SBVR: Summation of BitVector Representation for Efficient LLM Quantization

Wonjun Bang, Jongseok Park, Hongseung Yu, Kyungmin Bin*, Kyunghan Lee†

[1,3,†]Seoul National University
[2]University of California, Berkeley
[4] Samsung SDS
wjbang@snu.ac.kr js_park@berkeley.edu hongseung@snu.ac.kr
kyungmin.bin@samsung.com kyunghanlee@snu.ac.kr

Abstract

With the advent of large language models (LLMs), numerous Post-Training Quantization (PTQ) strategies have been proposed to alleviate deployment barriers created by their enormous parameter counts. Quantization achieves compression by limiting the number of representable points in the data. Therefore, the key to achieving efficient quantization is selecting the optimal combination of representation points, or codes, for the given data. Existing PTQ solutions adopt two major approaches to this problem: Round-To-Nearest (RTN)-based methods and codebook-based methods. RTN-based methods map LLM weights onto uniformly distributed integer grids, failing to account for the Gaussian-like weight distribution of LLM weights. Codebook-based methods mitigate this issue by constructing distribution-aware codebooks; however, they suffer from random and strided memory access patterns, resulting in degraded inference speed that is exacerbated by the limited size of GPU L1 cache. To overcome these limitations, we propose a novel LLM quantization method, SBVR (Summation of BitVector Representation), that enables Gaussian-like code representation in a hardware-friendly manner for fast inference. SBVR maps weight values to non-uniform representation points whose distribution follows the actual distribution of LLM weights, enabling more accurate compression. Additionally, we design a custom CUDA kernel that allows matrix-vector multiplication directly in the SBVR format without decompression, thereby enabling high-performance execution of SBVR-compressed models. Our evaluations of SBVR on various models demonstrate state-of-the-art perplexity and accuracy benchmark performance while delivering a 2.21×–3.04× end-to-end token-generation speedup over naive FP16 models in the 4-bit quantization regime.

1 Introduction

Large language models (LLMs) (Achiam et al. 2023; Dubey et al. 2024; Bai et al. 2023) have demonstrated remarkable performance across a wide range of tasks, including code generation, document summarization, and even solving complex mathematical problems. Since there are strong evidences indicating that model performance is roughly proportional to model size (Kaplan et al. 2020; Wei et al. 2022), many LLM researchers and vendors have focused on scaling up their models. However, this extreme scaling has led to substantial memory and computational resource requirements, thereby hindering easy access to LLMs.

To mitigate this issue, myriad quantization and compression methods have been proposed to reduce the memory footprint of LLMs. Most of these techniques employ post-training quantization (PTQ), which lowers the model's precision after it has been trained with full-precision weights.

One large group of post-training quantization methods relies on solutions based on Round-To-Nearest (RTN) quantization. These methods map LLM weights to uniformly distributed integer points while applying various techniques that reduce quantization-induced error. Since RTN only requires scale and quantized integer weights for dequantization and further computation, it is fast and hardware-friendly. However, uniformly distributed integers do not effectively capture LLM weights, which follow a Gaussian-like distribution, thereby possessing inherent inaccuracy.

Another group of methods treats weights as sets of multi-dimensional vectors and generates specific codebooks to encode the weight vectors. Optimized codebook design can mitigate the discrepancy between the actual weight distribution and the quantization points. However, large codebook sizes and random access to codebooks for dequantization often incur significant overhead, which causes inference speed degradation.

To address these issues from previous quantization methods, we propose SBVR, a "weighted sum of bitvectors" representation that enables codebook-level weight distribution capturing capability while demonstrating RTN-like hardware efficiency for fast inference. Here, bitvectors are vectors whose elements consist of 0 or 1.

Given a vector and a set of coefficients $\{a_1, a_2, \ldots, a_K\}$, a set of bitvectors can be selected such that the coefficient-weighted sum of these bitvectors closely approximates the vector by minimizing the Mean Squared Error (MSE). **Figure 1** provides an simplified example of these bitvector assignments.

To approximate LLM weights that follow a Gaussian-like distribution, the coefficients should be carefully determined to ensure that the subset sum of the coefficients can effectively represent the weight distribution. We empirically ob-

*This work was done during the PhD program at Seoul National University

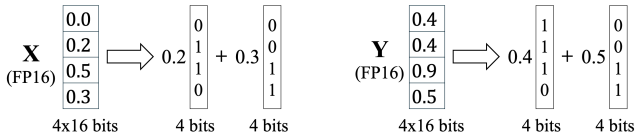
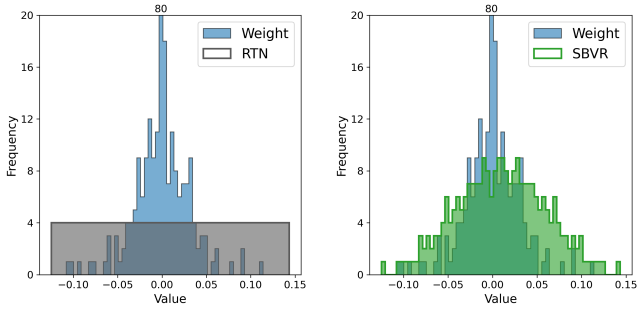


Figure 1: SBVR conversion of a given vector. Bitvectors for the weighted sum representation of a vector are obtained by identifying the set of bitvectors that minimizes the mean squared error (MSE) between the original vector and its weighted sum representation.



