# PolarQuant: Vector Quantization with Polar Transformation

**Insu Han**
KAIST
insu.han@kaist.ac.kr

**Praneeth Kacham**
Google Research
pkacham@google.com

**Amin Karbasi**
Foundation AI–Cisco Systems Inc
karbasi@cisco.com

**Vahab Mirrokni**
Google Research
mirrokni@google.com

**Amir Zandieh**
Google Research
zandieh@google.com

## Abstract

Vector quantization is a prevalent technique for reducing the memory footprint in a wide range of computational problems, such as the training and deployment of deep learning models and vector search systems. We introduce PolarQuant, a novel online vector quantization method that leverages random preconditioning and polar transformation. Our approach efficiently transforms Euclidean vectors into polar coordinates using a recursive algorithm, and then quantizes the resulting angles. A key insight is that, following random preconditioning, the angles in the polar representation exhibit a tightly bounded, highly concentrated, and analytically computable distribution, independent of the input data. This nice distribution eliminates the need for explicit normalization and learned data-dependent quantization codebooks, steps that introduce significant memory and runtime overhead in traditional product/scalar quantization methods. By circumventing this data-dependent step, PolarQuant achieves substantial memory and runtime savings, making it highly suitable for online scenarios such as KV cache compression. The long-context evaluation demonstrates that PolarQuant compresses the KV cache by over $\times \mathbf{4.2}$ while achieving the best quality scores compared to the state-of-the-art methods.

Authors are listed in alphabetical order. Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. 

## 1 INTRODUCTION

Vector quantization is a fundamental technique in large-scale machine learning systems, playing a critical role in compressing high-dimensional data such as embedding vectors for storage, retrieval, and efficient computation. Applications span across domains including similarity search (Jegou et al., 2010; Guo et al., 2020), recommendation systems (Covington et al., 2016; Zhu et al., 2024; Liu et al., 2024a), and real-time inference pipelines (Bin et al., 2025). Recently, it has been applied to a caching mechanism in large language models (LLMs) with Transformer-based architecture (Vaswani et al., 2017), particularly for compressing the key and value (KV) embeddings stored during autoregressive generation. However, traditional quantization methods suffer from practical limitations, which require metadata for data normalization and it can introduce substantial memory overhead.

These inefficiencies become particularly problematic in scenarios where quantization must be performed online or at scale. In such settings, computing normalization statistics (e.g., per-block scales and zero-points) and solving optimization procedures (e.g., k-means clustering for product quantization) can be prohibitive. Moreover, storing quantization parameters alongside quantized values often adds substantial metadata overheads.

In this work, we propose a general-purpose vector quantization method that resolves the overhead issues. Our approach applies a random preconditioning matrix to each input vector prior to quantization. This transformation reshapes the angular distribution of the data into a fixed, highly concentrated form, which enables the use of a data-independent codebook optimized once and for all. The result is a fast, scalable quantization scheme with no per-dataset tuning and minimal mem-

ory overhead.

Although our method is broadly applicable, we demonstrate its effectiveness on the KV cache compression problem in large language models (LLMs), i.e., it involves a setting where vector quantization offers a natural and impactful application. Our method achieves strong compression-accuracy tradeoffs while requiring no data-dependent training or normalization overhead.

### 1.1 Related Work

A rich line of research has studied quantization techniques for compressing high-dimensional vectors, particularly in the context of approximate nearest neighbor (ANN) search. Product quantization (PQ) (Jegou et al., 2010) and its variants (Babenko and Lempitsky, 2014; Ge et al., 2013; Guo et al., 2020) partition vectors into subspaces and learn subcodebooks using data-driven methods such as k-means. Though they are effective in reducing search costs, they incur high computational costs during training and often require per-block normalization, which introduces extra memory overhead and latency. Scalar and vector quantization schemes that use learned normalization constants or per-token scale/shift parameters also suffer from metadata overhead and limited generalization.

Recent efforts have explored bypassing data-dependent optimization using random projections or sketching-based techniques. These approaches improve runtime and generalization but often sacrifice quantization fidelity or still require partial normalization steps. Our work differs by achieving competitive accuracy while eliminating both normalization and codebook learning entirely, through a structured random preconditioning step.

Quantization has recently been explored as a solution for reducing the memory footprint of key-value (KV) caches in autoregressive Transformers (Yue et al., 2024; Yang et al., 2024; Dong et al., 2024; Kang et al., 2024; Zhang et al., 2024; Liu et al., 2024b). Among these, QJL (Zandieh et al., 2024) proposes a 1-bit data-oblivious scheme using sketching, while Lexico (Kim et al., 2024) applies dictionary learning with sparse encodings. However, many of these approaches either rely on specialized assumptions about KV structures or require expensive encoding procedures. Our method is more general and can be directly applied to KV cache embeddings without additional costs for calibrating or fine-tuning.

### 1.2 Contributions

We propose quantizing Euclidean vectors in polar coordinates instead of Cartesian coordinates. This shift enables more efficient representation and compression of vector datasets.

**Random Preconditioning.** We apply a random rotation to the vectors before quantization, which preserves inner products while randomizing the distribution of each vector. This preconditioning causes the angles in polar coordinates to follow a fixed and data-independent and highly concentrated distribution, allowing us to quantize them with high precision using small bit-widths. We derive the analytical distribution of angles after preconditioning and leverage this insight to construct an optimized quantization codebook, minimizing quantization error.

**Recursive Polar Transformation.** We introduce a computationally efficient recursive polar transformation that converts vectors into polar coordinates, enabling practical deployment of our approach.

**Performance on Long-Context Tasks.** We evaluate PolarQuant on long-context tasks (LongBench-E) and demonstrate that it achieves the best quality scores compared to competing methods while compressing the KV cache memory by over $\times\mathbf{4.2}$.

## 2 PRELIMINARIES

**Inference throughput and latency for LLMs.** Inference in a decoder-only Transformer-based large language model involves two phases: (1) **prefill** and (2) **decoding**. In the prefill phase, all tokens in the prompt are processed in parallel to compute the corresponding KV caches of the prompt. If the prompt has $T$ tokens, the number of matrix-multiplication FLOPs required in prefill is approximately $2 \cdot T \cdot$ (number of activated params) plus additional attention computation. When $T$ is reasonably large (e.g., hundreds of tokens), modern accelerators such as GPUs or TPUs can achieve high FLOP utilization.

In the decoding phase, tokens are generated one at a time. Each new token is processed by passing it through the model and using the cached KV embeddings to compute attention outputs. The compute cost per token is approximately $2 \cdot (\#$ activated params $+$ size of KV caches). When decoding one token, the model must load all activated parameters from HBM to perform a matrix multiplication with a single vector, resulting in poor compute utilization. To mitigate this, serving systems typically batch multiple decoding requests, so the cost of expensive loading of parameters can be amortized over the number of tokens generated. However, a key bottleneck to increasing concurrency is the size of the KV caches, which limits the number of concurrent KV caches that can be held in the HBM. Thus, quantizing KV caches will let us decrease the latency of a single request in the long-context setting

(by decreasing the amount of transfer necessary from HBM to compute) or increase the throughput of the serving systems by letting us serve a large number of decoding requests at the same time.

**Random Preconditioning.** A critical step in the PolarQuant algorithm involves the random preconditioning of vectors prior to quantization. This involves applying a random projection matrix to the embedding vectors before quantizing them. To analyze the algorithm effectively, we rely on specific facts and properties of multivariate normal random variables, which are outlined below.

**Fact 2.1.** *For any positive integer $d$, if $\boldsymbol{x} \in \mathbb{R}^d$ is a zero mean unit variance isotropic Gaussian random variable in dimension $d$, i.e., $\boldsymbol{x} \sim \mathcal{N}(0, \boldsymbol{I}_d)$, then its 2-norm, denoted by $r := \|\boldsymbol{x}\|_2$, follows a generalized gamma distribution with the following probability density for any $r \geq 0$:*

$$f_R(r) = \frac{2}{2^{d/2} \cdot \Gamma(d/2)} r^{d-1} \exp\left(-r^2/2\right) \tag{1}$$

The proof of Fact 2.1 and all subsequent proofs can be found in Appendix A. We also use the following facts about the moments of the univariate normal distribution.

**Fact 2.2** (Moments of Normal Random Variable)**.** *If $x$ is a normal random variable with zero mean and unit variance $x \sim \mathcal{N}(0,1)$, then for any integer $\ell$, $\mathbb{E}_{x\sim\mathcal{N}(0,1)}\left[|x|^\ell\right] = 2^{\ell/2}\Gamma((\ell+1)/2)/\sqrt{\pi}$.*

The preconditioning we refer to involves multiplying each embedding vector by a shared random matrix $\boldsymbol{S}$ with i.i.d. normal entries where we call $\boldsymbol{S}$ the preconditioning matrix. By the Johnson-Lindenstrauss (JL) lemma (Dasgupta and Gupta, 2003), this preserves the norms and inner products of the embedding vectors with minimal distortion. A key property of this preconditioning, which we will leverage in our later analysis, is that the embedding vectors after the preconditioning follow a multivariate normal distribution. This is formalized in the following fact.

**Fact 2.3.** *For any vector $\boldsymbol{x} \in \mathbb{R}^d$ if $\boldsymbol{S} \in \mathbb{R}^{m\times d}$ is a random matrix with i.i.d. normal entries $\boldsymbol{S}_{i,j} \sim \mathcal{N}(0, 1/m)$, then the vector $\boldsymbol{S}\cdot\boldsymbol{x}$ has multivariate normal distribution $\boldsymbol{S}\cdot\boldsymbol{x} \sim \mathcal{N}\left(0, (\|\boldsymbol{x}\|_2/m)\cdot\boldsymbol{I}_m\right)$.*

The following lemma establishes the distribution of the polar angle of a point $(x, y)$ in dimension 2, where the $x$ and $y$ coordinates are independent samples from the Euclidean norm of multivariate normal random variables.

**Lemma 2.4** (Moments of Normal Random Variable)**.** *For any positive integer $d$, if $x, y \geq 0$ are two i.i.d. random variables with generalized gamma distribution with probability density function $f_Z(z) = \frac{2}{2^{d/2}\cdot\Gamma(d/2)} z^{d-1}\exp\left(-z^2/2\right)$, then the angle variable $\theta := \tan^{-1}(y/x)$ follows the probability density function:*

$$f_\Theta(\theta) = \frac{\Gamma(d)}{2^{d-2}\cdot\Gamma(d/2)^2}\cdot\sin^{d-1}(2\theta).$$

*Additionally, $\mathbb{E}[\Theta] = \pi/4$ and $\mathrm{Var}(\Theta) = O(1/\sqrt{d})$.*

# 3 POLARQUANT

## 3.1 Recursive Polar Transformation

Here we employ a polar transformation in $\mathbb{R}^d$ that can be recursively computed from the Cartesian coordinates of points. Throughout this work, we assume that $d$ is an integer power of 2.

