SqueezeLLM: Dense-and-Sparse Quantization

Sehoon Kim*
sehoonkim@berkeley.edu
UC Berkeley

Coleman Hooper*
chooper@berkeley.edu
UC Berkeley

Amir Gholami*†
amirgh@berkeley.edu
ICSI, UC Berkeley

Zhen Dong
zhendong@berkeley.edu
UC Berkeley

Xiuyu Li
xiuyu@berkeley.edu
UC Berkeley

Sheng Shen
sheng.s@berkeley.edu
UC Berkeley

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

Kurt Keutzer
keutzer@berkeley.edu
UC Berkeley

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 open-sourced and available online[1].

1 INTRODUCTION

Recent advances in Large Language Models (LLMs), with up to hundreds of billions of parameters and trained on massive text corpora, have showcased their remarkable problem-solving capabilities across various domains [4, 10, 16, 30, 51, 53, 57, 62, 64, 84, 84]. However, deploying these models for inference has been a significant challenge due to their demanding resource requirements. For instance, the LLaMA-65B [63] model requires at least 130GB of RAM to deploy in FP16, and this 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 [50]. 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 incurring performance degradation [75], where 8-bit quantization not only reduces 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 [19] which uses a training-free quantization technique that achieves near-lossless 4-bit quantization for large 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) such as the recent LLaMA [64] or its instruction-following variants [8, 22, 61].

In this paper, we conduct an extensive study of low-bit precision quantization and identify limitations in existing approaches. Building upon these insights, we propose a novel solution that achieves lossless compression and improved quantization performance for models of the same size, even at precisions as low as 3 bits.

Contributions. We start by presenting performance modeling results demonstrating that *the memory, rather than the compute, is the primary bottleneck* in LLM inference with generative tasks. Building on this insight, we introduce SqueezeLLM, a post-training quantization framework that incorporates a novel *sensitivity-based non-uniform quantization* technique and a *Dense-and-Sparse decomposition* method. These techniques enable ultra-low-bit precision quantization, while maintaining competitive model performance, significantly reducing the model sizes and inference time costs. In more detail, our main contributions can be summarized as follows:

[*] Equal contribution
[†] Corresponding author
[1] https://github.com/SqueezeAILab/SqueezeLLM

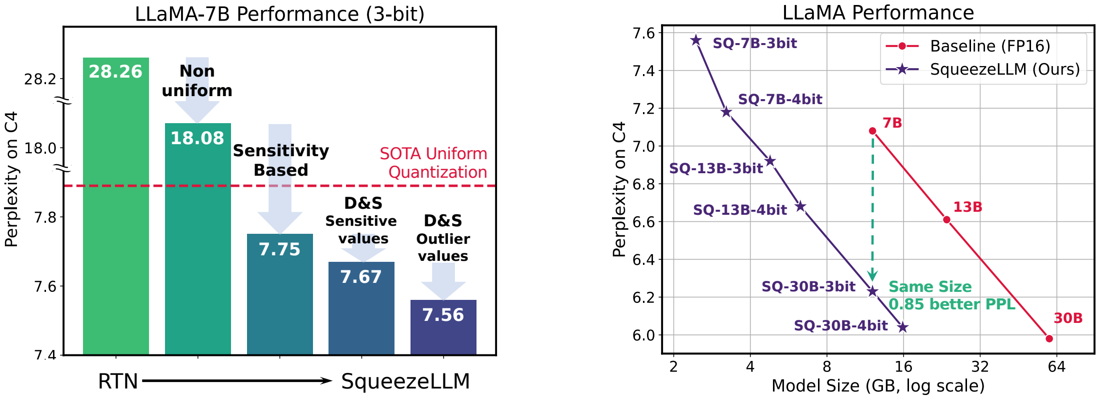

Figure 1: *(Left) We introduce SqueezeLLM, a post-training quantization scheme for LLM that outperforms the existing state-of-the-art methods [19, 45]. SqueezeLLM incorporates two key approaches: (i) sensitivity-based non-uniform quantization (Sec. 4.1), where quantization bins are allocated closer to 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, effectively constraining the quantization range and enhancing quantization performance. When applied to LLaMA-7B with 3-bit quantization, our method outperforms the state-of-the-art methods [19, 45] by a large perplexity margin of over 0.3 on the C4 benchmark. (Right) By applying our methods to LLaMA [64] models of varying sizes, we can achieve improved trade-offs between perplexity and model size. Notably, our 3-bit-quantized LLaMA-30B model outperforms the 7B models by a significant 0.85 perplexity improvement on the C4 benchmark while maintaining the same model size.*