(a) RTN Quantization (b) SBVR Quantization

Figure 2: Comparison of quantization methods applied to a 256-element weight group extracted from the Llama-3.1-8B-Instruct model: (a) RTN quantization, which maps weights onto a uniform grid; (b) SBVR quantization, which effectively captures the Gaussian-like distribution of the weights.

served that the subset sum from a geometric series closely follows the actual weight distribution. Leveraging this observation, we designed a system that carefully selects a geometric series and applies data-dependent scaling and bias to the series, thereby allowing the scaled-and-shifted geometric series serves as effective coefficients that accurately capture the given vector distribution. **Figure 2** provides concrete support for this observation and idea.

The SBVR format, which represents the weighted sum of bitvectors, also provides computational benefits. By converting two floating-point vectors into the SBVR format, the inner product between them can be computed using inexpensive bitwise operations—specifically, bitwise AND and population count—instead of relying on expensive fused multiply-add (FMA) operations. **Figure 3** demonstrates this process of decomposing the inner product between two vectors into bitwise AND and population count operations. Building upon this concept, we developed a GPU-friendly General Matrix-Vector Multiplication (GEMV) kernel for SBVR-formatted weights and activations to enable faster inference.

In summary, the principal contributions of this paper are as follows:

- We identify inherent limitations in post-training quantization methods that must be addressed to further improve accuracy and meet stringent speed requirements.

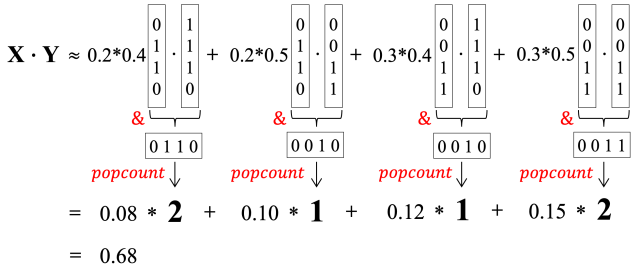


Figure 3: Computation of the inner product between two vectors represented as weighted sums of bitvectors. The inner product can be efficiently computed using a series of bitwise AND and population count operations.

- We propose SBVR, a novel quantization method based on a "weighted sum of bitvectors" representation, designed to achieve both high accuracy and fast inference for LLMs while effectively overcoming the aforementioned limitations.
- Experiments demonstrate that SBVR outperforms several prominent PTQ methods across various benchmarks, while also achieving considerable speedup in model inference.

2 Related Works

We organize the related work into three subsections: conventional RTN-based element-wise quantization approaches, codebook-based quantization approaches, and other approaches that do not directly fit into these two categories.

2.1 Conventional RTN-Based Quantizations

Many prominent 3-to-4-bit targeted quantization techniques rely on RTN-based quantization, employing unique strategies to mitigate quantization errors. Outlier suppression has emerged as a primary approach for reducing quantization error in RTN-based methods. One strategy involves giving attention to outlier channels (Lin et al. 2024), while another approach rotates model weights or activations to mitigate outlier values (Liu et al. 2024b; Ashkboos et al. 2024), both of which have achieved significant success. Error propagation methods for accuracy-preserving sequential quantization constitute another major branch of techniques. These methods utilize the Hessian matrix derived from inputs (Frantar et al. 2022) to minimize rounding errors from previous channels, or propagate quantization errors from each layer to subsequent layers (Arai and Ichikawa 2025) to optimize the quantization process. All the techniques above are valid for reducing quantization errors. However, since they all rely on Round-To-Nearest quantization that maps weights onto a uniformly distributed grid, this approach is naturally suboptimal according to the rate-distortion theorem, as it encodes Gaussian-like sources using codes from a uniform distribution.

2.2 Codebook-Based Quantizations

Methods that generate well-structured codebooks to capture the weight distribution of LLMs have been proposed and have become the new mainstream approach to quantization. AQLM (Egiazarian et al. 2024) proposes a multiple-codebook-based additive quantization method that performs effectively at extremely low bitrates. VPTQ (Liu et al. 2024a) further optimizes the codebook-based vector quantization method by maintaining additional codebooks for residuals and outliers. While a few methodologies attempt to mitigate the speed degradation caused by large codebook sizes (Tseng et al. 2024b), the majority of codebook-based approaches suffer from uncoalesced memory access problems and L1 cache misses due to exponentially growing codebooks.

2.3 Other Quantizaiton Techniques

Several methods attempt to compress LLMs by explicitly redefining the representation of floating-point numbers. (Micikevicius et al. 2022) introduce the FP8 format, which expresses floating-point numbers using 8 bits, such as E4M3 and E5M2. (Darvish Rouhani et al. 2023) propose the Microscaling format (MX), which uses a per-group scale factor and employs low-precision FP formats to represent each element within a group. By adopting shared scales and asymmetric representation for each element, this approach is well-suited for quantizing LLM weights, which exhibit non-uniform distributions.

3 Preliminaries

As stated in Section 1, SBVR is motivated by the observation that LLM weights follow a Gaussian-like distribution and are therefore more effectively compressed with Gaussian-shaped codepoints than with the uniformly spaced codepoints used by RTN quantization. Section 3.1 revisits this observation and presents the supporting information-theoretic background. Section 3.2 introduces essential GPU architectural concepts, offers guidelines for designing hardware-friendly kernels, and explains why codebook-based quantization methods become inefficient on GPUs due to non-consecutive memory access and related overheads.

