SqueezeLLM: Dense-and-Sparse Quantization

Sehoon Kim [*1] **Coleman Hooper** [*1] **Amir Gholami** [*1 2] **Zhen Dong** [1] **Xiuyu Li** [1] **Sheng Shen** [1]
Michael W. Mahoney [1 2 3] **Kurt Keutzer** [1]

Abstract

Generative Large Language Models (LLMs) have demonstrated remarkable results for a wide range of tasks. However, deploying these models for inference has been a significant challenge due to their unprecedented resource requirements. This has forced existing deployment frameworks to use multi-GPU inference pipelines, which are often complex and costly, or to use smaller and less performant models. In this work, we demonstrate that the main bottleneck for generative inference with LLMs is memory bandwidth, rather than compute, specifically for single batch inference. While quantization has emerged as a promising solution by representing weights with reduced precision, previous efforts have often resulted in notable performance degradation. To address this, we introduce SqueezeLLM, a post-training quantization framework that not only enables lossless compression to ultra-low precisions of up to 3-bit, but also achieves higher quantization performance under the same memory constraint. Our framework incorporates two novel ideas: (i) *sensitivity-based non-uniform quantization*, which searches for the optimal bit precision assignment based on second-order information; and (ii) the *Dense-and-Sparse decomposition* that stores outliers and sensitive weight values in an efficient sparse format. When applied to the LLaMA models, our 3-bit quantization significantly reduces the perplexity gap from the FP16 baseline by up to 2.1× as compared to the state-of-the-art methods with the same memory requirement. Furthermore, when deployed on an A6000 GPU, our quantized models achieve up to 2.3× speedup compared to the baseline. Our code is available at https://github.com/SqueezeAILab/SqueezeLLM.

[*]Equal contribution [1]UC Berkeley [2]ICSI [3]LBNL. Correspondence to: Amir Gholami <amirgh@berkeley.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1. Introduction

Recent advances in Large Language Models (LLMs) trained on massive text corpora, with up to hundreds of billions of parameters, have showcased their remarkable problem-solving capabilities across various domains (Brown et al., 2020; Chowdhery et al., 2023; Du et al., 2022; Hoffmann et al., 2022; Raffel et al., 2020; Scao et al., 2022; Smith et al., 2022; Thoppilan et al., 2022; Touvron et al., 2023a; Zhang et al., 2022). However, deploying these models for inference has been a significant challenge due to their demanding resource requirements. For instance, the LLaMA-65B model requires at least 130GB of RAM to deploy in FP16, which exceeds current GPU capacity. Even storing such large-sized models has become costly and complex.

As will be discussed in Sec. 3, the main performance bottleneck in LLM inference for generative tasks is memory bandwidth rather than compute. This means that the speed at which we can load and store parameters becomes the primary latency bottleneck for memory-bound problems, rather than arithmetic computations. However, recent advancements in memory bandwidth technology have been significantly slow, compared to the improvements in computes, leading to the phenomenon known as the Memory Wall (Gholami et al., 2024; Patterson, 2004). Consequently, researchers have turned their attention to exploring algorithmic methods to overcome this challenge.

One promising approach is quantization, where model parameters are stored at lower precision, instead of the typical 16 or 32-bit precision used for training. For instance, it has been demonstrated that LLM models can be stored in 8-bit precision without performance degradation (Yao et al., 2022), where 8-bit quantization not only improves the storage requirements by half but also has the potential to improve inference latency and throughput. As a result, there has been significant research interest in quantizing models to even lower precisions. A pioneering approach is GPTQ (Frantar et al., 2022) which uses a training-free quantization technique that achieves near-lossless 4-bit quantization for LLM models with over tens of billions of parameters. However, achieving high quantization performance remains challenging, particularly with lower bit precision and for relatively smaller models (e.g., < 50B parameters).

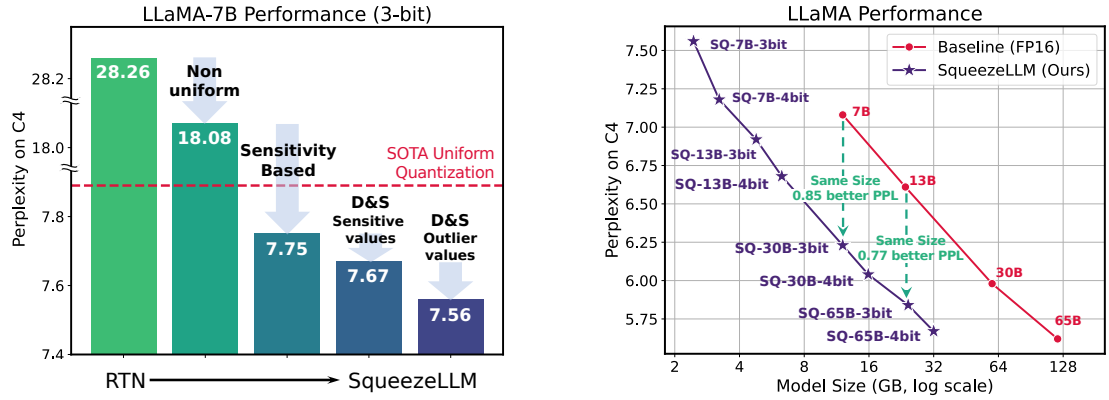

Figure 1. (Left) SqueezeLLM incorporates two key approaches: (i) sensitivity-based non-uniform quantization (Sec. 4.1), where quantization bins are allocated closer to sensitive values, and (ii) the Dense-and-Sparse decomposition (Sec. 4.2), which retains both sensitive values and outlier values as full-precision sparse format. When applied to LLaMA-7B with 3-bit quantization, our method outperforms the state-of-the-art methods (Frantar et al., 2022; Lin et al., 2023) by a large perplexity margin of over 0.3 on the C4 benchmark. (Right) By applying our methods to LLaMA models of varying sizes, we can achieve improved trade-offs between perplexity and model size.

Contributions. In this paper, we conduct an extensive study of low-bit precision quantization, and we identify limitations in existing approaches. Based on the insight that *the memory, rather than the compute, is the primary bottleneck* in LLM inference with generative tasks, we introduce SqueezeLLM, a post-training quantization framework with a novel *sensitivity-based non-uniform quantization* and *Dense-and-Sparse decomposition*. These techniques enable lossless compression even at precisions as low as 3 bits with reduced model sizes and faster inference without compromising model performance. Our detailed contributions include:

- **Sensitivity-based Non-Uniform Quantization:** We demonstrate that uniform quantization of prior works is sub-optimal for LLM inference for two reasons. First, the weight distributions in LLMs exhibit clear non-uniform patterns (Fig. 3). Second, the inference computation in prior works does not fully benefit from uniform quantization as the arithmetic is performed in FP16 precision, not in reduced precision. To address these, we propose a novel sensitivity-based non-uniform quantization method to achieve more optimal LLM quantization, which significantly improves the perplexity of 3-bit LLaMA-7B from 28.26 of uniform quantization to 7.75 on C4 (Sec. 4.1).

- **Dense-and-Sparse Quantization:** The weights in LLMs contain significant outliers, making low-bit quantization extremely challenging. To address this, we propose a simple solution that decomposes weights into dense and sparse components. The sparse part holds outlier values in full precision using efficient sparse storage methods, and the dense part can have a more compact range to aid quantization. By extracting only 0.45% of the weight values as the sparse component, we further improve the perplexity of LLaMA-7B from 7.75 to 7.58 on C4 (Sec. 4.2).

- **Evaluation:** We extensively test SqueezeLLM on various models on language modeling tasks using the C4 and WikiText2 datasets as well as on the MMLU (Hendrycks et al., 2021) and Vicuna benchmarks (Chiang et al., 2023) (Sec. 5.3). Furthermore, our deployed models on A6000 GPUs also exhibit significant latency gains of up to 2.4× compared to the FP16 baseline, showcasing the effectiveness of our method in terms of both quantization performance and inference efficiency (Sec. 5.4).

2. Related Work

LLM Quantization. In Appendix A, we offer an overview and related works of Transformer quantization, with an emphasis on Post-Training Quantization (PTQ), which is the primary focus of our work. With the increasing popularity of LLMs, *weight-only quantization* has surfaced as a promising approach to reduce memory consumption and enhance inference efficiency. GPTQ (Frantar et al., 2022) has been a pioneering work, and AWQ (Lin et al., 2023) and SpQR (Dettmers et al., 2023) have also suggested the weight-only quantization schemes concurrent to our work. Our work, however, is different in two key aspects. First, our work employs non-uniform quantization, as opposed to uniform quantization of the aforementioned works. In particular, our sensitivity-based non-uniform quantization not only better represents non-uniform distributions of weights, but it also strategically reduces the impact on more sensitive values, thereby enabling more aggressive quantization without performance degradation. Second, while previous works quantize weights in a way that layer-wise output activations remain unaffected, our approach targets preserving the model's final output. This strategy of minimizing the final loss, as shown in Appendix D.4, leads to better quantization performance since it is a direct measure of the end-to-end performance degradation after quantization.

Non-uniform Quantization. For low-bit LLM quantization, (Dettmers et al., 2024) has recently introduced the NF datatype, highlighting the importance of non-uniform quantization. However, our approach differs by offering a more dynamic non-uniform representation that accounts for both weight distributions and sensitivity of values, as opposed to the static, hard-coded NF datatype that assumes the normal distribution of the weights. While previous studies (Han et al., 2016; Xu et al., 2018) have used k-means clustering in quantization, our work pioneers its application in LLM quantization. Furthermore, we introduce the novel sensitivity-based weighted k-means clustering strategy, enabling lossless sub-4-bit quantization by significantly reducing performance degradation, in contrast to the sensitivity-agnostic counterpart (Fig. 1).

Outlier-Aware Quantization. Among the various challenges in low-bit Transformer quantization, one key issue is the presence of outliers (Kovaleva et al., 2021), which can unnecessarily increase the quantization range. To address this issue, outlier-aware quantization methods have been investigated (Bondarenko et al., 2021; Dettmers et al.; Wei et al., 2022; 2023; Xiao et al., 2023). Notably, (Dettmers et al.) keeps outlier activations in floating-point, while (Wei et al., 2022) transfers outlier factors to later layers without affecting functionality. These focus on activations, which is not a concern in our work where all activations are in floating-point. Our Dense-and-Sparse quantization instead tackles *weight* outliers for low-bit LLM quantization.

Concurrently to our work, SpQR (Dettmers et al., 2023) also explores outlier extraction in the context of weight quantization. While SpQR has shown a promising result on outlier extraction, SqueezeLLM, leveraging sensitivity-based non-uniform quantization, achieves precise quantization with significantly lower (e.g., 0.05%) or even zero sparsity levels. This is critical for both reducing model size and improving inference speed, as higher sparsity often degrades latency. Furthermore, SqueezeLLM uses outlier extraction as a direct solution to prevent outliers from negatively impacting quantization performance, bypassing the need for using the grouping strategy as an indirect solution. This contrasts with SpQR, which relies on fine-grained grouping that leads to increased model size and a more complex quantization process such as the bi-level quantization scheme.

Dense-and-Sparse Decomposition. Matrix decomposition into dense and sparse components has been explored in attention map decomposition (Chen et al., 2021; Dass et al., 2023), leveraging the fact that attention patterns often present low-rank characteristics with a few outliers. To the best of our knowledge, however, our research is the first to apply the dense-and-sparse decomposition strategy to weight matrices to improve quantization performance. Additionally, we uniquely incorporate both outlier and sensitive values within the sparse matrix, which yields considerable improvement in post-quantization performance.

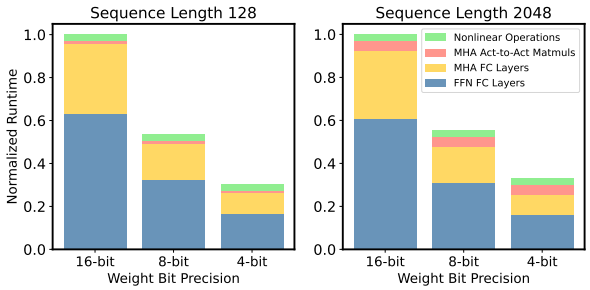

Figure 2. Normalized runtime for LLaMA-7B when reducing the bit precision for the weights with sequence lengths of 128 (left) and 2048 (right). Results were obtained using a roofline-based performance model for an A5000 GPU. Reducing only the precision of the weights (and not the activations) is sufficient to obtain significant latency reductions.