At a high level, our approach begins by grouping pairs of coordinates of a $d$-dimensional vector $\boldsymbol{x}$ and transforming each pair into 2D polar coordinates. This produces $d/2$ of radius and angle pairs. Next, we gather $d/2$ of radii and apply the polar transform to them. This procedure is recursively repeated $\log_2 d$ times and the final output consists of 1D radius and $1, \ldots, d/2$-dimensional angles. A formal definition is provided in Definition 3.1 and its visual diagram is shown in Figure 1.

**Definition 3.1** (RecursivePolar Transformation)**.** For any integer power of two $d$, the polar representation of any vector $\boldsymbol{x} \in \mathbb{R}^d$ includes $d-1$ angles and a radius. Angles are organized into a collection of $\log_2 d$ vector of angles $\psi^{(1)}, \psi^{(2)}, \ldots \psi^{(\log_2 d)}$ such that $\psi^{(1)} \in [0, 2\pi)^{d/2}$ and $\psi^{(\ell)} \in [0, \pi/2]^{d/2^\ell}$ for any $\ell \geq 2$. In other words, the angles are computed in $\log_2 d$ levels and there are $d/2^\ell$ angles in level $l$. These angles are defined by the following relation for $\ell \in \{2, 3, \ldots \log_2 d\}$:

$$\psi_j^{(1)} := \tan^{-1}\left(\boldsymbol{x}_{2j}/\boldsymbol{x}_{2j-1}\right) \quad \text{for } j \in [d/2],$$
$$\psi_j^{(\ell)} := \tan^{-1}\left(\frac{\left\|\boldsymbol{x}_{(j-1/2)2^\ell+1:j2^\ell}\right\|_2}{\left\|\boldsymbol{x}_{(j-1)2^\ell+1:(j-1/2)2^\ell}\right\|_2}\right) \quad \text{for } j \in [d/2^\ell].$$

The reverse of this transformation maps the angles and the radius of any point to its Cartesian vector representation using the following equation:

$$\boldsymbol{x}_i = \|\boldsymbol{x}\|_2 \cdot \prod_{\ell=1}^{\log_2 d}\left(\cos\psi^{(\ell)}_{\lfloor\frac{i}{2^\ell}\rfloor}\right)^{\mathbf{1}_{\{(i \bmod 2^\ell)\leq 2^{\ell-1}\}}} \cdot \prod_{\ell=1}^{\log_2 d}\left(\sin\psi^{(\ell)}_{\lfloor\frac{i}{2^\ell}\rfloor}\right)^{\mathbf{1}_{\{(i \bmod 2^\ell)> 2^{\ell-1}\}}}$$

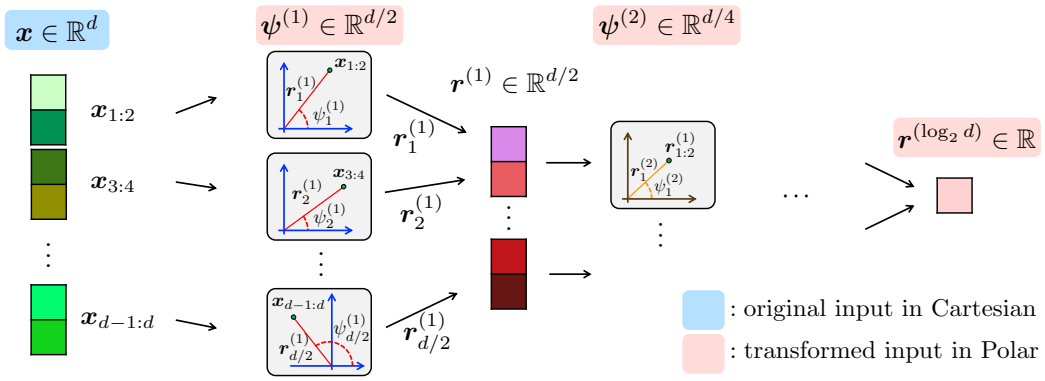



Figure 1: Overview of RecursivePolar transformation in Definition 3.1

In what follows, we analyze the distribution of angles generated in each level of polar transformation.

### 3.2 Distribution of Polar Angles under Random Preconditioning

One of our primary objectives is to eliminate the need for explicit normalization (e.g., minimum/maximum values) prior to quantization, thereby reducing quantization overhead. To achieve this, our algorithm applies random preconditioning to the embedding vectors. This preconditioning involves multiplying each embedding vector by a shared random matrix $\boldsymbol{S}$ with i.i.d. normal entries. By the Johnson-Lindenstrauss (JL) lemma (Dasgupta and Gupta, 2003), this preconditioning preserves the norms and inner products[1] of the embedding vectors with minimal distortion. A key property is that the embedding vectors after preconditioning follow a multivariate normal distribution, which has been formalized in Fact 2.3.

During the preconditioning stage, the matrix $\boldsymbol{S}$ is multiplied to all embedding vectors, allowing the analysis of PolarQuant to effectively treat the vectors being quantized as samples from a multivariate normal distribution. So for the analysis and design of PolarQuant we assume that our goal is to quantize a random vector with a multivariate Gaussian distribution. A critical insight is that the distribution of angles after random preconditioning becomes predictable and can be analytically derived, which enables the design of optimal quantization buckets. In the following lemma, we derive the polar distribution of a Gaussian vector.

**Lemma 3.2** (Distribution of a Gaussian Vector under RecursivePolar Transformation)**.** *For an integer power of two $d$, suppose that $\boldsymbol{x} \sim \mathcal{N}(0, I_d)$ is a random zero mean isotropic Gaussian random variable in dimension $d$. Let $\psi_d(\boldsymbol{x}) := \left(\psi^{(1)}, \psi^{(2)}, \dots \psi^{(\log_2 d)}\right)$ denote the set of polar angles obtained by applying the polar transformation defined in Definition 3.1 on $\boldsymbol{x}$. Denote the radius of $\boldsymbol{x}$ by $r = \|\boldsymbol{x}\|_2$. The joint probability density function for $\left(r, \psi^{(1)}, \psi^{(2)}, \dots \psi^{(\log_2 d)}\right)$ is the following:*

$$f_{R,\Psi_d}(r, \psi_d(\boldsymbol{x})) = f_R(r) \cdot \prod_{\ell=1}^{\log_2 d} f_{\Psi^{(\ell)}}\left(\psi^{(\ell)}\right), \quad (2)$$

*where $f_R(r)$ is the p.d.f. defined in Fact 2.1, $f_{\Psi^{(1)}}$ is p.d.f. of the uniform distribution over $[0, 2\pi)^{d/2}$:*

$$f_{\Psi^{(1)}} : [0, 2\pi)^{d/2} \to (2\pi)^{-d/2},$$

*and for every $\ell \in \{2, 3, \dots \log_2 d\}$ the p.d.f. $f_{\Psi^{(\ell)}}$ is the following:*

$$f_{\Psi^{(\ell)}}(\boldsymbol{\psi}) = \prod_{i=1}^{d/2^\ell} \frac{\Gamma(2^{\ell-1})}{2^{2^{\ell-1}-2} \cdot \Gamma(2^{\ell-2})^2} \sin^{(2^{\ell-1}-1)}(2\psi_i). \quad (3)$$

Lemma 3.2 demonstrates that the angles of Gaussian vectors in polar coordinates have independent distributions, as the probability density function is separable. Moreover, all angles within the same level share identical distributions. Specifically, at level $\ell$ all angles follow the distribution $\psi_i^{(\ell)} \sim \prod_{i=1}^{d/2^\ell} \frac{\Gamma(2^{\ell-1})}{2^{2^{\ell-1}-2} \cdot \Gamma(2^{\ell-2})^2} \sin^{2^{\ell-1}-1}\left(2\psi_i^{(\ell)}\right)$. This density becomes increasingly concentrated around $\pi/4$, particularly at higher levels $\ell$. This property is highly beneficial for reducing quantization error for the angles at higher levels.

Figure 2 shows the distributions of polar angles across levels $\ell \in \{1, 2, 3, 4\}$ with and without random preconditioning. Without preconditioning (Figure 2a), the level-1 angles are irregular and non-uniform, making quantization challenging. With random preconditioning (Figure 2b), the angles at level-1 become uniform,

[1] For our implementation, we use the Hadamard matrix for $\boldsymbol{S}$, which preserves norms and inner products exactly, but removes the independence across projected coordinates assumed in our theoretical analysis.

**Algorithm 1** PolarQuant: Quantization with RecursivePolar Transformation

**input:** embedding $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, precondition matrix $\boldsymbol{S} \in \mathbb{R}^{d \times d}$, bit width $b$
$\mathbf{R}_i, \boldsymbol{\Psi}_i^{(1)}, \ldots, \boldsymbol{\Psi}_i^{(\log_2 d)} \leftarrow \textsc{RecursivePolar}(\boldsymbol{X}_i \cdot \boldsymbol{S})$ for $i \in [n]$[2]
Find partition intervals and centroids $(I_k^{(\ell)}, \theta_k^{(\ell)})_{k \in [2^b]}$ of $\boldsymbol{\Psi}^{(\ell)}$ that optimize Equation (4)
$\boldsymbol{J}_i^{(\ell)} \leftarrow \text{Quantize}\boldsymbol{\Psi}_i^{(\ell)}$ using codebook $(I_k^{(\ell)}, \theta_k^{(\ell)})_{k \in [2^b]}$ for all $i \in [n]$ and $\ell \in [\log_2 d]$
**output:** $\mathbf{R}, \boldsymbol{\Psi}^{(1)}, \ldots, \boldsymbol{\Psi}^{(\log_2 d)}$, $(I_k^{(\ell)}, \theta_k^{(\ell)})_{k \in [2^b]}$ and $\boldsymbol{J}^{(1)}, \ldots, \boldsymbol{J}^{(\log_2 d)}$

**Procedure** $\textsc{DeQuant}(\boldsymbol{r}, (\boldsymbol{j}^{(\ell)})_{\ell \in [\log_2 d]}, (\theta_k^{(\ell)})_{k \in [2^b]}, \boldsymbol{S})$
**for** $\ell = \log_2 d, \ldots, 1$ **do**
  **for** $j = 1, \ldots, d/2^\ell$ **do**
    $i \leftarrow \boldsymbol{j}_j^{(\ell)}$
    $\boldsymbol{r}_{2j-1}^{(\ell-1)} \leftarrow \boldsymbol{r}_j^{(\ell)} \cdot \cos \theta_i^{(\ell)}$
    $\boldsymbol{r}_{2j}^{(\ell-1)} \leftarrow \boldsymbol{r}_j^{(\ell)} \cdot \sin \theta_i^{(\ell)}$
**output:** $\boldsymbol{r}^{(0)} \cdot \boldsymbol{S}^\top$

and deeper levels concentrate around $\pi/4$, matching Lemma 3.2. This confirms that preconditioning produces well-behaved angle distributions, enabling more effective data-independent quantization. More details can be found in Appendix B.1.

