KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization

Coleman Hooper
chooper@berkeley.edu
UC Berkeley

Sehoon Kim
sehoonkim@berkeley.edu
UC Berkeley

Hiva Mohammadzadeh
hiva@berkeley.edu
UC Berkeley

Michael W. Mahoney
mmahoney@stat.berkeley.edu
ICSI, LBNL, UC Berkeley

Yakun Sophia Shao
ysshao@berkeley.edu
UC Berkeley

Kurt Keutzer
keutzer@berkeley.edu
UC Berkeley

Amir Gholami
amirgh@berkeley.edu
ICSI, UC Berkeley

ABSTRACT

LLMs are seeing growing use for applications which require large context windows, and with these large context windows KV cache activations surface as the dominant contributor to memory consumption during inference. Quantization is a promising approach for compressing KV cache activations; however, existing solutions fail to represent activations accurately in sub-4-bit precision. Our work, KVQuant, facilitates low precision KV cache quantization by incorporating several novel methods: (i) *Per-Channel Key Quantization*, where we adjust the dimension along which we quantize the Key activations to better match the distribution; (ii) *Pre-RoPE Key Quantization*, where we quantize Key activations before the rotary positional embedding to mitigate its impact on quantization; (iii) *Non-Uniform KV Cache Quantization*, where we derive per-layer sensitivity-weighted non-uniform datatypes that better represent the distributions; (iv) *Per-Vector Dense-and-Sparse Quantization*, where we isolate outliers separately for each vector to minimize skews in quantization ranges; and (v) *Q-Norm*, where we normalize quantization centroids in order to mitigate distribution shift, providing additional benefits for 2-bit quantization. By applying our method to the LLaMA, LLaMA-2, and Mistral models, we achieve $<$ 0.1 perplexity degradation with 3-bit quantization on both Wikitext-2 and C4, outperforming existing approaches. Our method enables serving LLaMA-7B with a context length of up to **1 million on a single A100-80GB GPU** and up to **10 million on an 8-GPU system**. We develop custom CUDA kernels for KVQuant, showing that we can achieve up to ~1.4× speedups, compared to baseline fp16 matrix-vector multiplications, for the LLaMA-7B model. The code is available at https://github.com/SqueezeAILab/KVQuant/.

1 INTRODUCTION

Large language models (LLMs) have revolutionized many natural language processing (NLP) tasks. In order to improve the capabilities of LLMs, there is significant interest in increasing the context lengths of LLMs. Longer context lengths enable new applications, including long document summarization, retrieval for answering questions about long documents, extended multi-turn applications [4], and code analysis. To support this pull from applications, there have been significant recent advances in long-context length models in industry [1, 27], as well as in academia [4].

Given the importance of LLM workloads, there is strong motivation to improve their inference efficiency. LLM inference with large context lengths can be incredibly resource-intensive; serving LLMs requires high-end GPUs, and the largest LLMs require costly multi-GPU inference setups. When analyzing the computational nature of generative inference with LLMs, it becomes quickly apparent that, for relatively small batch sizes, the computation is memory bound [18]. With the growing divergence between computational speeds and memory speeds, this problem is only going to get worse over time [13]. This makes reducing the memory bottleneck preeminently important. Further analysis shows that the memory bottleneck is strongly related to context size. For short sequence lengths, the dominant contributor to memory consumption is the weight matrices, and therefore the optimal strategy is to minimize the model size in order to reduce memory consumption as well as bandwidth requirements [18, 19]. However, for long sequence lengths, the main bottleneck is the memory requirements for caching Key and Value (KV) activations throughout inference. In particular, the size of the KV cache can become the dominant contributor to memory footprint, even for a 32K context limit (see Table 1), making it challenging to perform long context length inference. This challenge is further exacerbated when one considers batched inference.

It is therefore crucial to develop methods for compressing the KV cache to enable efficient long-sequence length inference. Existing approaches lead to unacceptable accuracy degradation due to the outlier structures in KV cache activations as well as suboptimal bit allocation with existing uniform and non-uniform approaches. In this work, we perform an extensive analysis of KV cache activations in recent LLMs, revealing patterns which can be exploited to enable ultra-low precision quantization with minimal accuracy loss. In particular, we make the following contributions (summarized in Figure 1):

- We find that the Key matrices exhibit structured outliers in specific channels before applying RoPE. However, the outlier channel magnitudes become less consistent after applying RoPE, posing a distinct challenge for low precision quantization. Based on these observations, we use per-channel quantization for Keys,

and we quantize Keys before RoPE is applied (see Section 3.1 and Section 3.2).
- We find that existing uniform and non-uniform quantization methods result in sub-optimal quantization signpost placement. Instead, we propose a Non-Uniform Quantization (NUQ) method which considers sensitivity and not just magnitude when quantizing activations. We show that one can apply sensitivity-weighted non-uniform quantization offline on a calibration set to derive accurate datatypes for KV cache quantization (see Section 3.3).
- Even with the above, we find that outlier values in cached KV activations can significantly degrade quantization resolution. Unlike for weights, it is non-trivial to extract outlier values from activations, given the dynamic nature of activations. However, we find that we can efficiently and accurately identify and compress outlier values in order to store them compactly in a separate sparse representation. We also find that per-vector outlier detection outperforms per-matrix outlier detection with no additional memory overhead. With this method, we can attain under 0.1 perplexity degradation for 3-bit KV cache quantization on both Wikitext-2 and C4 by only removing 1% of outliers, thereby facilitating accurate inference with 4.8× longer context length.
- For ultra low-bit precision, we find that the quantized activations can deviate significantly from their corresponding fp16 values. To address this, we propose a lightweight Q-Norm layer which shifts and scales the distribution after de-quantization to match the mean and standard deviation of corresponding fp16 values. Interestingly, the Q-Norm layer can be fused with non-uniform quantization values resulting in no overhead during inference. This was particularly helpful for 2-bit quantization (see Section 3.5).
- We implement custom CUDA kernels to perform activation quantization efficiently during inference, achieving up to ∼1.4× speedups for Key and Value matrix-vector multiplications for LLaMA-7B at 4-bit precision relative to the fp16 baseline (see Section 3.7 and 4.2). These results demonstrate how our methodology allows for accurate and efficient low-bit KV cache quantization.

2 BACKGROUND

2.1 LLM Inference

When inferring a decoder-only LLM, inference proceeds in two distinct phases. In the prefill phase, the model takes in an input prompt, which it processes *in parallel*. During the generation phase, the model then generates the output sequence autoregressively, meaning that each token generation is dependent on all previously generated tokens. As such, for small batch sizes, the generation phase of LLM inference is typically *memory-bandwidth bound*, as the only available parallelism is across different sequences in a given batch.

Additionally, during generation, the model needs to store intermediate Key and Value activations in order to condition generations on previously generated output tokens. Otherwise, we would need to recompute all prior Keys and Values at each timestep, which would be prohibitively expensive. For each prior token, we need to store the Keys and Values at each layer in order to use these activations when generating future tokens. These stored activations are referred to as the *Key-Value (KV) cache*. Throughout this paper, we will capitalize Key and Value to distinguish when we are referring to the KV cache tensors. Assuming a model with n layers and h attention heads with dimension d, the KV cache size for batch size b and sequence length l is $2 \cdot n \cdot h \cdot d \cdot b \cdot l$, meaning that it grows linearly with both batch size and sequence length. As shown in Table 1, the KV cache becomes the dominant contributor to memory consumption for longer sequence lengths and larger batch sizes. Note that since each sequence in batched inference depends on separate past context, there is no available batch-level parallelism when loading the cached Keys and Values for their respective computations in batched inference. KV cache loading is therefore always *memory-bandwidth bound*. This motivates pursuing methods to optimally compress the KV cache, even at the expense of a more complex dequantization process.

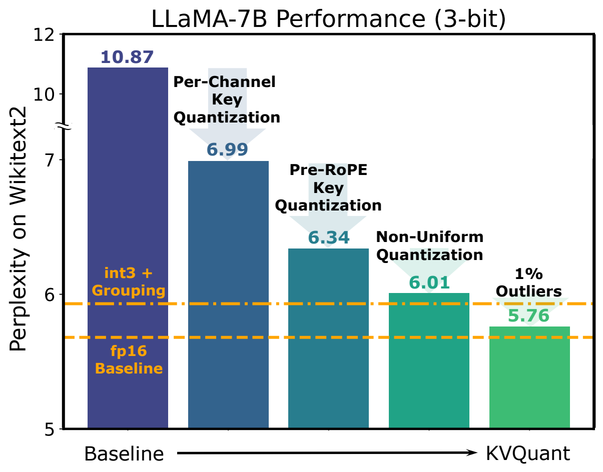
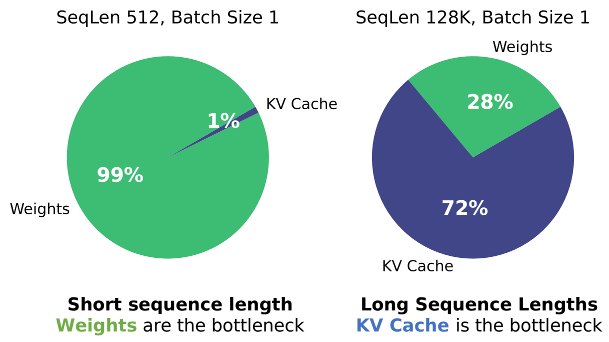


Figure 1: *Overview of the different components used in KVQuant that result in less than 0.1 perplexity degradation over the fp16 baseline when quantizing the KV cache to 3-bit precision. As shown in Table 2, our 3-bit approach results in 4.8× reduction in cached activation memory footprint.*

2.2 LLM Quantization

KV Cache Quantization. There have been many prior works on LLM quantization. Several have focused on weight-only quantization for LLMs, due to the greater contribution to memory consumption and runtime for fairly small sequence length and batch size [7, 18, 22]. There has also been work on quantizing both weights and activations (including KV cache) [29, 36]. However, there is still a significant perplexity degradation when quantizing KV cache activations to low precision; [30, 38] quantized KV cache activations to 4-bits, but required fine-grained grouping for 4-bit quantization, while still observing some perplexity degradation, and [30] observed that 3-bit KV cache quantization with fine-grained grouping leads to unacceptable accuracy loss. Other works quantized KV cache activations to 4-bits but required retraining to maintain performance [23]. One concurrent work also explores low precision KV cache quantization in order to enable larger batch size inference by reducing the KV cache size [25]. In this work, we introduce a

Table 1: *Model size and activation memory size for different sequence lengths and batch sizes (BS) for different LLaMA models. For long sequence lengths and larger batch sizes, activation memory is the main bottleneck (particularly when weights are already quantized to low precision). At a sequence length of 128K with the LLaMA-7B model, the KV cache is the main bottleneck (see below left). Additionally, if model weights are quantized, then the KV cache is the main bottleneck even at a sequence length of 32K. By compressing the KV cache to 3-bit precision, we can enable* ***1M context length inference with the LLaMA-7B model on a single A100-80GB GPU****, and we can also enable* ***10M context length inference with the LLaMA-7B model on an 8-GPU system****.*



BS	# Params	Model Size (GB) 16-bit → 3-bit	fp16 KV Cache Size with different seq len (GB) 32K	128K	1M	10M (16-bit → 3-bit)
1	7B	12.6 → 2.4	8.0	32.0	244.1	2441 → 459.0
	13B	24.7 → 4.6	12.5	50.0	381.5	3815 → 716.7
	30B	61.0 → 11.4	24.4	97.5	743.9	7439 → 1397
	65B	122.4 → 23.0	40.0	160.0	1221	12207 → 2292
4	7B	12.6 → 2.4	32.0	128.0	976.6	9766 → 1836
	13B	24.7 → 4.6	50.0	200.0	1526	15259 → 2867
	30B	61.0 → 11.4	97.5	390.0	2976	29755 → 5588
	65B	122.4 → 23.0	160.0	640.0	4883	48828 → 9167

method for near-lossless low-bit KV cache quantization that minimizes performance degradation without the need for finetuning.

Outlier-Aware LLM Quantization. LLMs have been known to have distinct outliers both in weights and activations [5, 7, 18]. SqueezeLLM and SpQR both decompose the weight matrix into a sparse matrix containing a small portion of outliers and a dense matrix that can be accurately quantized to low precision (referred to as dense-and-sparse or sparse-quantized representation) [7, 18]. LLM.int8() [5] handled particular outlier channels separately in higher precision, and SmoothQuant [36] migrates quantization difficulty due to outlier channels to weights in order to support joint weight-activation quantization. Other works reconsidered the dimension along which we quantize in order to reduce quantization error (or else added per-channel compensation to improve quantization performance) [2, 16, 34, 35]. In this work, we demonstrate that per-channel pre-RoPE Key quantization provides significant accuracy benefits given the outlier structure in Keys, and that dense-and-sparse quantization can be efficiently applied for KV cache quantization.