- **Memory Wall:** We conduct a comprehensive performance benchmark for generative LLM inference to analyze the impact of different components on the overall runtime, and find that memory bandwidth, rather than compute, is the primary bottleneck. For instance, we discover that by simply reducing the bit precision of the weights, while maintaining activations and compute at FP16 precision, we can achieve linear speed-ups in runtime proportional to the bit precision used (see Sec. 3 and Fig. 2).
- **Sensitivity based Non-Uniform Quantization:** We demonstrate that uniform quantization, as commonly adopted in prior works, is sub-optimal for LLM inference for two reasons. First, the weight distributions of different layers in LLMs exhibit clear non-uniform patterns, as illustrated in Fig. 3. Second, the inference computation in prior works does not benefit from uniform quantization as the arithmetic is performed in FP16 precision, and quantization is only used for memory requirement reduction. To address these two limitations, we propose a novel sensitivity-based non-uniform quantization method to achieve a more optimal quantization scheme for LLMs. Our approach significantly improves the perplexity of the LLaMA-7B model at 3-bit precision, yielding perplexity improvement from 28.26 of uniform quantization to 7.75 on the C4 dataset (see Sec. 4.1).
- **Dense-and-Sparse Quantization:** We observe that weight matrices in many LLMs contain significant outliers, making low-bit precision quantization extremely challenging. These outliers also affect our non-uniform quantization scheme, as they skew the bit allocation towards these extreme values. To address this issue, we propose a simple solution that decomposes the model weights into a dense component and a sparse component, where the latter extracts the outlier values from the original weights. By separating the outliers, the dense part exhibits a more compact range of up to 10×, allowing for improved quantization accuracy. The sparse part is stored in full precision using efficient sparse storage methods like Compressed Sparse Row (CSR). This approach introduces minimal overhead, as we can leverage efficient sparse kernels for the sparse part and parallelize the computation alongside the dense part. By only extracting 0.45% of the weight values as the sparse component, we further improve the perplexity of the LLaMA-7B model from 7.75 to 7.58 on the C4 dataset (see Sec. 4.2).
- **Performance Evaluation:** We extensively test SqueezeLLM on LLaMA-7B, 13B, and 30B on language modeling tasks using the C4 and WikiText2 benchmarks, where we find that SqueezeLLM consistently outperforms existing quantization methods by a large margin across different bit precisions (see Tab. 1 and Fig. 5).
- **Quantization of Instruction Following Models:** We also demonstrate the potential of SqueezeLLM in quantizing instruction following models by applying it to the Vicuna-7B and 13B models [8]. We use two evaluation methods. First, we evaluate the generation quality on the MMLU dataset [29], a multi-task benchmark that assesses a model's knowledge and problem-solving abilities. Additionally, following the evaluation methodology introduced in Vicuna [8], we employ GPT-4 to rank the generation quality of our quantized models compared to the FP16 baseline. In both evaluations, SqueezeLLM consistently outperforms current state-of-the-art methods, including GPTQ and AWQ. Notably, our 4-bit quantized model achieves performance on par with the baseline in both evaluations (see Tab. 2 and Fig. 6).
- **Model Deployment and Profiling:** Our deployed models on A6000 GPUs not only demonstrate improved quantization performance but also exhibit significant gains in latency. For LLaMA-7B and 13B, we observe speedups of up to 2.3× compared to baseline FP16 inference. Furthermore, our approach achieves up to 4× faster latency as compared to GPTQ, showcasing the effectiveness of our method in terms of both quantization performance and inference efficiency (see Tab. 3).

2 RELATED WORK

2.1 Efficient Transformer Inference

Various approaches have been proposed to reduce the latency and memory footprint of Transformer inference. While some approaches concentrate on enhancing the efficiency of the decoding process [6, 26, 37, 54, 70], another line of research focuses on making the Transformer architecture itself more efficient. This can be broadly categorized into efficient architecture design [32, 38, 42, 60, 67, 73], pruning [18, 20, 40, 41, 48, 52, 65, 79], neural architecture search [7, 58, 59, 66, 74, 78], and quantization [35, 55, 81, 82]. Out of these, the latter has been shown to result in very promising results in both reducing the memory footprint and improving latency and throughput which we will briefly discuss below.

2.2 Quantization of Transformer-based Models

Quantization methods can be broadly categorized based on two factors. The first factor is whether retraining is required or not [23]. Quantization-Aware Training (QAT) requires retraining the model to adapt its weights to help recover accuracy after quantization [2, 35, 55, 82, 85, 86], whereas Post-Training Quantization (PTQ) performs quantization without any retraining [5, 44, 49, 56, 87]. While QAT can often result in better accuracy, it is often infeasible for LLMs due to the expensive cost of retraining the model and/or lack of access to the training data and infrastructure. As such, most works on LLM quantization have focused on PTQ methods [12, 19, 45, 75, 80]. Our work also focuses on the PTQ approach.

Another important factor to classify quantization methods is uniform versus non-uniform quantization [23]. In uniform quantization [13, 19, 31, 35, 45, 46, 55, 82], the range of weights is uniformly splited into equally sized 2^b bins where b is the bit precision. Uniform quantization has gained popularity since it allows for faster computation by performing arithmetic in quantized precision rather than in full precision. However, recent hardware trends indicate that increasing computational speed does not necessarily translate to improved end-to-end latency or throughput [24], 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 is the case for general neural networks as well as LLMs (Fig. 3).

Therefore, in this work, we focus on non-uniform quantization, which allocates quantization bins in a non-uniform manner without any constraints. The high-level idea is to assign more bins to the regions where the weights are concentrated, allowing for a more precise representation of the weights with smaller quantization errors at a given bit precision. While non-uniform quantization does not support fixed-point or 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 computational operations. Among non-uniform quantization methods [11, 33, 81], the most similar work to ours is GOBO [81], which introduces a k-means clustering-based look-up table method for non-uniform quantization. Our work introduces two novel methods as compared to GOBO: (i) sensitivity-aware and (ii) Dense-and-Sparse quantization methodologies, which yield substantial improvements within the k-means-based non-uniform quantization framework.

2.3 Outlier-Aware Quantization

One of the challenges in low-bit Transformer quantization is the presence of outliers [39], which can unnecessarily increase the quantization range. To address this issue, outlier-aware quantization methods have been investigated [3, 12, 68, 69]. Notably, [12] proposes to retain outlier activations as floating-point representations, while [69] suggests migrating outlier factors to subsequent layers without altering the functionality. All of these methods focus on handling outliers in activations. This is not a concern in our work where all activations are maintained as floating point numbers. Instead, our Dense-and-Sparse quantization scheme is a general methodology to address outliers in weight values for low-bit LLM quantization.