### 3.3 PolarQuant and Main Theorem

RecursivePolar transforms a given vector into radius and $\log_2 d$ angles. Due to the bounded nature of angular values, quantization can be applied without requiring the storage of additional metadata such as minimum and maximum values, effectively eliminating quantization overheads.

In addition, Lemma 3.2 shows that the angular components at different levels after the preconditioning are independent. Hence, each angle can be quantized independently, minimizing the overall quantization error without requiring joint quantization. Therefore, we can independently optimize the quantization scheme for angles at each level $l \in \{1, 2, \ldots, \log_2 d\}$ to minimize the quantization error, leading to both computational efficiency and high accuracy. Furthermore, Lemma 3.2 provides an explicit input distribution to quantization. The quantization performance can be improved when the input distribution is known apriori because one can construct an optimized codebook.

[2]Here $\boldsymbol{X}_i, \mathbf{R}_i, \boldsymbol{\Psi}_i^{(\ell)}$ and $\boldsymbol{J}_i^{(\ell)}$ denote the $i$-th row vectors of $\boldsymbol{X}, \mathbf{R}, \boldsymbol{\Psi}^{(\ell)}$ and $\boldsymbol{J}^{(\ell)}$, respectively.

Motivated by these observations, we propose PolarQuant, which first applies the random preconditioning to the input vectors, and recursively transforms them into polar coordinates, and finally quantizes the angles obtained from different polar transform levels. We provide a pseudocode of PolarQuant in Algorithm 1.

Our main result and error bound are proved in the following.

**Theorem 3.3.** *For any $d$-dimensional vector $\boldsymbol{x}$, the polar quantization scheme in Algorithm 1 with $\boldsymbol{S}$ having independent entries from $\mathcal{N}(0, 1/d)$ uses $O(\log 1/\varepsilon)$ bits per coordinate + the space necessary to store $\|\boldsymbol{x}\|_2$, while reconstructing a vector $\boldsymbol{x}'$ from such a representation satisfying*

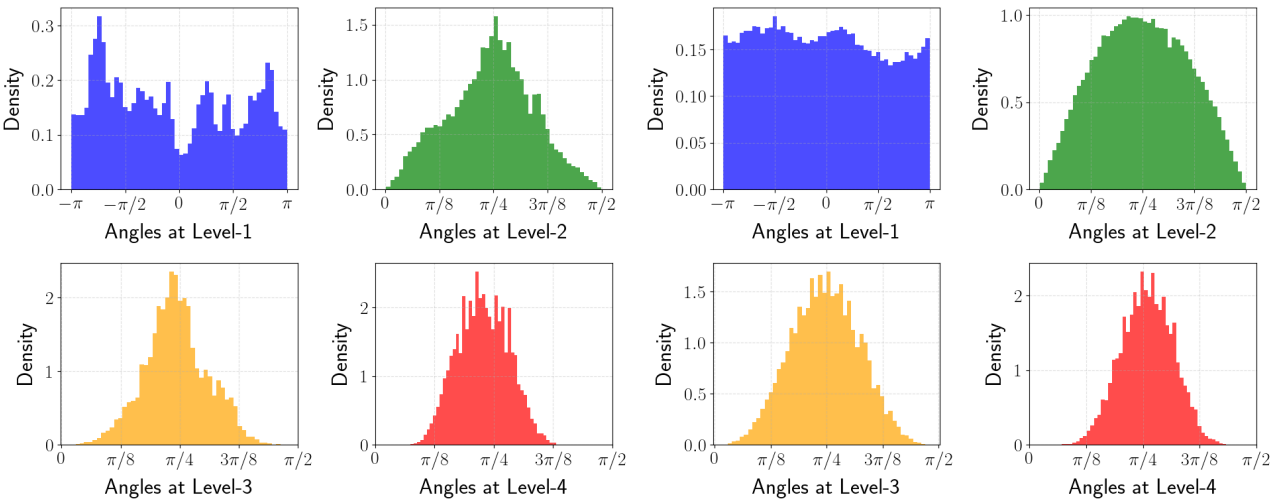
$$\mathop{\mathbb{E}}_{\boldsymbol{S}}[\|\boldsymbol{S}\boldsymbol{x} - \boldsymbol{x}'\|_2^2] = \varepsilon \cdot \|\boldsymbol{x}\|_2^2.$$

By Johnson-Lindenstrauss lemma, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are arbitrary vectors and $\boldsymbol{S}$ is sampled independently of $\boldsymbol{x}$ and $\boldsymbol{y}$, then $\langle \boldsymbol{S}\boldsymbol{x}, \boldsymbol{S}\boldsymbol{y} \rangle$ approximates $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ with a large probability and hence the reconstructions of $\boldsymbol{S}\boldsymbol{x}$ and $\boldsymbol{S}\boldsymbol{y}$ also approximate $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ from the above theorem. We note that a scheme which uses a deterministic $\varepsilon$-net $\mathcal{N}$ of the unit sphere $\mathbb{S}^{d-1}$, with $|\mathcal{N}| = O(1/\varepsilon)^d$ and rounds the vector $\hat{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|_2$ also uses $O(\log 1/\varepsilon)$ bits per coordinate while achieving the above bounds in the worst case instead of in expectation over the Gaussian distribution. But our construction (i) gives the flexibility to vary the size of the codebook used per each level depending on the resource constraints and as the above theorem shows, can approach the same quality as pinning to an $\varepsilon$-net on average, (ii) does not need to store a $|\mathcal{N}|$-size codebook which is impractical even for modest sizes of $d$ and (iii) has a fast decoding/encoding implementation.

**Empirical Method for Codebook Construction.** Consider an angle $\psi_i^{(\ell)}$ at some level $\ell$. According to Lemma 3.2, its values lie within the range $[0, \pi/2]$ for $\ell \geq 2$ and $[0, 2\pi)$ for $\ell = 1$ with a probability density function given by $f_\ell(\psi_i^{(\ell)})$ described above. The goal of quantization to $b$-bits is to partition the range $[0, \pi/2]$ (or $[0, 2\pi)$ in case of $\ell = 1$) into $2^b$ intervals $I_1^{(\ell)}, I_2^{(\ell)}, \cdots I_{2^b}^{(\ell)}$ and find corresponding centroids $\theta_1^{(\ell)}, \theta_2^{(\ell)}, \ldots \theta_{2^b}^{(\ell)}$ such that the following is mean squared error is minimized:

$$\mathop{\mathbb{E}}_{\psi_i^{(\ell)} \sim f_\ell(\psi_i^{(\ell)})} \left[ \sum_{j \in [2^b]:\ \psi_i^{(\ell)} \in I_j^{(\ell)}} \left| \psi_i^{(\ell)} - \theta_j^{(\ell)} \right|^2 \right]. \quad (4)$$

This problem involves a continuous domain and it is hard to find the optimal solution in general.




(a) without random preconditioning

(b) with random preconditioning

Figure 2: Distributions of angles of polar transformed key embeddings (a) with and (b) without random preconditioning. Preconditioning flattens the angle distribution and removes outliers which allows angle quantization more accurately. Angles at deeper levels concentrate around $\pi/4$.

Since we have an explicit formula for the p.d.f. of angle $\psi_i^{(\ell)} \sim f_\ell(\psi_i^{(\ell)}) = \frac{\Gamma(2^{\ell-1})}{2^{2^{\ell-1}-2}\cdot\Gamma(2^{\ell-2})^2} \sin^{2^{\ell-1}-1}\left(2\psi_i^{(\ell)}\right)$ the (sub)optimal interval partitions and centroids can be efficiently computed using numerical methods. We make use of k-means clustering on 1D angle values.

## 4 KV CACHE QUANTIZATION VIA POLARQUANT

To validate the effectiveness of PolarQuant, we apply it to the KV cache mechanism, a common optimization used in autoregressive Transformers to enable efficient token generation. Formally, given a stream of $(\boldsymbol{q}_1, \boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{q}_n, \boldsymbol{k}_n, \boldsymbol{v}_n)$, where $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i \in \mathbb{R}^d$ are query, key and value embeddings at the $i$-th generation step for $i \in [n]$. Let $\boldsymbol{K}_{:i}, \boldsymbol{V}_{:i} \in \mathbb{R}^{i\times d}$ be the matrices formed by stacking $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_i$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_j$ row-wise, respectively. The output is computed as:

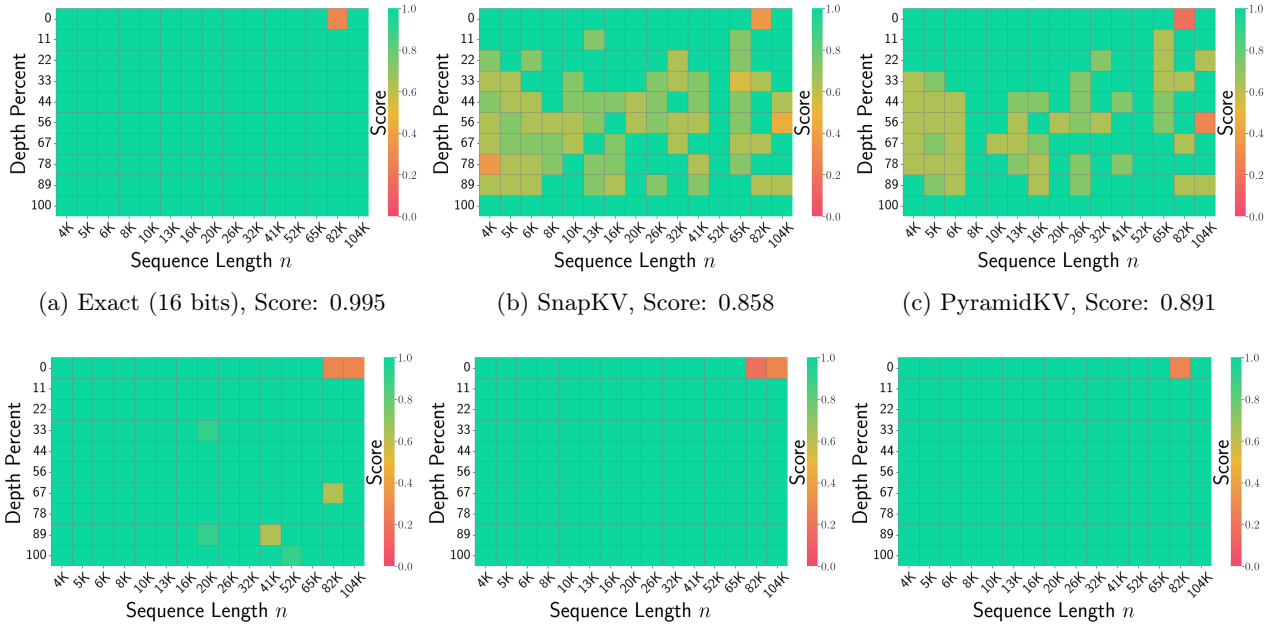
$$\text{softmax}\left(\frac{\boldsymbol{q}_i \boldsymbol{K}_{:i}^\top}{\sqrt{d}}\right)\boldsymbol{V}_{:i}. \tag{5}$$

