-4	**4.25**	18.38	21.16	37.19	31.67	26.90	7.85	58.32	51.99	**31.68 (-0.03)**
***PolarQuant*$_{44}$**	**4.25**	18.40	21.37	35.05	30.18	27.92	8.59	58.82	51.95	**31.54 (-0.17)**
Int-3	**3.25**	16.51	21.41	36.59	29.34	27.59	9.74	57.75	51.42	**31.29 (-0.42)**
***ZipCache*$_3$**	**3.25**	18.73	20.11	34.32	28.50	26.53	8.39	57.51	51.42	**30.69 (-1.02)**
QJL	**3.13**	19.13	20.51	35.93	30.74	25.60	5.79	57.79	50.92	**30.80 (-0.91)**
KIVI-2	**3.00**	18.79	20.46	35.51	27.52	26.36	8.12	56.82	50.26	**30.48 (-1.23)**
***PolarQuant*$_{33}$**	**3.25**	19.75	18.26	35.47	31.15	26.60	7.68	58.26	52.41	**31.20 (-0.51)**
					Llama-3.1-8B-Instruct (128K)					
Bf16	**16**	31.38	46.65	56.81	49.46	57.85	32.63	62.88	56.43	**49.26**
Int-4	**4.25**	31.76	45.49	56.47	49.52	57.67	32.82	63.17	55.46	**49.05 (-0.21)**
***ZipCache*$_4$**	**4.25**	32.26	45.97	56.77	49.50	58.67	33.03	63.41	55.97	**49.45 (+0.19)**
KIVI-4	**4.25**	31.23	47.15	57.14	49.14	58.05	32.67	63.05	56.45	**49.36 (+0.10)**
***PolarQuant*$_{44}$**	**4.25**	31.36	46.78	56.72	49.47	58.54	32.23	63.28	56.73	**49.39 (+0.13)**
Int-3	**3.25**	29.66	45.07	55.15	49.79	58.31	32.73	60.56	54.80	**48.26 (-1.00)**
***ZipCache*$_3$**	**3.25**	31.98	44.70	55.61	49.16	58.33	31.53	61.93	54.19	**48.43 (-0.83)**
QJL	**3.13**	32.41	44.75	56.18	48.50	57.34	32.07	61.66	55.99	**48.61 (-0.65)**
KIVI-2	**3.00**	31.90	45.39	54.96	49.88	58.08	32.25	62.03	54.93	**48.68 (-0.58)**
***PolarQuant*$_{33}$**	**3.25**	32.49	46.72	56.56	49.96	58.33	32.20	63.45	56.54	**49.53 (+0.27)**

Beyond long-context processing, we also apply PolarQuant on LLMs to examine their generative abilities with normal-length inputs. To explore how quantization affects the emergent capabilities of LLMs, such as chain-of-thought reasoning [30] and in-context learning [6], we benchmark PolarQuant's performance on the 5-shot CoT GSM8K [8]. The results, presented in Table 2, show that PolarQuant effectively supports both short- and long-context inputs, without compromising performance on reasoning or knowledge-intensive tasks.

Table 2: Evaluations on 5-shot CoT GSM8K.

Llama-2 7B-Chat	**Quantization**	**Bf16**	**Int-4**	**ZipCache-4**	**KIVI-4**	**PolarQuant$_{44}$**
	Acc. (Bits)	20.92 (16)	19.79 (4.25)	23.12 (4.25)	21.61 (4.25)	22.61 (4.25)
Llama-3.1 8B-Instruct	**Quantization**	**Bf16**	**Int-4**	**ZipCache-4**	**KIVI-4**	**PolarQuant$_{44}$**
	Acc. (Bits)	78.85 (16)	76.35 (4.25)	78.70 (4.25)	78.32 (4.25)	78.77 (4.25)

Quantizing reasoning models. Large reasoning models (LRMs) exhibit remarkable capability in solving complex problems by long chains of thought. Recent studies [18] have shown that complex tasks—such as mathematics [16, 8] and reasoning [23]—are sensitive to the accumulation of quantization errors. Given that LRMs rely on both long-context processing and the generation of lengthy outputs, quantization techniques are particularly critical for their application. Moreover, the

underlying mechanisms of LRMs differ from those of LLMs, making quantization more challenging and offering valuable insights into the effectiveness of different approaches.