Non-uniform LLM Quantization. Non-uniform quantization has also been applied in the context of LLMs. Non-uniform quantization allows for more flexible quantization signpost placement relative to uniform quantization methods, enabling improved accuracy for the same bit precision [6, 18]. Building on the observation that model parameters tend to be approximately normally-distributed, prior work has proposed the NormalFloat datatype [6]. SqueezeLLM [18] derived per-channel non-uniform quantization signposts using a sensitivity-weighted k-means approach. In this work, we show that we can derive accurate per-layer non-uniform datatypes using a sensitivity-weighted k-means approach with KV cache activations.

2.3 KV Cache Compression

There have also been several prior works on compressing the KV cache. Some of these methods aim to only store important tokens in the KV cache and to evict less important tokens, thereby maintaining low memory usage [12, 24, 37]. Other methods aim to only retrieve a subset of tokens at each step to achieve memory bandwidth savings [28]. In this work, we explore KV cache quantization as an orthogonal direction for compressing the KV cache in order to enable long context inference.

3 METHOD

3.1 Per-Channel Key Quantization

To inform our approach, we first performed a detailed analysis to understand the KV cache distributions. Figure 2 shows sample distributions for the KV cache activations. We observe that the Key matrices tend to have distinct outlier channels, which have larger average magnitudes than other channels; this corroborates previous observations about outlier channels in LLM activations [5, 36]. The Value matrices exhibit both outlier channels as well as outlier tokens (although these outliers are less extreme than the outlier Key channels).

Existing KV cache quantization approaches perform per-token quantization (meaning that the scaling factor and zero-point are shared by elements in the same token) [30, 38]. However, due to the differing average magnitudes between channels, the values within a channel are easier to quantize when grouped together than the values across different channels. As such, to better match the distributions, we investigate *per-channel* KV cache quantization, meaning that the scaling factor and zero-point are shared by elements in the same channel. By sharing the scaling factor and zero-point along the channel dimension, this will naturally group together values with similar magnitudes, thereby mitigating the impacts of outlier channels on other channels when quantizing to low precision. As outlined in Appendix D, we find that per-channel quantization provides significant accuracy benefits for Keys but not for Values. By leveraging per-channel quantization for Keys and per-token quantization for Values, we observe a 3.88 perplexity improvement on Wikitext-2 for 3-bit LLaMA-7B quantization. Note that this can potentially add runtime overhead since the quantization dimension is now misaligned with the reduction dimension for the Keys when performing matrix-vector multiplications. However,

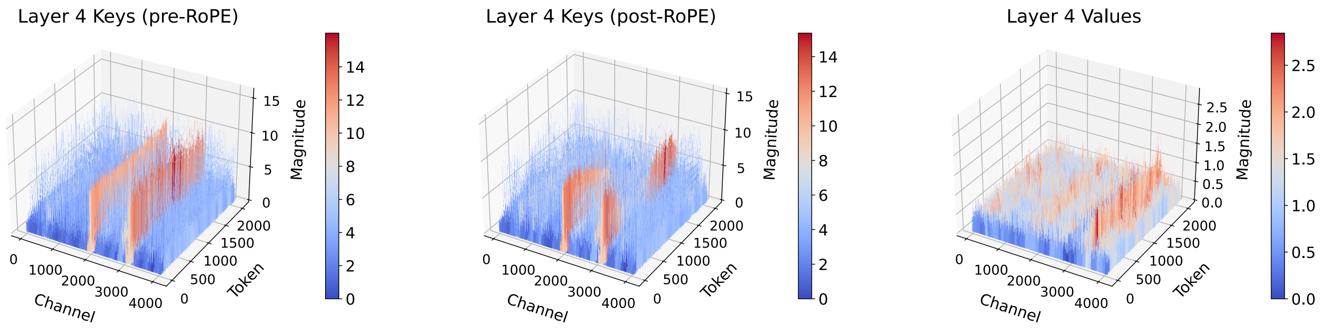


Figure 2: *Example distributions of the activation values for Keys pre-RoPE, Keys post-RoPE, and Values for LLaMA-7B on a sample with 2K sequence length from Wikitext-2. We observe several patterns: (i) Keys pre-RoPE exhibit clear outliers in specific channels across different tokens; (ii) after applying RoPE, the distribution becomes less structured and there are less consistent magnitudes for outlier channels (this is expected, as RoPE applies a rotation operation between pairs of channels); and (iii) Values exhibit no fixed outlier pattern with outlier values across channels and tokens.*

we find that we are able to efficiently dequantize Keys and perform the Query-Key matrix-vector multiplication without adding runtime overhead, as shown in Section 4.2. Additionally, as outlined in Section 3.6, per-channel quantization can also be challenging due to the need to recompute scaling factors as tokens are added to the Key cache. We show that we can calibrate offline for scaling factors, thereby avoiding expensive online recomputation.

Per-channel Key quantization was also explored in another concurrent work [25], which leveraged similar intuition about grouping together large magnitude values in the same channel to minimize quantization error. Their methodology requires fine-grained grouping for per-channel quantization while maintaining a residual subset of the KV cache in fp16 (until all elements for that group have been added to the KV cache). In our work, we demonstrate that by leveraging offline calibration, we can accurately perform per-channel quantization without grouping.

3.2 Pre-RoPE Key Quantization

One issue when quantizing Keys is handling the rotary positional embedding (RoPE), which is applied to Keys and Queries in most public LLMs, including LLaMA and LLaMA-2 [31]. Given Query and Key vectors $Q_m = W_q * x_m$ and $K_n = W_k * x_n$ at positions m and n in the sequence, RoPE is applied as position-dependent rotations to each of these vectors to obtain $\tilde{Q}_m$ and $\tilde{K}_n$. This embeds the relative position between a Query and Key vector as an amount of an angle that is a multiple of its position index. Formally, RoPE is applied in self-attention as follows:

$$\tilde{Q}_m \tilde{K}_n^\top = (R^d_{\theta,m} \cdot Q_m)(R^d_{\theta,n} \cdot K_n)^\top. \tag{1}$$

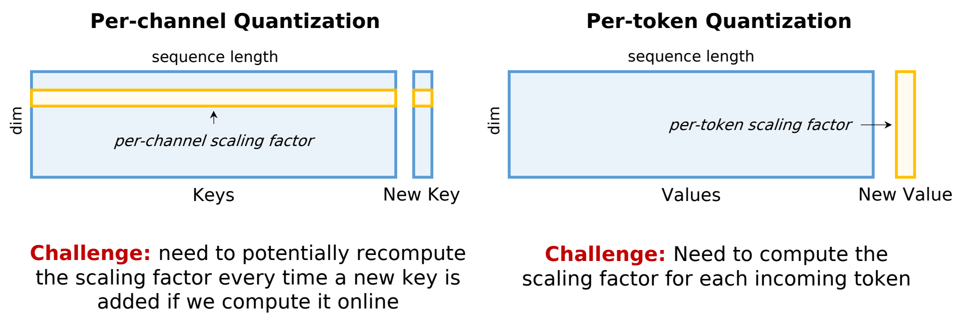
The Query vectors computed at each iteration will have RoPE applied ($\tilde{Q}_m$). When caching Key vectors, we need to either cache $\tilde{K}_n$, or else we need to cache K_n and apply $R^d_{\theta,n}$ on-the-fly during inference. The challenge with caching Key vectors after applying this rotation is that it leads to mixing pairs of channels by different amounts for different positions in the sequence, as shown in Appendix A (since it jointly rotates pairs of channels by different angles depending on the position in the sequence). The post-RoPE activation distribution is also shown in Figure 2, demonstrating how the rotation between pairs of channels leads to less consistent channel magnitudes. This makes it harder to quantize Key activation channels which would typically have consistent large-magnitude values. This motivated our investigation into whether we could perform *pre-RoPE* Key quantization (meaning that we quantize K_n) and then efficiently apply the positional embeddings on-the-fly after dequantization. The benefits of pre-RoPE Key quantization are highlighted in Appendix E, yielding 0.65 perplexity improvement on Wikitext-2 for 3-bit LLaMA-7B quantization. To be able to quantize Keys pre-RoPE, we develop a fused kernel to efficiently apply RoPE post-dequantization (the details of this approach will be discussed in Section 3.7).

3.3 nuqX: An X-Bit Per-Layer Sensitivity-Weighted Non-Uniform Datatype

Uniform quantization is suboptimal for KV cache quantization since the Query and Key activations are non-uniform. Additionally, KV cache loading is memory bandwidth bound, regardless of batch size or sequence length, meaning that the dequantization overhead introduced by non-uniform quantization methods is not problematic (since the added computation does not introduce any additional latency). It is therefore desirable to leverage non-uniform quantization methods for KV cache quantization.

In [18], the authors computed non-uniform quantization signposts using a sensitivity-weighted k-means approach. However, this is challenging to apply online during inference due to its computational cost, and it is also difficult to estimate sensitivity for activations online. We therefore facilitate efficient online non-uniform KV cache quantization by computing sensitivity-weighted quantization signposts offline on a calibration set prior to inference. Using the diagonal Fisher information matrix (derived in Appendix C), along with the quantization error for activation A, we formulate the error minimization objective as:

$$Q(A)^* \simeq \arg\min_Q \sum_{i=1}^{N} \mathcal{F}_{ii} \big(A - Q(A)\big)^2. \tag{2}$$



Per-channel Quantization
sequence length
dim
per-channel scaling factor
Keys
New Key
Challenge: need to potentially recompute the scaling factor every time a new key is added if we compute it online
Per-token Quantization
sequence length
dim
per-token scaling factor
Values
New Value
Challenge: Need to compute the scaling factor for each incoming token

Figure 3: *One typically achieves better performance when the scaling factor/zero point are computed online. However, this is quite challenging to do for per-channel quantization, as these factors will not only need to be recomputed for every new Key appended to the Key cache, but also all the prior cached Keys will need to be updated. As such, we use a calibration set to compute per-channel scaling factors offline. A similar challenge exists for per-token quantization, but online calibration for this does not require updating prior cached Values. In Section 3.6 and Appendix I, we discuss how we are able to efficiently compute outlier thresholds / scaling factors for per-token calibration, thereby enabling online computation.*

In order to derive a per-matrix non-uniform datatype, we first normalize each vector to the range $[-1, 1]$. We then minimize the objective in Equation 2 offline on a calibration set using a k-means solver in order to obtain the quantization signposts for the non-uniform datatype for each Key or Value layer. Appendix F compares our non-uniform quantization approach with existing uniform and non-uniform quantization baselines [6], demonstrating how our non-uniform approach provides 0.33 perplexity improvement on Wikitext-2 for LLaMA-7B relative to 3-bit uniform methods. Table 15 in Appendix I shows how computing the required Fisher information for the LLaMA-65B model takes only a few minutes, and how using the k-means solver takes only a few minutes per layer (with the computation for each layer being parallelizable). In Appendix N, we also demonstrate that we can derive a metric for accurate one-shot mixed-precision assignment (where different layers are assigned different bit widths) using the quantization error weighted by sensitivity information.

3.4 Per-Vector Dense-and-Sparse Quantization

Figure 5 in Appendix B shows the portion of elements falling within different percentiles of the dynamic range. For both Keys and Values, the majority of elements are contained within a small percentage of the dynamic range. This means that by leveraging dense-and-sparse quantization, as demonstrated in [18], in order to isolate a small percentage of numerical outliers, we can restrict the range that we need to represent, thereby allowing us to represent the remaining elements with greater precision.

Additionally, when looking at the Key and Value distributions in Figure 2, different channels and tokens have different average magnitudes. Therefore, an element which counts as an outlier in one channel may not be an outlier in another channel (since that channel may have a greater average magnitude). It is therefore crucial to directly target the outlier values that skew the dynamic range *at the granularity that we are quantizing* in order to address the values that are exaggerating the range along that particular dimension. In this work, we leverage *per-vector* dense-and-sparse quantization, where we use a different outlier threshold per-vector (either a separate threshold per-channel for per-channel quantization, or a separate threshold per-token for per-token quantization), rather than a single outlier threshold for each layer.