To enable fast token generation, the KV embeddings $\boldsymbol{K}_{:i}$ and $\boldsymbol{V}_{:i}$ are cached. However, storing them in floating-point precision requires substantial memory, especially for long-context inputs. To address this issue, we leverage PolarQuant to quantize the embeddings prior to caching, significantly reducing memory usage. Let $\widehat{\boldsymbol{K}}_{:i}, \widehat{\boldsymbol{V}}_{:i} \in \mathbb{R}^{i\times d}$ be their dequantizated forms using the DEQUANT procedure with a preconditioning matrix $\boldsymbol{S}$ in Algorithm 1. The output in Equation (5) is then approximated as:

$$\text{softmax}\left(\frac{\boldsymbol{q}_i \boldsymbol{S} \cdot \widehat{\boldsymbol{K}}_{:i}^\top}{\sqrt{d}}\right)\widehat{\boldsymbol{V}}_{:i}\boldsymbol{S}^\top. \tag{6}$$

In the naïve case, storing each $d$-dimensional embedding requires $d \cdot b_{\text{FPN}}$ bits, where $b_{\text{FPN}}$ denotes the number of bits used for floating-point representation. Using PolarQuant, we quantize the angles across $\log_2 d$ levels with $b$ bits each, while retaining the centroid values in $b_{\text{FPN}}$ bits. This reduces the per-vector memory requirement to $(b_{\text{FPN}} + (d-1)b)$ bits. For instance, in the case of `Llama-3.1-8B-Instruct` with $d = 128$ and $b_{\text{FPN}} = 16$, setting $b = 3$ achieves a compression ratio approximately 5.16. As shown in Section 5, this reduction comes with only minimal performance degradation when applied to the KV cache in LLMs across various tasks.

**Practical Implementation.** PolarQuant recursively reduces the dimension of radii by half until the input reaches dimension 1. We apply the polar transformation recursively for a fixed number of level $L = 4$. As a result, for an embedding of dimension $d$, we obtain $d/16$ radii values and $15d/16$ angle values. We assign different bit widths to each level: $b = 4$ bits for the first level, and $b = 2$ bits for the remaining levels. This is because the range of angle at the first level $[0, 2\pi)$ is 4 times wider than the subsequent levels, which are constrained to $[0, \pi/2]$. Hence, the representation of a block of 16 coordinates requires $b_{\text{FPN}} + 32 + 8 + 4 + 2 = b_{\text{FPN}} + 46$ bits that translates to $62/16 = 3.875$ bits per coordinate when $b_{\text{FPN}} = 16$ bits.




(a) Exact (16 bits), Score: 0.995 (b) SnapKV, Score: 0.858 (c) PyramidKV, Score: 0.891

(d) KIVI, Score: 0.984 (e) HQQ, Score: 0.990 (f) PolarQuant, Score: 0.995

Figure 3: Needle-In-A-Haystack test using `Llama-3.1-8B-Instruct`. The test spans different depths and context lengths ranging from 4K to 104K. Green/red colors indicate high/low recall scores (higher is better). PolarQuant achieves the best performance without any degradation in accuracy.

Although our analysis assumes a random Gaussian matrix, in practice we use a random orthonormal matrix $\boldsymbol{S} \in \mathbb{R}^{d\times d}$ generated via the QR decomposition of a matrix with i.i.d. $\mathcal{N}(0,1)$ entries in practice. Various choices for the preconditioning matrix are further explored in Appendix B.3.

We implement PolarQuant using the PyTorch framework. Since the smallest supported data type is represented in 8 bits (i.e., `torch.uint8`), we pack quantized angle indices into 8-bit units. To accelerate computation on GPUs, we implement custom CUDA kernels for two key operations: (1) the product of a query vector with the dequantized key cache and (2) the product of attention scores with the dequantized value cache as per Equation (6). We provide an in-depth runtime analysis in Section 5.3 and Appendix B.4, comparing PolarQuant with other quantization methods, and observe that our kernel implementations achieve latency comparable to the full precision baseline. A more optimized CUDA kernel can be expected to beat the latency of full precision baseline.

# 5 EXPERIMENTS

Experiments are performed with a single NVIDIA RTX A6000 GPU with 48GB VRAM. More detailed implementation and various ablations are in Appendix B.

## 5.1 Needle-In-A-Haystack

We first evaluate our method using the "Needle-In-A-Haystack" test (Kamradt, 2023), which measures LLM's ability to retrieve a specific sentence (the "needle") placed in an arbitrary location of a long document (the "haystack"). Following the experimental setup of Fu et al. (2024), we conduct the evaluation using the `Llama-3.1-8B-Instruct` model, with input sequence lengths ranging from 4K to 104K tokens. The evaluation is based on the recall score by comparing the hidden sentence. We compare PolarQuant to SnapKV (Li et al., 2024) and PyramidKV (Cai et al., 2024) as well as quantization methods including KIVI (Liu et al., 2024b) and Half-Quadratic Quantization (HQQ) (Badri and Shaji, 2023), where we use their public implementations provided by MInference (Jiang et al., 2024) for all methods except HQQ. HQQ is evaluated using its original repository. All methods are set to a compression ratio of approximately 0.25, i.e., required memory is roughly $\times 0.25$ the full KV cache. As shown in Figure 3, quantization-based methods (e.g., KIVI, HQQ, PolarQuant) consistently outperform token-level compression techniques (e.g., SnapKV, PyramidKV). Among the quantization methods, PolarQuant achieves the highest recall, matching the performance of the baseline with 16-bit precision, demonstrating the effectiveness of our proposed approach.

Table 1: LongBench-E (Bai et al., 2023) results of various KV cache compression methods on `Llama-3.1-8B-Instruct` and `Qwen2.5-14B-Instruct`. The compression ratio is set to approximately 0.25 for all methods. Bold indicates the best-performing compression method for each model.

| Method | Single-QA | Multi-QA | Summarization | Few-shot | Synthetic | Code | Average |
|---|---|---|---|---|---|---|---|
| `Llama-3.1-8B-Instruct` | | | | | | | |
| Exact (16 bits) | 45.71 | 45.32 | 26.69 | 68.62 | 59.25 | 46.17 | 48.63 |
| StreamingLLM | 25.68 | 35.79 | 20.90 | 56.91 | 58.81 | 32.07 | 38.36 |
| PyramidKV | 36.80 | 41.54 | 18.91 | 64.88 | 59.68 | 42.38 | 44.03 |
| SnapKV | 38.23 | 42.61 | 19.07 | 64.65 | 59.60 | 43.28 | 44.57 |
| KIVI | 43.38 | 37.81 | **27.44** | 68.60 | 58.67 | 44.29 | 46.70 |
| HQQ | **45.02** | 42.78 | 22.10 | 67.84 | 56.62 | 45.13 | 46.58 |
| PolarQuant (ours) | 45.01 | **45.44** | 26.59 | **68.81** | **59.76** | **46.68** | **48.71** |
| `Qwen2.5-14B-Instruct` | | | | | | | |
| Exact (16 bits) | 49.30 | 63.31 | 25.25 | 70.29 | 61.17 | 55.06 | 54.06 |
| StreamingLLM | 26.81 | 46.31 | 20.15 | 54.80 | 37.50 | 39.05 | 37.44 |
| PyramidKV | 39.61 | 56.52 | 18.51 | 64.25 | **61.00** | 52.43 | 48.72 |
| SnapKV | 40.02 | 56.36 | 19.13 | 64.24 | 60.67 | 53.79 | 49.03 |
| KIVI | 46.33 | 58.62 | 24.62 | 69.16 | 58.83 | 54.35 | 51.99 |
| HQQ | 48.28 | **59.17** | 25.21 | 69.05 | 59.83 | 54.22 | 52.63 |
| PolarQuant (ours) | **48.52** | 58.82 | **28.85** | **69.86** | 60.50 | **54.65** | **53.53** |

### 5.2 End-to-end Generation on LongBench

We run various KV cache compression algorithms for LongBench datasets (Bai et al., 2023), which encompass diverse long-text scenarios including single/multi-document question-answering, summarization, few-shot learning, synthetic tasks, and code completion. We particularly choose a test set with a more uniform length distribution (LongBench-E). Since the number of generated tokens is small compared to the input sequence length across all datasets, we preserve all new streamed query, key, and value pairs from the generation stage in full precision (16 bits) for all methods. We evaluate PolarQuant against the baseline methods using in Section 5.1 as well as StreamingLLM (Xiao et al., 2023) on both `Llama-3.1-8B-Instruct` and `Qwen2.5-14B-Instruct`.

As reported in Table 1, PolarQuant consistently achieves the highest average performance across all evaluated tasks outperforming the next-best method (KIVI) by 2.01 on `Llama-3.1-8B-Instruct` and 1.54 on `Qwen2.5-14B-Instruct`. These improvements demonstrate the effectiveness of our polar-coordinate quantization strategy in preserving model quality under aggressive memory constraints.

### 5.3 Latency Analysis of Operations

We evaluate the latency of core operations involving the quantized KV cache during the autoregressive decoding phase. We measure the wall-clock time of matrix-vector multiplications involving $\boldsymbol{qK}^\top$ and $\boldsymbol{aV}$ where $\boldsymbol{a} = \text{softmax}(\boldsymbol{qK}^\top/\sqrt{d}) \in \mathbb{R}^n$. We generate synthetic query and KV embeddings with varying sequence lengths $n$ from 8K to 1M, run 500 independent trials, and report the average latency. We mimic the setting in `Llama-3.1-8B-Instruct` where $d = 128$ and the KV cache operates under grouped-query attention (GQA) (Ainslie et al., 2023), i.e., the query has 32 heads and the KV have 8 heads. As shown in Figure 4, for $\boldsymbol{qK}^\top$ computation (with inner dimension $d = 128$), PolarQuant exhibits slightly higher latency than the full-precision baseline. However, it significantly outperforms other quantization methods such as HQQ and KIVI. For the $\boldsymbol{aV}$ operation (with inner dimension $n \gg d$), PolarQuant matches the runtime of full-precision computation while still maintaining its memory-saving advantage. These results demonstrate the efficiency of our optimized kernel implementation. We believe that with further low-level kernel optimization, the latency of PolarQuant can be reduced even more, offering even greater speed-accuracy trade-offs in real-world applications. We additionally conduct the end-to-end latency analysis and provide the results in Appendix B.4.