Concurrently to our work, the recent study in SpQR [13] also explores a method for extracting outliers in the context of quantization. However, SpQR employs a different sensitivity metric based on the Optimal Brain Surgeon (OBS) framework [27, 28], where the weights are quantized in a way that the output activations of each layer are not perturbed. On the other hand, our approach is based on the Optimal Brain Damage (OBD) framework [14, 15, 43, 55, 76] where the weights are quantized to preserve the final output of the model without perturbation. While both approaches show promise, we have observed that the OBD method yields better quantization performance since it is a direct measure of the end-to-end performance degradation after quantization (Sec. 8.2.4).

More importantly, SpQR requires methods that can introduce high overhead and complexity to achieve lossless quantization. In contrast, SqueezeLLM addresses this issue in two key ways. First, SqueezeLLM does not incorporate grouping. Our Dense-and-Sparse scheme provides a *direct* solution to prevent outliers and sensitive values from negatively impacting quantization performance, eliminating the need for grouping as an indirect and suboptimal solution (Sec. 8.2.3). In contrast, SpQR requires fine-grained grouping (e.g., group size 16) which increases the model size and complicates the quantization pipeline by necessitating the bi-level quantization scheme. Second, the sensitivity-based non-uniform quantization in SqueezeLLM allows for much smaller (e.g., 0.05%) or even zero sparsity levels to achieve accurate quantization. This is crucial for reducing the model size as well as improving inference speed since higher sparsity levels (e.g., 1%) can degrade inference latency (Sec. 5.4). Taken together, by avoiding grouping and utilizing smaller sparsity levels, SqueezeLLM achieves accurate and fast quantization while pushing the average bit precision down to 3-bit, all while employing a simpler quantization pipeline and implementation.

Another concurrent work is AWQ [45] which improves the weight-only quantization scheme for LLMs by introducing scaling factors to reduce the quantization error of a few important weights. However, their approach is also based on the OBS framework, where sensitivity is determined by the magnitude of activations. In Sec. 5, we demonstrate that our method consistently outperforms AWQ in terms of quantization performance across various models and application scenarios.

3 MEMORY WALL

When analyzing neural network inference on a target hardware platform, it is crucial to consider whether the network will be

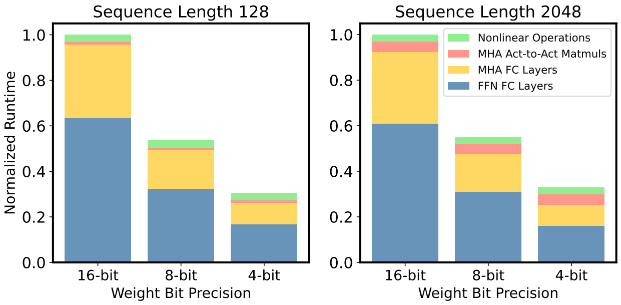

Figure 2: *Normalized runtime for the LLaMA-7B network when reducing the bit precision for the weights with sequence lengths of 128 (left) and 2048 (right). Results were obtained using a roofline-based performance model for an A5000 GPU. For both short and long generations, the primary bottleneck is with the fully connected layers. Additionally, reducing only the precision of the weights (and not the activations) is sufficient to obtain significant latency reductions.*

primarily *compute-bound* or *memory-bound*. Compute-bound inference is limited by the peak computational throughput of the hardware, whereas memory-bound inference is bottlenecked by the rate at which data can be fed into the processing cores from memory. A typical metric used to assess whether a kernel is compute-bound or memory-bound is arithmetic intensity [71], which is the ratio of compute operations to memory operations in that kernel. High arithmetic intensity means that there are very few memory operations but a lot of compute operations. This results in a compute-bound problem that can benefit from faster computation algorithms and hardware with higher peak computational throughput. Conversely, a low arithmetic intensity means that there are very few compute operations relative to the number of memory operations. In this case, the problem is memory-bound and speed-up can be achieved by reducing the memory traffic and not necessarily by reducing compute. This is because, in low arithmetic intensity workloads, the compute units in hardware are often underutilized, as the processors are idle waiting to receive data from memory.

We find that generative workloads for LLM inference (i.e., decoding workloads) exhibit extremely low arithmetic intensity relative to other neural network workloads[2]. For example, the ratio of compute operations to memory operations in matrix multiplications in generative inference is just 2, meaning that for every 2 multiplication/addition, we need to perform a single memory operation [36]. This is because generative inference consists almost entirely of matrix-vector operations, which limits the achievable data reuse as each weight matrix load is only used to process a single vector. This is in contrast with encoding workloads in which the entire input sequence is processed in parallel and a weight matrix load can be amortized across the multiple activation vectors for different tokens in the sequence.

This arithmetic intensity of 2 needs to be contrasted with the ratio of memory operations to compute operations on a typical GPU, which is orders of magnitude higher. For example, an A5000 GPU has peak computational throughput that is 290× higher than its DRAM bandwidth[3]. This means that *memory bandwidth is the bottleneck rather than compute* in generative LLM inference. The disparity between compute and memory bandwidth, in conjunction with the growing memory requirements of deep learning, has been termed the *Memory Wall* problem [24]. Since LLM inference is memory bandwidth-bound, adding more computational overhead will not hurt performance. In particular, this implies that we are free to explore more computationally intensive but memory bandwidth-efficient strategies for representing data [75].