We apply key quantization to the distillation-based reasoning models of DeepSeek-R1 [20]. The accuracy scores across different quantization methods and tasks are presented in Table 3. The quantized reasoning models of PolarQuant achieve substantial improvements over the baselines, which provides strong supplementary evidence of PolarQuant's superiority.

Table 3: Overall performance comparison of quantized DeepSeek-R1-Distill models across various reasoning benchmarks. Results for *ZipCache* on Qwen2.5 are omitted due to its performance collapse. Cell colors represent the degree of performance degradation compared to the backbone model. Parenthetical values in the last column denote the performance drop of the quantized model relative to its backbone. Best results are highlighted in bold.

Quantization	AIME			MATH	GPQA	Overall ↑
	AIME24	AIME25	AVG.			
			DeepSeek-R1-Distill-Qwen-1.5B			
Bf16	**36.67**	**23.33**	**30.00**	**85.20**	**39.90**	**51.70**
ZipCache-4			N.A			
KIVI-4	20.00	23.33	21.67	80.40	33.84	45.30 (-6.40)
PolarQuant$_{44}$	30.00	20.00	25.00	80.20	37.88	**47.69 (-3.31)**
			DeepSeek-R1-Distill-Llama-8B			
Bf16	**50.00**	**36.67**	**43.33**	**91.20**	**51.52**	**62.01**
ZipCache-4	43.33	43.33	43.33	91.60	48.48	61.13 (-0.88)
KIVI-4	43.33	33.33	38.33	89.80	51.01	59.71 (-2.30)
PolarQuant$_{44}$	60.00	36.67	48.33	89.00	50.00	**62.44 (+0.43)**

4.2 Superior efficiency of PolarQuant

To evaluate the efficiency of the customized decoding algorithm, we provide a comprehensive time breakdown analysis of our PolarQuant implementation. In all experiments, we use the Llama-3.1-8B-Instruct model configuration, which includes 32 query heads of dimension 128, and 8 key/value heads (grouped-query attention [2]). We benchmark the latency of our tailored kernel implementation, as well as the end-to-end generation throughput.

Latency for query-key multiplication kernel. We make comparisons of the query-key multiplication implementations for LLM decoding. Specifically, we evaluate the wall-clock latency across different sequence lengths and batch size settings. We benchmark the runtime by summing across 10K iterations of the calculations, and report results for: *Fp16*, *KIVI-4*, *KIVI-2*, *PolarQuant*$_{44}$ and *PolarQuant*$_{33}$. Figure 3 illustrates the performance comparison among kernel implementations, while Table 4 reports the multiplication latency with a batch size of 8. We see that, PolarQuant achieves up to **2.7**× speedup compared with KIVI and **1.6**× speedup compared with Fp16 Torch implementation.

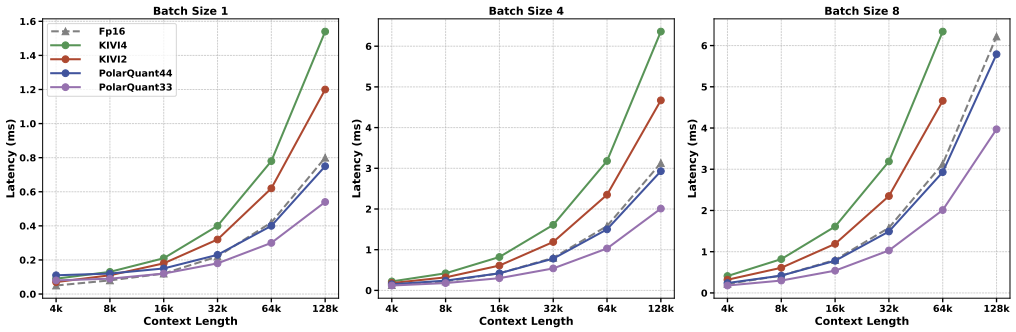

Figure 3: Latency comparisons of PolarQuant across varying batch sizes and context lengths