### 5.4 Generalization to Vector Search

To evaluate the versatility of PolarQuant on non-LLM tasks, we conducted experiments on the SIFT1M vector search benchmark Jegou et al. (2010). We compared PolarQuant against standard uniform quantization under a strict 4-bit budget. As shown Table 2, PolarQuant significantly outperforms uniform quantization across all retrieval metrics. For instance, Recall@1 improves from 0.2498 to 0.6428, while Recall@32 reaches

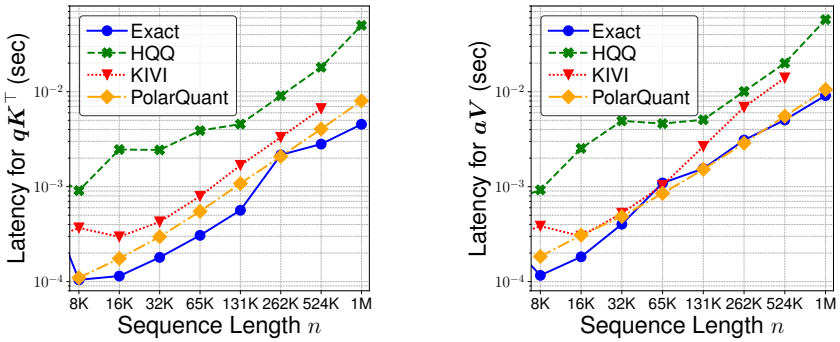



Figure 4: Latency comparison of core operations involving quantization, i.e., $\boldsymbol{qK}^\top$ (left) and $\boldsymbol{aV}$ (right). PolarQuant achieves competitive latency compared to full-precision and is superior to others.

Table 2: Retrieval performance comparison on the SIFT1M vector search benchmark under a 4-bit quantization budget.

| **Method** | **Recall@1** | **Recall@2** | **Recall@4** | **Recall@8** | **Recall@16** | **Recall@32** |
|---|---|---|---|---|---|---|
| Uniform Quantization | 0.2498 | 0.3283 | 0.4108 | 0.4949 | 0.5778 | 0.6498 |
| PolarQuant (ours) | **0.6428** | **0.8077** | **0.9165** | **0.9690** | **0.9899** | **0.9964** |

near-perfect levels (0.9964). These results demonstrate that angle-space quantization preserves the underlying nearest-neighbor structure of high-dimensional embeddings far more effectively than Cartesian-based uniform methods.

## 6 CONCLUSION

We propose PolarQuant, a novel quantization method applied to angles in multi-level polar coordinates. We connect it to the random preconditioning which allows us to formalize the angle distribution to be quantized. We provide rigorous theoretical bounds on quantization error. When applied to the KV cache compression problem, PolarQuant significantly reduces memory requirements during LLM inference while maintaining model performance. The principles underlying our method extend beyond KV cache compression, offering potential applications in LLM weight quantization and general vector similarity search problems.

## Acknowledgements

IH was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00406715 and No. RS-2025-23523958).

## References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. (2023). Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901.

Babenko, A. and Lempitsky, V. (2014). Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938.

Badri, H. and Shaji, A. (2023). Half-quadratic quantization of large machine learning models.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. (2023). Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Bhattacharya, A., Freund, Y., and Jaiswal, R. (2022). On the k-means/median cost function. *Information Processing Letters*, 177:106252.

Bin, X., Cui, J., Yan, W., Zhao, Z., Han, X., Yan, C., Zhang, F., Zhou, X., Wu, Q., and Liu, Z. (2025). Real-time indexing for large-scale recommendation by streaming vector quantization retriever. *arXiv preprint arXiv:2501.08695*.

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

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198.

Dasgupta, S. and Gupta, A. (2003). An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65.

Dong, S., Cheng, W., Qin, J., and Wang, W. (2024). Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*.

Fu, Y., Panda, R., Niu, X., Yue, X., Hajishirzi, H., Kim, Y., and Peng, H. (2024). Data engineering for scaling language models to 128k context. *arXiv preprint arXiv:2402.10171*.

Ge, T., He, K., Ke, Q., and Sun, J. (2013). Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2946–2953.

Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., and Kumar, S. (2020). Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR.

Jegou, H., Douze, M., and Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Jiang, H., LI, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A. H., Li, D., Lin, C.-Y., et al. (2024). Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Kamradt, G. (2023). Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

Kang, H., Zhang, Q., Kundu, S., Jeong, G., Liu, Z., Krishna, T., and Zhao, T. (2024). Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*.

Kim, J., Park, J., Cho, J., and Papailiopoulos, D. (2024). Lexico: Extreme kv cache compression via sparse coding over universal dictionaries. *arXiv preprint arXiv:2412.08890*.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. (2024). Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*.

Liu, Q., Dong, X., Xiao, J., Chen, N., Hu, H., Zhu, J., Zhu, C., Sakai, T., and Wu, X.-M. (2024a). Vector quantization for recommender systems: a review and outlook. *arXiv preprint arXiv:2405.03110*.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. (2024b). Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Vargaftik, S., Basat, R. B., Portnoy, A., Mendelson, G., Itzhak, Y. B., and Mitzenmacher, M. (2022). Eden: Communication-efficient and robust distributed mean estimation for federated learning. In *International Conference on Machine Learning*, pages 21984–22014. PMLR.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *NeurIPS*.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2023). Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Yang, J. Y., Kim, B., Bae, J., Kwon, B., Park, G., Yang, E., Kwon, S. J., and Lee, D. (2024). No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*.

Yue, Y., Yuan, Z., Duanmu, H., Zhou, S., Wu, J., and Nie, L. (2024). Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*.

Zandieh, A., Daliri, M., and Han, I. (2024). Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. *arXiv preprint arXiv:2406.03482*.

Zhang, T., Yi, J., Xu, Z., and Shrivastava, A. (2024). Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*.

Zhu, J., Jin, M., Liu, Q., Qiu, Z., Dong, Z., and Li, X. (2024). Cost: Contrastive quantization based semantic tokenization for generative recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 969–974.

## Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Not Applicable
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Not Applicable
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

# Instructions for Paper Submissions to AISTATS 2026: Supplementary Materials

## A Omitted Proofs

### A.1 Proof of Fact 2.1

**Fact A.1.** *For any positive integer $d$, if $\boldsymbol{x} \in \mathbb{R}^d$ is a zero mean unit variance isotropic Gaussian random variable in dimension $d$, i.e., $\boldsymbol{x} \sim \mathcal{N}(0, \boldsymbol{I}_d)$, then its 2-norm, denoted by $r := \|\boldsymbol{x}\|_2$, follows a generalized gamma distribution with the following probability density for any $r \geq 0$:*

$$f_R(r) = \frac{2}{2^{d/2} \cdot \Gamma(d/2)} r^{d-1} \exp\left(-r^2/2\right) \tag{1}$$

*Proof.* The cumulative distribution function (c.d.f) of the random variable $R$ can be computed as follows:

$$F_R(r) := \Pr_{\boldsymbol{x}}[\|\boldsymbol{x}\|_2 \leq r] = \Pr_{\boldsymbol{x}}[\|\boldsymbol{x}\|_2^2 \leq r^2] = \frac{1}{\Gamma(d/2)} \gamma(d/2, r^2/2),$$

where the last equality is because the squared norm of $\boldsymbol{x}$, by definition, is Chi-squared random variable. Differentiating the above c.d.f gives us the p.d.f of $R$:

$$f_R(r) = \frac{2}{2^{d/2} \cdot \Gamma(d/2)} r^{d-1} \exp\left(-r^2/2\right). \qquad \square$$

### A.2 Error Bounds for Polar Quantization

**Lemma A.2.** *If $X$ and $Y$ are independent non-negative random variables that are sampled from the generalized gamma distribution with probability density function $f_Z(z) = \frac{2}{2^{d/2} \cdot \Gamma(d/2)} z^{d-1} \exp(-z^2/2)$ so that $X^2, Y^2 \sim \chi_d^2$, then the distribution of $\Theta = \tan^{-1}(Y/X)$ has the pdf*

$$f_\Theta(\theta) = \frac{\Gamma(d)}{2^{d-2}\Gamma(d/2)^2} \cdot \sin^{d-1}(2\theta), \quad 0 \leq \theta \leq \pi/2.$$

*We also have that $\mu_\Theta = \mathbb{E}[\Theta] = \pi/4$ and $\sigma_\Theta = \sqrt{\mathrm{Var}(\Theta)} = O(1/\sqrt{d})$.*

*Proof.* Since $X$ and $Y$ are i.i.d., their joint distribution function is the following:

$$f_{X,Y}(x, y) = f_Z(x) \cdot f_Z(y) = \left(\frac{2}{2^{d/2} \cdot \Gamma(d/2)}\right)^2 (x \cdot y)^{d-1} \exp\left(-(x^2 + y^2)/2\right).$$

Now by changing the coordinates from Cartesian to polar we can represent the above distribution as a joint distribution over variables $r = \sqrt{x^2 + y^2}, \theta = \tan^{-1}(y/x)$ with $r \geq 0$ and $\theta \in [0, \pi/2)$ as follows:

$$f_{R,\Theta}(r, \theta) = r \cdot f_{X,Y}(r\cos\theta, r\sin\theta) = \frac{r^{2d-1}}{2^{2d-3} \cdot \Gamma(d/2)^2} \exp\left(-r^2/2\right) \cdot \sin^{d-1}(2\theta).$$

Since the joint probability distribution of $r, \theta$ is a separable function, we can deduce the marginal probability

distribution of $\theta$ as follows:

$$\begin{aligned}
f_\Theta(\theta) &= \int_0^\infty f_{R,\Theta}(r,\theta)dr \\
&= \frac{\sin^{d-1}(2\theta)}{2^{2d-3}\cdot\Gamma(d/2)^2}\cdot\int_0^\infty r^{2d-1}\exp\left(-r^2/2\right)dr \\
&= \frac{\sqrt{2\pi}\cdot\sin^{d-1}(2\theta)}{2^{2d-2}\cdot\Gamma(d/2)^2}\cdot\int_{-\infty}^\infty \frac{|r|^{2d-1}}{\sqrt{2\pi}}e^{-r^2/2}dr \\
&= \frac{\sqrt{2\pi}\cdot\sin^{d-1}(2\theta)}{2^{2d-2}\cdot\Gamma(d/2)^2}\cdot\mathop{\mathbb{E}}_{r\sim\mathcal{N}(0,1)}\left[|r|^{2d-1}\right] \\
&= \frac{\Gamma(d)}{2^{d-2}\cdot\Gamma(d/2)^2}\cdot\sin^{d-1}(2\theta),
\end{aligned}$$

where the last equality above follows from Fact 2.2. For a proof of the mean and variance bound see Lemma A.2. From the symmetry of $f_\Theta(\cdot)$ around $\theta=\pi/4$, it is clear that $\mu_\Theta=\pi/4$. We now bound $\sigma_\Theta$.