To further illustrate the Memory Wall problem in generative LLMs, we used a simple roofline-based performance modeling approach [36] to study LLaMA-7B's runtime on an A5000 GPU. The results are illustrated in Fig. 2, where we plot the runtime for different bit precisions used for the weight values. Here, we assume that both the activations and the computations are kept at FP16 for all cases. Despite this, we can clearly see that the runtime goes down linearly as we reduce the bit precision. This result is expected as the main bottleneck is memory and not compute.

Moreover, as depicted in Fig. 2, it is evident that the fully connected layers are the primary bottleneck in generative inference, surpassing the impact of activation-to-activation matrix multiplications or nonlinear operations. Significantly, the majority of memory operations in LLaMA are associated with loading the weight matrices in the fully connected layers. Specifically, in the case of LLaMA-7B, weight matrices account for 99.7% of the total memory operations for a sequence length of 128. Even for longer sequence lengths, such as 2048, this value remains significant at 95.8%.

In summary, in generative LLM inference, loading weight matrices into memory is the primary bottleneck, while the cost of dequantization and computation in the FP16 domain is relatively insignificant. Thus, by quantizing just the weights to lower precision, while leaving the activations in full precision, we can attain significant speedup, in addition to the reduction in model size. This observation also aligns with the findings in GPTQ [19] and ZeroQuant [75], which concludes that even though quantized weights are dequantized and computed within the floating-point domain, a significant latency improvement still comes from reduced memory traffic when loading weights. Given this insight, the appropriate strategy is to *minimize the memory size even if it may add overhead to arithmetic operations.*

4 METHODOLOGY

Here we discuss the two main ideas incorporated in SqueezeLLM of (i) sensitivity based non-uniform quantization (Sec. 4.1), and (ii) dense-and-sparse quantization (Sec. 4.2). We then show how this approach can be efficiently implemented in hardware (Sec. 4.3).

4.1 Sensitivity-Based Non-uniform Quantization

In Fig. 3 (Top), we plot an exemplary weight distribution of parameters in LLaMA-7B. The distribution clearly demonstrates a non-uniform pattern, where the majority of weight values are centered around zero, while there are a few outliers. Therefore, the main

[2] To be precise, we limit this discussion to single batch inference with decoder-only models where the arithmetic involves matrix-vector operations. For large batch inference or different model architectures, compute can become important.

[3] The A5000 GPU has peak computational throughput of 222 TeraFLOPs per second and a peak memory bandwidth of 768 GigaBytes per second.

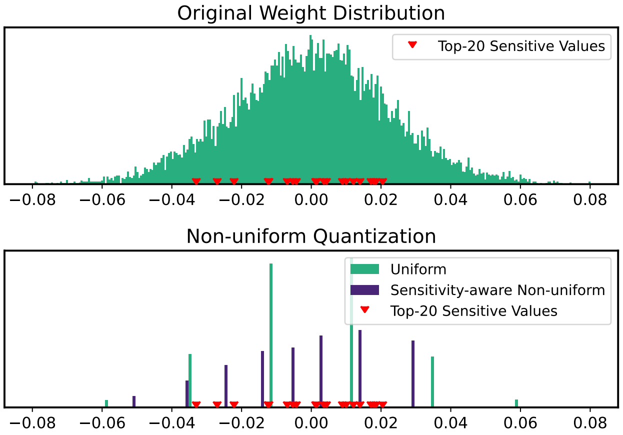

Figure 3: *(Top) The original weight distribution of the non-quantized LLaMA-7B model. The weight values are from a single output channel in the final down projection layer, and the top-20 most sensitive values, as determined by Eq. 4, are marked in red. (Bottom) The weight distributions after 3-bit quantization using two different clustering methods: (non-weighted) k-means clustering and sensitivity-based weighted k-means clustering. Note that in the latter case, the quantized values are more clustered around the sensitive values.*

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 and capabilities. As we discussed before, a widely used approach including the recent LLM quantization works [13, 19, 45] is uniform quantization where the weight range is evenly divided into bins, and each bin is represented by a single integer number.

There are two main issues with this approach for LLM quantization. First, uniformly distributing quantized values is sub-optimal because the weight distribution in neural networks is typically non-uniform, as also illustrated in Fig. 3. Second, while the main advantage of uniform quantization is fast and efficient reduced precision computation, this does not lead to end-to-end latency improvement in memory-bound LLM inference. Given the aforementioned limitations, we have chosen non-uniform quantization [9, 25, 34, 72, 83], which allows for a more flexible allocation of the representative values without any constraints.

It is straightforward that searching for 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., with k=8 for 3-bit quantization). This optimization problem for non-uniform quantization can be formulated as follows:

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

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

Sensitivity-Based K-means Clustering. The objective of quantizing a model is to represent the model weights with low-bit precision while ensuring minimal perturbation in the model output [14]. While quantization introduces errors or perturbations in each layer, we need to minimize the overall perturbation with respect to the *final loss term*, rather than focusing on individual layers, as it provides a more direct measure of the end-to-end performance degradation after quantization [43]. To achieve this, we need to place the k-means centroids closer to 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 which values are more sensitive, we can perform Taylor series expansion to analyze how the model output changes in response to perturbations in the parameters W:

$$\begin{aligned}\mathcal{L}(W_Q) &\simeq \mathcal{L}(W) - g^\top (W - W_Q) + \frac{1}{2}(W - W_Q)^\top H(W - W_Q) \\ &\simeq \mathcal{L}(W) + \frac{1}{2}(W - W_Q)^\top H(W - W_Q),\end{aligned} \tag{2}$$

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

$$Q(w)^* = \arg\min_{Q} (W - W_Q)^\top H(W - W_Q). \tag{3}$$

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 that have 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.