3. Memory Wall

Inference behavior broadly falls into two categories: *compute-bound* inference that is limited by computational throughput, and *memory-bound* inference that is bottlenecked by the rate at which data can be fed into the processing cores from memory. *Arithmetic intensity*, the ratio of compute to memory operations, is a typical metric used to assess this behavior. High and low arithmetic intensity indicates a compute-bound and memory-bound problem, respectively. For memory-bound problems, the speedup can be achieved by reducing the memory traffic rather than compute since the compute units in hardware are often underutilized waiting to receive data from memory.

Generative LLM inference exhibits extremely low arithmetic intensity compared to other workloads[1] (Kim et al., 2023). This is because it consists almost entirely of matrix-vector operations, which limits the data reuse as each weight load can only process a single vector for a single token, and cannot be amortized across the multiple vectors for different tokens. This low arithmetic intensity needs to be contrasted with the compute operations on a typical GPU which is orders of magnitude higher than the memory operations.[2] The disparity between compute and memory bandwidth, along with the growing memory requirements of deep learning, has been termed the *Memory Wall* problem (Gholami et al., 2024). To further illustrate this problem, we used a simple roofline-based performance modeling approach (Kim et al.,

[1]To be precise, we limit this discussion to single batch inference where the arithmetic involves matrix-vector operations. For large batch inference, compute can become important.

[2]For instance, A5000 GPU has peak computational throughput of 222 TeraFLOPs per second, which is 290× higher than the peak memory bandwidth of 768 GigaBytes per second.

2023) to study LLaMA-7B's runtime on an A5000 GPU with different bit precisions (Fig. 2). While we assume that all computations are kept at FP16, we see that the latency decreases linearly as we reduce the bit precision, indicating that the main bottleneck is memory, not compute.

In summary, in generative LLM inference, loading weights into memory is the primary bottleneck, while the cost of dequantization and FP16 computation is relatively small. Thus, by quantizing just the weights to lower precision, while leaving the activations in full precision, we can attain significant speedup as well as reduced model size. Given this insight, the appropriate strategy is to *minimize the memory size even if it may add overhead to arithmetic operations*.

4. Methodology

4.1. Sensitivity-Based Non-uniform Quantization

As in Fig. 3 (Top), weight distributions in LLMs demonstrate non-uniform patterns. The main task for quantization is to find an optimal way to allocate distinct quantized values (e.g., 8 for 3 bits) in a way that preserves model performance. A widely used approach in LLM quantization works is uniform quantization, where the weight range is evenly divided into bins. This has two main issues. First, uniformly distributing quantized values is sub-optimal as weight distributions are typically non-uniform. Second, while the main advantage of uniform quantization is efficient integer computation, this does not lead to end-to-end latency improvement in memory-bound LLM inference. Therefore, we have chosen non-uniform quantization, which allows for a more flexible allocation of the representative values.

Finding an optimal non-uniform quantization configuration translates into solving a k-means problem. Given a weight distribution, the goal is to determine k centroids that best represent the values (e.g., k=8 for 3-bit). This optimization problem for non-uniform quantization can be formulated as

$$Q(w)^* = \arg\min_Q \|W - W_Q\|_2^2, \tag{1}$$

where W denotes the weights and W_Q is the corresponding quantized weights (i.e., $[Q(w)$ for $w \in W]$), represented by k distinct values $\{q_1, \cdots, q_k\}$. Here, the optimal solution $Q(w)^*$ can be obtained by 1-dimensional k-means clustering, which clusters the parameters into k clusters and assign the centroid of each cluster as q_j's. While this already outperforms uniform quantization, we propose an improved *sensitivity-based* clustering algorithm.

Sensitivity-Based K-means Clustering. The quantization objective is to represent the model weights with low-bit precision with minimal perturbation in the model output (Dong et al., 2019). While quantization introduces perturbations in each layer, we need to minimize the overall perturbation

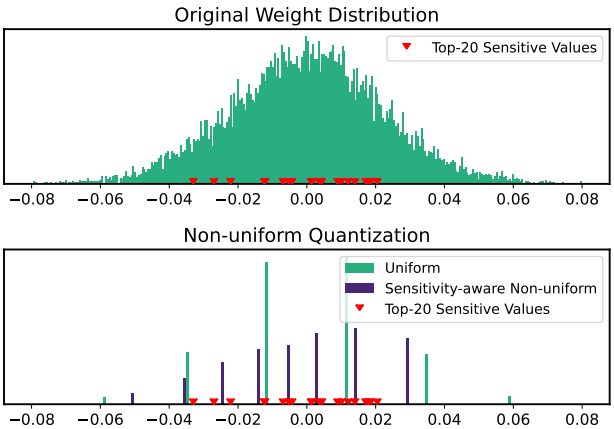

Figure 3. (Top) The weight distribution of one output channel in LLaMA-7B. The top-20 sensitive values are marked in red. (Bottom) Weight distributions after 3-bit quantization using uniform and sensitivity-based non-uniform quantization. In the latter case, the quantized values are clustered around the sensitive values.

with respect to the *final loss*, rather than focusing on individual layers, as it provides a more direct measure of the end-to-end performance degradation after quantization (LeCun et al., 1990). To achieve this, we need to place the k-means centroids near the values that are more sensitive with respect to the final loss, rather than treating all weight values equally, as in Eq. 1. To determine more sensitive values, we perform Taylor expansion to analyze how the loss changes in response to perturbations in the weights W:

$$\mathcal{L}(W_Q) \simeq \mathcal{L}(W) - g^\top (W - W_Q) \tag{2}$$

$$+ \frac{1}{2}(W - W_Q)^\top H (W - W_Q) \tag{3}$$

where g and $H = \mathbb{E}[\frac{\partial^2}{\partial W^2}\mathcal{L}(W)]$ are the gradient and Hessian of the loss at W. Assuming that the model has converged, the gradient g can be approximated as zero which gives us the following formula for computing how much the model gets perturbed after quantization:

$$Q(w)^* = \arg\min_Q (W - W_Q)^\top H (W - W_Q). \tag{4}$$

In the new optimization target, as compared to Eq. 1, the perturbation of each weight after quantization, i.e., $W - W_Q$, is weighted by the scaling factor introduced by the second-order derivative, H. This highlights the importance of minimizing perturbations for weights with large Hessian values, as they have a greater impact on the overall perturbation of the final output. In other words, the second-order derivative serves as a measure of importance for each weight value.

Due to the cost of computing the Hessian, we use an approximation to the Hessian based on the Fisher information matrix $\mathcal{F}$, which can be calculated over a sample dataset D

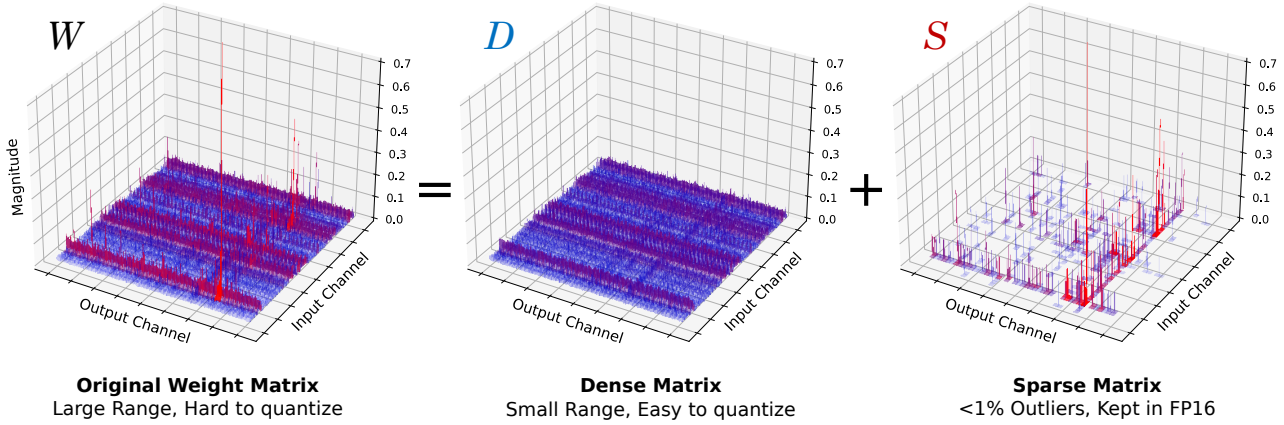

Figure 4. The illustration of the Dense-and-Sparse decomposition. The left figure plots the magnitude of a weight matrix (W) in the LLaMA 65B model, which contains a few outliers. These outliers contribute to the large range of values in the original weight matrix which significantly degrades the quantization performance. This matrix, however, can be decomposed into a sparse matrix S (Right) that contains the outliers and the remaining dense matrix D (Middle). The dense matrix D then exhibits a significantly smaller range, making accurate quantization much easier. The sparse matrix S can be kept in full precision with minimal memory and runtime overhead.

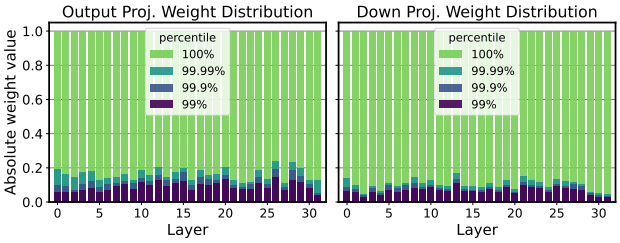

Figure 5. The distributions of the (normalized) absolute weight values, for the output layers in MHA and the down layers in FFN across different layers in LLaMA-7B. Note that the distributions exhibit outlier patterns across all layers, with 99% of the values clustered within ∼10% of the entire range.

as $H \simeq \mathcal{F} = \frac{1}{|D|}\sum_{d \in D} g_d g_d^\top$. This only requires computing gradient for a set of samples, which can be calculated efficiently with existing frameworks. To make the optimization objective in Eq. 4 more feasible, we further approximate the Fisher information matrix as a diagonal matrix by assuming that the cross-weight interactions are negligible. This simplifies our objective target as follows:

$$Q(w)^* \simeq \arg\min_Q (W - W_Q)^\top \mathrm{diag}(\mathcal{F})(W - W_Q) \quad (5)$$

$$= \arg\min_Q \sum_{i=1}^{N} \mathcal{F}_{ii}\big(w_i - Q(w_i)\big)^2. \quad (6)$$

An important consequence of Eq. 5 is the *weighted* k-means clustering setting, where the centroids will be pulled closer to these sensitive weight values. In Fig. 3, we illustrate the top-20 sensitive values based on the Fisher information of the exemplary weight distribution. At the bottom, the quantized values assigned by uniform quantization (green) are compared to those assigned by the sensitivity-based k-means approach (purple), which achieves a better trade-off by placing centroids near sensitive values, effectively minimizing quantization error. With 3-bit LLaMA-7B, sensitivity-based non-uniform quantization achieves much lower perplexity of 7.75 compared to the 28.26 perplexity of round-to-nearest uniform quantization on C4 (Fig. 1 and Sec. 5.2)

4.2. Dense-and-Sparse Quantization

Another challenge in low-bit LLM quantization is outlier values (Bondarenko et al., 2021; Dettmers et al.; Wei et al., 2022; 2023). In Fig. 5, we plot the normalized weight distributions of different layers in LLaMA-7B, which demonstrate that ∼99.9% of the weights are concentrated in a narrow range of ∼10% of the entire distribution. Naively quantizing the weights with a large range will significantly degrade performance, especially at low precisions. However, this also implies opportunity as the range of the weight values can be contracted by a factor of 10 simply by removing a small number of outlier values (e.g., 0.1%), yielding a significant improvement in quantization resolution. This will then help the sensitivity-based k-means centroids to focus more on the sensitive values rather than a few outliers.