3.1 Distribution of LLM Model Weights

Numerous studies have shown that the weight data of large language models (LLMs) closely follow a Gaussian distribution (Tseng et al. 2024a,b; Si, Jiang, and Shen 2025). Fortunately, lossy compression of a data source following a Gaussian distribution is a classic topic of information theory, where the rate-distortion theory (Shannon 1959) provides a theoretical lower bound to the achievable data rate under a fixed distortion when compressing a Gaussian source.

Given a Gaussian source $X \sim \mathcal{N}(\mu, \sigma^2)$, data rate of $R(D)$, and distortion of D from quantizer $\hat{X}$ measured by squared error, it is proven from the rate distortion theory that

$$R(D) \geq \frac{1}{2} log_2 \frac{\sigma^2}{D}, \tag{1}$$

where the equality holds when $\hat{X} \sim \mathcal{N}(\mu, \sigma^2 - D)$.

From this result, we can also observe that given a fixed data rate budget, or *bitrate*, of R, the lowest distortion D would be achieved when the quantizer closely follows a Gaussian distribution with a variance slightly below that of the source data distribution. However, conventional integer round-to-nearest (RTN) quantization is ill-suited to achieve this. In b-bit RTN quantization, the weight elements are partitioned into groups, where each group is scaled and biased to a predefined range of integers, and each element of the group is mapped to its nearest integer using the following equation:

$$\Delta = \frac{2^{b-1} - 1}{\max_i |w_i|} \tag{2}$$

$$q_i = \text{clip}\Big(\text{round}\big(\tfrac{w_i}{\Delta}\big),\ -(2^{b-1}-1),\ 2^{b-1}-1\Big) \tag{3}$$

Here, $\text{clip}(x, a, b) := \max\big(\min(x, b), a\big)$. While the biasing and scaling of conventional RTN quantization allow close reconstruction of the mean and the variance of the LLM weight data, rounding to a nearest integer in a fixed range fails to leverage the Gaussian distribution of the LLM weights, as it yields a uniformly distributed quantizer, which is proven to be suboptimal when compressing a Gaussian source.

3.2 GPU-friendly Kernel

A GPU-friendly kernel is essential for efficient execution of quantized LLMs. GPUs consist of numerous Streaming Multiprocessors (SMs), each capable of massively parallel computation across thousands of threads. GPUs feature a hierarchical memory structure with L1 and L2 caches, as well as global memory (DRAM), where access speed decreases from L1 cache to global memory. GPUs efficiently handle memory transactions through coalescing, accessing consecutive memory regions within thread groups (warps), thereby reducing transaction counts. Therefore, data layout that arranges related data consecutively in memory and kernel designs that leverage this memory layout are crucial for performance.

RTN-based quantization supports this requirement by storing scalar scales, biases, and quantized weights consecutively in memory, which can be computed with minimal memory overhead. Conversely, codebook-based quantization methods face inherent inefficiencies due to their non-consecutive memory accesses. Moreover, their codebooks scale exponentially with target bitwidth (2^L), often exceeding GPU's L1 cache capacity, leading to increased cache misses, extensive indexing overhead, codebook lookups, and reconstruction processes, which significantly degrade overall inference performance.

4 SBVR Design

This section describes our distribution-aware efficient LLM quantization algorithm, **SBVR** (Summation of BitVector Representation), which guarantees fast and accurate inference.

4.1 Design Overview

To allow hardware-efficient encoding of Gaussian LLM weights, SBVR is composed of two main components: the bit vectors and a set of coefficients shared by the bit vectors. Using the weighted sum of the coefficients and the bit vectors, which we denote as SBVR, we can efficiently implement tuning of the representation points while simultaneously enabling each data element to select the representation point with minimal error. For example, given a coefficient set $[a, b, c]$, our representation points become $[0, a, b, c, a+b, a+c, b+c, a+b+c]$, and each element of the vector can freely select one of these points by assigning either 0 or 1 to the three bit vector entries weighted by the coefficients. This yields a 3-bit[1] SBVR quantization, where we can arbitrarily select the distribution that best suits our objectives by adjusting a, b, and c. Although finding the optimal coefficient values for arbitrary data is an NP-hard problem, we observe that Gaussian-distributed LLM weights can be efficiently represented by setting the coefficients to a scaled-and-shifted version of a geometric series, as described in Section 4.2.

In terms of accuracy, we find it beneficial to group multiple weight elements, typically 128, into separate vectors and apply SBVR individually, similar to the grouped bias and scaling approaches used in existing RTN methods. This grouping facilitates parallelism during the encoding step and enhances accuracy by allowing optimization of the coefficient set for each vector. While the weight elements can be grouped in any orientation, we group weights along the GEMV inner-product dimension to optimize inner products within our fast inference kernel. Similarly, activations must also be converted into SBVR format to utilize our inference kernel efficiently. Thus, we propose two types of SBVR quantization algorithms: a slower but highly bit-efficient algorithm for offline weight encoding, and a lightweight, less bit-efficient algorithm for online activation encoding, as detailed in Sections 4.2 and 4.3.

The SBVR-formatted weights and activations are sent to a GPU-friendly, high-performance General Matrix-Vector Multiplication (GEMV) kernel. The kernel eliminates the need for explicit dequantization of weights and activations and replaces costly Fused Multiply-Add (FMA) operations with relatively inexpensive bitwise operations, as elaborated in Section 4.4. Additionally, we apply complementary techniques from other research alongside SBVR quantization to further enhance its quantization performance. Further details are provided in Section 4.5.