$$\begin{aligned}
\mathrm{Var}(\Theta) &= \frac{\Gamma(d)}{2^{d-2}\Gamma(d/2)^2}\int_0^{\pi/2}(\theta-\pi/4)^2\sin^{d-1}(2\theta)\,\mathrm{d}\theta \\
&= \frac{\Gamma(d)}{2^{d-2}\Gamma(d/2)^2}\int_{-\pi/4}^{\pi/4}\theta^2\sin^{d-1}(2\theta+\pi/2)\,\mathrm{d}\theta \\
&= \frac{2\Gamma(d)}{2^{d-2}\Gamma(d/2)^2}\int_0^{\pi/4}\theta^2\cos^{d-1}(2\theta)\,\mathrm{d}\theta \\
&= \frac{\Gamma(d)}{2^{d-3}\Gamma(d/2)^2}\int_0^{\pi/4}\theta^2(1-2\sin^2\theta)^{d-1}\mathrm{d}\theta \\
&\le \frac{\Gamma(d)}{2^{d-3}\Gamma(d/2)^2}\int_0^{\pi/4}\theta^2(1-\frac{8\theta^2}{\pi^2})^{d-1}\mathrm{d}\theta \qquad (\text{since } \sin(x)\ge 2x/\pi \text{ for } 0\le x\le\pi/2) \\
&\le \frac{\Gamma(d)}{2^{d-3}\Gamma(d/2)^2}\int_0^{\pi/4}\theta^2\exp\left(-\frac{8(d-1)\theta^2}{\pi^2}\right)\mathrm{d}\theta \qquad (\text{since } 1-x\le\exp(-x)) \\
&\le \frac{\Gamma(d)}{2^{d-3}\Gamma(d/2)^2}\int_0^{\infty}\theta^2\exp\left(-\frac{8(d-1)\theta^2}{\pi^2}\right)\mathrm{d}\theta\ .
\end{aligned}$$

Substituting $\beta=8(d-1)\theta^2/\pi^2$, we have

$$\mathrm{Var}(\Theta)\le\frac{\Gamma(d)}{2^{d-3}\Gamma(d/2)^2}\cdot\frac{\pi^3}{32\sqrt{2}(d-1)^{3/2}}\int_0^\infty\beta^{1/2}\exp(-\beta)\,\mathrm{d}\beta\le C\frac{\Gamma(d)}{2^{d-3}\Gamma(d/2)^2(d-1)^{3/2}}.$$

Assuming $d$ is even and using Stirling approximation, we get

$$\Gamma(d)=\frac{d!}{d}\approx\frac{\sqrt{2\pi d}\,(d/e)^d}{d}\,\text{and}\,\Gamma(d/2)=\frac{(d/2)!}{(d/2)}\approx\frac{\sqrt{2\pi(d/2)}(d/2e)^{d/2}}{d/2}.$$

Therefore,

$$\mathrm{Var}(\Theta)\le C'\frac{2^d\sqrt{d}}{2^{d-3}(d-1)^{3/2}}\le\frac{C''}{d-1}$$

where $C''$ is a universal constant independent of $d$ hence proving the theorem. □

### A.3 Proof of Lemma 3.2

**Lemma A.3** (Distribution of a Gaussian Vector under RecursivePolar Transformation)**.** *For an integer power of two $d$, suppose that $\boldsymbol{x}\sim\mathcal{N}(0,I_d)$ is a random zero mean isotropic Gaussian random variable in dimension $d$. Let $\psi_d(\boldsymbol{x}):=\left(\psi^{(1)},\psi^{(2)},\dots\psi^{(\log_2 d)}\right)$ denote the set of polar angles obtained by applying the polar transformation*

*defined in Definition 3.1 on* $\boldsymbol{x}$*. Denote the radius of* $\boldsymbol{x}$ *by* $r = \|\boldsymbol{x}\|_2$*. The joint probability density function for* $\left(r, \psi^{(1)}, \psi^{(2)}, \dots \psi^{(\log_2 d)}\right)$ *is the following:*

$$f_{R,\Psi_d}(r, \psi_d(\boldsymbol{x})) = f_R(r) \cdot \prod_{\ell=1}^{\log_2 d} f_{\Psi^{(\ell)}}\left(\psi^{(\ell)}\right), \tag{2}$$

*where* $f_R(r)$ *is the p.d.f. defined in Fact 2.1,* $f_{\Psi^{(1)}}$ *is p.d.f. of the uniform distribution over* $[0, 2\pi)^{d/2}$*:*

$$f_{\Psi^{(1)}} : [0, 2\pi)^{d/2} \to (2\pi)^{-d/2},$$

*and for every* $\ell \in \{2, 3, \dots \log_2 d\}$ *the p.d.f.* $f_{\Psi^{(\ell)}}$ *is the following:*

$$f_{\Psi^{(\ell)}}(\boldsymbol{\psi}) = \prod_{i=1}^{d/2^\ell} \frac{\Gamma(2^{\ell-1})}{2^{2^{\ell-1}-2} \cdot \Gamma(2^{\ell-2})^2} \sin^{(2^{\ell-1}-1)}(2\psi_i). \tag{3}$$

*Proof.* The proof is by induction on $d$. First for the base of induction we prove the result in dimension $d = 2$. So we prove that for a 2-dimensional random Gaussian vector $\boldsymbol{y} = (y_1, y_2) \in \mathbb{R}^2$ if $(r, \theta)$ is the polar representation of this vector then following holds:

$$f_{R,\Theta}(r, \theta) = \frac{1}{2\pi} \cdot r \exp\left(-r^2/2\right),$$

To prove this, let $f_Y(\boldsymbol{y})$ be the probability density function of the vector random variable $\boldsymbol{y}$. We know $\boldsymbol{y}$ has a normal distribution so we have:

$$f_{R,\Theta}(r, \theta) = r \cdot f_Y(\boldsymbol{y}) = r \cdot \frac{1}{2\pi} e^{-\frac{y_1^2+y_2^2}{2}} = \frac{1}{2\pi} \cdot r e^{-\frac{r^2}{2}},$$

where the first equality above follows from the change of variable from $(y_1, y_2)$ to $r = \sqrt{y_1^2 + y_2^2}$ and $\theta = \tan^{-1}(y_2/y_1)$. This proves the base of induction for $d = 2$.

Now we prove the inductive step. Suppose that the lemma holds for dimension $d/2$ and we want to prove it for dimension $d$. Denote $\theta := \psi^{(\log_2 d)}$, $\boldsymbol{\phi}_1 := \left(\psi^{(l)}_{1:d/2^{l+1}}\right)_{\ell=1}^{\log_2 d-1}$, $\boldsymbol{\phi}_2 := \left(\psi^{(l)}_{d/2^{\ell+1}+1:d/2^\ell}\right)_{\ell=1}^{\log_2 d-1}$, $r_1 := \left\|\boldsymbol{x}_{1:d/2}\right\|$, and $r_2 := \left\|\boldsymbol{x}_{d/2+1:d}\right\|$. Essentially we sliced all the angle vectors $\psi^{(\ell)}$ in half and named the collection of first half vectors $\boldsymbol{\phi}_1$ and the collection of second halves $\boldsymbol{\phi}_2$. Using the definition of $\psi^{(\ell)}$'s in Definition 3.1, $\boldsymbol{\phi}_1$ is exactly the polar transformation of $\boldsymbol{x}_{1:d/2}$, and $\boldsymbol{\phi}_2$ is the polar transformation of $\boldsymbol{x}_{d/2+1:d}$, so by the definition of $\psi_d(\boldsymbol{x})$ in the lemma statement we have $\boldsymbol{\phi}_1 = \psi_{d/2}(\boldsymbol{x}_{1:d/2})$ and $\boldsymbol{\phi}_2 = \psi_{d/2}(\boldsymbol{x}_{d/2+1:d})$. Thus, we can write:

$$\begin{aligned}
f_{R,\Psi_d}(r, \psi_d(\boldsymbol{x})) &= f_{R,\Theta,\Phi_1,\Phi_2}(r, \theta, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \\
&= r \cdot f_{R_1,R_2,\Phi_1,\Phi_2}(r\cos\theta, r\sin\theta, \boldsymbol{\phi}_1, \boldsymbol{\phi}_2) \\
&= r \cdot f_{R_1,\Phi_1}(r\cos\theta, \boldsymbol{\phi}_1) \cdot f_{R_2,\Phi_2}(r\sin\theta, \boldsymbol{\phi}_2) \\
&= r \cdot f_{R,\Psi_{d/2}}(r_1, \boldsymbol{\phi}_1) \cdot f_{R,\Psi_{d/2}}(r_2, \boldsymbol{\phi}_2),
\end{aligned} \tag{7}$$

where the third line above follows from the change of variable from $(r_1, r_2) = (r\cos\theta, r\sin\theta)$ to $r = \sqrt{r_1^2 + r_2^2}$ and $\theta = \tan^{-1}(r_2/r_1)$. In the fourth line above we used the definition of $\theta = \psi^{(\log_2 d)} = \tan^{-1}\left(\frac{\left\|\boldsymbol{x}_{d/2+1:d}\right\|_2}{\left\|\boldsymbol{x}_{1:d/2}\right\|_2}\right)$ from Definition 3.1.