Throughput comparison. We incorporate our custom kernels into the generation pipeline and measure their end-to-end performance. We determine the maximum supported batch size with a sequence length of 32K tokens, with Hugging Face's implementation. We fix the input length at 256 tokens and measure throughput across different generation lengths. Table 4 presents the results; entries marked with † indicate the application of 2-bit value quantization. As is shown in Table 4, PolarQuant achieves up to **3.18**× throughput improvements, and demonstrates significant improvements over KIVI in both latency and throughput. Values in parentheses denote the speedup.

Table 4: Latency, throughput and memory usage comparisons.

Operation	**Latency (s)**			
	4K	**8K**	**32K**	**128K**
Fp16	**0.22**	**0.42**	**1.58**	**6.22**
KIVI-4	0.41 (0.54×)	0.82 (0.51×)	3.19 (0.50×)	N.A
PolarQuant$_{44}$	0.24 (0.92×)	0.42 (1.00×)	1.49 (1.06×)	5.79 (1.07×)
KIVI-2	0.32 (0.69×)	0.61 (0.69×)	2.35 (0.67×)	N.A
PolarQuant$_{33}$	0.18 (1.22×)	0.30 (1.40×)	1.03 (1.53×)	3.97 (1.57×)

Configuration	**TP. (token/s) / Mem. (GB)**			
	4K	**8K**	**16K**	**32K**
Fp16	**119.1 / 20.78**	**84.5 / 25.21**	**53.6 / 34.07**	**15.3 / 51.79**
KIVI-4	129.0 (1.09×) / 19.03	96.9 (1.15×) / 21.88	65.1 (1.21×) / 27.59	39.0 (2.54×) / 38.96
PolarQuant$_{44}$	138.8 (1.17×) / 19.04	108.5 (1.28×) / 21.90	75.9 (1.42×) / 27.61	46.8 (3.05×) / 39.06
KIVI-2	133.2 (1.12×) / 18.89	99.9 (1.18×) / 21.55	67.7 (1.26×) / 26.91	40.5 (2.64×) / 37.62
PolarQuant$_{33}$	144.8 (1.22×) / 19.04	111.1 (1.31×) / 21.90	78.6 (1.46×) / 27.61	48.7 (3.18×) / 39.06
KIVI-4†	129.0 (1.09×) / 17.08	115.8 (1.37×) / 17.99	90.8 (1.69×) / 19.92	30.13 (1.97×) / 23.60
PolarQuant$^{\dagger}_{44}$	144.1 (1.21×) / 17.06	128.0 (1.51×) / 17.97	111.6 (2.08×) / 19.81	46.86 (3.06×) / 23.46

5 Discussions

5.1 Ablation studies of PolarQuant

Effect of group size g. PolarQuant applies group-wise quantization along the channel dimension. We conduct ablation studies to investigate the impact of the group size g on model performance.

Table 5: Ablation study of group size g on LongBench.

	Group Size	32	64	128	256
LongBench	**KIVI-4**	49.48 (5.00)	49.47 (4.50)	49.36 (4.25)	49.52 (4.13)
(Bits)	**PolarQuant**$_{44}$	49.50 (5.00)	49.33 (4.50)	49.39 (4.25)	49.58 (4.13)

Specifically, we benchmark PolarQuant and KIVI on LongBench using the Llama-3.1-8B-Instruct model. The results, presented in Table 5, indicate that PolarQuant performs competitively with KIVI across all tested group sizes. It is noteworthy that the quantization parameters occupy $32/g$ bits; therefore, smaller values of g result in higher average bit widths. To strike an optimal balance between performance and parameter overhead, we set g as 128 throughout this paper.