While it is possible to compute Hessian information with a second backprop [77], this approach can be costly for LLMs due to the increased memory requirements. As such, we use an approximation to the Hessian based on the so-called 1-sampled gradients or Fisher information matrix. In particular, the Fisher information can be calculated over a sample dataset D as:

$$H \simeq \mathcal{F} = \frac{1}{|D|}\sum_{d \in D} g_d g_d^\top. \tag{4}$$

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. 3 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:

$$\begin{aligned}Q(w)^* &\simeq \arg\min_{Q} (W - W_Q)^\top \operatorname{diag}(\mathcal{F})(W - W_Q) \\ &= \arg\min_{Q} \sum_{i=1}^{N} \mathcal{F}_{ii}\big(w_i - Q(w_i)\big)^2.\end{aligned} \tag{5}$$

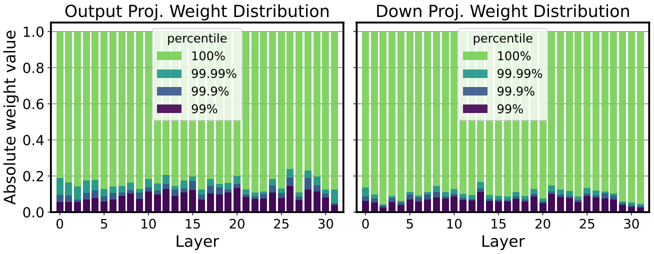

Figure 4: *The distributions of the absolute weight values, normalized by their maximum values, for the output projections in MHA modules and the down projections in FFN modules across different layers in the LLaMA-7B model. Note that the distributions exhibit outlier patterns across all layers, with 99% of the values clustered within ~10% of the entire range.*

An important consequence of using 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 provide an example of a single output channel of LLaMA-7B. The top figure shows the original weight distribution before quantization, where we also denote the top 20 sensitive values in the x-axis (computed based on Fisher information). In the bottom figure, the quantized values assigned by uniform quantization (green bars) are compared to those assigned by the sensitivity-based k-means approach (purple bar). We can clearly see that our sensitivity-based approach achieves a better trade-off by placing centroids near sensitive values, effectively minimizing quantization error.

To ensure the efficacy of this approach, we performed an extensive set of ablation studies with the LLaMA-7B model on the C4 dataset using various quantization methods. The baseline model in FP16 achieves a perplexity of 7.08. With 3-bits uniform quantization using round-to-nearest (RTN) leads to 28.26 perplexity value. Employing non-uniform quantization without considering the sensitivity improves this perplexity to 18.08. Remarkably, the sensitivity-based k-means approach yielded a significant improvement, reducing the perplexity to 7.75, as shown in Fig. 1. More details on these ablation experiments are provided in Sec. 8.2.

4.2 Dense-and-Sparse Quantization

Another challenge when quantizing weights in LLMs into low-bit precision involves outlier values [3, 12, 68, 69]. In Fig 4, we plot the weight value distributions, normalized by their maximum values, for the output projections in MHA modules and the down projections in FFN modules across different layers in the LLaMA 7B model. The figure demonstrates that approximately 99.9% of the weight values are concentrated in a narrow range of less than or around 10% of the entire weight distribution. Naively quantizing the weights with such large range, will significantly degrade performance, especially at low precisions such as 3-bits. However, the observation in Fig. 4 also implies opportunity. 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 result in the sensitivity-based k-means centroids to focus more on the sensitive values rather than a few outliers.

Motivated by this observation, we introduce a method to filter out outliers from the weight matrix W by performing a very simple, yet effective, decomposition of the weight matrix into a dense (D) and sparse matrix (S). The sparse part is calculated by computing the outliers in a given layer, and taking it out of the weight matrix. The remainder is a dense matrix that can be quantized much more effectively thanks to its significantly reduced range of values, which in some cases is more than 10×:

$$W = D + S \ \text{ s.t. } \ D = W[T_{\min} \le w \le T_{\max}] \\ \text{and } \ S = W[w < T_{\min} \ \text{ or } \ w > T_{\max}], \tag{6}$$

where $T_{\min}$, and $T_{\max}$ are thresholds that define outliers and can be calculated based on the percentile of the distribution.

Importantly, note that the overhead of this decomposition is minimal, since the number of outlier values is small. Even in the most aggressive quantization experiments, we did not find it necessary to use more than 0.5% of sparsity. As a consequence, the sparse matrix can be stored efficiently using methods like compressed sparse row (CSR) format. Inference is also straightforward with the Dense-and-Sparse decomposition. The matrix operations are easily parallelizable as $WX = DX + SX$, which will allow us to overlap the two kernels effectively. In particular, the dense part (DX) can be calculated efficiently using the non-uniform quantization scheme and the sparse part (SX) can benefit from sparse libraries [1].

Sensitivity-Based Sparse Matrix. In addition to extracting outliers as a sparse matrix, we found it helpful to also extract a small number of highly sensitive values in the weight matrix to make sure those values are represented exactly without any error. These values can be easily computed based on the Fisher information discussed in Sec. 4.1. This offers two benefits. First, by preserving these sensitive values with FP16 precision, we can minimize their impact on the final output perturbation of the model. Second, we prevent the centroids of Eq. 5 from being skewed toward the sensitive values, thereby improving the quantization errors for the less sensitive weight values as well. We have observed that extracting only 0.05% of these sensitive values across the layers can significantly improve the quantization performance (Sec. 8.2).