Motivated by this, we introduce a method to filter out outliers from the weight matrix W by performing a simple yet effective decomposition into a sparse matrix (S) containing the outliers and the remaining dense matrix (D) that can be quantized much more effectively thanks to its significantly reduced range of values. That is, $W = D + S$ where $D = W[T_{\min} \leq w \leq T_{\max}]$ and $S = W[w <$

$T_{\min}$ or $w > T_{\max}$]. Here, $T_{\min/\max}$ are thresholds that define outliers based on the percentile of the distribution. This Dense-and-Sparse decomposition process is visually illustrated in Fig. 4.

Importantly, the overhead of this decomposition is minimal, since the number of outlier values is small (e.g., 0.5% of the entire values). Therefore, the sparse matrix can be stored efficiently using methods like the compressed sparse row (CSR) format. Inference is also straightforward with the decomposition as in $WX = DX + SX$, two kernels for dense and sparse multiplication can be overlapped, and the sparse part (SX) can benefit from sparse kernels (Sec. 4.3).

Sensitivity-Based Sparse Matrix. In addition to isolating outliers into a sparse matrix, we've also discovered the advantage of precisely representing a small number of highly sensitive weight matrix values. These values can be easily identified based on the Fisher information (Sec. 4.1). This not only maintains sensitive values with FP16 to avoid their impact on the model output, but also prevents the centroids of Eq. 5 from skewing towards the sensitive values. We have observed that extracting only 0.05% of these sensitive values across layers substantially enhances quantization performance (Appendix D). Altogether, with 3-bit LLaMA-7B, extracting 0.45% of outlier and sensitive values further reduces the perplexity from 7.67 to 7.56 (Fig. 1 and Sec. 5.2).

4.3. Dense-and-Sparse Kernel Implementation

To efficiently process non-uniformly quantized values, we implement 3/4-bit CUDA LUT-based kernels for matrix-vector multiplication between compressed weight matrices and uncompressed activation vectors. These kernels load the compressed weights and dequantize them piece-by-piece to minimize memory bandwidth utilization. The compressed matrices store 3/4-bit indices, which correspond to LUT entries containing FP16 values associated with the bins obtained from non-uniform quantization. After dequantization, all arithmetic is performed in FP16.

To optimize the handling of our Dense-and-Sparse representation, we develop kernels for sparse matrix-vector multiplication that load a matrix in CSR format and a dense activation vector, inspired by (Evtushenko, 2019). Since the non-zero entry distributions are highly skewed across rows (Appendix C), assigning a single thread per row can be inefficient due to uneven workload distribution among threads. Thus, we implement *balanced hybrid kernels* based on (Flegar & Quintana-Ortí, 2017) by assigning an equal number of nonzeros per thread; this leads to additional synchronization across threads due to rows being processed by multiple threads, but leads to a more balanced work assignment. We set the number of threads such that there were 10 nonzeros per thread. The dense non-uniform kernel and balanced sparse kernels are launched in one call to avoid overhead from summing the outputs from these separate operations.

5. Evaluations

5.1. Experiment Setup

Below is our experiment setup, with more details in Appendix B.

Models and Datasets. We have conducted comprehensive evaluations of SqueezeLLM using various models including LLaMA, LLaMA2 (Touvron et al., 2023a;b), OPT (Zhang et al., 2022) and Vicuna (Chiang et al., 2023) (v1.1 and v1.3). We conduct language modeling evaluation using the C4 (Raffel et al., 2020) and WikiText2 (Merity et al., 2016) datasets. We further evaluate the domain-specific knowledge and problem-solving ability using MMLU (Hendrycks et al., 2021), and the instruction-following ability using the methodology in (Chiang et al., 2023).

Baseline Methods. We compare SqueezeLLM against various PTQ methods for LLMs including RTN, GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023) and SpQR (Dettmers et al., 2023). To ensure a fair comparison, we use GPTQ *with* activation ordering throughout all experiments unless specified, which addresses the significant performance drop that would otherwise occur.

Quantization Details. For SqueezeLLM, we adopt channel-wise quantization where each output channel is assigned a separate lookup table. We use 2 different sparsity levels: 0% (dense-only) and 0.45% (0.05% sensitive values and 0.4% outlier values, as discussed in Sec. 4.2). For measuring sensitivity, we use 100 random samples from the Vicuna training set for Vicuna models and C4 training set for the others. While grouping can also be incorporated with our method, we found it sub-optimal as compared to extracting sensitive/outlier values with sparsity (Appendix D.3).

Latency Profiling. We measure the latency and peak memory usage for generating 128 and 1024 tokens on an A6000 machine using the Torch CUDA profiler. As an official implementation of GPTQ (in particular, the grouped version) is not available, we implement an optimized kernel for single-batch inference based on the most active open-source codebase (GPTQ-For-LLaMA).

5.2. Main Results

Tab. 1 shows quantization results for LLaMA along with the baseline methods. The models are grouped based on their size to better compare size-perplexity trade-offs. See Fig. 6 for a visual illustration. Below we use LLaMA-7B as the main example to discuss the impact of dense-only and Dense-and-Sparse quantization, and we then discuss how these trends extend to larger models. We provide the full evaluation result on all LLaMA models in Tab. H.13.

Table 1. Perplexity comparison of LLaMA models quantized into 3 and 4 bits using different methods including RTN, GPTQ, AWQ and SpQR on C4 and WikiText-2. We compare the performance of different methodologies by grouping them based on their model sizes. In the first group, we compare dense-only SqueezeLLM with non-grouped GPTQ. In the second group, we compare SqueezeLLM with a sparsity level of 0.45% to GPTQ and AWQ with a group size of 128. For comparison, we add speedup and peak memory usage numbers, which we provide more details in Tab. H.13. Further results for **LLaMA-30/65B** can be found in Tab. H.13, and results on other models including **LLaMA-2 7/13/70B** are provided in Appendix H.1.

LLaMA-7B	**3-bit**					**4-bit**				
Method	**Avg. Bits** (comp. rate)	**PPL (↓)** C4	Wiki	**Speedup** (↑)	**Mem.** (GB, ↓)	**Avg. Bits** (comp. rate)	**PPL (↓)** C4	Wiki	**Speedup** (↑)	**Mem.** (GB, ↓)
Baseline	16	7.08	5.68	1×	12.7	16	7.08	5.68	1×	12.7
RTN	3 (5.33)	28.26	25.61	2.3×	2.9	4 (4.00)	7.73	6.29	2.0×	3.7
GPTQ	3 (5.33)	9.55	7.55	2.3×	2.9	4 (4.00)	7.43	5.94	2.0×	3.7
SpQR	-	-	-	-	-	3.94 (4.06)	7.28	5.87	1.2×[†]	N/A
SqueezeLLM	3.02 (5.29)	**7.75**	**6.32**	2.1×	2.9	4.05 (3.95)	**7.21**	**5.79**	1.8×	3.8
GPTQ (g128, no reorder)[‡]	3.24 (4.93)	10.09	8.85	2.0×	3.0	4.24 (3.77)	7.80	6.07	1.6×	3.8
GPTQ (g128)[‡]	3.24 (4.93)	7.89	6.27	0.2×	3.0	4.24 (3.77)	7.21	5.78	0.4×	3.8
AWQ (g128)	3.24 (4.93)	7.90	6.44	2.0×	3.0	4.24 (3.77)	7.22	5.82	1.6×	3.8
SqueezeLLM (0.45%)	3.24 (4.93)	**7.56**	**6.13**	1.9×	3.1	4.27 (3.75)	**7.18**	**5.77**	1.7×	4.0

LLaMA-13B	**3-bit**					**4-bit**				
Method	**Avg. Bits** (comp. rate)	**PPL (↓)** C4	Wiki	**Speedup** (↑)	**Mem.** (GB, ↓)	**Avg. Bits** (comp. rate)	**PPL (↓)** C4	Wiki	**Speedup** (↑)	**Mem.** (GB, ↓)
Baseline	16	6.61	5.09	1×	24.6	16	6.61	5.09	1×	24.6
RTN	3 (5.33)	13.24	11.78	2.7×	5.3	4 (4.00)	6.99	5.53	2.3×	6.8
GPTQ	3 (5.33)	8.22	6.22	2.7×	5.3	4 (4.00)	6.84	5.29	2.3×	6.8
SpQR	-	-	-	-	-	3.96 (4.04)	6.72	5.22	1.2×[†]	N/A
SqueezeLLM	3.02 (5.30)	**7.08**	**5.60**	2.4×	5.4	4.04 (3.96)	**6.71**	**5.18**	2.0×	6.9
GPTQ (g128, no reorder)[‡]	3.25 (4.92)	7.16	5.53	2.2×	5.7	4.25 (3.77)	6.71	5.18	1.9×	7.2
GPTQ (g128)[‡]	3.25 (4.92)	7.12	5.47	0.2×	5.6	4.25 (3.77)	6.70	5.17	0.4×	7.0
AWQ (g128)	3.25 (4.92)	7.08	5.52	2.2×	5.7	4.25 (3.77)	6.70	5.21	1.9×	7.2
SqueezeLLM (0.45%)	3.24 (4.94)	**6.92**	**5.45**	2.2×	5.8	4.26 (3.76)	**6.68**	**5.17**	1.9×	7.3

Dense-only Quantization. In Tab. 1 (Top), we compare dense-only SqueezeLLM with 0% sparsity level and GPTQ without grouping. With 4-bit quantization, our method exhibits minimal degradation compared to the FP16 baseline, with only ∼0.1 perplexity degradation on C4 and WikiText2, while reducing the model size by 3.95×. Moreover, when compared to non-grouped GPTQ our method shows significant perplexity improvement of up to 0.22.

The performance gap between the two methods becomes more pronounced with 3-bit quantization. SqueezeLLM outperforms GPTQ by a substantial margin of 1.80/1.22 points on C4/WikiText2 with a 5.29× compression rate. This is only 0.67/0.55 points off from the FP16 baseline. This demonstrates the effectiveness of the sensitivity-based non-uniform method for ultra-low-bit quantization.

[†]Since SpQR does not release their kernel implementation, we conduct our best-effort comparison using their reported speedup numbers. See Appendix B for details.

[‡]GPTQ with activation ordering incurs a significant latency penalty as elements in the same channel are associated with different scaling factors, resulting in distributed memory accesses (Sec. 5.4). GPTQ *without* activation ordering alleviates the latency issue at the cost of a substantial perplexity degradation.

Dense-and-Sparse Quantization. By leveraging the Dense-and-Sparse quantization, we achieve a further reduction in the perplexity gap from the FP16 baseline, as shown in Tab. 1. This improvement is particularly significant with 3-bit quantization, where extracting just 0.45% of the values yields around 0.2 perplexity improvement. This enables nearly lossless compression with less than 0.1/0.5 perplexity deviation from the FP16 baseline for 4/3-bit, respectively.

Both GPTQ and AWQ use a grouping strategy to enhance performance with a slight overhead in model size. However, we demonstrate that SqueezeLLM with a sparsity level of 0.45% consistently outperforms both GPTQ/AWQ with a group size of 128 in all scenarios with comparable model sizes. This is more pronounced for 3-bit quantization, where SqueezeLLM with a 0.45% sparsity level outperforms both GPTQ/AWQ with a group size of 128 by up ∼0.3 perplexity.

Results on Larger Models. In Tab. 1 (13B) and Tab. H.13 (30/65B), we observe that the trend in 7B extends to larger models, where SqueezeLLM consistently outperforms other PTQ methods across all models and bit widths. Such a trend is also illustrated in Fig. 6 for 3-bit quantization where even *dense-only* SqueezeLLM achieves comparable perplexity to *grouped* GPTQ/AWQ. With sparsity, we can further improve perplexity, reducing the gap from the FP16 baseline

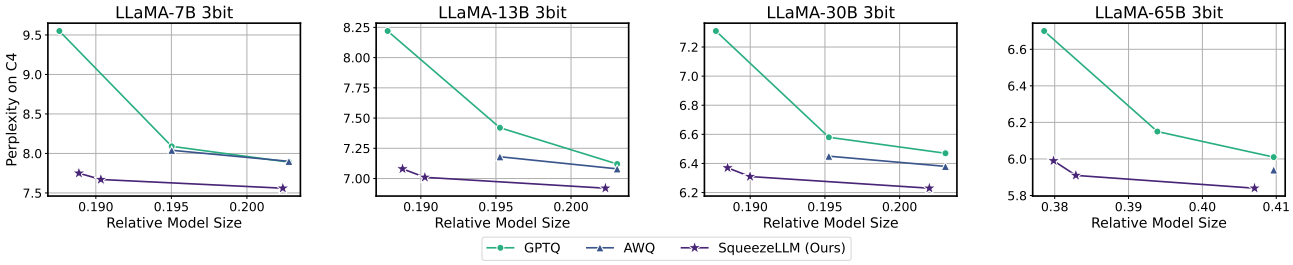

Figure 6. Perplexity comparison PTQ methods for 3-bit LLaMA quantization, evaluated on C4. The x-axes are the relative model sizes with respect to the model size in FP16. Different size-perplexity trade-offs are achieved by adjusting the group size for GPTQ and AWQ and the sparsity level for ours. Our quantization method consistently and significantly outperforms GPTQ and AWQ across all model size regimes, with a more pronounced gap in lower-bit and smaller model sizes.