4.2 Offline Weight Quantization

We employ three variables—ratio (r), scale (s), and bias (b)—to generate a search space for our coefficient set. Each variable changes a different characteristic in the distribution of the representable points: the ratio changes the shape, the scale changes the variance, and the bias change the mean of the distribution. By adjusting each variable, we can fit our representable point distribution to that of the weight data.

[1]The memory overhead of the coefficient set is easily amortized by the large vector size.

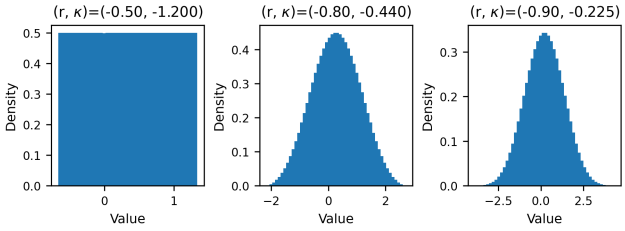


Figure 4: Generated representation point distributions for three distinct r values. The s and b values are set to 1 and 0, respectively, for demonstration purposes. Excess kurtosis κ represents the "tail thickness" of a distribution. The Gaussian distribution has zero excess kurtosis, while the uniform distribution has an excess kurtosis of -1.2. By changing the r value from -1 to -0.5, we can tune the excess kurtosis to better fit the representation points to the data distribution.

We denote the number of coefficients in the coefficient set as K, which in turn determines the target bitrate of the SBVR quantization.

$$\begin{gathered}\left\{\, s_j r_i^0 + b_k,\ s_j r_i^1 + b_k,\ \cdots,\ s_j r_i^{K-1} + b_k \,\right\} \\ 1 \le i \le |R|, \quad 1 \le j \le |S|, \quad 1 \le k \le |B|\end{gathered} \tag{4}$$

Eq 4 shows the coefficient set generated using a scaled-and-shifted geometric series with variables r_i, s_j, and b_k from the set of cadidate variables R, S and B, repectively. The size of the SBVR parameter search space is therefore $|R|\times|S|\times|B|$. **Figure 4** illustrates the effect of changing r on the generated representation points. As illustrated in the figure, entries in the search space can generate various forms of distributions, ranging from Uniform to Gaussian-like distributions. As such, SBVR is strictly a superset of RTN quantization and can handle many variants of Gaussian-like distributions in addition, enabling accurate quantization of the Gaussian-like LLM weights.

As stated in Eq 4, R, S and B determine the overall landscape of the search space. Therefore, it is crucial to judiciously determine the candidate sets R, S, and B for the parameters (r, s, b). We used the following equation 5 to 11 to define the candidate sets for the parameters.

$$R = \{\, r_i \mid 0 \le i < N_{\text{ratio}} \} \tag{5}$$

$$S = \{\, s_{\min} + (j+1)s_{\text{gran}} \mid 0 \le j < N_{\text{scale}} \} \tag{6}$$

$$B = \{\, b_{\min} + k \times b_{\text{gran}} \mid 0 \le k < N_{\text{bias}} \} \tag{7}$$

$$s_{\max} = 1.1\big(\max(D) - \min(D)\big), \tag{8}$$

$$b_{\max} = 2\,\text{abs}(\text{avg}(D))/K \tag{9}$$

$$s_{\min} = 2\,q_{95}(D), \quad b_{\min} = -b_{\max} \tag{10}$$

$$s_{\text{gran}} = \frac{s_{\max} - s_{\min}}{N_{\text{scale}}}, \quad b_{\text{gran}} = \frac{b_{\max} - b_{\min}}{N_{\text{bias}}} \tag{11}$$

Here, N_{ratio}, N_{scale}, and N_{bias} are the size of each search dimension that can be tuned, and D is the target weight data.

Algorithm 1: SBVR Encoding

Input: weight vector X, search space $\mathbb{S}$
Output: entry $e^\star$ minimizing mean-squared error (MSE)

$e^\star \leftarrow$ nil, bestMSE $\leftarrow \infty$
for all $e \in \mathbb{S}$ **do**
  sums $\leftarrow$ AllSubsetSums(e)
  errorSum $\leftarrow 0$
  for all $x \in X$ **do**
    $n \leftarrow$ Nearest(sums, x)
    errorSum $\leftarrow$ errorSum $+ (x - n)^2$
  end for
  mse $\leftarrow$ errorSum$/|X|$
  if mse $<$ bestMSE **then**
    $e^\star \leftarrow e$, bestMSE $\leftarrow$ mse
  end if
end for
return $e^\star$

Notes
(1) AllSubsetSums: Retrieves all subset sums from entry e
(2) Nearest: Return the sum that is closest to x

R consists of a superset of two disjoint sets, where each set is generated using the same procedure employed to generate S and B. $q_{95}(D)$ denotes the 95th percentile of the given data. Further details, such as the actual values used for the experiments, can be found in the Technical Appendix.

Algorithm 1 illustrates the process of determining the optimal entry from the search space for each weight vector. After selecting the entry that minimizes the mean squared error (MSE) for the given weight vector, the coefficients and bitvectors can be determined. Each element in the selected entry becomes a coefficient. For each element x_i in the weight vector, we identify the subset of elements from the entry whose subset sum is closest to x_i. We then mark the selected elements as 1 and the others as 0, forming a binary array. When using K-bit SBVR (SBVR-K), this K-dimensional binary array represents the array of the i^{th} elements of the bitvectors.