Note that computing outlier thresholds for per-vector dense-and-sparse quantization poses potential accuracy and efficiency challenges. However, in Section 3.6, we show that we are able to accurately calibrate for per-channel outlier thresholds offline and efficiently compute per-token outlier thresholds online. After determining the upper and lower outlier thresholds, the remaining numbers in the vector are normalized to the range $[-1, 1]$, and we then minimize Equation 2 (ignoring outliers) in order to obtain the quantization signposts for the non-uniform datatype for the remaining numbers. Appendix G will demonstrate the benefits of removing a small percentage of numerical outliers and keeping them in full precision, as well as the advantages of per-vector dense-and-sparse quantization over using a single outlier threshold for each layer. As shown in Figure 1, by removing 1% of numerical outliers using per-vector outlier thresholds, we achieve an additional 0.25 perplexity improvement on Wikitext-2 for 3-bit LLaMA-7B quantization, which is within 0.08 perplexity of the fp16 baseline.

3.5 Mitigating Distribution Shift using Q-Norm

When pushing to extremely low bit widths like 2-bit quantization, we begin to observe accuracy degradation due to *distribution shift*, meaning that the post-quantization distribution has a different mean and standard deviation than the pre-quantization distribution. As shown in Appendix H, this distribution shift can lead to greater error accumulation at later layers. To combat this, we introduce *Q-Normalization* (shortened to *Q-Norm*), where we normalize the quantization centroids obtained from k-means in order to ensure the post-quantization distribution has the same mean and standard deviation as the pre-quantization distribution. Our solution is inspired by [21], which demonstrated that ensuring that the activation distributions post-weight quantization have similar mean and standard deviation to the activation distributions pre-weight quantization can help reduce accuracy degradation. Given mean μ_1 and standard deviation σ_1 for the pre-quantization distribution and mean μ_2 and standard deviation σ_2 for the post-quantization distribution (computed offline on a calibration set), we normalize

Table 2: *Evaluation of our method for different models using the perplexity on Wikitext-2. Non-uniform quantization results are using pre-RoPE per-channel quantization for Keys. "gs64/128" refers to baseline experiments using grouping with group size 64/128. KV cache sizes assume a sequence length of 128K (ignoring context length limits for the models). We incorporate Q-Norm for nuq2 experiments.* † *and* ‡ *denote the methods used in ATOM [38] and FlexGen [30], respectively. Note that we used post-RoPE quantization for all baseline methods since it achieves higher accuracy when quantizing Keys per-token as shown in Appendix L.*

Method	**LLaMA-7B**		**LLaMA-13B**		**LLaMA-30B**		**LLaMA-65B**	
	Perplexity	KV Cache (GB)	Perplexity	KV Cache (GB)	Perplexity	KV Cache (GB)	Perplexity	KV Cache (GB)
baseline	5.68	32.0	5.09	50.0	4.10	97.5	3.53	160.0
int4	5.98	8.0	5.32	12.5	4.34	24.4	3.73	40.0
nf4	5.87	8.0	5.23	12.5	4.25	24.4	3.63	40.0
int4-gs128†	5.77	8.3	5.16	13.0	4.16	25.3	3.57	41.6
int4-gs64‡	5.73	8.6	5.14	13.5	4.14	26.3	3.56	43.1
nf4-gs128	5.77	8.5	5.17	13.3	4.17	25.9	3.58	42.5
nuq4	5.73	8.0	5.15	12.5	4.16	24.4	3.57	40.0
nuq4-1%	**5.70**	8.6	**5.11**	13.5	**4.12**	26.3	**3.54**	43.2
int3	10.87	6.0	8.69	9.4	6.82	18.3	6.37	30.0
nf3	7.33	6.0	6.21	9.4	5.46	18.3	4.44	30.0
int3-gs128	6.17	6.3	5.47	9.8	4.44	19.2	3.78	31.5
int3-gs64	5.93	6.6	5.29	10.3	4.26	20.1	3.66	33.0
nf3-gs128	6.26	6.5	5.52	10.2	4.54	19.8	3.83	32.5
nuq3	6.01	6.0	5.34	9.4	4.41	18.3	3.74	30.0
nuq3-1%	**5.76**	6.6	**5.15**	10.4	**4.17**	20.3	**3.59**	33.2
int2	11779	4.0	69965	6.3	1470	12.2	7272	20.0
nf2	3210	4.0	5786	6.3	2044	12.2	5367	20.0
int2-gs128	37.37	4.3	41.77	6.7	16.49	13.0	13.63	21.4
int2-gs64	11.09	4.6	9.84	7.1	6.60	13.9	5.54	22.8
nf2-gs128	351.23	4.5	141.19	7.0	60.97	13.7	31.69	22.5
nuq2	8.17	4.0	7.29	6.3	7.05	12.2	27.17	20.0
nuq2-1%	**6.06**	4.6	**5.40**	7.3	**4.43**	14.2	**3.76**	23.2

the quantization centroids C_i to $\hat{C}_i$ as follows:

$$\hat{C}_i = \frac{(C_i - \mu_2)\sigma_1}{\sigma_2} + \mu_1. \quad (3)$$

During dequantization, we use $\hat{C}_i$ in place of C_i such that the mean and standard deviation of dequantized values matches the original fp16 distribution. As shown in Appendix H, Q-Norm provides significant accuracy benefits for 2-bit quantization with no added inference cost.

3.6 Offline Calibration versus Online Computation

A crucial challenge for activation quantization is that we either need to compute statistics on-the-fly (which is potentially expensive) or else we need to use offline calibration data (which potentially has negative accuracy implications). The challenges with computing scaling factors (and zero-points) online versus offline for both Keys and Values are shown in Figure 3. In per-channel quantization, it is challenging to update scaling factors online since the scaling factors corresponding to each incoming channel would potentially need to be updated whenever a new token is added to the KV cache. It is therefore desirable to be able to compute statistics offline (i.e., using calibration data before running inference). While this can have negative effects on model accuracy, in Appendix I we show that we can effectively calibrate offline for per-channel quantization, obviating the need for online updates of scaling factors for per-channel quantization. For per-token quantization, it is challenging to calibrate for scaling factors offline due to the presence of outlier Value tokens. It is therefore desirable to be able to compute scaling factors and outlier thresholds online for each incoming token. As shown in Appendix I, we can efficiently compute outlier thresholds online per-token by offloading to the CPU. By leveraging custom quantization function implementations for compressing activations, we are able to perform online per-token Value quantization without compromising on performance.

3.7 Kernel Implementation

In order to efficiently perform activation quantization on-the-fly, we leverage dedicated kernel implementations with our 4-bit quantization method for compressing vectors to reduced precision and extracting the sparse outliers, performing matrix-vector multiplications using the compressed vectors, and performing sparse matrix-dense vector multiplications using the sparse outliers. We store the quantized Key and Value matrices as 4-bit elements which are used as indices into lookup tables to recover the corresponding fp16 values. We store the sparse outlier matrices in either Compressed-Sparse Row (CSR) or Compressed-Sparse Column (CSC) format (depending on which aligns better with appending new Key and Value tokens). The kernels for the Key matrix-vector operations apply RoPE on-the-fly in order to support pre-RoPE quantization. More kernel implementation details are provided in Appendix O.

4 RESULTS

4.1 Main Results

4.1.1 Main Evaluation. We used the LLaMA-7B/13B/30B/65B, LLaMA-2-7B/13B/70B, and Mistral-7B models to evaluate our methodology by measuring perplexity on both Wikitext-2 and C4 [17, 32, 33]; see Appendix J for details on our experimental setup. Table 2 shows the results for LLaMA models for the Wikitext-2 dataset. We compared our method with per-token quantization with and without grouping. The baseline configurations used by Atom and FlexGen are included for reference [30, 38]. We did not include results for

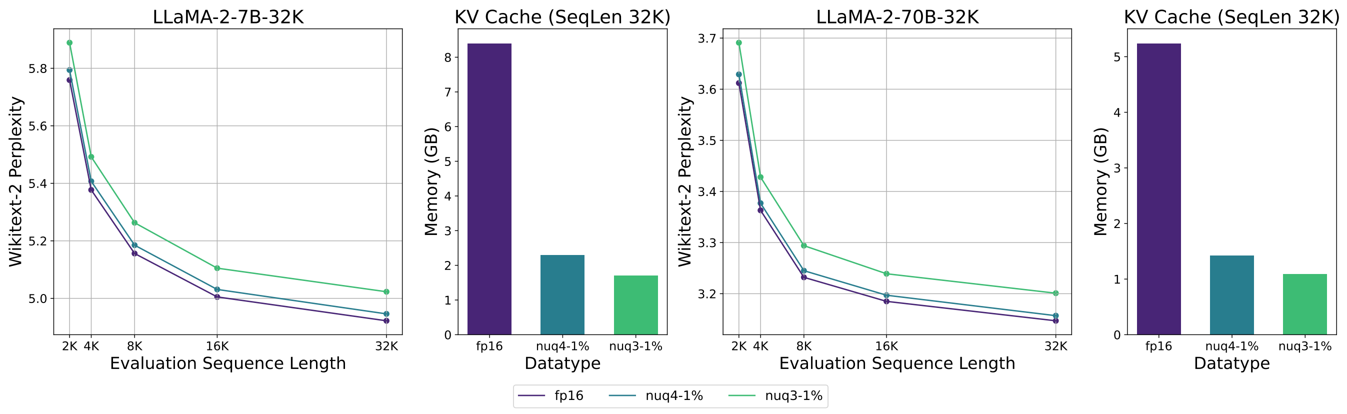


Figure 4: *Perplexity results for the LLaMA-2-7B-32K model [3] as well as the LLaMA-2-70B-32K LongLoRA model [4] on the Wikitext-2 dataset, evaluated using different sequence lengths. Figure 8 in Appendix M provides additional results with long sequence length evaluation for 2-bit quantization including Q-Norm.*

KIVI [25] since they did not report perplexity values. We find that our method consistently outperforms baseline approaches by an especially large margin with 3-bit and 2-bit quantization. Once we incorporate outliers, we further push the performance of low-precision quantization, achieving 4-bit quantization with less than 0.02 perplexity degradation, 3-bit quantization with under 0.09 perplexity degradation, and 2-bit quantization with under 0.43 perplexity degradation relative to the fp16 baseline across all models and datasets (while attaining 3.7×, 4.8×, and 6.9× memory savings, respectively). Tables 16 and 17 in Appendix K show full perplexity evaluation on Wikitext-2 and C4, showing the consistent performance of our approach across different models and datasets. Table 18 in Appendix K also provides zero-shot MMLU evaluation with 3-bit quantization, demonstrating how our method maintains performance on downstream tasks.

4.1.2 Long Context Length Evaluation. We evaluated long context length performance using the LLaMA-2-7B-32K model (uptrained for long sequence lengths using positional interpolation [3]) as well as the LLaMA-2-70B-32K LongLoRA model [4]. For evaluating performance on longer context lengths, we first evaluated perplexity on Wikitext-2 using larger amounts of input context, as shown in Figure 4 [4, 14]. The results demonstrate how our method maintains accuracy even for longer amounts of input context, thereby enabling efficient and accurate long sequence length inference. Additional long sequence length perplexity evaluation results for 2-bit quantization (including Q-Norm) are provided in Appendix M.

We also evaluated the performance of our quantization method on passkey retrieval to assess the model's ability to use its context. Passkey retrieval involves evaluating the model's capacity to locate specific information in long texts [20], and this can be used to effectively measure the maximum distance over which a token can attend during the inference stage. We used the passkey evaluation framework from [39] (which is based on the methodology from [26]) to evaluate retrieval performance. Table 3 shows passkey retrieval results for the LLaMA-2-7B-32K and LLaMA-2-70B-32K models. These results indicate that with 4-bit and 3-bit quantization, KVQuant is able to maintain retrieval performance of the fp16 model. We observe some degradation with 2-bit quantization especially with LLaMA-2-70B, indicating potential room for improvement in 2-bit quantization techniques.