Table 2. Comparison of PTQ methods on zero-shot MMLU accuracy applied to Vicuna v1.1 and v1.3. We add peak memory usage in GB for comparison. Additional results on 5-shot MMLU evaluation can be found in Appendix H.2.

Method	Avg. bit	Vicuna-7B (v1.1)		Vicuna-13B (v1.1)		Vicuna-7B (v1.3)		Vicuna-13B (v1.3)		Vicuna-33B (v1.3)	
		Acc (↑)	Mem (GB, ↓)	Acc (↑)	Mem (GB, ↓)	Acc (↑)	Mem (GB, ↓)	Acc (↑)	Mem (GB, ↓)	Acc (↑)	Mem (GB, ↓)
Baseline	16	39.1%	12.7	41.2%	24.6	40.2%	12.7	43.3%	24.6	49.5%	OOM
AWQ (g128)	4.25	38.0%	3.8	40.4%	7.2	**39.6%**	3.8	42.2%	7.2	49.5%	17.2
SqueezeLLM	4.05	38.8%	3.8	39.2%	6.9	39.3%	3.8	**44.1%**	6.9	48.0%	17.5
SqueezeLLM (0.45%)	4.26	**39.4%**	4.0	**41.0%**	7.3	39.5%	4.0	43.8%	7.3	**49.9%**	18.7
AWQ (g128)	3.25	36.5%	3.0	37.6%	5.7	37.4%	3.0	40.7%	5.7	46.4%	13.2
SqueezeLLM	3.02	36.0%	2.9	37.2%	5.4	35.1%	2.9	40.5%	5.4	46.2%	12.5
SqueezeLLM (0.45%)	3.24	**37.7%**	3.1	**39.4%**	5.8	**37.6%**	3.1	**40.8%**	5.8	**47.7%**	14.7

to less than 0.1/0.4 perplexity points for 4/3-bit quantization. Notably, with 3-bit quantization, our approach achieves up to a 2.1× reduction in perplexity gap from the FP16 baseline compared to existing methods. Further ablation studies on our design choices are provided in Appendix D, and additional results on the LLaMA2 and OPT models can be found in Appendix H.1.

5.3. Quantization of Instruction Following Models

Instruction tuning has emerged as a method for improving the model's ability to respond to user commands. We explore the quantization of instruction-following models to demonstrate the benefits of SqueezeLLM in terms of accuracy preservation by applying it to the Vicuna models, and evaluating the performance with the following approaches.

MMLU Evaluation. We first evaluate the baseline and quantized models on the MMLU benchmark where the weighted accuracy in the zero-shot setting is provided in Tab. 2 for Vicuna models. As we can see, 3-bit SqueezeLLM achieves higher accuracy for all models compared to AWQ and also preserves the FP16 baseline accuracy with 4-bit quantization. 5-shot results are provided in Appendix H.2.

Instruction-Following Ability. Another approach for evaluating instruction-following ability is to ask GPT-4 to rank the generated responses as presented in (Chiang et al., 2023). As shown in Fig. 7, SqueezeLLM without sparsity achieves near-perfect performance (i.e., 50/50 split) with 4-bit quantization for both Vicuna-7B and 13B, outperforming GPTQ with the same model size. In the case of 3-bit quantization, SqueezeLLM outperforms both GPTQ and AWQ with comparable model sizes. In the case of the Vicuna-13B model, achieving a near-perfect 50/50 split for 3-bit quantization.

5.4. Hardware Deployment and Profiling

We show the latency and peak GPU memory usage of SqueezeLLM in Tab. 3 on an A6000 GPU for different configurations when generating 128 tokens. We observe that the LUT-based non-uniform approach in SqueezeLLM (3rd row) shows up to 2.4× speedup compared to the FP16 baseline, and exhibits comparable latency and peak memory usage to the uniform quantization of non-grouped GPTQ (2nd row). This indicates that the overhead associated with LUT-based dequantization is small, especially considering the significant perplexity gains it enables.

Additionally, when incorporating sparsity, we still observed latency gains relative to the FP16 baseline. As shown in Tab. 3, keeping 0.45% of parameters in FP16 (4th row) only adds around 10% latency overhead relative to the dense-only implementation, while still resulting in up to 2.2× speed up compared to the FP16 baseline. In contrast, when accounting for permutation, the GPTQ runtime is degraded heavily (5th row). This latency penalty is due to permutation, which means that elements in the same channel need to be scaled using different scaling factors (which are accessed using group indices); it is challenging for these distributed memory accesses to be performed efficiently, as GPUs rely

Table 3. Latency (s) and peak memory usage (GB) of 3-bit LLaMA when generating 128 tokens on an A6000 GPU. The table compares the FP16 baseline, non-grouped and grouped GPTQ with activation ordering, and SqueezeLLM with different sparsity levels. For comparison, we include bitwidth and perplexity on the C4 benchmark. See Tab. G.11 for additional results on generating 1024 tokens, and see Tab. G.12 for additional benchmarking results on an A100 GPU.

Method	Bit width	7B			13B			30B			65B		
		PPL (C4)	Lat (s)	Mem (G)	PPL (C4)	Lat (s)	Mem (G)	PPL (C4)	Lat (s)	Mem (G)	PPL (C4)	Lat (s)	Mem (G)
Baseline	16	7.08	3.2	12.7	6.61	5.6	24.6	5.98	OOM	OOM	5.62	OOM	OOM
GPTQ	3	9.55	1.4	2.9	8.22	2.1	5.3	7.31	4.0	12.3	6.70	6.7	24.0
SqueezeLLM	3.02	7.75	1.5	2.9	7.08	2.4	5.4	6.37	4.0	12.5	5.99	7.6	24.5
GPTQ (g128)	3.25	7.89	13.7	3.0	7.12	24.2	5.6	6.47	61.9	12.9	6.01	117.8	25.1
SqueezeLLM (0.45%)	3.24	7.56	1.7	3.1	6.92	2.5	5.8	6.23	4.4	14.7	5.84	8.8	28.0

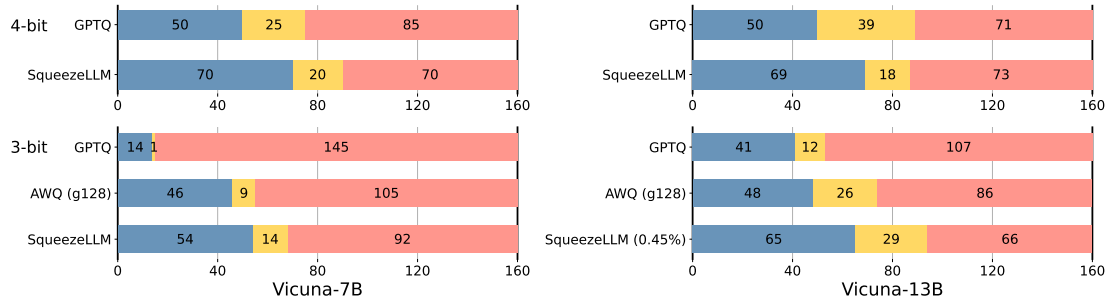

Figure 7. Comparison of PTQ methods applied to Vicuna v1.1. Blue / yellow / red represent the number of times that the quantized model won / tied / lost against the baseline FP16 model. This evaluation was performed using the methodology from Vicuna.

heavily on coalesced memory accesses in order to optimally use memory bandwidth. This shows how our Dense-and-Sparse quantization methodology allows for both higher accuracy as well as better performance relative to GPTQ. Additional evaluation results on generating 1024 tokens are provided in Tab. G.11, where we observe a similar trend.

6. Conclusion

We have presented SqueezeLLM which attempts to address the Memory Wall problem associated with generative LLM inference that is memory-bound. SqueezeLLM incorporates two novel ideas that allow ultra-low precision quantization of LLMs with negligible degradation in generation performance: the sensitivity-based non-uniform quantization method; and the Dense-and-Sparse decomposition that resolves the outlier issue. We have evaluated SqueezeLLM on a wide range of models and datasets that assess language modeling, problem-solving, and instruction-following capabilities of quantized models, where we have demonstrated that our quantization method can consistently outperform the previous state-of-the-art methodologies.

Impact Statement

This paper introduces advancements in machine learning through a method that results in more lightweight models by improving computational efficiency. While this technique will enable broader access to the applications of machine learning technologies across diverse sectors, we do not foresee direct negative social consequences that require specific highlights. Our work aims at fostering innovation and inclusivity in the field, making advanced technologies more available to a wider range of users and developers.

Acknowledgements

The authors would like to acknowledge Karttikeya Mangalam, Nicholas Lee, and Thanakul Wattanawong for helpful discussions and brainstorming. We acknowledge gracious support from Google Cloud, Google TRC team, and specifically Jonathan Caton, Jing Li, Jiayu Ye, and Prof. David Patterson. Prof. Keutzer's lab is sponsored by Intel corporation, Intel VLAB team, Intel One-API center of excellence, as well as gracious funding from Furiosa, Berkeley Deep Drive, and BAIR. Michael W. Mahoney would also like to acknowledge a J. P. Morgan Chase Faculty Research Award as well as the DOE, NSF, and IARPA. Sehoon Kim would like to acknowledge the support from the Korea Foundation for Advanced Studies (KFAS). Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

References

Bai, H., Zhang, W., Hou, L., Shang, L., Jin, J., Jiang, X., Liu, Q., Lyu, M., and King, I. BinaryBERT: Pushing the limit of BERT quantization. *arXiv preprint arXiv:2012.15701*, 2020.

Bondarenko, Y., Nagel, M., and Blankevoort, T. Understanding and overcoming the challenges of efficient Transformer quantization. *arXiv preprint arXiv:2109.12948*, 2021.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. ZeroQ: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13169–13178, 2020.

Chee, J., Cai, Y., Kuleshov, V., and De Sa, C. M. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36, 2024.

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34: 17413–17426, 2021.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Chung, I., Kim, B., Choi, Y., Kwon, S. J., Jeon, Y., Park, B., Kim, S., and Lee, D. Extremely low bit transformer quantization for on-device neural machine translation. *arXiv preprint arXiv:2009.07453*, 2020.

Dass, J., Wu, S., Shi, H., Li, C., Ye, Z., Wang, Z., and Lin, Y. Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 415–428. IEEE, 2023.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Dong, Z., Yao, Z., Arfeen, D., Gholami, A., Mahoney, M. W., and Keutzer, K. HAWQ-V2: Hessian Aware trace-Weighted Quantization of neural networks. *NeurIPS'19 workshop on Beyond First-Order Optimization Methods in Machine Learning.*, 2019.

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

Egiazarian, V., Panferov, A., Kuznedelev, D., Frantar, E., Babenko, A., and Alistarh, D. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Evtushenko, G. Sparse Matrix-Vector Multiplication with CUDA. *https://medium.com/analytics-vidhya/sparse-matrix-vector-multiplication-with-cuda-42d191878e8f*, 2019.

Flegar, G. and Quintana-Ortí, E. S. 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*, pp. 697–709. Springer, 2017.

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

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., et al. A framework for few-shot language model evaluation, 2021.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

Gholami, A., Yao, Z., Kim, S., Hooper, C., Mahoney, M. W., and Keutzer, K. Ai and memory wall. *IEEE Micro*, pp. 1–5, 2024.