We observe that many vectors in the SBVR format share similar coefficient sets. To accelerate the search process, we maintain a cache of previously selected variables r, s, and b. The search algorithm first checks the cache to determine if any previously selected combinations of r, s, and b provide sufficient accuracy[2] before exploring the entire search space. In many cases, the cache hit rate reaches up to 90%, greatly reducing quantization time and significantly decreasing the size of the coefficient cache.

4.3 Online Activation Conversion on Decoding Stage

Unlike LLM weights, which are converted to the SBVR format offline, activations must be converted online. Consequently, the search-space-based method employed for weight conversion is computationally infeasible for activations due to its prohibitive overhead. Therefore, we adopt the conventional integer-grid method for activation conversion, which, while suboptimal in terms of accuracy, ensures significantly lower computational costs compared to exhaustive search-space traversal. First, we partition the activations into a set of vectors and predetermine their SBVR-format coefficients as power-of-two values. Then, for each activation vector, we compute a per-vector scale factor by dividing the absolute maximum value of the vector by 2^l, where l denotes the target bitwidth for the activations. Finally, during the actual conversion process, we multiply this per-vector scale factor by the power-of-two coefficients.

Equation 12 illustrates the final coefficients obtained after this conversion, with s representing the per-vector scale factor and the bias set to 0. We set l to 8 for near-lossless quantization of activations (Dettmers et al. 2022), minimizing the accuracy degradation inherent in integer-grid RTN quantization. The corresponding bitvectors are then obtained using the same procedure as in weight quantization.

$$\left\{-2^{(b-1)}s,\ s,\ 2s,\ \cdots,\ 2^{(b-2)}s\right\} \tag{12}$$

4.4 GPU-Friendly Kernel Design of SBVR

The goal of SBVR is not only to achieve accurate quantization of LLM weights but also to enable fast LLM inference during the decoding stage. To achieve high inference speed, we designed a GPU-friendly GEMV kernel. After converting weights and activations to SBVR, we separately store coefficients and bitvectors. Specifically, the bitvectors are partitioned into 32-bit integers and stored as a tensor in a contiguous memory region. For coefficients, we utilize a coefficient cache containing all coefficient sets required for decoding, as well as a coefficient index that identifies the specific coefficient set used by each K-bit bitvector set. For activations, we maintain predefined power-of-two coefficients in the GPU's constant memory, storing only the bitvectors and per-vector scale factors. At runtime, these scale factors are multiplied by the predefined coefficients to reconstruct the final coefficients.

During GEMV—the inner product between an activation vector and each weight row—the kernel first fetches the relevant bitvectors, coefficients, and scales, loading the SBVR-formatted weights and activations into registers. Then, the kernel multiplies corresponding coefficients and performs the inner product between the bitvectors of weights and activations. The inner products between bitvectors can be decomposed into two bitwise operations: bitwise AND and population count. After performing the AND operation between bitvectors, we apply the population count, which counts the number of set bits (1's) resulting from the bitwise AND operation performed immediately prior. Finally, we accumulate all multiplication results of the coefficients and population counts into a single FP16 element. For clarity, readers are encouraged to revisit **Figure 3**, which provides a comprehensive example illustrating the entire computation process.

[2]We maintain a moving average of the quantization error. If the error from the cached values is below this moving average, the cached values are used.

This design offers two key benefits. First, it eliminates the need for per-element de-quantization steps during computation. Second, when employing K-bit and N-bit SBVR quantization for weights and activations, respectively, only KN Fused Multiply-Add (FMA) operations are required per inner product between these two vectors. For instance, in the W4A8 configuration with 128-dimensional vectors, this approach reduces the number of FMA operations by a factor of $4 \times 8/128 = 4$. Consequently, this kernel design not only reduces memory traffic by loading only uint32-packed bitvectors but also decreases computationally expensive FMA operations by replacing them with efficient bitwise operations.

4.5 Additional Techniques Applied to SBVR

Since SBVR does not have inherent dependencies on other existing techniques, methods such as outlier suppression and error propagation can be combined orthogonally with SBVR to further enhance quantization accuracy. To ensure that LLM weights closely follow a Gaussian-like distribution and to mitigate the negative impact of outliers on quantization performance, we applied random Hadamard rotation merging (Liu et al. 2024b) to the weights prior to quantization. Additionally, we employed a coarse-grained variant of GPTQ (Frantar et al. 2022), which propagates quantization errors across multiple columns simultaneously, thereby further reducing quantization-induced errors.

5 Experiment

In this section, we introduce the experimental setup, including the models we used, evaluation metrics, baseline methods, test environment, and implementation details for the experiments. We also compare the benchmark results of our methodology, SBVR, with other baseline quantization methods.

5.1 Experiment Setting

Models. We test our methodology on various models with different sizes. We used Llama3 (Dubey et al. 2024) series models: Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct, and Llama-3.1-70B-Instruct. We also used the latest Qwen (Bai et al. 2023) models: Qwen3-0.6B, Qwen3-1.7B, Qwen3-8B, and Qwen3-14B. Finally, we tested the DeepSeek-R1 (Guo et al. 2025) distilled model, DeepSeek-R1-Distil-Qwen-7B.