In terms of model performance, when applied to 3-bit quantization of LLaMA-7B on C4, filtering out only 0.05% of the sensitive values yields a significant perplexity improvement from 7.75 to 7.67. By further removing 0.4% of outlier values, thus resulting in the overall sparsity level of 0.45%, we further reduce the perplexity from 7.67 to 7.56 (Fig. 1). To the best of our knowledge, this is the best-reported result for the 3-bit quantization of the LLaMA model.

4.3 Dense-and-Sparse Kernel Implementation

A natural question to consider is the impact of both the non-uniform and Dense-and-Sparse decomposition on latency. We found that it is pretty straightforward to implement these methods efficiently. We first implemented 3-bit and 4-bit dense lookup table-based kernels in order to perform matrix-vector multiplication between the compressed matrix and an uncompressed activation vector. These kernels are optimized to load the weight matrix in compressed format and dequantize it piece-by-piece to minimize memory bandwidth utilization. The compressed matrices store 3-bit or 4-bit indices, which correspond to entries in lookup tables containing FP16 values associated with the bins obtained from non-uniform quantization. After dequantizing using the lookup table, all arithmetic is performed in full precision.

In order to process our Dense-and-Sparse representation efficiently, we also develop CUDA kernels for Dense-and-Sparse matrix-vector multiplication. We have employed custom sparse kernels to load a matrix in CSR format as well as a dense activation vector, inspired by the implementations in [17]. Additionally, we implemented a hybrid kernel to handle cases where there was a skewed nonzero distribution in the weight matrix where a small number of channels in the weight matrix had a large number of nonzeros (see Fig. 7 in Appendix 8.1). We processed this small number of rows using a dense matrix-vector operation in parallel with our sparse matrix-vector operation to reduce the latency of the sparse matrix-dense vector operation. In our experiments, we isolated out the 10 rows with the greatest number of nonzeros in each weight matrix to be processed in parallel. The dense non-uniform kernel and hybrid sparse CSR kernels are launched in one call to avoid overhead from summing the output vectors from these separate operations.

5 EVALUATIONS

5.1 Experiment Setup

Models and Datasets. We have conducted comprehensive evaluations of SqueezeLLM on various tasks using the LLaMA [63] and Vicuna (v1.1) [8] models. First, in the language modeling evaluation, we apply SqueezeLLM to the LLaMA-7B, 13B, and 30B models and measure the perplexity of the quantized models on the C4 [51] and WikiText2 [47] datasets with a chunk size of 2048. We also evaluate the domain-specific knowledge and problem-solving ability through zero-shot MMLU [29] using the instruction-tuned Vicuna-7B and 13B models. We used the Language Model Evaluation Harness to run zero-shot evaluation across all tasks [21]. Finally, we evaluate the instruction following ability following the methodology presented in [8]. 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 several PTQ methodologies for LLMs including the round-to-nearest (RTN) method as well as the state-of-the-art GPTQ [19] and AWQ [45]. To ensure a fair performance comparison, we use GPTQ with activation ordering throughout all experiments, 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 3 different sparsity levels: 0% (dense-only), 0.05% by excluding sensitive values, and 0.45% by additionally excluding outlier values (based on their magnitude). For measuring sensitivity, we use 100 random samples from the C4 training set for the LLaMA models, and the Vicuna training set for the Vicuna models. While grouping can also be incorporated with our method, we found it sub-optimal as compared to extracting sensitive and outlier values with sparsity (see Appendix 8.2.3).

Deployment and Profiling. To reduce static memory consumption, we employ a compressed memory format to store the quantized models. Additionally, we integrate our custom kernels into PyTorch to enable end-to-end inference directly on the compressed format. In order to evaluate the performance of SqueezeLLM, we measure the latency and peak memory usage for generating 128 and 1024 tokens on a single A6000 machine using 3-bit and 4-bit quantization with different sparsity levels, and compare against both full 16-bit precision inference and GPTQ. As an official implementation of GPTQ (in particular, the grouped version) from [19] is not available, we implement an optimized kernel for single-batch inference based on the most active open-source codebase[4].

5.2 Main Results

Table 1 shows quantization results for LLaMA-7B, 13B, and 30B along with comparison with round-to-nearest (RTN), and state-of-the-art PTQ methods including GPTQ [19] and AWQ [45]. The models are grouped based on their average bitwidth (i.e., model size) for a better comparison of size-perplexity trade-offs. Figure 5 also illustrates the trade-off between model size and perplexity specifically for the C4 dataset using 3-bit quantization. Below we use LLaMA-7B as the main example for the discussions for the impact of dense-only quantization and Dense-and-Sparse quantization, and subsequently discuss how these trends extend to larger models.

Dense-only Quantization. In the first group of Tab. 1 (a), we compare dense-only SqueezeLLM with 0% sparsity level and GPTQ without grouping applied to LLaMA-7B. 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 and 1.22 points on C4 and WikiText2 with a 5.29× compression rate. This is only 0.67 and 0.55 points off from the FP16 baseline. These results demonstrate the effectiveness of the sensitivity-based non-uniform method for ultra-low-bit quantization.