GPTQ-For-LLaMA. https://github.com/qwopqwop200/gptq-for-llama.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Huang, Y., Yang, H., Dong, Z., Gudovskiy, D., Okuno, T., Nakata, Y., Du, Y., Zhang, S., and Keutzer, K. Output sensitivity-aware detr quantization. 2023.

Jeon, Y., Lee, C., Cho, E., and Ro, Y. Mr. BiQ: Post-training non-uniform quantization based on minimizing the reconstruction error. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12329–12338, 2022.

Kim, S., Gholami, A., Yao, Z., Mahoney, M. W., and Keutzer, K. I-BERT: Integer-only bert quantization. *arXiv preprint arXiv:2101.01321*, 2021.

Kim, S., Hooper, C., Wattanawong, T., Kang, M., Yan, R., Genc, H., Dinh, G., Huang, Q., Keutzer, K., Mahoney, M. W., Shao, S., and Gholami, A. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*, 2023.

Kovaleva, O., Kulshreshtha, S., Rogers, A., and Rumshisky, A. Bert busters: Outlier dimensions that disrupt transformers. *arXiv preprint arXiv:2105.06990*, 2021.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Li, X., Liu, Y., Lian, L., Yang, H., Dong, Z., Kang, D., Zhang, S., and Keutzer, K. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17535–17545, October 2023.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. 2023.

Liu, Y., Yang, H., Dong, Z., Keutzer, K., Du, L., and Zhang, S. NoisyQuant: Noisy bias-enhanced post-training activation quantization for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20321–20330, 2023.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.

Oh, S., Sim, H., Kim, J., and Lee, J. Non-uniform step size quantization for accurate post-training quantization. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pp. 658–673. Springer, 2022.

Patterson, D. A. Latency lags bandwith. *Communications of the ACM*, 47(10):71–75, 2004.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Shao, W., Chen, M., Zhang, Z., Xu, P., Zhao, L., Li, Z., Zhang, K., Gao, P., Qiao, Y., and Luo, P. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137*, 2023.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Q-BERT: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8815–8821, 2020.

Shomron, G., Gabbay, F., Kurzum, S., and Weiser, U. Post-training sparsity-aware quantization. *Advances in Neural Information Processing Systems*, 34:17737–17748, 2021.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E.,

Azhar, F., et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wei, X., Zhang, Y., Zhang, X., Gong, R., Zhang, S., Zhang, Q., Yu, F., and Liu, X. Outlier suppression: Pushing the limit of low-bit transformer language models. *arXiv preprint arXiv:2209.13325*, 2022.

Wei, X., Zhang, Y., Li, Y., Zhang, X., Gong, R., Guo, J., and Liu, X. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *arXiv preprint arXiv:2304.09145*, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099. PMLR, 23–29 Jul 2023.

Xu, Y., Wang, Y., Zhou, A., Lin, W., and Xiong, H. Deep neural network compression with single and multiple level quantization, 2018.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., and He, Y. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *arXiv preprint arXiv:2206.01861*, 2022.

Yuan, Z., Niu, L., Liu, J., Liu, W., Wang, X., Shang, Y., Sun, G., Wu, Q., Wu, J., and Wu, B. RPTQ: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023.

Zadeh, A. H., Edo, I., Awad, O. M., and Moshovos, A. GOBO: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 811–824. IEEE, 2020.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. Q8BERT: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, W., Hou, L., Yin, Y., Shang, L., Chen, X., Jiang, X., and Liu, Q. TernaryBERT: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

Zhang, Y., Dong, Z., Yang, H., Lu, M., Tseng, C.-C., Guo, Y., Keutzer, K., Du, L., and Zhang, S. Qd-bev: Quantization-aware view-guided distillation for multi-view 3d object detection. 2023.

Zhao, R., Hu, Y., Dotzel, J., De Sa, C., and Zhang, Z. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pp. 7543–7552. PMLR, 2019.

A. Related Works on Transformer Quantization

Quantization methods can be broadly categorized based whether retraining is required or not (Gholami et al., 2021). Quantization-Aware Training (QAT) requires retraining the model to adapt its weights to help recover accuracy after quantization (Bai et al., 2020; Kim et al., 2021; Shen et al., 2020; Zafrir et al., 2019; Zhang et al., 2020; 2023), whereas Post-Training Quantization (PTQ) does not involve retraining (Cai et al., 2020; Li et al., 2023; Oh et al., 2022; Shomron et al., 2021; Zhao et al., 2019). While QAT often results in better accuracy, it is often infeasible for LLMs due to the expensive retraining cost and/or lack of access to the training data and infrastructure. As such, most works on LLM quantization have focused on PTQ (Dettmers et al.; Frantar et al., 2022; Lin et al., 2023; Xiao et al., 2023; Yao et al., 2022; Yuan et al., 2023). Our work also focuses on the PTQ approach.

Quantization methods can be also classified as uniform or non-uniform (Gholami et al., 2021). Uniform quantization (Dettmers et al., 2023; Frantar et al., 2022; Huang et al., 2023; Kim et al., 2021; Lin et al., 2023; Liu et al., 2023; Shen et al., 2020; Zafrir et al., 2019), which uniformly divides weight ranges into bins, has gained popularity since it allows faster computation by using quantized precision arithmetic. However, recent hardware trends indicate that faster computation does not necessarily translate to improved end-to-end latency or throughput (Gholami et al., 2024), particularly in memory-bound tasks like generative LLM inference (Sec. 3). Furthermore, uniform quantization can be sub-optimal when the weight distribution is non-uniform, as in LLMs (Fig. 3).

Hence, we focus on non-uniform quantization, which non-uniformly allocates quantization bins without constraints for a more accurate representation of weights and smaller quantization errors. While it does not support integer arithmetic for computational acceleration, this drawback is not significant for memory-bound problems, as in our focus, where the primary bottleneck lies in memory bandwidth rather than computation. Among non-uniform quantization methods (Chung et al., 2020; Jeon et al., 2022), the most similar work to ours is GOBO (Zadeh et al., 2020), which introduces a k-means clustering-based look-up table approach. Our work introduces two novel methods as compared to GOBO: (i) sensitivity-based methods; and (ii) Dense-and-Sparse quantization methodologies, which yield substantial improvements within the k-means-based non-uniform quantization framework.

B. Experiment Setup (Details)

Models and Datasets. We have conducted comprehensive evaluations of SqueezeLLM using various models on different tasks. First, in the language modeling evaluation, we apply SqueezeLLM to the LLaMA (Touvron et al., 2023a), LLaMA2 (Touvron et al., 2023b) and OPT (Zhang et al., 2022) models and measure the perplexity of the quantized models on the C4 (Raffel et al., 2020) and WikiText2 (Merity et al., 2016) datasets with a chunk size of 2048. We also evaluate the domain-specific knowledge and problem-solving ability through the MMLU benchmark (Hendrycks et al., 2021) using the instruction-tuned Vicuna (v1.1 and v1.3) models. We used the Language Model Evaluation Harness to run zero-shot evaluation across all tasks (Gao et al., 2021). Finally, we evaluate the instruction following ability following the methodology presented in (Chiang et al., 2023). To do so, we generate answers for 80 sample questions and compared them to the answers generated by the FP16 counterpart using the GPT-4 score. To minimize the ordering effect, we provide the answers to GPT-4 in both orders, resulting in a total of 160 queries.

Baseline Methods. We compare SqueezeLLM against PTQ methods for LLMs including RTN as well as state-of-the-art methods including GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023) and SpQR (Dettmers et al., 2023). To ensure a fair comparison, we use GPTQ *with* activation ordering throughout all experiments unless specified, which addresses the significant performance drop that would otherwise occur. For AWQ, we use official quantized models or reproduce using their official code if they are not available except for LLaMA 65B with group size 256, which ran into OOM even on A100-80G. Evaluations are then conducted based on our perplexity method. For SpQR, we rely on the paper's reported numbers since their perplexity evaluation methodology is identical to ours. SpQR aims to enhance 3-bit and 4-bit models by introducing grouping, bi-level quantization, and sparsity, making them approximately 4 and 4.6 bits on average for LLaMA. In contrast, SqueezeLLM aims to preserve 3 and 4-bit as closely as possible, minimizing any extra model size overhead. Therefore, we present our best-effort comparison of SpQR and SqueezeLLM by comparing 3-bit SpQR models, which average around 4 bits, and our 4-bit models, both of which possess similar model sizes.

Latency Profiling. We measure the latency and peak memory usage for generating 128 and 1024 tokens on an A6000 machine using the Torch CUDA profiler. As an official implementation of GPTQ (in particular, the grouped version)

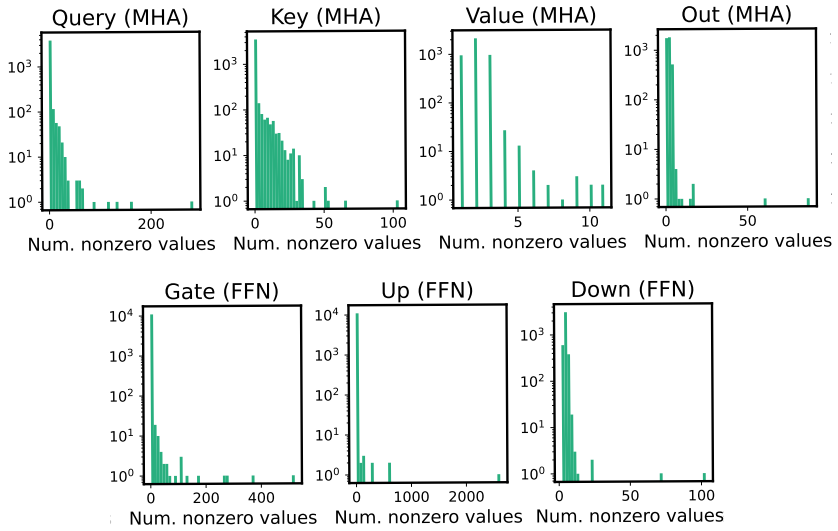

Figure C.1. Histograms of the number of non-zero entries per output channel in 7 different linear layers in the first LLaMA-7B block. The histograms reveal the presence of a few channels that contain significantly more non-zero entries than others, highlighting the skew in the sparsity patterns across different channels within the linear layers.

Table C.1. Hardware profiling of latency and memory usage using different kernel implementations for LLaMA 7B, 13B, 30B, and 65B quantized into 3-bit when generating 128 tokens on an A6000 GPU. The first row shows the performance of SqueezeLLM without sparsity as a reference. The second row shows the performance of SqueezeLLM with a sparsity level of 0.45% using a standard kernel for processing a CSR matrix. The third row shows the performance of SqueezeLLM with a sparsity level of 0.45% using a balanced sparse kernel that allocates 10 nonzeros per thread, thereby more efficiently handling skewed sparse matrices.

Sparse Kernel	**Sparsity Level**	**Latency (Seconds)** 7B	13B	30B	65B	**Peak Memory (GB)** 7B	13B	30B	65B
-	0%	1.5	2.4	4.0	7.6	2.9	5.4	12.5	24.5
Standard	0.45%	3.9	6.2	12.5	14.4	3.2	5.8	13.7	28.0
Balanced	0.45%	1.7	2.6	4.4	8.8	3.1	5.8	14.7	28.0

is not available, we implement an optimized kernel for single-batch inference based on the most active open-source codebase (GPTQ-For-LLaMA).

To compare latency with SpQR, we rely on their reported speedup numbers to make our best-effort comparison, as their kernel implementation is not publicly available. Regarding AWQ, we use the GPTQ kernel without activation ordering since they exhibit identical behavior during inference. Although AWQ has released their own kernel implementation, their 3-bit kernels are not publicly available. Furthermore, they have incorporated optimizations that are unrelated to quantization, such as LayerNorm and positional embedding, which are universally applicable to other quantization methods. To ensure a fair comparison with other methods, we refrained from using their released kernels.