Evaluation Metrics. To quantify the accuracy of the quantized model, we measure perplexity (PPL) on WikiText-2 (Merity et al. 2017) and evaluate performance by measuring zero-shot accuracy on the following six commonsense benchmarks for the models listed in Section 5.1: ARC-Easy, ARC-Challange (Clark et al. 2018), CommonsenseQA (Talmor et al. 2019), HellaSwag (Zellers et al. 2019), PIQA (Bisk et al. 2020), and WinoGrande (Sakaguchi et al. 2021). To measure inference speed during the decoding stage, we evaluated the Time-Per-Output-Token (TPOT) of the models when quantization was applied.

Baseline Methods. We compare our proposed method, SBVR, against several popular and state-of-the-art baseline quantization methods in the 4-bit regime, including GPTQ (Frantar et al. 2022), AWQ (Lin et al. 2024), SpinQuant (Liu et al. 2024b), and VPTQ (Liu et al. 2024a). For inference speed evaluation, we use RTN-based methods equipped with custom kernels optimized for speed (GPTQ, AWQ), as well as the codebook-based method VPTQ as baselines. To evaluate the effectiveness of the SBVR kernel against other W4A8 methods, we also include QQQ (Zhang et al. 2024) in our speed comparison experiments. Note that due to the limited availability of publicly released VPTQ models on Hugging Face (Wolf et al. 2019), some benchmarks could not be measured for this method.

Test Environment & Implementation Details. All speed measurements are conducted on a single machine equipped with NVIDIA GeForce RTX 3090 GPUs. The SBVR encoding algorithm is implemented using PyTorch. Customized kernels for GEMV operations and online activation conversion are implemented in C++ and CUDA, and are imported into PyTorch as an extension. For the decoding stage, we fix the SBVR configuration to W4A8, with SBVR-4 for weights and SBVR-8 for activations. Unlike the decoding stage, we can leverage tensor cores to accelerate massive General Matrix Multiplication (GEMM) operations, amortizing the dequantization overhead across multiple tokens. Therefore, we design a prefill kernel that decompresses SBVR weights into FP16 and transfers the recovered weight segments to tensor cores for GEMM computation.

5.2 Experimental Results

This section presents the evaluation results for the accuracy and inference speed of our quantization method.

Evaluation on Llama Models. Table 1 presents the perplexity (PPL) and commonsense benchmark results for the Llama 3 model series. For both the 1B and 8B models, SBVR outperforms RTN-based quantization methods, including AWQ, GPTQ, and SpinQuant, and achieves results comparable to the codebook-based quantization method VPTQ. Specifically, SBVR achieves lower PPL and superior performance across various commonsense benchmarks.

We further conduct an ablation study to assess the effectiveness of the SBVR coding scheme by implementing a quantization method that retains all SBVR techniques, such as Hadamard rotations and GPTQ error propagation, but replaces the Gaussian representation points with the integer grid employed by existing RTN-based methods. As shown in the table, SBVR consistently surpasses this ablation implementation in terms of both PPL and commonsense task performance, providing additional evidence of the effectiveness of our quantization approach relative to RTN-based quantization methods.

Evaluation on Qwen Models. Table 2 presents the perplexity and commonsense benchmark results for the Qwen3 model series and the DeepSeek-R1-Distill-Qwen-7B model. Except for the 0.6B model, SBVR demonstrates the lowest PPL among the baseline methods for every model included in the table. SBVR also ranks at or near the top across diverse commonsense benchmarks. For instance, the Qwen3-8B model achieves the highest scores across all evaluated

Model	Method	Bits	PPL↓	CQA↑	ARC-C↑	ARC-E↑	HS↑	PIQA↑	WG↑
3.2-1B Instruct	FP16	16	13.72	55.36	35.84	68.52	45.21	74.37	59.67
	AWQ	4	15.05	52.33	34.98	66.96	44.07	72.63	59.19
	GPTQ	4	90.11	20.23	23.38	51.68	31.18	63.55	55.09
	SpinQuant	4	14.58	49.14	34.81	67.34	43.75	72.69	59.35
	Ablation	4	14.58	49.14	34.04	67.17	43.71	72.52	58.56
	SBVR	4	14.41	51.84	34.22	66.96	44.30	72.69	59.91
3.2-3B Instruct	FP16	16	11.45	67.90	43.69	74.20	52.24	75.73	67.32
	AWQ	4	12.15	65.19	41.55	73.02	51.60	74.86	67.48
	GPTQ	4	18.97	60.36	39.08	70.88	49.95	72.36	61.88
	SpinQuant	4	11.65	65.03	41.38	73.36	51.43	74.27	67.48
	Ablation	4	12.49	67.40	41.47	72.73	51.01	73.83	67.40
	SBVR	4	12.10	65.85	40.96	73.23	51.29	74.05	67.17
3.1-8B Instruct	FP16	16	7.50	77.23	51.79	81.82	59.11	79.92	73.88
	AWQ	4	7.85	76.74	51.19	80.93	58.43	79.21	73.72
	GPTQ	4	7.91	71.58	48.72	78.70	56.88	78.84	71.67
	SpinQuant	4	7.76	75.92	51.45	80.72	57.90	79.49	73.95
	VPTQ	4	7.70	76.66	51.45	81.23	58.68	80.20	73.40
	Ablation	4	7.81	75.68	50.26	80.30	58.33	78.84	73.40
	SBVR	4	7.71	76.00	51.88	81.44	58.27	79.49	73.32
3.1-70B Instruct	FP16	16	4.40	80.59	62.46	86.74	65.18	82.97	79.48
	AWQ	4	4.82	80.84	62.12	86.57	64.89	83.41	79.32
	GPTQ	4	4.77	79.93	61.18	86.07	64.65	82.86	77.98
	VPTQ	4	4.65	80.02	61.01	86.32	64.92	82.86	78.77
	SBVR	4	4.71	79.85	62.63	86.53	64.49	82.92	80.27