Dense-and-Sparse Quantization. By leveraging the Dense-and-Sparse quantization scheme, we achieve a further reduction in the perplexity gap between the FP16 baseline and quantized models, as shown in the second and third groups of Tab. 1 (a). This improvement is particularly significant in the case of 3-bit quantization, where excluding just 0.05% and 0.45% of the values yields around 0.1 and 0.2 perplexity improvement, respectively. With the Dense-and-Sparse decomposition, we achieve nearly lossless compression with less than 0.1 and 0.5 perplexity deviation from the FP16 baseline for 4-bit and 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 sparsity levels of 0.05% and 0.45% consistently outperforms both GPTQ and AWQ with group sizes of 256 and 128, respectively, in all scenarios while maintaining

[4] https://github.com/qwopqwop200/GPTQ-for-LLaMa

Table 1: *Perplexity Comparison of LLaMA-7B, 13B, and 30B quantized into 4-bit and 3-bit using different methods including round-to-nearest (RTN), GPTQ [19] and AWQ [45] on C4 and WikiText-2. The average bitwidths and compression rates are also included for comparison. 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. See Fig. 5 for a visual representation.*

(a) *LLaMA-7B*

Bitwidth	4-bit			3-bit		
Method	**Avg. Bits**	**PPL** (↓)		**Avg. Bits**	**PPL** (↓)	
	(comp. rate)	C4	Wiki	(comp. rate)	C4	Wiki
Baseline	16	7.08	5.68	16	7.08	5.68
RTN	4 (4.00×)	7.73	6.29	3 (5.33×)	28.26	25.61
GPTQ	4 (4.00×)	7.43	5.94	3 (5.33×)	9.55	7.55
SqueezeLLM	4.05 (3.95×)	**7.21**	**5.79**	3.02 (5.29×)	**7.75**	**6.32**
GPTQ (g256)	4.12 (3.89×)	7.25	5.81	3.12 (5.13×)	8.09	6.43
AWQ (g256)	4.12 (3.89×)	7.29	5.87	3.12 (5.13×)	8.04	6.51
SqueezeLLM (0.05%)	4.07 (3.93×)	**7.20**	**5.79**	3.05 (5.25×)	**7.67**	**6.20**
GPTQ (g128)	4.24 (3.77×)	7.21	5.78	3.24 (4.93×)	7.89	6.27
AWQ (g128)	4.24 (3.77×)	7.22	5.82	3.24 (4.93×)	7.90	6.44
SqueezeLLM (0.45%)	4.27 (3.75×)	**7.18**	**5.77**	3.24 (4.93×)	**7.56**	**6.13**

(b) *LLaMA-13B*

Bitwidth	4-bit			3-bit		
Method	**Avg. Bits**	**PPL** (↓)		**Avg. Bits**	**PPL** (↓)	
	(comp. rate)	C4	Wiki	(comp. rate)	C4	Wiki
Baseline	16	6.61	5.09	16	6.61	5.09
RTN	4 (4.00×)	6.99	5.53	3 (5.33×)	13.24	11.78
GPTQ	4 (4.00×)	6.84	5.29	3 (5.33×)	8.22	6.22
SqueezeLLM	4.04 (3.96×)	**6.71**	**5.18**	3.02 (5.30×)	**7.08**	**5.60**
GPTQ (g256)	4.12 (3.88×)	6.74	5.20	3.12 (5.12×)	7.42	5.68
AWQ (g256)	4.12 (3.88×)	6.72	5.20	3.12 (5.12×)	7.18	5.63
SqueezeLLM (0.05%)	4.07 (3.94×)	**6.69**	**5.17**	3.04 (5.26×)	**7.01**	**5.51**
GPTQ (g128)	4.25 (3.77×)	6.70	5.17	3.25 (4.92×)	7.12	5.47
AWQ (g128)	4.25 (3.77×)	6.70	5.21	3.25 (4.92×)	7.08	5.52
SqueezeLLM (0.45%)	4.26 (3.76×)	**6.68**	**5.17**	3.24 (4.94×)	**6.92**	**5.45**

(c) *LLaMA-30B*

Bitwidth	4-bit			3-bit		
Method	**Avg. Bits**	**PPL** (↓)		**Avg. Bits**	**PPL** (↓)	
	(comp. rate)	C4	Wiki	(comp. rate)	C4	Wiki
Baseline	16	5.98	4.10	16	5.98	4.10
RTN	4 (4.00×)	6.33	4.54	3 (5.33×)	28.53	14.89
GPTQ	4 (4.00×)	6.20	4.43	3 (5.33×)	7.31	5.76
SqueezeLLM	4.03 (3.97×)	**6.06**	**4.22**	3.02 (5.31×)	**6.37**	**4.66**
GPTQ (g256)	4.12 (3.88×)	6.08	4.26	3.12 (5.12×)	6.58	4.87
AWQ (g256)	4.12 (3.88×)	6.07	4.24	3.12 (5.12×)	6.45	4.71
SqueezeLLM (0.05%)	4.06 (3.94×)	**6.05**	**4.20**	3.04 (5.26×)	**6.31**	**4.56**
GPTQ (g128)	4.25 (3.77×)	6.07	4.24	3.25 (4.92×)	6.47	4.83
AWQ (g128)	4.25 (3.77×)	6.05	4.21	3.25 (4.92×)	6.38	4.63
SqueezeLLM (0.45%)	4.25 (3.77×)	**6.04**	**4.18**	3.25 (4.92×)	**6.23**	**4.44**

Table 2: *Comparison of PTQ methods on zero-shot MMLU [29] accuracy when applied to the Vicuna-7B and 13B models.*

Method	**Avg. Bit**	**Vicuna-7B** (↑)	**Vicuna-13B** (↑)
Baseline	16	39.1%	41.2%
AWQ (g128)	4.25	38.0%	40.4%
SqueezeLLM	4.05	38.8%	39.2%
SqueezeLLM (0.45%)	4.26	**39.4%**	**41.0%**
AWQ (g128)	3.25	36.5%	37.6%
SqueezeLLM	3.02	36.0%	37.2%
SqueezeLLM (0.45%)	3.24	**37.7%**	**39.4%**

smaller or comparable model sizes. This is more pronounced for 3-bit quantization, where SqueezeLLM with a 0.45% sparsity level outperforms both GPTQ and AWQ with a group size of 128 by up to more than 0.3 perplexity points.