C. Data Skew in Per-channel Sparsity Pattern

Fig. C.1 provides the distribution of nonzero entries per output channel across different linear layers in the first LLaMA-7B block. This plot shows that the nonzero distribution is heavily skewed, with a few channels containing a much larger proportion of nonzero values. This skewed distribution makes it challenging to efficiently perform computations using the sparse matrix, as it is difficult to distribute the nonzero elements evenly across parallel processing units. This motivates our modified kernel for handling channels with a large number of outliers in order to reduce the runtime overhead of the sparse matrices. As outlined in Tab. C.1, we observed over 100% added runtime overhead when employing a standard CSR-based

Table D.2. Ablation study comparing sensitivity-agnostic and sensitivity-based non-uniform quantization on the LLaMA-7B model with 3-bit quantization, measured by perplexity on the C4 benchmark. The baseline model in FP16 achieves a perplexity of 7.08.

Method	Sensitivity-Agnostic (↓)	Sensitivity-Based (↓)
SqueezeLLM	18.08	**7.75**
SqueezeLLM (0.05%)	8.10	**7.67**
SqueezeLLM (0.45%)	7.61	**7.56**

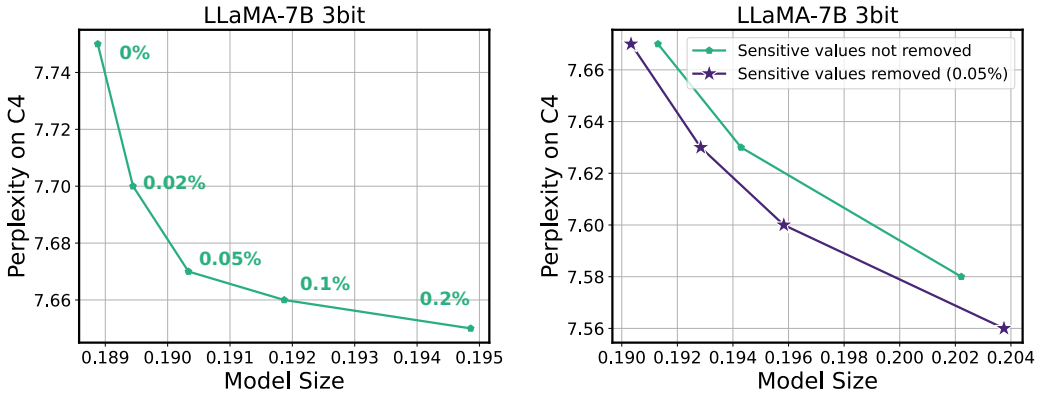

Figure D.2. (Left) Model size (normalized by the size of the FP16 model) and perplexity trade-off with different percentages of sensitive values included in the sparse matrix. Here, no outlier values are included in the sparse matrix. (Right) Comparison of the performance when the sensitive values are not removed as the sparse matrix (only outlier values are removed) to the case where 0.05% of the sensitive values are removed. In both cases, the trade-offs are obtained by controlling the percentage of outlier values included in the sparse matrix.

kernel. However, if we allocate each thread to process a fixed number of nonzeros (rather than having each thread process an entire row) we were able to drastically reduce the runtime overhead to 10-20% with both sensitive values and outliers.

D. Ablation Studies

D.1. Sensitivity-Based Quantization.

In our ablation study, we investigate the impact of sensitivity-based weighted clustering on the performance of non-uniform quantization. In Tab. D.2, we compared the performance of sensitivity-based and sensitivity-agnostic approaches in the context of 3-bit quantization of the LLaMA-7B model. For sensitivity-agnostic quantization, we apply non-weighted k-means clustering at sparsity levels of 0%, 0.05%, and 0.45%. The results demonstrate that while non-uniform quantization alone can reduce the perplexity from 28.26 (of RTN uniform quantization) to 18.08 without considering sensitivity, incorporating sensitivity-based clustering is critical in reducing the perplexity to 7.75. This improvement is consistent across all sparsity levels.

D.2. Impact of Sparsity Levels on SqueezeLLM

In Fig. D.2 (Left), we present the perplexity results of the 3-bit quantized LLaMA-7B model on the C4 benchmarks, with varying percentages of sensitive values extracted as the sparse matrix, ranging from 0% to 0.2%. The plot demonstrates that the perplexity gain diminishes as the sparsity level of the sensitive values exceeds 0.05%. Therefore, we maintain a fixed sparsity level of 0.05% for the sensitive values throughout all experiments.

Furthermore, in Figure D.2 (Right), we compare the performance when the sensitive values are not removed as the sparse matrix (only outlier values are removed) to the case where 0.05% of the sensitive values are removed. In both scenarios, we control the sparsity level by increasing the percentage of outlier values included in the sparse matrix to obtain the trade-off curves. The results indicate that the sparsity configuration with both sensitive values and outlier values consistently outperforms the configuration with only outlier values.

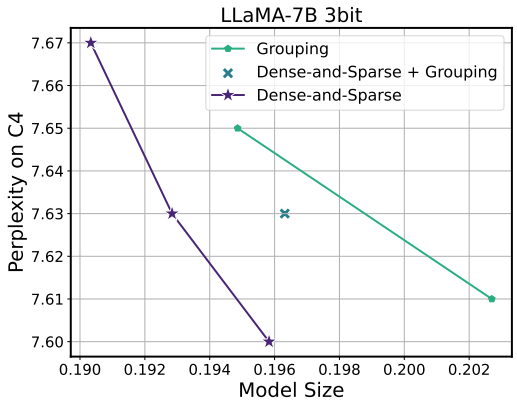

Figure D.3. Model size (normalized by the size of the FP16 model) and perplexity trade-offs of grouping and the Dense-and-Sparse decomposition on 3-bit quantization of LLaMA-7B. Here, we compare SqueezeLLM with (i) grouping using group sizes of 1024 and 512 (green), (ii) a hybrid approach that combines a group size of 1024 with a sparsity level of 0.05% (blue), and (iii) the Dense-and-Sparse decomposition approach with varying sparsity levels (violet). The pure Dense-and-Sparse decomposition always outperforms both grouping and the hybrid approach.

D.3. Impact of Grouping on SqueezeLLM

In Fig. D.3, we explore the effectiveness of incorporating grouping into SqueezeLLM as an alternative approach to improve quantization performance. We compare three configurations: SqueezeLLM with (i) grouping using group sizes of 1024 and 512 (green), (ii) a hybrid approach that combines a group size of 1024 with a sparsity level of 0.05% (blue), and (iii) the Dense-and-Sparse decomposition approach with varying sparsity levels (violet), where 0.05% of sensitive values are kept and the percentage of outlier values is adjusted. The results clearly demonstrate that both grouping and the hybrid approach result in suboptimal trade-offs compared to the pure Dense-and-Sparse decomposition approach.

This can be attributed to two factors. First, the Dense-and-Sparse decomposition is a direct solution to the outlier issue. In contrast, while grouping can mitigate the impact of outliers to some extent by isolating them within individual groups, it does not provide a direct solution to this issue. Second, grouping can introduce significant overhead in terms of storage requirements when combined with non-uniform quantization, since it needs to store one LUT per group. This can be a considerable overhead compared to the uniform quantization approach, where only a scaling and zero point value per group need to be stored.

D.4. Comparison of Optimization Objectives for Non-uniform Quantization: Minimizing Layer-wise Perturbation versus Final Output Perturbation

While our method targets minimizing the perturbation of the final output of the model during quantization, it is worth noting that minimizing the layer-wise perturbation can also be considered as an alternative. Most existing solutions for LLM quantization including GPTQ (Frantar et al., 2022), AWQ (Lin et al., 2023), and SpQR (Dettmers et al., 2023) have used the latter objective, which aims to minimize the perturbation of output activations in individual layers. In this ablation study, we demonstrate that minimizing the final output perturbation is a superior objective to minimizing the layer-wise perturbation.

When minimizing the layer-wise perturbation, the optimization objective for determining the non-uniform quantization configuration can be reformulated as $\arg\min_Q \|WX - W_QX\|_2^2$, where X denotes a batch of input activations. This object can be approximated as a weighted k-means clustering problem, where each weight is weighted by the square of the corresponding input activation size. This indeed results in the activation-based sensitivity/importance metric as in the AWQ framework (Lin et al., 2023).

In Fig. D.4, we compare the perplexity on the C4 dataset for 3-bit quantization of the LLaMA-7B model using both objectives. Across all sparsity levels obtained by adjusting the number of outliers being extracted, SqueezeLLM based on final loss perturbation minimization outperforms the alternative of using layer-wise perturbation minimization by a large margin of up to around 0.3 perplexity points.

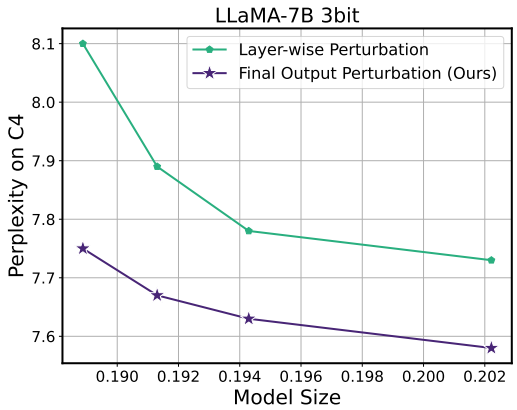

Figure D.4. Model size (normalized by the size of the FP16 model) and perplexity trade-offs for 3-bit quantization of the LLaMA-7B model using layer-wise perturbation minimization versus final output perturbation minimization as a non-uniform quantization objective. The trade-off is obtained by adjusting the sparsity level of the outliers being extracted. Across all sparsity levels, the OBD framework, which is the foundation for SqueezeLLM, consistently outperforms the OBS framework as an alternative approach.

Table D.3. Perplexity scores on Wikitext2 for the LLaMA-2 7B model, quantized using non-uniform (SqueezeLLM's sensitivity-based quantization) and uniform (RTN) approaches with 3 and 4-bit precision with varying levels of sparsity.

Bit Width	Sparsity Level (%)	Avg. Bit Width	Uniform (PPL)	Nonuniform (PPL)
16-bit	0	16	5.47	**5.47**
4-bit	0	4.04	6.12	**5.62**
	0.05	4.09	5.95	**5.59**
	0.45	4.26	5.95	**5.57**
	2	5.01	5.95	**5.55**
	4.5	6.20	5.94	**5.53**
3-bit	0	3.02	542.00	**6.18**
	0.05	3.07	27.38	**6.05**
	0.45	3.24	26.58	**5.96**
	1.5	3.98	25.97	**5.81**
	4.5	5.18	23.58	**5.73**

D.5. Impact of Non-uniform Quantization versus Dense-and-Sparse Decomposition

In Tab. D.3, we perform a detailed analysis to further disambiguate the impact of non-uniform quantization and the Dense-and-Sparse decomposition.

Uniform vs. Non-uniform Quantization. As can be seen in Tab. D.3, across all bitwidths and sparsity levels, our non-uniform quantization has noticeable improvements over uniform quantization.

Sparsity Levels. Furthermore, we also report the results with varying sparsity levels of the Dense-and-Sparse decomposition in Tab. D.3. As expected, higher levels of sparsity consistently result in improved performance in any scenario. However, there are diminishing returns for larger values of sparse decomposition since only a small portion of the weight values are outliers or sensitive. As a consequence, saving additional values into the sparse format does not help as much beyond a certain level, and instead results in higher average bitwidth. This is in line with the conclusions in the main experiments where we found a sparsity level of 0.45% sufficient for the performance gain.

D.6. Impact of Dense-and-Sparse Decomposition versus Precision

In Tab. D.4, we additionally demonstrate that increasing the bit width of the dense component results in higher improvement in perplexity compared to increasing the sparsity level. Note that 4-bit LLaMA-2 7B model without any sparsity outperforms the 3-bit counterparts with sparsity levels of 1.5% and 2.5% that have similar or even larger model sizes. This observation aligns with the sensitivity level ablation study in Appendix D.5, since the Dense-and-Sparse decomposition is only effective

Table D.4. Perplexity scores on C4 and WikiText2 for the LLaMA-2 7B model, quantized using SqueezeLLM with 4-bit and 3-bit with different sparsity level. In particular, the sparsity levels of 3-bit quantization are selected to match their average bit widths to that of 4-bit quantization without sparsity.