Table 1: Benchmark results for the Llama-3.1 and 3.2 Instruct model series. CQA, HS, WG represent CommonsenseQA, HellaSwag, and Winogrande, respectively. Lower perplexity values indicate better performance, while higher commonsense benchmark accuracy indicates better performance. Ablation refers to the SBVR implementation without using the Guassian representation points.

tasks except PIQA, where it attains a nearly identical score to the best-performing AWQ method.

Speed Measurements. We measure the inference speed of SBVR-quantized models and compare them with models quantized using other methods. For a fair comparison, we use the model implementation from Huggingface (Wolf et al. 2019) and only replace the linear layers with layers containing customized kernels for each method (ModelCloud.ai and qubitium@modelcloud.ai 2024; Hansen and contributors 2025). Additionally, we apply CUDA-Graph (Corporation 2025) to all quantized models to reduce excessive overhead from GPU kernel launching. We fix the batch size to 1 for all inference measurements.

Table 3 compares the time-per-output-token (TPOT) across the Llama model series. Although AWQ achieves the best overall performance, SBVR attains comparable performance to AWQ and consistently outperforms the GPTQ kernel across all evaluated models, while delivering superior accuracy compared to RTN-based methods. Furthermore, when compared to the codebook-based method VPTQ,

[3]3-8B was used instead of 3.1-8B as QQQ does not support 3.1 models. Since 3-8B and 3.1-8B have identical structures except for the minimal RoPE embedding overhead, their execution times can be considered equivalent.

Model	Method	Bits	PPL↓	CQA↑	ARC-C↑	ARC-E↑	HS↑	PIQA↑	WG↑
0.6B	FP16	16	22.24	46.60	31.40	60.86	37.54	67.68	56.20
	AWQ	4	26.04	36.94	28.75	51.09	36.27	66.00	56.12
	GPTQ	4	23.64	35.71	29.27	57.53	36.11	64.47	55.01
	SBVR	4	25.09	40.05	28.75	56.61	36.09	65.40	55.09
1.7B	FP16	16	17.67	64.21	39.85	72.18	46.14	72.47	61.09
	AWQ	4	19.14	63.72	38.14	71.30	44.77	70.24	61.64
	GPTQ	4	18.83	60.28	35.15	64.90	43.31	70.40	59.12
	SBVR	4	18.40	60.28	41.30	73.40	44.97	70.78	60.85
8B	FP16	16	10.26	78.71	55.80	83.50	57.15	76.82	67.72
	AWQ	4	10.71	77.97	54.01	82.41	56.24	77.04	67.40
	GPTQ	4	10.56	74.94	51.28	80.81	55.66	75.52	67.88
	SBVR	4	10.35	79.03	54.01	82.53	56.42	76.55	67.40
14B*	FP16	16	9.14	80.18	58.70	84.26	61.05	80.03	72.69
	AWQ	4	9.5	78.71	58.11	83.54	60.35	80.03	71.90
	GPTQ	4	9.5	79.69	57.94	83.29	59.78	79.16	72.38
	SBVR	4	9.28	79.61	58.02	84.09	60.35	79.60	72.61
DS-7B*	FP16	16	26.81	56.02	42.41	69.65	46.64	71.00	60.46
	AWQ	4	29.26	56.67	41.04	67.68	45.51	70.51	59.67
	GPTQ	4	28.41	53.89	39.51	67.00	45.53	70.89	61.09
	SBVR	4	27.97	53.81	41.64	67.72	46.02	69.64	60.46

Table 2: Benchmark results for the Qwen3 model series and DeepSeek-R1-Distill-Qwen-7B. The table follows the same format as Table 1. Hadamard rotation was not applied to the 14B and DS-7B models since there is no matching Hadamard matrix for their hidden dimensions.

Model/Methods	Naive Torch (ms)	SBVR (ms)	GPTQ (ms)	AWQ (ms)	VPTQ (ms)	QQQ (ms)
3.2-1B	10.81	3.56	4.18	2.59	x	x
3.2-3B	18.25	6.73	7.09	5.00	x	x
3.1-8B	22.68	10.24	10.51	8.61	28.49	12.66[3]

Table 3: Time-per-output-token (TPOT) comparison of various quantization methods for Llama models. Naive Torch represents Hugging Face models with FP16 torch kernels, while the other methods represent 4-bit quantized models with their own custom kernels. X denotes publicly unavailable models.

SBVR demonstrates substantial acceleration, achieving up to a 2.78× speedup and surpassing the performance of the W4A8 kernel utilized by QQQ.

6 Conclusion

This study introduces SBVR, a novel quantization technique for large language models (LLMs) that effectively maintains quantization accuracy while achieving high inference speed. SBVR maps the weights of LLMs onto quantization points approximating a Gaussian distribution and introduces a specialized data format and computational kernel optimized for rapid GEMV operations on GPUs. Extensive experimental results confirm that SBVR accurately captures the Gaussian-like characteristics of LLM weight distributions and effectively leverages GPU architectures, resulting in significantly enhanced inference speed.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Arai, Y.; and Ichikawa, Y. 2025. Quantization Error Propagation: Revisiting Layer-Wise Post-Training Quantization. *arXiv preprint arXiv:2504.09629*.