Impact of RoPE configuration. We conduct experiments to test the sensitivity of PolarQuant to different RoPE configurations and draw the following conclusions:

(1) PolarQuant exhibits consistent performance across LLMs with different RoPE base frequencies (see Table 1). The experimental results for basic values of $\{10000, 500000, 1000000\}$, highlight PolarQuant's insensitivity to the choice of base frequency.

(2) PolarQuant is adaptable to different RoPE variants. We employ NTK RoPE scaling [24] to extend the LLM's context and apply PolarQuant to the key cache. Critically, mo significant performance drop is observed. We provide detailed experimental results and setups in Appendix C.

Bitwidth allocation for radii and polar angles. In PolarQuant, we assign bitwidth asymmetrically between radii and angles. To determine which component requires higher precision, we conduct ablation studies on various bitwidth configurations, enabling a more flexible allocation strategy.

Table 6: Ablation study on asymmetrical bitwidth allocation in PolarQuant.

LongBench	Bits	$(\mathbf{r5}, \mathbf{t3})$	$(\mathbf{r4}, \mathbf{t4})$	$(\mathbf{r3}, \mathbf{t5})$	Bits	$(\mathbf{r4}, \mathbf{t2})$	$(\mathbf{r3}, \mathbf{t3})$	$(\mathbf{r2}, \mathbf{t4})$
	4.25	49.41	49.39	49.51	3.25	47.66	49.53	49.00

We take Llama-3.1-8B-Instruct as backbone and benchmark PolarQuant on LongBench, to assess the impact of different bitwidth settings. From Table 6, we have the following observations:

***Observation* 1:** Angle quantization is more sensitive to bitwidth. Allocating fewer than 3 bits to angles often results in a significant drop in performance.

***Observation* 2:** Despite the asymmetry in bitwidth allocation, a more balanced distribution between radii and angles can still achieve strong performance.

5.2 Compatibility with existing KV cache compression techniques

In this section, we explore the integration of existing KV compression techniques with PolarQuant, which allows for further reductions in KV cache memory occupation. We present the experimental results of Llama-3.1-8B-Instruct on LongBench. As stated in Section 4.1, we retain the value cache in full precision, as key cache is more sensitive to low-precision quantization. Appendix D provides further analysis to support this. We combine value quantization with PolarQuant to verify its compatibility. A standard token-wise quantization is applied to the value cache, consistent with KIVI [19]. Table 7 presents the results. The introduction of value quantization results in only marginal performance degradation, even at 2-bit precision.

Table 7: LongBench results of PolarQuant with value cache quantization .

Quantization	Value Bits.	NtrvQA	Qasper	MF-en	2Wiki	Hotpot	Musique	Lcc	RepoBench	Avg.
PolarQuant_{44}	**16**	31.36	46.78	56.72	49.47	58.54	32.23	63.28	56.73	**49.39**
	4	31.67	46.51	56.48	49.74	58.48	32.41	63.50	56.59	**49.42 (+0.03)**
	2	31.26	46.69	57.59	47.88	58.51	32.98	63.24	55.95	**49.26 (-0.13)**

We further explore the integration of PolarQuant with token-eviction strategies [15]. As shown in Table 8, PolarQuant does not exhibit significant performance degradation. We left it for future work to combine PolarQuant with existing mixed-precision quantization for further memory saving [3, 10, 33].

Table 8: LongBench Evaluations of PolarQuant with SnapKV.