Table 3: *Passkey retrieval results across different context lengths for the LLaMA-2-7B-32K model (uptrained for long sequence lengths using positional interpolation [3]) as well as the LLaMA-2-70B-32K LongLoRA model [4]. The values reported are the success rate for retrieving the passkey, computed over 50 samples. 2-bit results are with per-matrix Q-Norm enabled.*

Model	Datatype	2K	4K	8K	16K	32K
LLaMA-2-7B-32K	fp16	1	1	1	1	1
	nuq4-1%	1	1	1	1	1
	nuq3-1%	1	1	1	1	1
	nuq2-1%	1	1	1	0.92	0.98
LLaMA-2-70B-32K	fp16	1	1	1	1	1
	nuq4-1%	1	0.98	1	1	1
	nuq3-1%	1	0.98	1	1	1
	nuq2-1%	0.82	0.8	0.82	0.8	0.76

4.1.3 Joint Weight and KV Cache Quantization. Table 4 shows results for our KV cache quantization method when the weights are also quantized using the methodology in SqueezeLLM [18]. We observe minimal perplexity degradation when leveraging our KV cache quantization approach, even when weights are also quantized to reduced precision (achieving within 0.02 perplexity of 4-bit weight-only quantization when quantizing the KV cache using nuq4-1% for the LLaMA-7B and LLaMA-13B models). These results demonstrate how our method is compatible with existing weight-only quantization methods.

4.2 Performance Analysis

Table 5 shows kernel benchmarking results using a batch size of 1 for the 4-bit dense-and-sparse compression and matrix-vector kernel implementations. We show results across different sequence lengths to assess the performance of the kernels at different points during generation. We report latency benchmarked on an A6000 GPU and averaged over 1000 runs. The results show that for the

Table 4: *KV cache quantization results when applied in conjunction with the weight quantization methodology in SqueezeLLM [18]. w4-s45 and w3-s45 refer to the 4-bit and 3-bit dense-and-sparse weight quantization approaches in [18], respectively. See Appendix J for experimental details.*

Weights	KV Cache	LLaMA-7B	LLaMA-13B	Avg. Bits (KV Cache)
fp16	fp16	5.68	5.09	16
w4-s45	fp16	5.77	5.17	16
	nuq4	5.83	5.22	4.00
	nuq4-1%	**5.79**	**5.18**	4.32
w3-s45	fp16	6.13	5.45	16
	nuq3	6.55	5.77	3.00
	nuq3-1%	**6.26**	**5.54**	3.32

Table 5: *Average latency (in microseconds) for the Key and Value nuq4-1% kernels, benchmarked on an A6000 GPU for the LLaMA-7B model across different sequence lengths (l). fp16 matrix-vector multiplication latencies are included for reference, and the fp16 Key multiplication time also includes the time to apply RoPE to the newly appended Key vector. Section 3.7 and Appendix O provide additional details for our kernel implementation, and Table 20 provides a detailed breakdown of kernel runtime.*

Activation	Operation	l=2K	l=4K	l=16K
Key	fp16 Matvec	66.4	116.1	402.3
Key	nuq4-1%	60.2	96.7	344.1
Value	fp16 Matvec	56.0	104.2	389.3
Value	nuq4-1%	40.8	85.8	293.4

Key and Value multiplications, we can achieve 1.1-1.2× and 1.2-1.4× latency savings, respectively, relative to the baseline. We have integrated these kernels into an end-to-end generation pipeline that is able to compress activations dynamically during inference, thereby achieving significant memory savings and allowing for either larger batch sizes or longer sequence lengths.

4.3 Pushing the Context Length Limit

Table 6 shows the KV cache memory requirements for 128K, 1M, and 10M sequence lengths, with the KV cache stored in fp16 as well as 4-bit, 3-bit, and 2-bit precision with KVQuant. As one can see, our method provides 3.7× KV cache compression (nuq4-1%) and enables serving the quantized LLaMA-65B model with a context length of 32K tokens on a single A100-80GB GPU (requiring 30GB for the model weights compressed to 4-bit, and 33GB for the KV cache when compressed with nuq4-1%), and even serving the LLaMA-7B model with a context length of **1M tokens** on a single A100 GPU (requiring 3GB for the model weights in 4-bit precision and 66GB for the KV cache with nuq4-1%). Additionally, when considering an 8-GPU serving system, we enable serving the LLaMA-7B model with **10M context length** (with nuq3-1%) or the LLaMA-65B model, with 1M context length (with nuq4-1%). While currently there are no models or datasets to measure accuracy with such large context sizes, our results on smaller context lengths show little degradation compared to baseline inference, demonstrating the benefits of our approach for enabling long sequence length inference.

Table 6: *Activation memory size (GB) for 128K, 1M, and 10M sequence length (l) for different LLaMA models. By compressing the KV cache to 3-bit precision, we can enable 1M context length inference with the LLaMA-7B model on a single A100-80GB GPU, and we can also enable 10M context length inference with the LLaMA-7B model on an 8-GPU system.*

Model	Operation	l=128K	l=1M	l=10M
LLaMA-7B	fp16	32.0	244.1	2441.4
	nuq4-1%	8.6	66.0	659.8
	nuq3-1%	6.6	50.7	507.2
	nuq2-1%	4.6	35.5	354.6
LLaMA-65B	fp16	160.0	1220.7	12207
	nuq4-1%	43.2	329.8	3297.5
	nuq3-1%	33.2	253.5	2534.6
	nuq2-1%	23.2	177.2	1771.6

5 CONCLUSION

As context lengths in LLMs increase, the KV cache activations surface as the dominant contributor to memory consumption. Quantization is a promising approach to reduce the size of KV cache activations, but prior solutions failed to represent activations accurately in ultra-low precisions, such as sub-4-bit. In contrast, we achieve accurate ultra-low precision KV cache quantization. By quantizing Keys per-channel before applying RoPE, we are able to better match the outlier distribution and mitigate the impacts of RoPE on quantization (due to it mixing pairs of channels which may have different average magnitudes). We use non-uniform quantization to better allocate the small number of quantization signposts at low precision. We observe significant accuracy improvements when employing dense-and-sparse quantization, particularly when detecting outliers at the same granularity as we compute quantization scaling factors. Crucially, we demonstrate that we can perform accurate calibration offline for Keys, as well as efficient online scaling factor and outlier threshold computation for Values. By leveraging these methods, we are able to enable accurate low-precision activation quantization, achieving 4.8x compression (nuq3-1% outliers) with only 0.1 perplexity degradation across different LLaMA, LLaMA-2, and Mistral models. Additionally, we show that Q-Norm improves accuracy for 2-bit quantization by helping mitigate distribution shift. Our methodology therefore supports inferring the LLaMA-7B model with a context length of **10M** on an 8-GPU serving system. Through our efficient kernel implementation, we are able to show improved latency relative to the fp16 baseline, demonstrating how our method allows for improved latency in addition to the memory savings.

6 LIMITATIONS

While our work enables accurate long-context length inference by reducing the memory requirements, there is significant work required for training long context length models with greater than 100K context length. This work is orthogonal to our efforts, which are constrained to efficient inference with long context length models. Additionally, our latency benchmarking results currently focus on memory-bandwidth bound generation rather than prompt processing (where we need to compress multiple Keys and Values at once). In future work, we plan to develop dedicated efficient kernels for block Key/Value compression in order to efficiently compress multiple Keys and Values at once. Finally, in the current end-to-end implementation, there are inefficiencies in how memory allocation is handled for updating the sparse matrix (where the data corresponding to the previous tokens have to be copied when concatenating them with the data from the new token). In future work, we plan to optimize this by doing blocked allocation to avoid overheads from reallocating memory.

7 ACKNOWLEDGEMENTS.

The authors would like to acknowledge Nicholas Lee for helpful discussions and feedback. We acknowledge gracious support from Intel, Furiosa, Apple, and Samsung SAIT. We also appreciate the support from Microsoft through their Accelerating Foundation Model Research, including great support from Sean Kuno. Furthermore, we appreciate support from Google Cloud, the Google TRC team, and specifically Jonathan Caton, and Prof. David Patterson. Prof. Keutzer's lab is sponsored by the Intel corporation, Intel One-API, Intel VLAB team, the Intel One-API center of excellence, as well as funding through BDD and BAIR. We appreciate great feedback and support from Ellick Chan, Saurabh Tangri, Andres Rodriguez, and Kittur Ganesh. Sehoon Kim would like to acknowledge the support from the Korea Foundation for Advanced Studies (KFAS). Amir Gholami was supported through funding from Samsung SAIT. Michael W. Mahoney would also like to acknowledge a J. P. Morgan Chase Faculty Research Award as well as the DOE, NSF, and ONR. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

REFERENCES

[1] Anthropic. Introducing claude 2.1, Nov 2023.

[2] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

[3] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

[4] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

[5] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[7] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

[8] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

[9] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302, 2019.

[10] Goran Flegar and Enrique S Quintana-Ortí. Balanced csr sparse matrix-vector product on graphics processors. In *Euro-Par 2017: Parallel Processing: 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28–September 1, 2017, Proceedings 23*, pages 697–709. Springer, 2017.

[11] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021.

[12] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

[13] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael Mahoney, and Kurt Keutzer. Ai and memory wall. *RiseLab Medium Post*, 2021.

[14] Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

[15] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[16] Jung Hwan Heo, Jeonghoon Kim, Beomseok Kwon, Byeongwook Kim, Se Jung Kwon, and Dongsoo Lee. Rethinking channel dimensions to isolate outliers for low-bit weight quantization of large language models. *arXiv preprint arXiv:2309.15531*, 2023.

[17] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[18] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

[19] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*, 2023.

[20] Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

[21] Liang Li, Qingyuan Li, Bo Zhang, and Xiangxiang Chu. Norm tweaking: High-performance low-bit quantization of large language models. *arXiv preprint arXiv:2309.02784*, 2023.

[22] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

[23] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.

[24] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.

[25] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization. 2023.

[26] Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*, 2023.

[27] OpenAI. New models and developer products announced at devday 2023, Nov 2023.

[28] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

[29] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.

[30] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[31] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[32] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[33] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[34] Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.

[35] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414, 2022.
[36] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.
[37] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.
[38] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. *arXiv preprint arXiv:2310.19102*, 2023.
[39] Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. Pose: Efficient context window extension of llms via positional skip-wise training. *arXiv preprint arXiv:2309.10400*, 2023.

A ROPE EQUATION

The rotation matrix for RoPE is provided in Equation 4, where c and s are cosine and sine functions, $\theta_i = 10000^{-2(i-1)/d}$, d is the attention head dimension, and n is the current position in the sequence:

$$\begin{bmatrix} \mathrm{c}(n\theta_1) & -\mathrm{s}(n\theta_1) & \cdots & 0 & 0 \\ \mathrm{s}(n\theta_1) & \mathrm{c}(n\theta_1) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \mathrm{c}(n\theta_{d/2}) & -\mathrm{s}(n\theta_{d/2}) \\ 0 & 0 & \cdots & \mathrm{s}(n\theta_{d/2}) & \mathrm{c}(n\theta_{d/2}) \end{bmatrix} \tag{4}$$

The Query vectors computed at each iteration will have RoPE applied (to obtain $\tilde{Q}_m = R^d_{\theta,m} * Q_m$). When caching Key vectors, we therefore need to either cache $\tilde{K}_n = R^d_{\theta,n} * K_n$, or else we need to cache K_n and apply $R^d_{\theta,n}$ on-the-fly during inference. In order to apply $R^d_{\theta,n}$ efficiently on-the-fly, we leverage the element-wise formulation of RoPE rather than the matrix-multiplication formulation from Equation 4. The element-wise formulation for $R^d_{\theta,n}x$ is as follows, where $\odot$ is the element-wise multiplication operator (note that the formulation that we use matches the implementation in Huggingface for LLaMA, and it is a different but equivalent formulation to the element-wise expression in [31]):

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{\frac{d}{2}} \\ x_{\frac{d}{2}+1} \\ \vdots \\ x_{d-1} \\ x_d \end{bmatrix} \odot \begin{bmatrix} \mathrm{c}(\theta_1 n) \\ \mathrm{c}(\theta_2 n) \\ \vdots \\ \mathrm{c}(\theta_{\frac{d}{2}} n) \\ \mathrm{c}(\theta_1 n) \\ \vdots \\ \mathrm{c}(\theta_{\frac{d}{2}-1} n) \\ \mathrm{c}(\theta_{\frac{d}{2}} n) \end{bmatrix} + \begin{bmatrix} -x_{\frac{d}{2}+1} \\ -x_{\frac{d}{2}+2} \\ \vdots \\ -x_d \\ x_1 \\ \vdots \\ x_{\frac{d}{2}-1} \\ x_{\frac{d}{2}} \end{bmatrix} \odot \begin{bmatrix} \mathrm{s}(\theta_1 n) \\ \mathrm{s}(\theta_2 n) \\ \vdots \\ \mathrm{s}(\theta_{\frac{d}{2}} n) \\ \mathrm{s}(\theta_1 n) \\ \vdots \\ \mathrm{s}(\theta_{\frac{d}{2}-1} n) \\ \mathrm{s}(\theta_{\frac{d}{2}} n) \end{bmatrix} \tag{5}$$

By leveraging this element-wise implementation, we can apply RoPE on-the-fly to the Key activations (after dequantizing the Key activations and before multiplying them with the corresponding elements in the Query vector).