Ashkboos, S.; Mohtashami, A.; Croci, M. L.; Li, B.; Cameron, P.; Jaggi, M.; Alistarh, D.; Hoefler, T.; and Hensman, J. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *In Proceedings of NeurIPS*, 37: 100213–100240.

Bai, J.; Bai, S.; Chu, Y.; Cui, Z.; Dang, K.; Deng, X.; Fan, Y.; Ge, W.; Han, Y.; Huang, F.; et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Bisk, Y.; Zellers, R.; Gao, J.; Choi, Y.; et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of AAAI*, volume 34, 7432–7439.

Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; and Tafjord, O. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Corporation, N. 2025. *CUDA C++ Programming Guide, ver. 12.9: §12.4 CUDA Graphs*. Accessed 2025-07-26.

Darvish Rouhani, B.; Garegrat, N.; Savell, T.; More, A.; Han, K.; Zhao, R.; Hall, M.; Klar, J.; Chung, E.; Yu, Y.; et al. 2023. OCP Microscaling Formats (MX) Specification Version 1.0. Specification, Open Compute Project Foundation, San Jose, CA. Accessed 26 Jul 2025.

Dettmers, T.; Lewis, M.; Belkada, Y.; and Zettlemoyer, L. 2022. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Proceedings of NeurIPS*, NIPS '22. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713871088.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Egiazarian, V.; Panferov, A.; Kuznedelev, D.; Frantar, E.; Babenko, A.; and Alistarh, D. 2024. Extreme compression of large language models via additive quantization. In *Proceedings of ICML*, ICML'24. JMLR.org.

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hansen, C.; and contributors, A. 2025. AutoAWQ: Easy-to-use 4-bit Activation-aware Weight Quantization Library. https://github.com/casper-hansen/AutoAWQ. Commit 88e4c76 (archived 2025-05-11), accessed 2025-07-26.

Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T. B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; and Amodei, D. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Lin, J.; Tang, J.; Tang, H.; Yang, S.; Chen, W.-M.; Wang, W.-C.; Xiao, G.; Dang, X.; Gan, C.; and Han, S. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *In Proceedings of MLSys*, 6: 87–100.

Liu, Y.; Wen, J.; Wang, Y.; Ye, S.; Zhang, L. L.; Cao, T.; Li, C.; and Yang, M. 2024a. VPTQ: Extreme Low-bit Vector Post-Training Quantization for Large Language Models. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *In Proceedings of EMNLP*, 8181–8196. Miami, Florida, USA: Association for Computational Linguistics.

Liu, Z.; Zhao, C.; Fedorov, I.; Soran, B.; Choudhary, D.; Krishnamoorthi, R.; Chandra, V.; Tian, Y.; and Blankevoort, T. 2024b. Spinquant: Llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2017. Pointer Sentinel Mixture Models. In *Proceedings of ICLR*.

Micikevicius, P.; Stosic, D.; Burgess, N.; Cornea, M.; Dubey, P.; Grisenthwaite, R.; Ha, S.; Heinecke, A.; Judd, P.; Kamalu, J.; et al. 2022. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*.

ModelCloud.ai; and qubitium@modelcloud.ai. 2024. GPTQModel. https://github.com/modelcloud/gptqmodel. Contact: qubitium@modelcloud.ai.

Sakaguchi, K.; Bras, R. L.; Bhagavatula, C.; and Choi, Y. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9): 99–106.

Shannon, C. E. 1959. Coding Theorems for a Discrete Source With a Fidelity Criterion. In *IRE National Convention Record*, volume 7, 142–163. Institute of Radio Engineers.

Si, C.; Jiang, J.; and Shen, W. 2025. Unveiling the mystery of weight in large foundation models: Gaussian distribution never fades. *arXiv preprint arXiv:2501.10661*.

Talmor, A.; Herzig, J.; Lourie, N.; and Berant, J. 2019. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In Burstein, J.; Doran, C.; and Solorio, T., eds., *In Proceedings of NAACL*, 4149–4158. Minneapolis, Minnesota: Association for Computational Linguistics.

Tseng, A.; Chee, J.; Sun, Q.; Kuleshov, V.; and Sa, C. D. 2024a. QuIP$\#$: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks. In *Proceedings of ICML*.

Tseng, A.; Sun, Q.; Hou, D.; and Sa, C. D. 2024b. QTIP: Quantization with Trellises and Incoherence Processing. In *Proceedings of NeurIPS*.

Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; Chi, E. H.; Hashimoto, T.; Vinyals, O.; Liang, P.; Dean, J.; and Fedus, W. 2022. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*. Survey Certification.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; and Choi, Y. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? In Korhonen, A.; Traum, D.; and Màrquez, L., eds., *In Proceedings of ACL*, 4791–4800. Florence, Italy: Association for Computational Linguistics.

Zhang, Y.; Zhang, P.; Huang, M.; Xiang, J.; Wang, Y.; Wang, C.; Zhang, Y.; Yu, L.; Liu, C.; and Lin, W. 2024. Qqq: Quality quattuor-bit quantization for large language models. *arXiv preprint arXiv:2406.09904*.