Now if we let $f_{\Psi_{d/2}}(\boldsymbol{\phi}_1) := \prod_{\ell=1}^{\log_2 d-1} f_{\Psi^{(\ell)}}\left(\psi^{(\ell)}_{1:d/2^{\ell+1}}\right)$ and $f_{\Psi_{d/2}}(\boldsymbol{\phi}_2) := \prod_{\ell=1}^{\log_2 d-1} f_{\Psi^{(\ell)}}\left(\psi^{(\ell)}_{d/2^{\ell+1}+1:d/2^\ell}\right)$, by the inductive hypothesis we have $f_{R,\Psi_{d/2}}(r_1, \boldsymbol{\phi}_1) = \frac{2}{2^{d/4} \cdot \Gamma(d/4)} r_1^{d/2-1} \exp\left(-r_1^2/2\right) \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_1)$ and $f_{R,\Psi_{d/2}}(r_2, \boldsymbol{\phi}_2) =$

$\frac{2}{2^{d/4}\cdot\Gamma(d/4)} r_2^{d/2-1} \exp\left(-r_2^2/2\right) \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_2)$. Plugging these values into Equation (7) gives:

$$\begin{aligned} f_{R,\Psi_d}(r, \psi_d(\boldsymbol{x})) &= \frac{4\cdot(r_1 r_2)^{d-1}}{2^{d/2}\Gamma(d/4)^2} \exp\left(-r^2/2\right) \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_1) \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_2) \\ &= \frac{2r^{d-1}\cdot \sin^{d/2-1}(2\theta)}{2^{3d/2-2}\cdot\Gamma(d/4)^2} e^{-r^2/2} \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_1) \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_2) \\ &= f_R(r) \cdot \frac{\Gamma(d/2)\cdot \sin^{d/2-1}(2\theta)}{2^{d/2-2}\cdot\Gamma(d/4)^2} f_{\Psi_{d/2}}(\boldsymbol{\phi}_1) \cdot f_{\Psi_{d/2}}(\boldsymbol{\phi}_2) \\ &= f_R(r) \cdot f_{\Psi_d}\left(\psi_d(\boldsymbol{x})\right), \end{aligned} \tag{8}$$

which completes the inductive proof of this lemma. □

### A.4 Proof of Theorem 3.3

**Theorem 3.3.** *For any $d$-dimensional vector $\boldsymbol{x}$, the polar quantization scheme in Algorithm 1 with $\boldsymbol{S}$ having independent entries from $\mathcal{N}(0, 1/d)$ uses $O(\log 1/\varepsilon)$ bits per coordinate + the space necessary to store $\|\boldsymbol{x}\|_2$, while reconstructing a vector $\boldsymbol{x}'$ from such a representation satisfying*

$$\mathop{\mathbb{E}}_{\boldsymbol{S}}[\|\boldsymbol{S}\boldsymbol{x} - \boldsymbol{x}'\|_2^2] = \varepsilon \cdot \|\boldsymbol{x}\|_2^2.$$

Let $\boldsymbol{x} \in \mathbb{R}^d$ be a unit vector without loss of generality and let $\boldsymbol{S}$ be a $d \times d$ random Gaussian matrix with entries all being drawn from $\mathcal{N}(0, 1/d)$. Now, $\boldsymbol{S}\boldsymbol{x}$ is the vector that is quantized by the recursive scheme. By the 2-stability of Gaussian random variables, all the entries of $\boldsymbol{S}\boldsymbol{x}$ are independent and are distributed as random variables drawn from $\mathcal{N}(0, 1/d)$.

Now we consider the error of quantizing a vector with independent normal random variables. Note that scaling the vector does not change the quantization analysis and hence is valid for the quantization error for $\boldsymbol{S}\boldsymbol{x}$. Suppose that $X, Y$ are random variables as in Lemma A.2 so that $X^2, Y^2 \sim \chi_d^2$ and define $R = \sqrt{X^2 + Y^2}$ and $\Theta = \tan^{-1}(Y/X)$. Let $\{\theta_1, \ldots, \theta_k\} \subseteq [0, \pi/2]$ be a codebook and define $\Theta' = \text{round}(\Theta)$ to be the $\theta_i$ nearest to $\Theta$ and $R'$ be an approximation of $R$. We then define

$$X' = R' \cdot \cos(\text{round}(\Theta)) \qquad \text{and} \qquad Y' = R' \cdot \sin(\text{round}(\Theta))$$

to be the reconstructions of $X$ and $Y$ respectively from the rounding scheme, which rounds $\Theta$ using the codebook and the radius $R$ using a recursive approximation. The reconstruction error is defined as

$$\begin{aligned} (X - X')^2 + (Y - Y')^2 &= (R\cos(\Theta) - R'\cos(\Theta'))^2 + (R\sin(\Theta) - R'\sin(\Theta'))^2 \\ &= (R\cos(\Theta) - R\cos(\Theta') + R\cos(\Theta') - R'\cos(\Theta'))^2 \\ &\quad + (R\sin(\Theta) - R\sin(\Theta') + R\sin(\Theta') - R'\sin(\Theta'))^2. \end{aligned}$$

Using the fact that $2ab \le (1/\alpha)\cdot a^2 + \alpha \cdot b^2$ for any $\alpha > 0$, we get $(a+b)^2 = a^2 + 2ab + b^2 \le (1+1/\alpha)a^2 + (1+\alpha)b^2$ for any $\alpha > 0$. Therefore we have that for any $\alpha > 0$,

$$\begin{aligned} (X - X')^2 + (Y - Y')^2 &\le (1+1/\alpha)R^2(\cos(\Theta) - \cos(\Theta'))^2 + (1+\alpha)(R - R')^2\cos(\Theta')^2 \\ &\quad + (1+1/\alpha)R^2(\sin(\Theta) - \sin(\Theta'))^2 + (1+\alpha)(R - R')^2\sin(\Theta')^2 \\ &\le (2 + 2/\alpha)R^2(\Theta - \Theta')^2 + (1+\alpha)(R - R')^2, \end{aligned}$$

where we used that fact that $\sin(\cdot)$ and $\cos(\cdot)$ are 1-Lipschitz and that $\sin^2(\Theta') + \cos^2(\Theta') = 1$.

Now, restricting $\alpha \in (0, 1)$ we get that

$$\mathbb{E}[(X - X')^2 + (Y - Y')^2] \le \frac{4\,\mathbb{E}[R^2]}{\alpha}\,\mathbb{E}[(\Theta - \Theta')^2] + (1+\alpha)\,\mathbb{E}[(R - R')^2]$$

using the independence of $R$ and $\Theta$ and the fact that $\Theta'$ is a deterministic function of $\Theta$ given the codebook $\{\theta_1, \ldots, \theta_k\}$.

When $d = 1$, we call $\mathbb{E}[(X - X')^2 + (Y - Y')^2]$ to be $\text{error}_0$ since it is the expected error in the reconstruction at level 0 and similarly, we call $\mathbb{E}[(R-R')^2] = \text{error}_1$ since it is the error in level 1. We also call $\mathbb{E}[(\Theta-\Theta')^2] = \text{quant}_1$ since it is the error of quantizing angles in that level. We therefore have

$$\text{error}_0 \leq \frac{4\,\mathbb{E}[R^2]}{\alpha} \cdot \text{quant}_1 + (1+\alpha) \cdot \text{error}_1.$$

Given a vector $(X_1, \ldots, X_d)$, where each $X_i \sim N(0,1)$, let $(X'_1, \ldots, X'_d)$ be the reconstructions using $t = \log_2(d)$-level quantization as described in the introduction. Extending the above definitions, we say that $\text{error}_i$ is the total expected error in the reconstructions of level $i$ coordinates in the quantization scheme. Using the above, inequality, we get

$$\begin{aligned}\text{error}_0 \leq \frac{4d}{\alpha} \cdot \text{quant}_1 + \frac{4d}{\alpha}(1+\alpha) \cdot \text{quant}_2 + \frac{4d}{\alpha}(1+\alpha)^3 \cdot \text{quant}_2 \\ + \cdots + \frac{4d}{\alpha}(1+\alpha)^t \cdot \text{quant}_t + (1+\alpha)^{t+1} \cdot \text{error}_{t+1},\end{aligned}$$

where we use the fact that $\mathbb{E}[X_1^2 + \cdots + X_d^2] = d$. Since we store the top-level radius exactly, we have $\text{error}_{t+1} = 0$ and therefore,

$$\frac{\text{error}_0}{d} \leq \frac{4}{\alpha}\left(\text{quant}_1 + (1+\alpha) \cdot \text{quant}_2 + \cdots + (1+\alpha)^t \cdot \text{quant}_t\right).$$

For each level $i$, given $\varepsilon > 0$, we will now upper bound the size of the codebook for level $i$ so that $\text{quant}_i \leq \varepsilon$. It is clear that a codebook $\{0, \sqrt{\varepsilon}, 2\sqrt{\varepsilon}, \ldots, 2\pi\}$ has a size $\lceil 2\pi/\sqrt{\varepsilon} \rceil + 1$ and has $\text{quant}_i \leq \varepsilon$ irrespective of the distribution of the angles. We would like to use the fact that the distribution of angles gets concentrated with increasing level $i$ to obtain better bounds on the size of the codebook. To that end, we prove the following lemma.

**Lemma A.4.** *Let $X \in [0, \pi/2]$ be an arbitrary random variable with $Var(X) = \sigma^2$. Given $x$ and a set $S = \{x_1, \ldots, x_k\}$, define $d(x, S) = \min_{i\in[k]} |x_i - x|$. Define*

$$Var_k(X) := \min_{S:|S|=k} E[d(X,S)^2].$$

*Given $\varepsilon > 0$, for $k = \Omega(\log(1/\sigma)/\sqrt{\varepsilon})$, we have $Var_k(X) \leq \varepsilon \cdot Var_1(X) = \varepsilon \cdot \sigma^2$.*

The lemma shows that as variance decreases, we can get tighter approximation bounds using the same value of $k$. The proof of this lemma is similar to that of Lemma 2 in Bhattacharya et al. (2022).

*Proof.* Let $\mu = \mathbb{E}[X]$. Consider the interval $[\mu-\sigma, \mu+\sigma]$ and consider the points $S_1 = \{\mu, \mu \pm \varepsilon\sigma, \mu \pm 2\varepsilon\sigma, \ldots, \mu \pm \sigma\}$. Note that $|S_1| = 3 + 2/\varepsilon$. We have

$$\mathbb{E}[d(X, S_1)^2 \mid X \in [\mu - \sigma, \mu + \sigma]] \leq \varepsilon^2\sigma^2$$

since every point in the interval $[\mu - \sigma, \mu + \sigma]$ has some point in $S_1$ that is at most $\varepsilon\sigma$ away.