B KEY AND VALUE DYNAMIC RANGE

Figure 5 shows the portion of the elements contained within difference percentages of the dynamic range for both Keys and Values. The majority of values ($\sim 99\%$) are contained in a small portion of the dynamic range, and a small portion of numerical outliers skew the dynamic range that must be represented. This motivates our dense-and-sparse approach which removes numerical outliers and stores them in a separate sparse matrix, thereby restricting the range that needs to be represented in the dense component.

C DERIVATION FOR SENSITIVITY ANALYSIS

The following derivation is adapted from [18], with the only difference being that it is estimating sensitivity with respect to activations rather than gradients. Let the activation at a given layer for input data X be denoted as A, and let the loss of the neural network with respect to activation A be represented as $\mathcal{L}(A)$. Let the gradient and Hessian of the loss function with respect to activation A be denoted as $g = \frac{\partial}{\partial A}\mathcal{L}(A)$ and $H = \frac{\partial^2}{\partial A^2}\mathcal{L}(A)$, respectively. Let $A_Q = Q(A)$ be the application of the quantization function Q to A, and let $A - A_Q$ represent the quantization error. Note that the subsequent derivations assume that the activation and gradient matrices are all flattened to one dimension.

By performing Taylor expansion of the loss with respect to the Hessian for data X, we get:

$$\mathcal{L}(A_Q) \simeq \mathcal{L}(A) - g^\top (A - A_Q) + \frac{1}{2}(A - A_Q)^\top H (A - A_Q). \tag{6}$$

Assuming that the loss has converged to a local minimum, g can be approximated as zero, which (omitting constants) yields the following formula for the increased error in the loss E due to perturbation in A [18]:

$$E = (A - A_Q)^\top H (A - A_Q). \tag{7}$$

Prior work [8, 9] has used either the largest Hessian eigenvalue or the Hessian trace to estimate the sharpness of the loss landscape with respect to a particular layer. However, the full Hessian is prohibitively expensive to compute for LLMs, and it is challenging to even estimate the Hessian eigenvalues. We therefore aim to leverage the Fisher information approximation for the Hessian since it is easier to compute. Given activation A and the corresponding Hessian H, we can approximate the Hessian as the Fisher information matrix $H \simeq \mathcal{F} = gg^\top$,

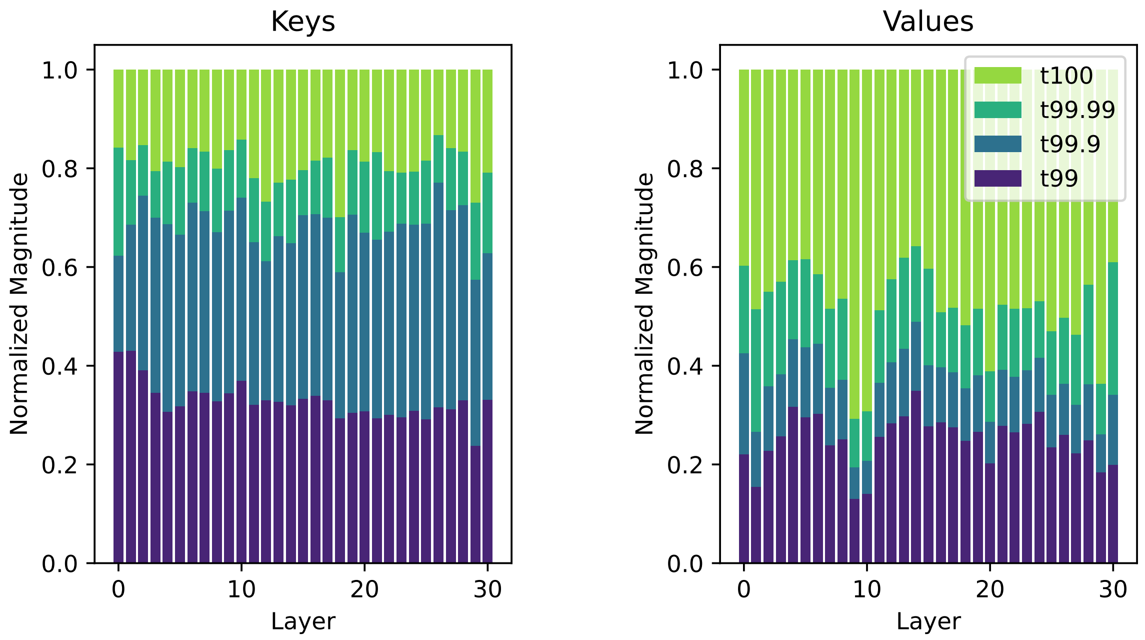


Figure 5: *Distribution of the magnitude of elements of Key (Pre-RoPE) and Value activations for different layers of LLaMA-7B, computed on a single sample with sequence length 2K from the Wikitext-2 dataset. The normalized magnitude is computed by dividing by the largest magnitude value in that layer. As one can see, for both Key and Value activations, the majority of values lie in a small portion of the dynamic range, with a few numerical outliers skewing the dynamic range (and thereby reducing the fidelity when quantizing to low precision).*

Table 7: *Ablation Study: Perplexity comparison of per-token and per-channel quantization for KV cache activations for LLaMA-7B. PT refers to per-token quantization, and PC refers to per-channel quantization.*

Datatype	Key Dim.	Value Dim.	Perplexity	KV Cache Size (GB) Seqlen 128K
fp16	-	-	5.68	32.0
int3	PT	PT	10.87	6.0
int3	PC	PC	8.19	6.0
int3	PC	PT	**6.99**	6.0

where g are the gradients for activation A. We can then leverage the approximation that cross-element interactions are negligible in order to approximate the Fisher information matrix as a diagonal matrix such that it can be expressed as $\mathcal{F} = \text{diag}(g \odot g)$.

D PER-CHANNEL KEY QUANTIZATION ABLATIONS

We report results in Table 7 demonstrating the perplexity for different KV cache compression ratios, showing that per-channel quantization for Keys and per-token quantization for Values outperforms the standard per-token quantization approach for both Keys and Values, yielding an improvement of 3.88 perplexity for the LLaMA-7B model at 3-bit precision. This demonstrates the benefits of per-channel Key quantization to mitigate the large outlier channels in Keys. Additionally, although there are per-channel outliers in Values, we observe that per-channel quantization for Values actually performs worse than per-token quantization. We hypothesize that this behavior is because per-channel Value quantization leads to greater error accumulation in particular output values (since the result of the attention scores multiplied by one channel of the Values will be localized to a single value in the output vector), which leads to greater quantization error at later model layers. Another concurrent work, KIVI [25], observes similar behavior for per-channel Value quantization, which they attribute to the fact that per-token Value quantization confines the error to each token. Assuming that the output is a weighted sum of only a few important tokens (as only a few attention scores are large), a perturbation in these tokens can lead to significant degradation.

E PRE-ROPE KEY QUANTIZATION ABLATIONS

As shown in Table 8, pre-RoPE Key quantization achieves higher accuracy than post-RoPE quantization, with an improvement of 0.65 perplexity for 3-bit quantization with the LLaMA-7B model. These results show that the rotary positional embeddings make Key quantization

Table 8: *Ablation Study: Perplexity comparison of Pre-RoPE and post-RoPE Key quantization for LLaMA-7B (using per-channel Key quantization and per-token Value quantization). Pre-RoPE quantization leads to significant improvement (see Section 3.2 for more details).*

Datatype	Scheme	Perplexity	KV Cache Size (GB) Seqlen 128K
fp16	-	5.68	32.0
int3	post-RoPE	6.99	6.0
int3	pre-RoPE	**6.34**	6.0

more challenging due to mixing pairs of channels with different magnitudes. Pre-RoPE quantization thereby allows for more accurate quantization at low precision.

F SENSITIVITY-WEIGHTED NON-UNIFORM QUANTIZATION ABLATIONS

Table 9 shows perplexity evaluation results across different LLaMA, LLaMA-2, and Mistral models on Wikitext-2 using nf3, nuq3, and nuq3 without using sensitivity-weighting. These results demonstrate the benefits of our sensitivity-weighted non-uniform quantization approach, relative to NormalFloat quantization [6], as we achieve consistent accuracy improvements of up to 0.32 perplexity across different models (with particularly pronounced improvements for larger models). For 3-bit quantization with the LLaMA-7B model, we observe a 0.33 perplexity improvement relative to uniform quantization. The gains relative to uniform quantization are particularly noticeable for 3-bit and 2-bit quantization, where the benefits of non-uniform quantization are more pronounced due to the reduced precision. These results also demonstrate the necessity of our sensitivity-weighting approach in order to derive performant non-uniform datatypes using a k-means based approach.

Figure 6 shows the relationship between the Fisher information and the normalized activation values for the LLaMA-7B model. For the Value matrices, we observe that the average Fisher information is higher close to the middle, which leads to quantization centroids being pulled closer to the center of the distribution. For the Key matrices, we observe that the average sensitivity across different magnitudes is relatively constant, which leads to wider spacing for the centroids. These results show that using Fisher information to derive a sensitivity-weighted per-layer datatype allows for better representation of Keys and Values.

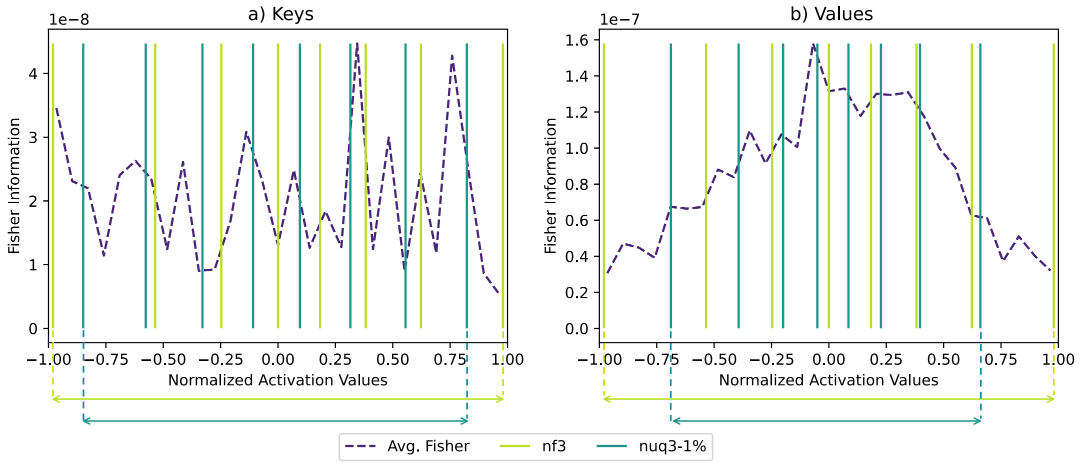


Figure 6: *Relationship between average Fisher information and normalized activation values for the 3-bit quantized LLaMA-7B model with 1% outliers, shown for layer 29. a) Fisher information versus normalized activation values for Keys (with 1% outliers). b) Fisher information versus normalized activation values for Values (with 1% outliers). These results demonstrate how sensitivity-weighted k-means shifts the quantization signposts inward relative to NormalFloat (with more condensed signposts across a slightly narrower range). Additionally, for the Value matrix, the values close to 0 have a higher average sensitivity. This leads to more condensed signpost assignment relative to the Key matrix, demonstrating how our non-uniform approach derives accurate datatypes for layers with differing sensitivity characteristics.*

G PER-VECTOR DENSE-AND-SPARSE QUANTIZATION ABLATIONS

Table 10 shows the performance improvements we observe when isolating a small portion of outliers and storing them in a sparse format. We provide results both with using a single per-matrix outlier threshold, as well as with applying separate outlier thresholds per-vector. In particular, we see greater improvements by employing outlier detection with a different threshold per-channel for Keys and per-token for Values. This provides additional benefits since some values which would be considered outliers for the entire matrix are not actually

Table 9: *Ablation Study: Ablation of our sensitivity-weighted non-uniform datatype for different models on Wikitext-2. All experiments use pre-RoPE per-channel quantization for Keys and per-token quantization for Values (meaning that all configurations are the same as in KVQuant, except for the datatype). We compare against both uniform (int3) and non-uniform (nf3) [6] approaches, as well as with using "unweighted" k-means (i.e., not sensitivity-weighted) to compute the non-uniform quantization signposts. Note that there is slight variation in average bitwidth across models due to the differing hidden dimensions. Results report perplexity with 2K/4K/8K sequence length for LLaMA, LLaMA-2, and Mistral, respectively.*