LLM	NtrvQA	Qasper	MF-en	2Wiki	Hotpot	Musique	Lcc	RepoBench	Avg.
Full KV	31.36	46.78	56.72	49.47	58.54	32.23	63.28	56.73	**49.39**
***SnapKV*: 4096**	31.31	46.51	57.03	49.71	57.99	32.79	62.90	55.75	49.25 (-0.14)
w. PolarQuant	31.21	46.38	56.45	49.38	58.05	31.99	62.90	55.75	49.01 (-0.38)
***SnapKV*: 1024**	31.51	43.45	56.14	49.61	57.78	32.01	62.45	56.00	48.62 (-0.77)
w. PolarQuant	31.27	42.61	55.18	48.60	57.49	31.12	61.54	54.57	47.80 (-1.59)

6 Conclusion

In this paper, we view the outliers in the key cache of LLMs from a novel polar-coordinate-based perspective, which provides an efficient and effective solution, PolarQuant, to reduce the complexity and quantization costs in previous methods. PolarQuant well preserves downstream performance even in long-context understanding and long chain-of-thought generation, comparable to previous works under 4-bit precision while achieving superior efficiency. We hope the polar coordinate view can inspire the community to advance new low-bit precision quantization techniques.

Acknowledgments

This work is supported by Meituan through Agentic system X Program. Songhao Wu is supported by "Qiushi Academic-Dongliang" Project of Renmin University of China (No. RUC24QSDL015). We appreciate all reviewers for their valuable comments and suggestions, which are crucial for improving our work. We are also grateful to Xintong Qiu, Jixiang Hong, Tao Tan, Jia-Nan Li and Yuxuan Liu for their insightful suggestions on the manuscript.

References

[1] AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

[3] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms, 2024. URL https://arxiv.org/abs/2404.00456.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016. URL https://arxiv.org/abs/1409.0473.

[5] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[7] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2024. URL https://arxiv.org/abs/2406.02069.

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

[9] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An,

Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

[10] Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache, 2024. URL https://arxiv.org/abs/2403.04643.

[11] Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. Lighteval: A lightweight framework for llm evaluation, 2023. URL https://github.com/huggingface/lighteval.

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

[13] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

[14] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm, 2024. URL https://arxiv.org/abs/2403.05527.

[15] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

[16] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

[17] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

[18] Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan, and Lu Hou. Quantization hurts reasoning? an empirical study on quantized reasoning models, 2025. URL https://arxiv.org/abs/2504.04823.

[19] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2025.

[20] Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision, 2024. URL https://arxiv.org/abs/2406.06592.

[21] OpenAI. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/, 2024. [Accessed 19-09-2024].

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

[23] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[24] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

[25] Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

[26] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

[27] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

[28] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[30] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[32] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[33] June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization, 2024. URL https://arxiv.org/abs/2402.18096.

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

[35] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data*, 3(3):1–30, 2025.

[36] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

[37] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of Machine Learning and Systems*, volume 6, pages 196–209, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/5edb57c05c81d04beb716ef1d542fe9e-Paper-Conference.pdf.

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: In the abstract, we claim that PolarQuant serve as a novel key cach quantization, while PolarQuant enhances the quantization performance of the downstream tasks.

   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 create a separate "Limitations" section, i.e., Section E in Appendix.

   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: [NA]

Justification: Our paper does not introduce theoretical results.

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: We have submited our code in the supplementary materials, enabling easy reproduction.

   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: We have submited our code in the supplementary materials, enabling easy reproduction.

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: See Section B in Appendix.

   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: The statistical significance of the experiment has not been reported in this work.

   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 detailed information on the computing resources required to reproduce the experiments in Section 4.2.

   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 have conformed to 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: [No]

    Justification: We have not included a discussion of the potential positive and negative societal impacts of the work performed.

    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: The paper poses no such risks.

    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: We have correctly cited all the data, scripts, and models we used.

    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: We have included a README document with our code.

    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: The 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: The 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: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

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.