Bit Width	Sparsity Level (%)	Avg. Bit Width	C4 (PPL)	WikiText2 (PPL)
16-bit	0	16	6.97	5.47
4-bit	0	4.04	7.12	5.62
3-bit	1.5	3.98	7.35	5.81
	2.5	4.22	7.32	5.80

Table E.5. Peak memory requirement in GB when quantizing different LLaMA models.

Model	Peak Memory (GB)
LLaMA-7B	33
LLaMA-13B	61
LLaMA-30B	149
LLaMA-65B	292

Table E.6. End-to-end latency breakdown of quantizing different LLaMA models. Latency is broken down into (i) Fisher information computation on a A100 system and (ii) sensitivity-based k-means clustering on Intel Xeon Gold 6126 with 48 cores. In the last column, we provide the end-to-end time for GPTQ as reported in the original paper.

Model	Fisher Computation (min)	K-means (min)	GPTQ (min)
LLaMA-7B	0.3	11	10
LLaMA-13B	0.6	17	21
LLaMA-30B	1.3	45	45
LLaMA-65B	2.5	80	96

to the extent of removing the outliers and sensitive values from the parameters. Increasing the sparsity level beyond that will not be effective and results in diminishing returns.

E. Quantization Cost Analysis

E.1. Memory Requirement

In Tab. E.5, we report the memory requirement of SqueezeLLM when quantizing different model sizes from 7B to 65B. Note that our method can have a higher memory requirement than GPTQ. This is because SqueezeLLM performs quantization based on minimizing the perturbation to the loss function of the model which requires computing the Fisher information matrix. GPTQ, on the other hand, performs quantization by minimizing the perturbation to the output activation of the individual layer, which does not require back-propagating the gradient through the model to compute the Fisher information matrix. However, this is a one-time cost, and as demonstrated below, this gradient computation process is fast, taking only 2-3 minutes even for the largest 65B model.

E.2. Quantization Time

In Tab. E.6, we additionally assess the end-to-end time for (i) computing the Fisher information on an A100 system and (ii) performing sensitivity-based K-means clustering on Intel Xeon Gold 6126 with 48 cores, which are two major procedures in SqueezeLLM. Note that the time for computing the Fisher information matrix is minimal, taking only 2.5 minutes with the largest 65B model. K-mean clustering can take 11 min for the 7B model and up to 80 min for the 65B model. Overall, the computational time requirement of SqueezeLLM is on par with that of GPTQ.

E.3. Data Efficiency

In Tab. E.7, we provide data efficiency analysis in terms of the number of data samples to calculate the Fisher information matrix (gradients) for sensitivity-based non-uniform quantization. While we used a calibration set of 100 data samples

Table E.7. Perplexity on C4 and Wikitext2 of the LLaMA2 7B model after 4-bit quantization, with varying sizes of the calibration dataset used for computing the Fisher information matrix.

# Data Examples	C4	Wikitext2
1	7.89	6.41
2	7.81	6.22
5	7.73	6.20
10	7.72	6.17
20	7.72	6.16
100	7.72	6.18

Table F.8. Perplexity on Wikitext2 of the LLaMA2 13B and 70B models quantized into 4, 3, and 2 bits using SqueezeLLM and QuIP (Chee et al., 2024). For QuIP, we use the perplexity numbers that are reported in the original paper as well as our own reproduction using the official codebase. Following the perplexity evaluation method of the QuIP paper, we use sequence length of 4096 (different from other experiments that use sequence length of 2048).

Model	Config.	Avg. Bit Width	LLaMA2-13B	LLaMA2-70B
QuIP (original paper)	4-bit	4	-	3.53
QuIP (our repr)	4-bit	4	4.81	3.65
SqueezeLLM	4-bit	4.05	**4.67**	**3.21**
QuIP (original paper)	3-bit	3	-	3.85
QuIP (our repr)	3-bit	3	5.25	3.84
SqueezeLLM	3-bit	3.02	**5.01**	**3.55**
QuIP (original paper)	2-bit	2	-	6.33
QuIP (our repr)	2-bit	2	**20.54**	**6.20**
SqueezeLLM	2-bit	2.01	61.25	10.86
SqueezeLLM	2-bit + 0.1%	2.05	**7.91**	**5.04**
SqueezeLLM	2-bit + 0.45%	2.22	**7.43**	**4.71**

throughout the paper, a calibration set with as few as 10 examples is typically sufficient to achieve the desired quantization performance. Note that both GPTQ and AWQ require 100-200 data points for calibration as reported in the AWQ paper (Lin et al., 2023).

F. Comparison with Other Weight-only Quantization Methods

In this section, we compare SqueezeLLM with more recent weight-only quantization methods including QuIP (Chee et al., 2024) and OmniQuant (Shao et al., 2023).

F.1. Comparison with QuIP

Here, we provide a quantitative comparison of our method to QUIP. Given that the QuIP paper only reports performance evaluation of LLaMA2-70B among all LLaMA models, we enrich our comparison by additionally incorporating our own reproduction based on their official codebase. Different from other experiments that use sequence length of 2048, we use sequence length of 4096, following the perplexity evaluation method of the QuIP paper. In Tab. F.8, we compare the perplexity scores on Wikitext2 for LLaMA2 13B and 70B models quantized to 4, 3, and 2-bit. Note that we did not include a comparison on LLaMA2 7B as we were unable to achieve reasonable performance with QuIP, as was also reported in (Egiazarian et al., 2024).

The table indicates that dense-only SqueezeLLM consistently achieves superior performance over QUIP, across all model sizes and quantization bitwidth. With 2bit quantization, we noticed that solely relying on dense-only quantization may not yield results as competitive as those of QuIP. However, by incorporating just 0.1% sparsity (additional 0.05 bit; 0.05% outlier values + 0.05% sensitive values), SqueezeLLM significantly outperforms QuIP by a considerable margin.

F.2. Comparison with OmniQuant

In Tab. F.9, we compare the perplexity of our method to OmniQuant on WikiText2 using sequence length of 2048. In particular, the table reports the perplexity numbers of 4 and 3-bit quantized models across all LLaMA and LLaMA2 models.

Table F.9. Perplexity on Wikitext2 of all LLaMA and LLaMA2 models quantized into 4 and 3 bits using SqueezeLLM and Omni-Quant (Chee et al., 2024). For OmniQuant, we directly use the perplexity numbers that are reported in the original paper.

Model	Config.	Avg. Bit Width	7B	13B	30B	65B	2-7B	2-13B	2-70B
Baseline	16-bit	16	5.68	5.09	4.1	3.53	5.47	4.88	3.32
Omniquant	4-bit	4	5.86	5.21	4.25	**3.71**	5.74	5.02	3.47
SqueezeLLM	4-bit	4.05	**5.79**	**5.18**	**4.22**	3.76	**5.62**	**4.99**	**3.41**
Omniquant	4-bit (g128)	4.24	5.77	5.17	4.19	**3.62**	5.58	**4.95**	3.4
SqueezeLLM	4-bit (0.45%)	4.27	**5.77**	**5.17**	**4.18**	3.63	**5.57**	4.96	**3.39**
Omniquant	3-bit	3	6.49	5.68	4.74	4.04	6.58	5.58	3.92
SqueezeLLM	3-bit	3.02	**6.32**	**5.60**	**4.66**	**4.05**	**6.18**	**5.36**	**3.77**
Omniquant	3-bit (g128)	3.24	6.15	**5.44**	4.56	3.94	6.03	5.28	3.78
SqueezeLLM	3-bit (0.45%)	3.24	**6.13**	5.45	**4.44**	**3.88**	**5.96**	**5.23**	**3.63**

Table F.10. Perplexity on Wikitext2 of all LLaMA2 models quantized into 2 bits using SqueezeLLM and OmniQuant (Chee et al., 2024). For OmniQuant, we directly use the perplexity numbers that are reported in the original paper.

Model	Config.	Avg. Bit Width	2-7B	2-13B	2-70B
Baseline	16-bit	16	5.47	4.88	3.32
OmniQuant	2-bit	2	37.37	**17.21**	**7.81**
SqueezeLLM	2-bit	2.01	**35.49**	41.02	9.44
SqueezeLLM	2-bit (0.1%)	2.05	**13.64**	**8.56**	**5.38**
OmniQuant	2-bit (g128)	2.24	11.06	8.26	6.55
SqueezeLLM	2-bit (0.45%)	2.22	**10.79**	**7.91**	**4.99**

Table G.11. Latency (s) and peak memory usage (GB) of 3-bit LLaMA when generating 1024 tokens on an A6000 GPU. The table compares the FP16 baseline, non-grouped and grouped GPTQ with activation ordering, and SqueezeLLM with different sparsity levels. For comparison, we include bitwidth and perplexity on the C4 benchmark.

Method	Bit width	7B			13B			30B			65B		
		PPL (C4)	Lat (s)	Mem (G)	PPL (C4)	Lat (s)	Mem (G)	PPL (C4)	Lat (s)	Mem (G)	PPL (C4)	Lat (s)	Mem (G)
Baseline	16	7.08	26.5	13.1	6.61	47.0	25.2	5.98	OOM	OOM	5.62	OOM	OOM
GPTQ	3	7.55	12.6	3.3	6.22	19.1	6.0	5.76	36.8	13.8	5.58	60.2	26.2
SqueezeLLM	3.02	6.32	13.6	3.4	5.60	21.2	6.1	4.66	37.8	16.1	4.05	66.9	29.9
GPTQ (g128)	3.25	6.27	110.7	3.4	5.47	176.1	6.2	4.83	500.8	14.3	4.55	955.2	27.3
SqueezeLLM (0.45%)	3.24	6.13	14.6	3.6	5.45	22.2	6.5	4.44	42.5	17.4	3.88	82.35	32.4

For OmniQuant, we directly use the numbers reported in the original paper. Omniquant and SqueezeLLM are grouped in the table so that their model sizes are roughly the same. This comparison demonstrates that SqueezeLLM generally outperforms OmniQuant with the same model size and memory constraints.

Additionally, Tab. F.10 demonstrates the same comparison using 2-bit quantization. With 2-bit quantization, the table shows that OmniQuant without grouping outperforms dense-only SqueezeLLM on the 13B and 70B models. This can be attributed to OmniQuant's learnable clipping ranges via a few iterations of training that effectively account for outliers. SqueezeLLM's sensitivity-based nonuniform quantization alone does not inherently address this. Handling outliers can be particularly critical for 2-bit quantization where weights should be represented with only four values. Nevertheless, introducing a 0.1% sparsity remarkably enhances SqueezeLLM's performance with a minimal memory overhead increase of 0.05 bit. This perplexity improvement is also persistent when comparing OmniQuant with a group size 128 and SqueezeLLM at a 0.45% sparsity level with roughly the same size.

G. Additional Hardware Profiling Results

In Tab. G.11, we provide additional hardware profiling results using a sequence length of 1024. All the experimental setups and details are identical to Sec. 5.4 and Tab. 3.

Table G.12. Matrix-vector kernel runtime (in seconds) for generating 128 tokens, benchmarked on an A100 GPU. Our kernel implementation attains 1.5-2.5× performance speedups relative to the fp16 matrix-vector multiply kernel across different model sizes without any additional optimizations or tuning. We include GPTQ (with group size 128) without reordering for comparison against the latency of uniform quantization with grouping.

Method	Bit Width	Model 7B	13B	30B
Baseline	16	1.21	2.32	5.56
GPTQ (g128)	4	0.92	1.51	3.24
SqueezeLLM	4	0.83	1.52	3.66
SqueezeLLM (0.45%)	4	1.09	1.87	4.25
GPTQ (g128)	3	0.62	1.03	2.39
SqueezeLLM	3	0.56	0.97	2.26
SqueezeLLM (0.45%)	3	0.83	1.32	2.86

Table H.13. Perplexity comparison of LLaMA-30B and 65B models quantized into 4 and 3 bits using different methods including RTN, GPTQ, AWQ and SpQR on C4 and WikiText-2. We compare the performance of GPTQ, AWQ, and SqueezeLLM in groups based on similar model sizes. In the first group, we compare dense-only SqueezeLLM with non-grouped GPTQ. In the subsequent groups, we compare SqueezeLLM with different levels of sparsity to GPTQ and AWQ with different group sizes.