Method	LLaMA-7b	LLaMA-13b	LLaMA-30b	LLaMA-65b	LLaMA-2-7b	LLaMA-2-13b	LLaMA-2-70b	Mistral-7b	Avg. Num. Bits
baseline	5.68	5.09	4.10	3.53	5.12	4.57	3.12	4.76	16
int3	6.34	5.62	4.69	4.16	6.10	5.59	3.46	5.52	3.00-3.01
nf3	6.05	5.42	4.51	3.84	5.55	5.15	3.27	5.13	3.00-3.01
nuq3 (unweighted)	6.84	6.16	5.37	4.57	8.52	7.66	3.67	5.29	3.00-3.01
nuq3	**6.01**	**5.34**	**4.41**	**3.74**	**5.49**	**4.83**	**3.26**	**5.03**	3.00-3.01

Table 10: *Ablation Study: Perplexity comparison of different outlier isolation methods for LLaMA-7B. Per-vector outlier detection allows for significant accuracy improvements relative to per-tensor outlier detection. All nuq4 and nuq3 experiments use per-token quantization for Values and per-channel quantization for Keys (pre-RoPE). "PV" refers to using per-vector outlier thresholds, and "PM" refers to using a single per-matrix outlier threshold.*

Datatype	% Outliers	Outlier Dim.	Perplexity	KV Cache Size (GB) Seqlen 128K
fp16	-	-	5.68	32.0
nuq3	-	-	6.01	6.0
nuq3	0.1%	PM	5.94	6.1
nuq3	0.1%	PV	**5.86**	6.1
nuq3	1%	PM	5.86	6.6
nuq3	1%	PV	**5.76**	6.6

outliers within a particular channel (so they are not hard to quantize). It is therefore better to directly target the outliers that will skew the quantization range for a particular channel. By removing 1% of outliers using per-vector thresholds, we can achieve an additional 0.25 reduction in perplexity for the LLaMA-7b model at 3 bits, **thereby enabling 3-bit quantization with under 0.1 degradation in perplexity**.

H Q-NORM ABLATIONS

Table 11 provides perplexity results for LLaMA-7B and LLaMA-13B with and without Q-Norm. As shown in the table, Q-Norm provides noticeable accuracy improvements for 2-bit quantization, but it doesn't provide significant accuracy improvements for 3-bit or 4-bit quantization. Figure 7 demonstrates how the minimization of distribution shift provided by Q-Norm leads to reduced quantization error (particularly at later layers in the network).

Additionally, we experimented with using *per-vector* Q-Norm, where we normalize each channel for Keys and each token for Values to ensure the distribution has the same mean and standard deviation post-quantization. For Keys, per-vector Q-Norm requires offline computation of the required normalization per-channel; and for Values, per-vector Q-Norm requires online computation of the required normalization per-token. With per-vector Q-Norm, the normalization parameters must also be stored separately from the per-vector outlier thresholds and cannot be fused with the NUQ datatype. Table 11 also compares employing per-vector Q-Norm with per-matrix Q-Norm. Although we observe similar accuracy with per-vector Q-Norm when incorporating dense-and-sparse quantization, we observe worsened performance when we aren't using dense-and-sparse quantization. We attribute this to the larger relative shift when applying normalization per-vector, as well as the impacts that significant changes to the centroids can have on outlier values (in the case where we aren't using dense-and-sparse quantization, large perturbations in these outlier values can lead to significant accuracy loss). The need to only slightly tweak normalization parameters when combating distribution shift is similar to [21], which describes how weight quantization can be improved by slightly adjusting layernorm parameters to avoid distribution shift. Due to the improved performance of per-matrix Q-Norm for dense-only quantization (as well as potential inference overheads with per-vector Q-Norm from having to separately rescale the non-uniform centroids per-vector and compute normalization statistics on-the-fly for Values), we focus on per-matrix Q-Norm. However, there is potential to improve accuracy through fine-grained normalization in future work; for example, Figure 8 demonstrates how per-vector Q-Norm provides improved performance for the LLaMA-2-70B-32K model when evaluating perplexity on long context lengths.

I CALIBRATION ABLATIONS

Table 12 shows accuracy results when using offline calibration for computing the scaling factors for the Keys. For 4-bit quantization, we observe no accuracy loss when calibrating scaling factors offline. For 3-bit quantization, we observe minor accuracy degradation when not employing outlier extraction methods. However, if we remove a small percentage of outliers, then the accuracy with offline calibration is the

Table 11: *Ablation Study: Perplexity with LLaMA-7B and LLaMA-13B with and without Q-Norm (including results for both per-matrix ("PM") and per-vector ("PV") Q-Norm). For 4-bit and 3-bit quantization, Q-Norm provides minimal perplexity improvements; however, at 2-bit quantization, Q-Norm improves performance by reducing distribution shift. Per-vector Q-Norm performs similarly to per-matrix Q-Norm when incorporating dense-and-sparse quantization, and performs worse than per-matrix Q-Norm without dense-and-sparse quantization. Note that there is slight variation in average bitwidth across models due to the differing hidden dimensions.*

Datatype	Norm	LLaMA-7B	LLaMA-13B	Avg. Bits (KV Cache)
fp16	-	5.68	5.09	16
nuq4	-	5.73	5.15	4.00
nuq4	PV	5.76	5.16	4.01
nuq4	PM	5.74	5.14	4.00
nuq4-1%	-	5.70	5.11	4.32-4.33
nuq4-1%	PV	5.70	5.10	4.33
nuq4-1%	PM	5.70	5.10	4.32-4.33
nuq3	-	6.01	5.34	3.00
nuq3	PV	6.11	5.42	3.01
nuq3	PM	5.97	5.34	3.00
nuq3-1%	-	5.76	5.15	3.32-3.33
nuq3-1%	PV	5.75	5.16	3.33
nuq3-1%	PM	5.75	5.16	3.32-3.33
nuq2	-	8.70	7.50	2.00
nuq2	PV	34.81	18.56	2.01
nuq2	PM	8.17	7.29	2.00
nuq2-1%	-	6.24	5.50	2.32-2.33
nuq2-1%	PV	6.06	5.41	2.33
nuq2-1%	PM	6.06	5.40	2.32-2.33

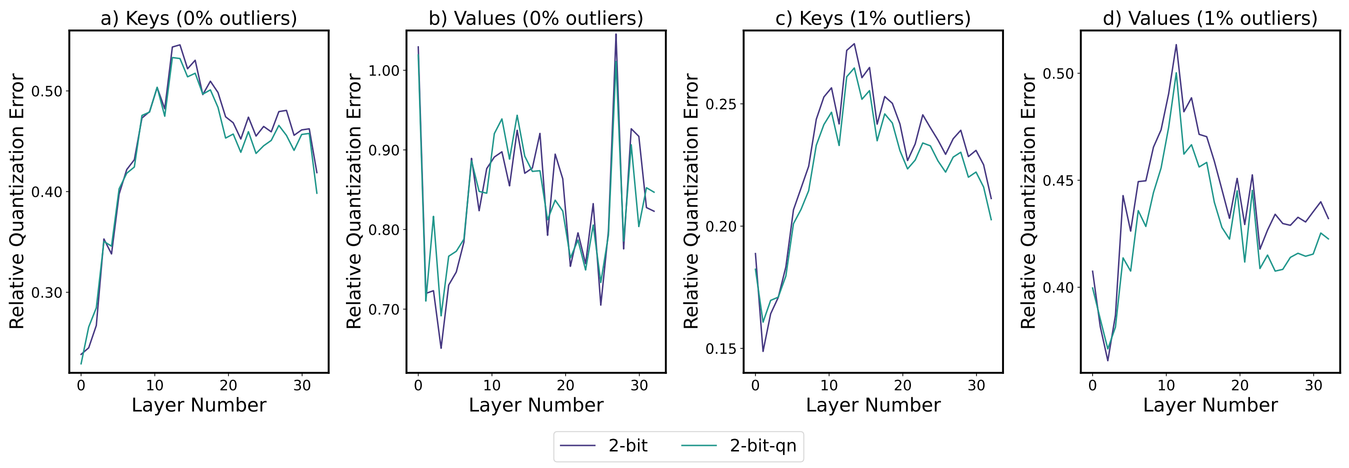


Figure 7: *Quantization error for 2-bit quantization with and without Q-Norm for the LLaMA-7B model, evaluated on a sample of length 2K from Wikitext-2. a) Quantization error for Keys (with 0% outliers). b) Quantization error for Values (with 0% outliers). c) Quantization error for Keys (with 1% outliers). d) Quantization error for Values (with 1% outliers). As one can see, using Q-Norm is helpful with/without dense-and-sparse quantization and results in improved quantization error, especially for later layers. See Table 11 for the corresponding perplexity results which clearly show the benefit of using Q-Norm.*

same as computing the scaling factors online per-channel during evaluation. This demonstrates that when incorporating outlier extraction methods, we are better able to perform offline calibration due to reduced sensitivity to outliers (either to outliers during calibration that exaggerate the quantization range, or to outliers during evaluation that cannot be represented accurately if there weren't large outliers observed during calibration).

Table 13 shows the runtime for the *topk* operation for the LLaMA-7B model (which is required for computing outlier thresholds online). It compares the runtime of the *topk* operation with the runtime for the QKV projections, finding that the *topk* runtime is 60% of the matrix-vector operation runtime for a single projection layer. The *topk* operation can also be performed efficiently on the CPU, so we can actually run this operation *in parallel* with the subsequent linear layer matrix-vector operations on the GPU (which is possible by computing the Value projection before the Key and Query projections). This allows us to compress the activations dynamically without added runtime overhead, thereby enabling online scaling factor computation for the Value tensors.

Table 14 shows the perplexity of the LLaMA-7B model using different numbers of samples during calibration. The results show that perplexity is similar across the range of the number of samples tested for each bit width. This shows that the calibration step does not require

Table 12: *Ablation Study: Model accuracy when using offline calibration for Keys with LLaMA-7B. When incorporating outlier detection, offline calibration for Keys is able to perform comparably with online calibration. All nf4 and nf3 experiments use per-token quantization for Values and per-channel quantization for Keys (pre-RoPE), and experiments with outliers use per-vector outlier detection.*

Datatype	% Outliers	Perplexity (Online for K)	Perplexity (Offline for K)
fp16	-	5.68	5.68
nuq4	-	5.73	5.73
nuq4	1%	5.70	5.70
nuq3	-	5.96	6.01
nuq3	1%	5.77	5.76

Table 13: *topk runtime on a vector of length 4096 for computing outlier thresholds when using 1% sparsity (compared with the runtime for the QKV matrix multiplications, which are 4096×4096 by 4096 matrix-vector multiplications for the LLaMA-7B model). The runtime is reported on a system with an A6000 GPU and an Intel Xeon Gold 6126 CPU. We find that the runtime for the topk operation is only 60% of the runtime of each matvec operation. Additionally, the topk operation can be performed efficiently on the CPU; we can therefore run this operation in parallel with subsequent linear layer operations on the GPU to compress the activations dynamically without added overhead. Note that the CPU runtime includes the time for copying the vector to the CPU.*

Operation	Device	Outlier %	Runtime (ms)
QKV Projection	GPU	-	0.308
topk	GPU	1%	0.073
topk	CPU	1%	0.059
QKV Projection / *topk* (Fused)	GPU / CPU	1%	0.309

Table 14: *Ablation Study: Perplexity on Wikitext-2 for the LLaMA-7B model when using different number of samples of length 2K during calibration.*

Method	Number of Samples						
	1	2	4	8	16	32	64
nuq4	5.75	5.74	5.73	5.75	5.73	5.74	5.74
nuq4-1%	5.69	5.70	5.70	5.70	5.70	5.70	5.70
nuq3	6.00	6.00	5.95	5.98	6.01	6.02	5.98
nuq3-1%	5.75	5.77	5.76	5.76	5.76	5.77	5.76
nuq2	8.34	8.32	8.26	8.70	8.61	8.93	8.45
nuq2-1%	6.20	6.22	6.23	6.23	6.24	6.22	6.25

Table 15: *Runtime for computing Fisher information as well as for calibration (including k-means) with 16 samples for LLaMA-65B quantization. Runtime for computing Fisher information was computed on an 8-GPU A100-80GB system. Runtime for calibration (including k-means) was performed on an Intel Xeon Gold 6442Y CPU, and is shown for a single layer. Note that calibration is independent for each layer, so it can be easily parallelized.*

Operation	Runtime (minutes)
Computing Fisher Information	2.8
4-bit Calibration Per-Layer (including k-means)	4.5
3-bit Calibration Per-Layer (including k-means)	2.7
2-bit Calibration Per-Layer (including k-means)	1.9

a large number of calibration samples to attain high accuracy. Additionally, Table 15 shows how both Fisher information computation and calibration (including k-means) per-layer take only a few minutes for the LLaMA-65B model on a typical server machine. Even if we perform calibration sequentially for each layer, the entire calibration process would take a maximum of 6 hours for the LLaMA-65B model at 4-bit precision.