A Accelerated Query-Key Inner Product with Cached Keys in PolarQuant

For a quantized representation $\Big(Q(\rho_n[j]), Q(\theta_n[j])\Big)$, we first dequantize the polar coordinates back into Cartesian coordinates, which serves as the at dimensions $2j$ and $2j+1$:

$$\begin{bmatrix} \widetilde{\mathbf{K}}_n[2j] \\ \widetilde{\mathbf{K}}_n[2j+1] \end{bmatrix}^\top = \begin{bmatrix} \tilde{\rho}_n[j] \cdot \cos\Big(\tilde{\theta}_n[j]\Big) \\ \tilde{\rho}_n[j] \cdot \sin\Big(\tilde{\theta}_n[j]\Big) \end{bmatrix}^\top,$$

where $\tilde{\rho}_n[j],\ \tilde{\theta}_n[j]$ is the dequantization for (ρ, θ) respectively. $\tilde{\rho}_n[j]$ and $\tilde{\theta}_n[j]$) are formulated as:

$$\tilde{\rho}_n[j] = \big(Q(\rho_n[j]) + \frac{1}{2}\big) \cdot s_{\rho[:]}[j] + z_{\rho[:]}[j], \quad \tilde{\theta}_n[j] = \big(Q(\theta_n[j]) + \frac{1}{2}\big) \cdot s_{\theta[:]}[j] + z_{\theta[:]}[j].$$

$\Big(Q(\rho_n[j]), Q(\theta_n[j])\Big)$ represents $\big[\widetilde{\mathbf{K}}_n[2j],\ \widetilde{\mathbf{K}}_n[2j+1]\big]$ as a state in the lookup table.

When computing the query-key inner product for the dimensions $2j$ and $2j+1$, the result is:

$$\mathbf{product}_{2j,2j+1} = \mathbf{Q}_n[2j] \cdot \widetilde{\mathbf{K}}_n[2j] + \mathbf{Q}_n[2j+1] \cdot \widetilde{\mathbf{K}}_t[2j+1],$$

and the final inner product is the sum:

$$\sum_{0 \le j < \frac{d}{2}} \mathbf{product}_{2j,2j+1}.$$

Figure 4 presents a naive PyTorch [22] implementation of the aforementioned dequantization-and-multiplication operation. We implement a fused Triton kernel that reproduces the functionality of the PyTorch code for improved computational efficiency. More details can be found in our code.

```
import torch


def attention_decode_forward_pytorch_impl(
    q,  # q's shape: (B, N, 1, 2, D)
    r, rscale, rmn,  # r's shape: (B, N, G, D)
    t, tscale, tmn,  # t's shape: (B, N, G, D)
    # rscale, rmn, tscale and tmn's shape: (B, N, 1, 1, D)
    rbits: int = 4, tbits: int = 4,
):
    # phi: finite set for theta
    phi = torch.arange(0, 2 ** tbits)[None, None, :, None, None]
    phi = (2 * phi + 1) / 2 * tscale + tmn

    # rho: finite set for rho
    rho = torch.arange(0, 2 ** rbits)[None, None, :, None, None]
    rho = (2 * rho + 1) / 2 * rscale + rmn

    phi = torch.cat([phi.cos(), phi.sin()], dim=-2)

    accumulator = torch.sum(q * phi, dim=-2)  # (B, N, 2^tbits, D)
    accumulator = torch.gather(accumulator, 2,
        t.unsqueeze(-1).expand_as(accumulator))
    accumulator *= torch.gather(rho.squeeze(-2), 2,
        r.unsqueeze(-1).expand_as(accumulator))

    accumulator = accumulator.sum(-1)

    return accumulator
```

Figure 4: Pytorch implementation of the accelerated Query-Key Inner Product in PolarQuant.

B Details experiment setup

In this section, we provide additional details about the experimental setups.

B.1 Setup of baseline methods

This section provides an overview of the baseline methods. Following this, we outline the quantization configurations and variants for both the baselines and our proposed PolarQuant. We also explain how the average number of bits for the quantization parameters is calculated.

Int-N applies token-wise N-bit quantization to the key states. This token-wise quantization incurs $32/d$ bits quantization parameters (16 bits for zero-points and 16 bits for scaling factors per token), which amounts to 0.25 bits per token when $d = 128$.

ZipCache-N [12] introduces a channel-separable, token-wise scheme for key quantization, where N denotes the quantization bits. Each key channel is normalized by the square root of its maximum magnitude before quantization. Similar to **Int**-N, this method performs token-wise quantization, allocating 0.25 bits for zero-points and scales.