Now define $S_2 = \{\mu \pm (1+\varepsilon)^0\sigma, \mu \pm (1+\varepsilon)^1\sigma, \ldots, \}$ where the exponent $i$ extends until we have $\mu + (1+\varepsilon)^i\sigma \geq \pi/2$ and $\mu - (1+\varepsilon)^i\sigma \leq 0$. Note that we have $|S_2| \leq O(\log(1/\sigma)/\varepsilon)$. Suppose $|X - \mu| \in [(1+\varepsilon)^i\sigma, (1+\varepsilon)^{i+1}\sigma]$, then $d(X, S_2) \leq \varepsilon(1+\varepsilon)^i\sigma \leq \varepsilon|X - \mu|$. Now,

$$\mathbb{E}[d(X, S_1 \cup S_2)^2] \leq \mathbb{E}[\max(\varepsilon\sigma, \varepsilon|X-\mu|)^2] \leq 2\varepsilon^2\sigma^2.$$

Thus by putting $\varepsilon \coloneqq \sqrt{\varepsilon/2}$ above, if $k = \Omega(\log(1/\sigma)/\sqrt{\varepsilon})$, then $\text{Var}_k(X) \leq \varepsilon\sigma^2$. □

Now we consider bounding $\text{quant}_i$ by picking an appropriate codebook for level $i$. For all $i > 0$, given a codebook $\mathcal{C} = \{\theta_1, \ldots \theta_{|\mathcal{C}|}\}$, we have $\text{quant}_i = \mathbb{E}[(\Theta - \text{round}(\Theta, \mathcal{C}))^2]$ where $\Theta$ is $\arctan(Y/X)$ with $Y^2, X^2 \sim \chi^2_{2^{i-1}}$. From the lemma above, when $i > 1$, we have $\text{Var}(\Theta) \leq C/(2^{i-1} - 1)$ and hence, the above lemma shows that there exists a codebook $\mathcal{C}$ of size $|\mathcal{C}| = \Theta(i/\sqrt{\varepsilon})$,

$$\text{quant}_i \leq \frac{\varepsilon}{2^{i-1}}. \tag{9}$$

If we use a codebook of size $O(1/\sqrt{\varepsilon})$ for level 1, we have $\text{quant}_1 \leq \varepsilon$ and as described above, for $i \geq 1$, if we use a codebook of size $|\mathcal{C}| = \Theta(i/\sqrt{\varepsilon})$, then $\text{quant}_i \leq \varepsilon/(2^{i-1})$ from which we obtain that

$$\frac{\text{error}_0}{d} \leq \frac{4}{\alpha}(2\varepsilon + \frac{1+\alpha}{\varepsilon}(2\varepsilon) + \left(\frac{1+\alpha}{2}\right)^2 (2\varepsilon) + \cdots) \leq O(\varepsilon)$$

picking $\alpha = 1/2$. Now we compute the number of bits necessary to store the quantized representation of a $d$-dimensional vector. In level $i$, we have $d/2^i$ independent samples of $\Theta$ to be quantized and therefore, we need to store $O(\frac{d}{2^i}\log(i/\sqrt{\varepsilon}))$ bits for indexing into the codebook for level $i$. Adding over all the levels, we store $O(d\log 1/\sqrt{\varepsilon})$ bits for representing the $d$-dimensional vector and therefore $O(\log 1/\varepsilon)$ bits per coordinate to achieve an average reconstruction error of $\varepsilon$.

# B Additional Experiments

## B.1 Effectiveness of Preconditioning

We explore the effectiveness of preconditioning, which makes the input distribution Gaussian. To verify practical effect of preconditioning, we choose a single prompt from Qasper dataset in LongBench-E (Bai et al., 2023) and extract the corresponding KV cache. To observe how preconditioning improves, we transform the KV cache into 4-level polar coordinates and plot their angle distributions of the key cache. Note that the first level angles are range in $[0, 2\pi)$ and the rest are in $[0, \pi/2]$. The results are illustrated in Figure 2. As shown in Lemma 3.2, the distribution of angles get predictably sharper around $\pi/4$ as the level increases. Moreover, we observe that at the first level the preconditioning flattens the angle distribution and removes outliers. This allows us to quantize angles in the KV cache more accurately.

## B.2 Offline versus Online Codebook Construction

We investigate two variants of PolarQuant: one using "online" codebook construction and the other using "offline" construction. The online variant builds the angle codebook during the prefill phase for each input prompt and layer, while the offline variant uses a single precomputed codebook shared across all prompts, layers, and attention heads. Our empirical findings suggest that, after preconditioning, the angle distribution remains largely consistent across inputs. This supports the use of a shared offline codebook. Although the online variant incurs a one-time clustering cost during each prefill phase, it can yield a better quantization. Table 3 compares the two variants. Surprisingly, the offline version achieves slightly better overall results. Furthermore, as shown in Table 4, the online construction significantly slows down the prefill stage, making the offline approach better choice in practice.

Table 3: Results on LongBench-E of variants of PolarQuant using `Llama-3.1-8B-Instruct`.

| Method | Single-QA | Multi-QA | Summarization | Few-shot | Synthetic | Code | Average |
|---|---|---|---|---|---|---|---|
| PolarQuant (Online, Hadamard) | 45.25 | 44.52 | 26.35 | 68.63 | 59.57 | 46.91 | 48.54 |
| **PolarQuant (Offline, Hadamard)** | 45.01 | 45.44 | 26.59 | 68.81 | 59.76 | 46.68 | 48.71 |
| PolarQuant (Offline, Random) | 44.71 | 44.72 | 26.43 | 68.58 | 60.08 | 45.20 | 48.29 |

## B.3 Choice of Preconditioning Matrix

We explore two types of preconditioning matrices: a random rotation matrix and a Hadamard matrix. The random rotation matrix is obtained by applying QR decomposition to a Gaussian matrix. Although our theoretical analysis does not directly apply to these matrices, both preserve norms and inner products almost exactly, which helps minimize performance degradation from quantization. In particular, we compare the two preconditioning matrices on the LongBench-E benchmark. As shown in Table 3, the Hadamard matrix yields better performance (48.71 vs. 48.29 for the random rotation), and is thus selected for our final evaluation. In addition, the Hadamard matrix supports faster vector multiplication in $O(d\log d)$ time, suggesting potential for further optimization.

### B.4 End-to-end Latency Analysis

We additionally evaluate the end-to-end wall-clock runtimes of both prefill and token generation stages. Using the `Llama-3.1-8B-Instruct` model with an input prompt length of 16,384, we measure the time to generate 1,024 tokens for each method. Table 4 summaries the result. Token eviction approaches (SnapKV, PyramidKV, and HeadKV) demonstrate faster generation times compared to exact and quantization methods, though at the cost of lower quality. Among quantization approaches, our PolarQuant algorithms achieve 14% faster generation time than the KIVI while maintaining superior performance. These results demonstrate that PolarQuant offers advantages in both computational efficiency and model performance. For faster prefill time, we suggest to use offline codebook construction as its runtime significantly decreases without k-means++ clustering, though sacrificing subtle performance. We remain a better codebook construction method for future work. To achieve faster prefill times, we recommend using offline codebook construction, as it significantly reduces runtime by eliminating the need for clustering, though this results in a modest performance trade-off. We leave even better codebook construction approaches for future research. In addition, the preconditioning adds negligible overhead.

Table 4: Wall-clock runtime comparisons of various KV cache compression methods. The input sequence length is $n = 16{,}384$ and the number of generated tokens is 1,024.

| Method | Prefill Time (sec) | Generation Time (sec) |
|---|---|---|
| Exact (16 bits) | 2.934 | 38.374 |
| SnapKV | 3.438 | 34.053 |
| PyramidKV | 3.428 | 32.732 |
| HeadKV | 3.300 | 34.401 |
| KIVI | 3.590 | 49.564 |
| PolarQuant (online) | 11.633 | 44.448 |
| PolarQuant (offline) | 3.364 | 44.097 |

### B.5 Comparison with RHT-based Vector Quantization

We further evaluate PolarQuant against a randomized Hadamard transform (RHT) based quantization method, named Eden Vargaftik et al. (2022). Eden is a high-performing RHT-based quantization method originally designed for distributed mean estimation but applicable to vector quantization tasks. Using the official Eden implementation[3], we quantized the KV cache across all layers and heads of `Llama-3.1-8B-Instruct` at 4 bits, while PolarQuant operated at 3.875 bits. Both methods leverage the Hadamard transform to improve quantization efficiency but differ fundamentally in their quantization domain: PolarQuant performs quantization on angles in polar coordinate space after recursive polar transformation, whereas Eden quantizes vectors directly in Euclidean coordinate space after rotation.  presents the accuracy results on the LongBench benchmark across six task categories. Despite using fewer bits, PolarQuant achieves slightly higher average performance (48.7140) compared to Eden (48.6198), while both methods maintain accuracy close to the full-precision baseline (48.6283). Beyond accuracy, PolarQuant demonstrates substantial computational advantages through our optimized CUDA kernel design. Table Y shows the latency comparison for computing the attention operation $QK^T$, where $K$ represents the reconstructed key from quantization. PolarQuant operates dramatically faster than Eden across all sequence lengths, achieving a 155.23× speedup at sequence length 1,048,576. This performance gain stems from our kernel's exploitation of the polar-coordinate quantization structure, which enables efficient angle-based dequantization directly integrated into matrix operations. Unlike Eden, which requires full vector reconstruction before computing attention scores, PolarQuant's polar representation allows for on-the-fly dequantization during dot product computation, significantly reducing memory bandwidth requirements and computational overhead.

### B.6

[3] https://github.com/amitport/EDEN-Distributed-Mean-Estimation

Table 5: Accuracy comparison on LongBench benchmark. PolarQuant achieves higher average performance than Eden despite using fewer bits (3.875 vs 4 bits).

| **Method** | **Single-QA** | **Multi-QA** | **Summarization** | **Few-shot** | **Synthetic** | **Code** | **Average** |
|---|---|---|---|---|---|---|---|
| Exact (16 bits) | 45.71 | 45.32 | 26.69 | 68.62 | 59.25 | 46.17 | 48.63 |
| PolarQuant (3.875 bits) | 45.01 | 45.44 | 26.59 | 68.81 | 59.76 | 46.68 | **48.71** |
| Eden Vargaftik et al. (2022) (4 bits) | 45.65 | 44.24 | 26.78 | 69.29 | 59.04 | 46.72 | 48.62 |

Table 6: Latency comparison (seconds) for $QK^T$ computation across different sequence lengths from 4K to 1M. PolarQuant achieves up to 155× speedup over Eden through optimized CUDA kernel design.

| **Method** | **4K** | **8K** | **16K** | **32K** | **65K** | **131K** | **262K** | **524K** | **1M** |
|---|---|---|---|---|---|---|---|---|---|
| PolarQuant (3.875 bits) | 7.69e-5 | 1.10e-4 | 1.74e-4 | 2.96e-4 | 5.48e-4 | 1.07e-3 | 2.08e-3 | 4.02e-3 | 8.00e-3 |
| Eden Vargaftik et al. (2022) (4 bits) | 5.10e-3 | 8.17e-3 | 1.66e-2 | 3.38e-2 | 6.92e-2 | 1.42e-1 | 2.93e-1 | 6.03e-1 | 1.24 |
| **Speedup** | 66.3× | 74.5× | 95.1× | 114.0× | 126.4× | 133.6× | 141.1× | 149.8× | 155.2× |