LLaMA-30B	3-bit			4-bit		
Method	Avg. Bits (comp. rate)	PPL (↓) C4	Wiki	Avg. Bits (comp. rate)	PPL (↓) C4	Wiki
Baseline	16	5.98	4.10	16	5.98	4.10
RTN	3 (5.33)	28.53	14.89	4 (4.00)	6.33	4.54
GPTQ	3 (5.33)	7.31	5.76	4 (4.00)	6.20	4.43
SpQR	-	-	-	3.89 (4.11)	6.08	4.25
SqueezeLLM	3.02 (5.31)	**6.37**	**4.66**	4.03 (3.97)	**6.06**	**4.22**
GPTQ (g128)	3.25 (4.92)	6.47	4.83	4.25 (3.77)	6.07	4.24
AWQ (g128)	3.25 (4.92)	6.38	4.63	4.25 (3.77)	6.05	4.21
SqueezeLLM (0.45%)	3.25 (4.92)	**6.23**	**4.44**	4.25 (3.77)	**6.04**	**4.18**

LLaMA-65B	3-bit			4-bit		
Method	Avg. Bits (comp. rate)	PPL (↓) C4	Wiki	Avg. Bits (comp. rate)	PPL (↓) C4	Wiki
Baseline	16	5.62	3.53	16	5.62	3.53
RTN	3 (5.33)	12.77	10.59	4 (4.00)	5.86	3.92
GPTQ	3 (5.33)	6.70	5.58	4 (4.00)	5.81	4.11
SpQR	3 (5.33)	-	4.2[†]	3.90 (4.10)	5.70	**3.68**
SqueezeLLM	3.02 (5.30)	**5.99**	**4.05**	4.04 (3.96)	**5.69**	3.76
GPTQ (g128)	3.25 (4.92)	6.01	4.55	4.25 (3.77)	5.69	3.76
AWQ (g128)	3.25 (4.92)	5.94	4.00	4.25 (3.77)	5.68	3.67
SqueezeLLM (0.45%)	3.24 (4.94)	**5.84**	**3.88**	4.26 (3.76)	**5.67**	**3.63**

Additionally, in Tab. G.12, we demonstrate that our custom CUDA kernels (both including and without including outliers) attain significant speedups of 1.5-2.5× relative to the fp16 baseline. These results were obtained without any additional optimizations or tuning specifically for the A100, demonstrating how our kernels are easily portable across different GPUs and do not introduce complexity.

H. Additional Experiment Results

H.1. Perplexity Evaluation

In Tab. H.13, we provide the full experimental results on LLaMA (Touvron et al., 2023a). Furthermore, in Tab. H.14 and H.15, we provide additional experimental results on LLaMA2 (Touvron et al., 2023b) and OPT (Zhang et al., 2022) models.

H.2. 5-shot MMLU Evaluation

In Tab. H.16, we provide additional results on 5-shot MMLU evaluation using the Vicuna v1.1 (7/13B) and Vicuna v1.3 (7/13/33B) models. We see a similar trend as the zero-shot MMLU evaluation results where SqueezeLLM consistently outperforms the baseline quantization methods with the same model size.

I. Limitations

While our empirical results primarily focus on generation tasks, the proposed ideas in this work are not inherently limited to decoder architectures. However, we have not yet conducted thorough assessments of our framework's effectiveness on

[†]SpQR does not report their near-3-bit performance. However, in the case of 65B model, its 3-bit perplexity on Wikitext-2 can be inferred from the trade-off curve in Figure 8 of their paper. This comparison indicates that the gap between SpQR and SqueezeLLM can be larger in the lower-bitwidth regimes.

Table H.14. Perplexity comparison of LLaMA2 models quantized into 4 and 3 bits using different methods including RTN, GPTQ, AWQ and SpQR on C4 and WikiText-2. We compare the performance of GPTQ, AWQ, and SqueezeLLM in groups based on similar model sizes. In the first group, we compare dense-only SqueezeLLM with non-grouped GPTQ. In the subsequent groups, we compare SqueezeLLM with different levels of sparsity to GPTQ and AWQ with different group sizes. Note that all GPTQ results are with activation reordering.

LLaMA2-7B	**3-bit**			**4-bit**		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	6.97	5.47	16	6.97	5.47
RTN	3 (5.33)	404.45	542.86	4 (4.00)	7.72	6.12
GPTQ	3 (5.33)	10.45	8.97	4 (4.00)	7.42	5.90
SqueezeLLM	3.02 (5.29)	**7.72**	**6.18**	4.05 (3.95)	**7.12**	**5.62**
GPTQ (g128)	3.24 (4.93)	7.97	6.25	4.24 (3.77)	7.23	5.72
AWQ (g128)	3.24 (4.93)	7.84	6.24	4.24 (3.77)	7.13	5.72
SqueezeLLM (0.45%)	3.24 (4.93)	**7.51**	**5.96**	4.27 (3.75)	**7.08**	**5.57**

LLaMA2-13B	**3-bit**			**4-bit**		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	6.47	4.88	16	6.47	4.88
RTN	3 (5.33)	12.50	10.68	4 (4.00)	6.83	5.20
GPTQ	3 (5.33)	8.27	6.17	4 (4.00)	6.74	5.08
SqueezeLLM	3.02 (5.30)	**6.97**	**5.36**	4.04 (3.96)	**6.57**	**4.99**
GPTQ (g128)	3.25 (4.92)	7.06	5.31	4.25 (3.77)	6.57	**4.96**
AWQ (g128)	3.25 (4.92)	6.94	5.32	4.25 (3.77)	6.56	4.97
SqueezeLLM (0.45%)	3.24 (4.94)	**6.82**	**5.23**	4.26 (3.76)	**6.54**	**4.96**

LLaMA2-70B	**3-bit**			**4-bit**		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	5.52	3.32	16	5.52	3.32
RTN	3 (5.33)	10.02	7.52	4 (4.00)	5.80	3.67
GPTQ	3 (5.33)	6.69	4.86	4 (4.00)	5.70	3.59
SqueezeLLM	3.02 (5.30)	**5.83**	**3.77**	4.04 (3.96)	**5.58**	**3.41**
GPTQ (g128)	3.25 (4.92)	5.87	3.88	4.25 (3.77)	5.59	3.42
AWQ (g128)	3.25 (4.92)	5.81	3.74	4.25 (3.77)	5.58	3.41
SqueezeLLM (0.45%)	3.24 (4.94)	**5.73**	**3.63**	4.26 (3.76)	**5.57**	**3.39**

encoder-only or encoder-decoder architectures, as well as other neural network architectures. Additionally, it is important to note that our hardware performance modeling approach relies on a simulation-based method using a roofline model, which entails making simplified assumptions about the hardware's inference pipeline.

Table H.15. Perplexity comparison of OPT models quantized into 4 and 3 bits using different methods including RTN, GPTQ, AWQ and SpQR on C4 and WikiText-2. We compare the performance of GPTQ, AWQ, and SqueezeLLM in groups based on similar model sizes. In the first group, we compare dense-only SqueezeLLM with non-grouped GPTQ. In the subsequent groups, we compare SqueezeLLM with different levels of sparsity to GPTQ and AWQ with different group sizes. Note that all GPTQ results are with activation reordering. "div" means that the perplexity is diverged.

OPT-1.3B	3-bit			4-bit		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	14.72	14.62	16	14.72	14.62
RTN	3 (5.43)	div.	div.	4 (4)	24.68	48.19
SqueezeLLM	3.04 (5.26)	**16.42**	**16.30**	4.09 (3.91)	**15.01**	**14.94**
AWQ (g128)	3.25 (4.93)	16.28	16.32	4.25 (3.77)	15.04	14.95
SqueezeLLM (0.5%)	3.25 (4.92)	**15.84**	**15.76**	4.30 (3.72)	**14.94**	**14.83**

OPT-2.7B	3-bit			4-bit		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	13.17	12.47	16	13.17	12.47
RTN	3 (5.33)	div.	div.	4 (4)	17.52	16.92
SqueezeLLM	3.04 (5.26)	**14.45**	**13.85**	4.07 (3.93)	**13.38**	**12.80**
AWQ (g128)	3.25 (4.93)	16.28	16.32	4.25 (3.77)	13.39	12.73
SqueezeLLM (0.5%)	3.25 (4.92)	**13.88**	**13.43**	4.29 (3.73)	**13.30**	**12.60**

OPT-6.7B	3-bit			4-bit		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	11.74	10.86	16	11.74	10.86
RTN	3 (5.33)	div.	div.	4 (4)	13.38	12.10
SpQR	-	-	-	3.94 (4.06)	11.98	11.04
SqueezeLLM	3.02 (5.29)	**12.44**	**11.70**	4.05 (3.96)	**11.85**	**11.03**
SpQR	-	-	-	4.27 (3.74)	11.88	**10.91**
AWQ (g128)	3.25 (4.92)	12.30	11.41	4.25 (3.77)	11.86	10.93
SqueezeLLM (0.5%)	3.26 (4.90)	**12.18**	**11.31**	4.28 (3.73)	**11.83**	10.92

OPT-13B	3-bit			4-bit		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	11.20	10.12	16	11.20	10.12
RTN	3 (5.33)	div.	div.	4 (4)	12.35	11.32
SpQR	-	-	-	3.93 (4.07)	11.34	10.28
SqueezeLLM	3.02 (5.29)	**12.69**	**11.76**	4.05 (3.96)	**11.29**	**10.24**
SpQR	-	-	-	4.27 (3.74)	11.27	**10.22**
AWQ (g128)	3.25 (4.92)	12.61	10.67	4.25 (3.77)	11.28	**10.22**
SqueezeLLM (0.5%)	3.26 (4.90)	**11.57**	**10.54**	4.28 (3.73)	**11.26**	**10.22**

OPT-30B	3-bit			4-bit		
Method	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki	**Avg. Bits** (comp. rate)	**PPL** (↓) C4	Wiki
Baseline	16	10.69	9.56	16	10.69	9.56
RTN	3 (5.33)	div.	div.	4 (4)	11.90	10.98
SpQR	-	-	-	3.94 (4.06)	10.78	**9.54**
SqueezeLLM	3.01 (5.31)	**11.10**	**10.17**	4.03 (3.97)	**10.75**	9.65
SpQR	-	-	-	4.26 (3.76)	10.73	**9.50**
AWQ (g128)	3.25 (4.92)	10.96	9.85	4.25 (3.77)	10.75	9.59
SqueezeLLM (0.5%)	3.26 (4.90)	**10.93**	**9.77**	4.28 (3.73)	**10.72**	9.61

Table H.16. Comparison of PTQ methods on five-shot MMLU accuracy applied to Vicuna v1.1 and v1.3. We add peak memory usage in GB for comparison.

Method	**Avg. bit**	**Vicuna-7B (v1.1)** Acc (↑)	Mem (GB, ↓)	**Vicuna-13B (v1.1)** Acc (↑)	Mem (GB, ↓)	**Vicuna-7B (v1.3)** Acc (↑)	Mem (GB, ↓)	**Vicuna-13B (v1.3)** Acc (↑)	Mem (GB, ↓)	**Vicuna-33B (v1.3)** Acc (↑)	Mem (GB, ↓)
Baseline	16	45.3%	12.7	50.0%	24.6	45.6%	12.7	51.6%	24.6	60.1%	OOM
AWQ (g128)	4.25	44.1%	3.8	48.8%	7.2	44.8%	3.8	50.7%	7.2	59.6%	17.2
SqueezeLLM	4.05	44.3%	3.8	48.4%	6.9	44.3%	3.8	50.5%	6.9	59.6%	16.5
SqueezeLLM (0.45%)	4.26	**44.7%**	4.0	**49.7%**	7.3	**44.9%**	4.0	**51.4%**	7.3	**60.0%**	17.7
AWQ (g128)	3.25	41.4%	3.0	46.3%	5.7	42.5%	3.0	48.4%	5.7	56.3%	13.3
SqueezeLLM	3.02	40.4%	2.9	45.6%	5.4	41.0%	2.9	47.4%	5.4	55.7%	12.4
SqueezeLLM (0.45%)	3.24	**42.2%**	3.1	**48.2%**	5.8	**43.2%**	3.1	**48.8%**	5.8	**58.2%**	13.7