KIVI-N [19] employs an asymmetric strategy for KV cache quantization, applying channel-wise quantization to the key cache and token-wise quantization to the value cache. For 4-bit quantization, we use **KIVI**-4 with a group size of 128. For 3-bit quantization, we use **KIVI**-2 with a group size of 32, as the official implementation does not support 3-bit quantization. This channel-wise quantization introduces $(16 + 16)d$ bits of quantization parameters per group, which increases the average bitwidth by $32/g$: 1 bit for $g = 32$ and 0.25 bits for $g = 128$ respectively.

QJL [34] applies Johnson-Lindenstrauss transformation to key states, removing the memory overheads associated with storing quantization constants. For a 3-bit quantization schema, this method achieves a key cache bitwidth of 3.13 bits.

PolarQuant$_{rt}$ assigns r bits for radii quantization and t bits for polar angles, resulting in $(r + t)/2$ bitwidth for key states quantization. PolarQuant also employs channel-wise quantization with grouping; this group partitioning adds an overhead of $32/g$ bits.

Nearly all methods discussed here require a buffer size or residual length for applying quantization. We exclude the contribution of these residual key states to bit counts for comparisons.

B.2 Configurations of the open-sourced LLMs

- **Qwen-2.5-1.5B-Instruct** [27] is the instruction-tuned 1.5B Qwen2.5 model, which supports a context length of up to 131,072 tokens and features a base RoPE frequency of 1,000,000.
- **Llama-2-7B-Chat** [28] has a context length of 4096 and base RoPE frequency of 10,000.
- **Llama-3.1-8B-Instruct** [1] is an 8 billion parameter language model, designed to handle a context length of up to 131,072 tokens, and has the base RoPE frequency set to 500,000.

B.3 Reasoning model evaluation

For fair comparison, the evaluation code is built on the Huggingface Lighteval framework [11]. We use the default generation configuration for datasets splitting, and the EM score is reported in Table 3. More implementation details can be found in our released code.

C NTK RoPE scaling experiment

We adopt NTK RoPE scaling [24] to extend the context window of the Llama-2-7B-Chat model from 4096 to 8192. Specifically, we implement dynamic RoPE updates based on the Hugging Face codebase. PolarQuantachieve an average performance of 32.44 on LongBench. Compared to the 32.15 score of the Bf16 baseline, PolarQuant's performance remains competitive.

D Sensitivity Analysis of Key-Value Quantization

In Section 5.2, we combine 4-bit key quantization PolarQuant$_{44}$ with 2-bit value quantization and observe minimal performance degradation. We further evaluate value quantization (KIVI, group size 128) on LongBench. By retaining the key cache at full precision, KIVI result in no performance drop. However, when the key is quantized to the same bitwidth while the value is kept at full precision, the performance drops significantly. Table 9 presents the results, the notation $(\mathbf{K_b}, \mathbf{V_c})$ denotes key quantization to b-bit and value quantization to c-bit.

Table 9: Impact of key and value quantization bitwidths on LongBench performance.

LongBench	$(\mathbf{K16}, \mathbf{V16})$	$(\mathbf{K16}, \mathbf{V4})$	$(\mathbf{K16}, \mathbf{V2})$	$(\mathbf{K2}, \mathbf{V16})$
	49.26	49.54	49.30	47.73

E Limitation

Although PolarQuant achieves promising results in reducing storage and computational resources, we also discuss the limitations of our current work. This work focuses exclusively on decoder-only Transformer-based large language models (LLMs) that utilize rotary position embedding (RoPE) as the underlying position encoding mechanism. RoPE has become a prevalent choice in many state-of-the-art open-source LLMs. However, the effectiveness of PolarQuant when applied to models with alternative position encoding methods or attention mechanisms remains an open question and warrants further investigation. Furthermore, exploring more recent LLM backbones [32] is necessary, but due to time and computational limitations, we leave this part of the work for future research.