J ADDITIONAL EXPERIMENTAL DETAILS

For our empirical evaluation, we use 16 calibration samples of sequence length 2K from the Wikitext-2 training set (as well as the corresponding gradients) to derive the per-channel scaling factors and zero-points, to derive the non-uniform datatypes for both Keys and Values, and to estimate layer-wise sensitivity for mixed-precision experiments. We measured perplexity on both Wikitext-2 and on C4 using a sequence length equal to the maximum context length of the model (2K for LLaMA, 4K for LLaMA-2, and 8K for Mistral-7B). For baseline experiments, we use post-RoPE quantization, both since this is required from an efficiency perspective without a dedicated kernel implementation, and because it provides better accuracy when quantizing Keys per-token as shown in Appendix L.

Table 16: *Evaluation of our method for different models using the perplexity (PPL) measured on Wikitext-2. Non-uniform quantization results are using pre-RoPE per-channel quantization for Keys. "gs64/128" refers to baseline experiments using grouping with group size 64/128. 2-bit results leverage per-matrix Q-Norm. Note that there is slight variation in average bitwidth across models due to the differing hidden dimensions.*

Method	LLaMA-7b	LLaMA-13b	LLaMA-30b	LLaMA-65b	LLaMA-2-7b	LLaMA-2-13b	LLaMA-2-70b	Mistral-7b	Avg. Num. Bits
baseline	5.68	5.09	4.10	3.53	5.12	4.57	3.12	4.76	16
int4	5.98	5.32	4.34	3.73	5.66	5.01	3.31	4.97	4.00-4.01
int4-gs128	5.77	5.16	4.16	3.57	5.32	4.71	3.16	4.82	4.16
int4-gs64	5.73	5.14	4.14	3.56	5.25	4.66	3.14	4.80	4.31
nf4	5.87	5.23	4.25	3.63	5.47	4.90	3.22	4.91	4.00-4.01
nf4-gs128	5.77	5.17	4.17	3.58	5.30	4.71	3.16	4.83	4.16
nuq4	5.73	5.15	4.16	3.57	5.18	4.63	3.15	4.81	4.00-4.02
+ 0.1% outliers	5.71	5.12	4.15	3.55	5.16	4.61	3.14	4.80	4.04-4.06
+ 0.5% outliers	5.70	5.11	4.12	3.54	5.14	4.60	3.13	4.78	4.16-4.19
+ 1.0% outliers	**5.70**	**5.11**	**4.12**	**3.54**	**5.14**	**4.59**	**3.13**	**4.78**	4.32-4.35
int3	10.87	8.69	6.82	6.37	22.71	18.26	7.68	7.64	3.00-3.01
int3-gs128	6.17	5.47	4.44	3.78	6.15	5.34	3.33	5.16	3.15
int3-gs64	5.93	5.29	4.26	3.66	5.64	4.98	3.23	5.00	3.30
nf3	7.33	6.21	5.46	4.44	9.96	9.50	4.06	6.30	3.00-3.01
nf3-gs128	6.26	5.52	4.54	3.83	6.21	5.43	3.38	5.23	3.15
nuq3	6.01	5.34	4.41	3.74	5.49	4.83	3.26	5.03	3.00-3.02
+ 0.1% outliers	5.86	5.28	4.27	3.64	5.32	4.71	3.23	4.96	3.04-3.06
+ 0.5% outliers	5.79	5.15	4.19	3.60	5.22	4.66	3.17	4.86	3.16-3.19
+ 1.0% outliers	**5.76**	**5.15**	**4.17**	**3.59**	**5.20**	**4.64**	**3.16**	**4.84**	3.32-3.35
int2	11779	69965	1470	7272	4708	3943	976	573	2.00-2.01
int2-gs128	37.37	41.77	16.49	13.63	117.88	93.09	18.31	51.96	2.14
int2-gs64	11.09	9.84	6.60	5.54	25.69	26.83	5.93	12.47	2.28
nf2	3210.5	5785.6	2044.2	5367.3	13601	4035.6	3680.3	902.51	2.00-2.01
nf2-gs128	351.23	141.19	60.97	31.69	634.59	642.44	71.21	252.85	2.14
nuq2	8.17	7.29	7.05	27.17	9.75	29.25	4.98	7.33	2.00-2.02
+ 0.1% outliers	6.91	5.84	4.94	4.13	6.47	5.54	3.76	6.05	2.04-2.06
+ 0.5% outliers	6.26	5.52	4.48	3.86	5.67	5.05	3.35	5.33	2.16-2.19
+ 1.0% outliers	**6.06**	**5.40**	**4.43**	**3.76**	**5.50**	**4.92**	**3.29**	**5.16**	2.32-2.35

We make several assumptions in order to estimate average bit widths and KV cache sizes for different approaches. We compute these estimates assuming a sequence length of 128K. For integer quantization, we assume a low-precision integer offset and a 16-bit scaling factor, whereas for NormalFloat and NUQ we assume that the zero-point and offset are each 16-bit. For the sparse matrices, 32-bit integers are assumed for the per-token indices (since we need to support long sequence lengths), and the elements and per-element indices are assumed to be 16-bit. This means that for CSR, the rows are assumed to be 32-bit and the columns and values are assumed to be 16-bit, whereas for CSC, the columns are assumed to be 32-bit and the rows and values are assumed to be 16-bit.

K FULL PERPLEXITY EVALUATION AND MMLU EVALUATION

Tables 16 and 17 show perplexity evaluation results across different LLaMA, LLaMA-2, and Mistral models on Wikitext-2 and C4, respectively. These results demonstrate the benefits of our approach for KV cache compression across different model sizes as well as across different language modeling datasets. Table 18 also provides zero-shot evaluation results on MMLU for LLaMA-7B and LLaMA-13B, demonstrating how we can maintain accuracy even for 3-bit KV cache quantization [15]. We used the Language Model Evaluation Harness to run zero-shot evaluation across all MMLU tasks [11].

L POST-ROPE PER-TOKEN QUANTIZATION ABLATION

Table 19 shows perplexity evaluation on Wikitext-2 for the LLaMA-7B model with uniform quantization, with Keys quantized pre-RoPE and post-RoPE. These results show that post-RoPE Key quantization is superior to pre-RoPE Key quantization when quantizing Keys per-token. This is because when rotating an outlier channel with large average magnitude and another channel with smaller average magnitude together, at some positions in the sequence part of the magnitude from the outlier channel will be shifted to the smaller channel. This partially mitigates the impact of the outlier channel on skewing the quantization range for some of the tokens in the sequence. As such, for our baseline comparisons, we use post-RoPE per-token Key quantization to serve as a stronger baseline.

M ADDITIONAL LONG SEQUENCE LENGTH EVALUATION

Figure 8 shows perplexity evaluation results with varying amounts of input context for 2-bit quantization. These results show that Q-Norm (and in particular, per-vector Q-Norm) can provide significant perplexity advantages for 2-bit quantization with long sequence length models. Additionally, for the LLaMA-2-70B-32K model, we observe perplexity degradation when going to a context length of 32K, likely due to error accumulation; however, this is largely mitigated by employing per-vector Q-Norm.

Table 17: *Evaluation of our method for different models using the perplexity (PPL) measured on C4. PC-K refers to using per-channel quantization for Keys. "gs64/128" refers to baseline experiments using grouping with group size 64/128. 2-bit results leverage per-matrix Q-Norm. Note that there is slight variation in average bitwidth across models due to the differing hidden dimensions.*

Method	LLaMA-7b	LLaMA-13b	LLaMA-30b	LLaMA-65b	LLaMA-2-7b	LLaMA-2-13b	LLaMA-2-70b	Mistral-7b	Avg. Num. Bits
baseline	7.08	6.61	5.98	5.62	6.63	6.05	4.97	5.71	16
int4	7.40	6.82	6.18	5.75	7.31	6.59	5.12	5.91	4.00-4.01
int4-gs128	7.16	6.67	6.02	5.65	6.87	6.20	5.00	5.76	4.16
int4-gs64	7.12	6.64	6.00	5.63	6.79	6.15	4.99	5.75	4.31
nf4	7.27	6.74	6.10	5.69	7.09	6.45	5.06	5.85	4.00-4.01
nf4-gs128	7.16	6.66	6.02	5.65	6.86	6.20	5.00	5.77	4.16
nuq4	7.13	6.65	6.02	5.64	6.70	6.11	5.00	5.75	4.00-4.02
+ 0.1% outliers	7.11	6.63	6.00	5.63	6.68	6.08	4.99	5.75	4.04-4.06
+ 0.5% outliers	7.10	6.62	5.99	5.62	6.66	6.07	4.98	5.73	4.16-4.19
+ 1.0% outliers	**7.09**	**6.62**	**5.99**	**5.62**	**6.65**	**6.06**	**4.98**	**5.72**	4.32-4.35
int3	12.97	10.95	9.13	8.27	30.14	28.57	16.00	8.84	3.00-3.01
int3-gs128	7.62	6.93	6.24	5.79	8.00	7.06	5.16	6.08	3.15
int3-gs64	7.34	6.78	6.11	5.70	7.29	6.59	5.08	5.92	3.30
nf3	8.90	7.84	7.43	6.37	14.92	13.75	5.96	7.27	3.00-3.01
nf3-gs128	7.65	6.99	6.29	5.82	8.03	7.12	5.24	6.16	3.15
nuq3	7.36	6.83	6.18	5.75	7.05	6.37	5.10	5.95	3.00-3.02
+ 0.1% outliers	7.24	6.72	6.08	5.68	6.87	6.20	5.06	5.88	3.04-3.06
+ 0.5% outliers	7.16	6.67	6.03	5.65	6.75	6.14	5.01	5.80	3.16-3.19
+ 1.0% outliers	**7.14**	**6.66**	**6.02**	**5.64**	**6.72**	**6.11**	**5.00**	**5.77**	3.32-3.35
int2	10892	100870	1411	7216	4708	4220	814	477	2.00-2.01
int2-gs128	43.49	56.25	21.07	17.05	113.49	97.04	23.67	50.73	2.14
int2-gs64	13.91	13.36	8.49	7.34	35.21	40.40	8.28	13.83	2.28
nf2	2850.1	4680.3	1617.1	5189.7	13081.2	4175.6	3216.9	1102.3	2.00-2.01
nf2-gs128	248.32	118.18	60.28	36.05	420.05	499.82	80.51	191.73	2.14
nuq2	10.28	9.05	9.27	60.87	15.16	43.77	7.24	8.40	2.00-2.02
+ 0.1% outliers	8.18	7.24	6.54	6.00	8.40	7.31	5.56	6.91	2.04-2.06
+ 0.5% outliers	7.50	6.91	6.21	5.78	7.28	6.53	5.16	6.23	2.16-2.19
+ 1.0% outliers	**7.38**	**6.83**	**6.15**	**5.73**	**7.06**	**6.38**	**5.11**	**6.08**	2.32-2.35

Table 18: *Zero-shot MMLU evaluation results for LLaMA-7B and LLaMA-13B when incorporating 3-bit KV cache quantization. The value reported is the weighted accuracy across all tasks. We observe minimal accuracy degradation with 3-bit quantization.*

Method	LLaMA-7b	LLaMA-13b	Avg. Num. Bits
baseline	32.2%	36.6%	16
nuq3-1%	**31.9%**	**36.2%**	3.32-3.33

Table 19: *Model accuracy when using pre-RoPE and post-RoPE quantization for LLaMA-7B with per-token Key quantization. Our experiments demonstrate that post-RoPE quantization is superior when using per-token Key quantization. Therefore, we decided to use these results for baseline comparison with per-token quantization.*

Datatype	Perplexity (Pre-RoPE)	Perplexity (Post-RoPE)
fp16	5.68	5.68
int4	6.02	5.98
int4-gs128	5.76	5.77
int3	14.68	10.87
int3-gs128	6.28	6.17

N MIXED-PRECISION QUANTIZATION

An additional method to optimize compression performance is to consider mixed-precision quantization, where different layers are assigned different bit widths. This can allow for more accurate compression down to low bit widths due to differing sensitivities to quantization error of different layers. Finding the best bit precision distribution for different layers is intractable with brute force methods as the search space is exponentially large. We therefore aim to derive a metric to determine which layers can be quantized to reduced precision with minimal degradation in model accuracy, thereby enabling efficient one-shot mixed-precision bit assignments.

Prior work on mixed-precision quantization has leveraged the largest Hessian eigenvalue or the Hessian trace to assess which layers are most sensitive to quantization [8, 9]; however, due to the computational challenges of computing the full Hessian or even estimating Hessian eigenvalues, we instead leverage the Fisher information approximation for the Hessian (as derived in Appendix C). Using the diagonal Fisher information matrix along with the quantization error, we can use the following sensitivity metric for a given layer (with activation A and quantized activation A_Q) to encompass both the quantization error as well as the Fisher information for that layer:

$$\Omega = \left(A - A_Q\right)^{\top} \mathcal{F} \left(A - A_Q\right). \tag{8}$$

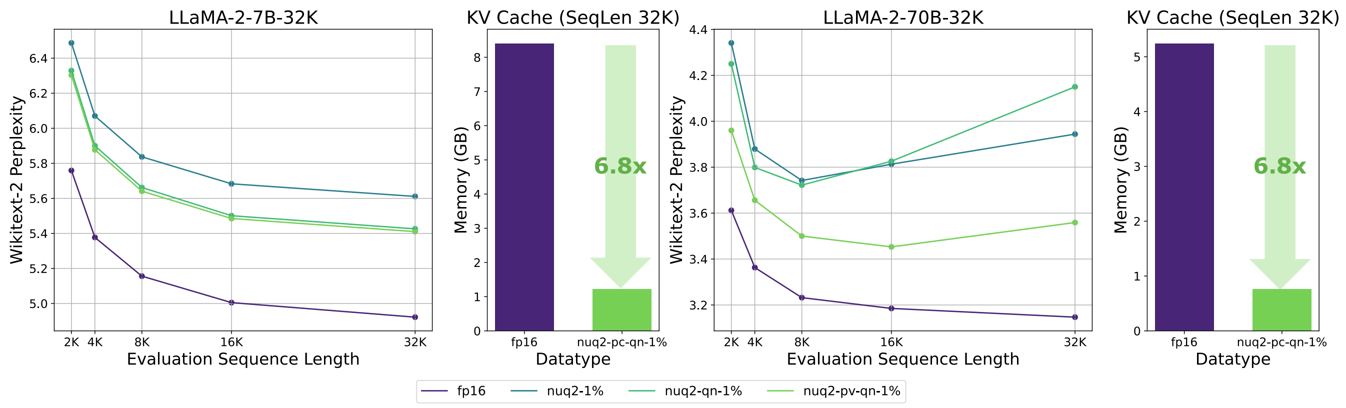


Figure 8: *Perplexity results for the LLaMA-2-7B-32K model (uptrained for long sequence lengths using positional interpolation [3]) as well as the LLaMA-2-70B-32K LongLoRA model [4] on the Wikitext-2 dataset using 2-bit quantization, evaluated using different amounts of input context. Results are shown using Q-Norm ("qn") as well as per-vector Q-Norm ("pv-qn"), and all results include 1% outliers. The memory reduction shown is for 2-bit quantization with per-vector Q-Norm. Our results show that Q-Norm is helpful for 2-bit quantization, and that per-vector Q-Norm provides additional benefits for 2-bit quantization with LLaMA-2-70B-32K. See Figure 4 for 4-bit and 3-bit quantization (which do not require Q-Norm).*

We use the quantization error computed at the lower precision (as well as sensitivity information computed in fp16) to determine which layers were most sensitive to being quantized to the lower precision level. Figure 9 shows the mixed-precision perplexity results on the Wikitext-2 dataset for the LLaMA-7B and LLaMA-13B models. We show mixed-precision results using our sensitivity metric, and we include both quantization error-based mixed-precision assignment and the inverse of the selection order from our sensitivity metric as baselines. We see improved performance from incorporating sensitivity-based analysis for determining activation precisions when employing mixed-precision quantization. Our sensitivity metric therefore allows for efficiently determining an accurate mixed-precision assignment in order to trade off KV cache size and model accuracy.

O KERNEL IMPLEMENTATION DETAILS

We implemented 4-bit lookup table-based kernels for matrix-vector multiplication between the Key or Value activations (packed as a lookup table (LUT) plus indices into the LUT per-element) and a full-precision activation vector. These kernels load the compressed Key and Value activations and dequantize them only as needed in order to minimize memory bandwidth utilization. All arithmetic is performed in fp16. The lookup table entries are the values of the sensitivity-weighted non-uniform datatype for that particular layer scaled according to the range of activations that need to be represented [6]. Note that it is possible to implement this using a single LUT shared across channels that is rescaled by a per-channel or per-token scaling factor and offset, but for simplicity we used a separate LUT per-channel or per-token for our initial implementation.

When selecting between the Compressed-Sparse Column (CSC format) and the Compressed-Sparse Row (CSR) format for storing the outliers for the Keys and Values, we needed to consider how easy it would be to append new vectors. When using CSC format for the Key matrix, we only need to append a single element to the column vector, as well as one new element to the row and value vectors per nonzero element in that new column. If we used CSR format, we would need to insert the new column and value elements in the middle of the existing column and value vectors, and we would need to recompute the elements of the row vector. When using CSR format for the Value matrix, we only need to append a single element to the row vector, as well as one new element to the column and value vectors per nonzero element in that new row. If we used CSC format, we would need to insert the new row and value elements in the middle of the existing row and value vectors, and we would need to recompute the elements of the column vector. We therefore used the CSC format for the Key matrices and the CSR format for the Value matrices.

One challenge with efficiently processing the sparse matrix-dense vector operation is that the sparsity distribution may be unbalanced. This poses a challenge for efficiently processing the sparse matrix on a GPU as there can be different numbers of nonzeros to process per thread. We therefore leverage a *balanced* sparse matrix-dense vector kernel based on [10, 18], which assigns an equal number of nonzeros per thread. This has greater synchronization overhead than assigning a single thread for an entire row or column when processing CSR/CSC matrices, but it leads to a more balanced work assignment between threads. We set the number of threads such that there were 10 nonzero values assigned to each thread. The dense non-uniform kernel and balanced sparse kernels are launched in one call to avoid overhead from summing the output vectors from these separate operations.

Table 20 shows a detailed breakdown of kernel runtime, including how much time is spent packing vectors into the compressed format and how much time is spent on the dense and sparse matrix-vector multiplications. We find that even with 1% sparsity, we can attain significant

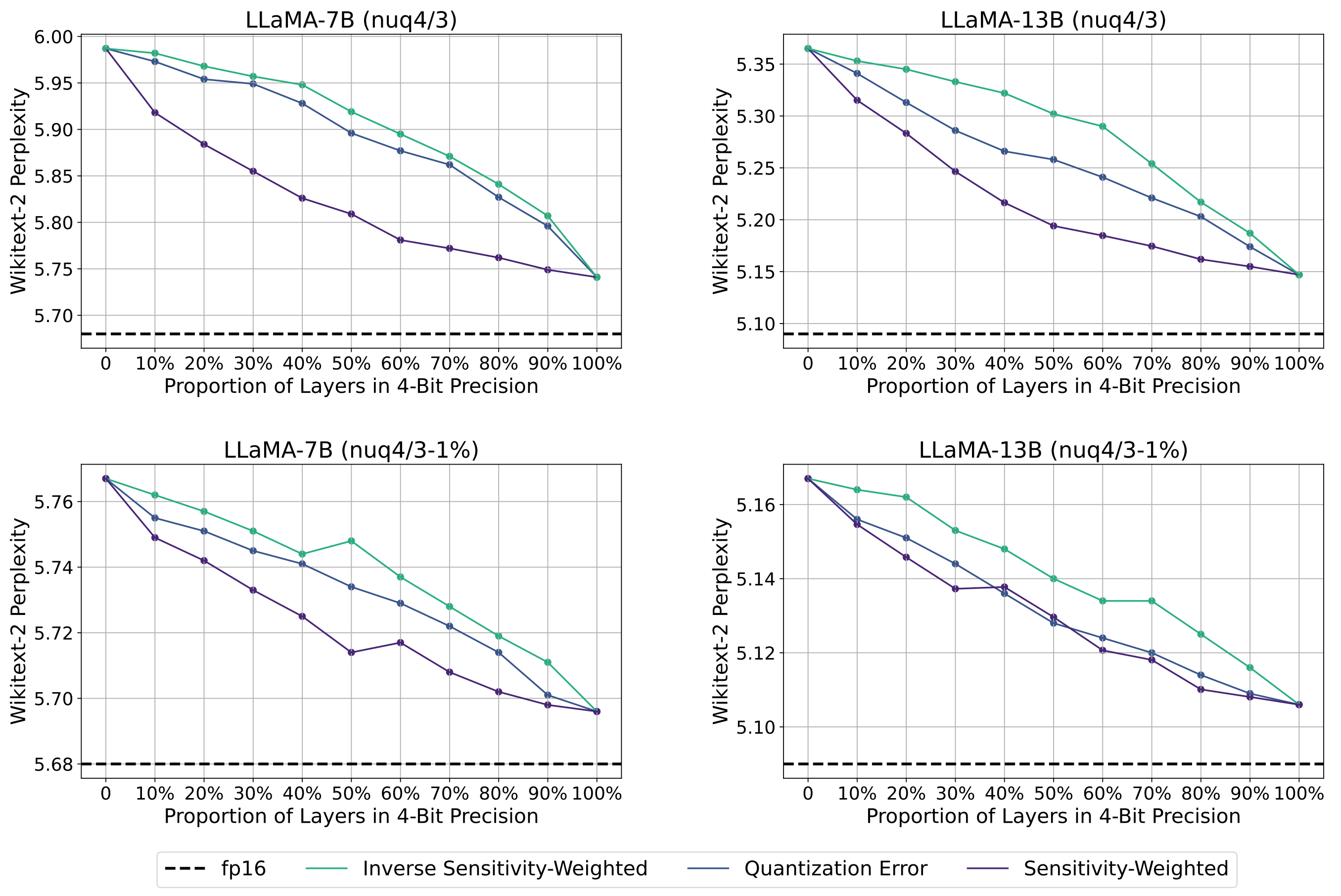


Figure 9: *Perplexity results for mixed-precision nuq4/3 quantization for the LLaMA-7B and LLaMA-13B models on the Wikitext-2 dataset, with model weights in fp16. Sensitivity-Weighted uses our metric for layer importance, Inverse Sensitivity-Weighted refers to selecting the lowest-importance layers using our metric, and Quantization Error uses the quantization error to sort layers without weighting using sensitivity information.*

speedups of up to 1.4× relative to the fp16 matrix-vector multiply kernels, demonstrating how our methodology facilitates efficient inference with a low-precision quantized KV cache.

Table 20: *Average latency (in microseconds) for the Key and Value dense nuq4 kernels as well as for the sparse kernels (with 1% outliers), benchmarked on an A6000 GPU for the LLaMA-7B model. Benchmarking results are reported for different sequence lengths (l). fp16 matrix-vector multiplication latencies are included for reference, and the Key multiplication time also includes the time to apply RoPE to the newly appended Key vector. We find that our dense-and-sparse approach (even with 1% outliers) provides latency benefits relative to the fp16 baseline, even when accounting for the time to compress activations online.*

Activation	**Operation**	***l*=2K**	***l*=4K**	***l*=16K**
Key	fp16 Matvec	66.4	116.1	402.3
Key (nuq4-1%)	Dense Packing	2.4	2.4	2.4
	Sparse Packing	4.5	4.5	4.5
	Dense Matvec	42.8	75.5	283.5
	Sparse Matvec	10.5	14.3	53.7
	Total Latency	60.2	96.7	344.1
Value	fp16 Matvec	56.0	104.2	389.3
Value (nuq4-1%)	Dense Packing	2.0	2.0	2.0
	Sparse Packing	4.2	4.2	4.2
	Dense Matvec	26.3	63.1	226.3
	Sparse Matvec	8.3	16.5	60.9
	Total Latency	40.8	85.8	293.4