Results on Larger Models. In Tab. 1 (b, c), we observe that the trend observed in LLaMA-7B extends to LLaMA-13B and 30B models. As also illustrated in Fig. 5, SqueezeLLM consistently outperforms state-of-the-art PTQ methods across all model sizes and quantization bit widths. Notably, even the dense-only version of SqueezeLLM achieves perplexity comparable to the grouped GPTQ and AWQ. By incorporating sparsity, we achieve further perplexity improvements, reducing the gap from the FP16 baseline to less than 0.1 and 0.4 perplexity points for 4-bit and 3-bit quantization, respectively. 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. To the best of our knowledge, this is the best results reported so far for quantizing these models.

Further ablation studies on our design choices, including sensitivity metrics, sparsity levels, and grouping, are provided in Appendix 8.2.

5.3 Quantization of Instruction Following Models

Instruction tuning has emerged as a common method for improving the model's ability to respond to user commands [8, 22, 61]. We explore the quantization of instruction-following models in order to demonstrate the benefits of our approach in terms of accuracy preservation by applying SqueezeLLM to Vicuna-7B and 13B [8], and evaluate the performance with the following approaches.

Zero-shot MMLU Evaluation. We first compare the baseline and quantized model on the zero-shot multitask problem-solving benchmark of MMLU [29]. The weighted accuracy across all tasks is provided in Tab. 2 for the baseline Vicuna 7B and 13B models, as well as models quantized using AWQ [45] and SqueezeLLM. As we can see, SqueezeLLM achieves higher accuracy for both Vicuna-7B and 13B as compared to the AWQ method and also preserve the accuracy of the FP16 baseline model with 4-bit quantization. Furthermore, it is noteworthy that the 4-bit quantized version of Vicuna-13B using SqueezeLLM has 2× smaller memory footprint than the 7B baseline model in FP16, while still achieving a 2% higher accuracy. Thus, SqueezeLLM not only improves performance but also reduces memory requirements.

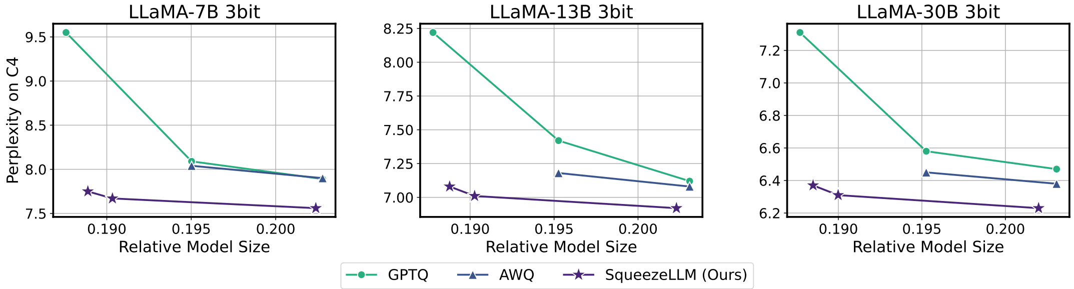

Figure 5: *Perplexity comparison of the LLaMA-7B, 13B, and 30B models quantized into and 3-bit using GPTQ [19], AWQ [45], and SqueezeLLM (ours), evaluated on the C4 benchmarks. The x-axes are the relative model sizes with respect to the model size in FP16. Different trade-offs between model size and perplexity are achieved by adjusting the group size for GPTQ and AWQ and the sparsity level for our method. 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. Note that all GPTQ results are with activation reordering.*

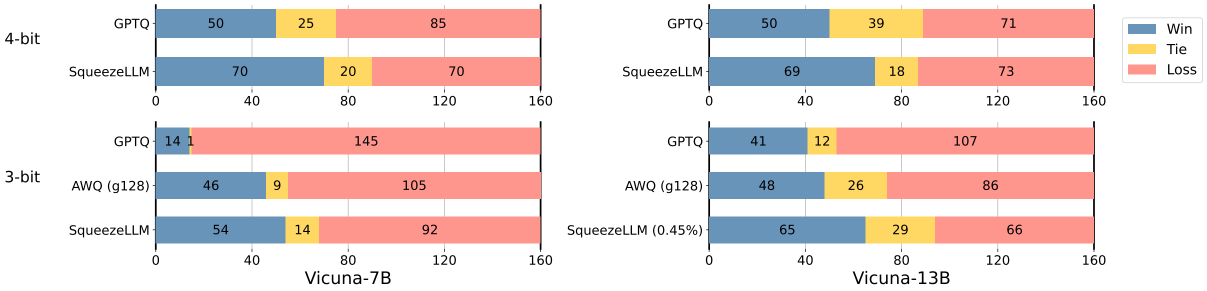

Figure 6: *Comparison of PTQ methods when applied to the instruction following Vicuna models. Wins / Ties / Losses represent the number of times that the quantized network won / tied / lost against the baseline FP16 Vicuna network. This evaluation was performed using the methodology from Vicuna [8] using v1.1 model weights. Both GPTQ and AWQ results are our reproduction based on the released code base.*

Instruction-Following Ability. Another approach for evaluating instruction-following ability is to feed the generated results to GPT-4 and ask it to rank the results, which is the method used by [8]. It is important to note, that the order of the models fed to GPT-4 can create a bias, as it has been observed that GPT-4 slightly favors the first model [8]. To address that, we repeat all comparisons twice by swapping the orders. The results are shown in Fig. 6. SqueezeLLM without sparsity achieves near-perfect performance (i.e., 50/50 split) with 4-bit quantization for both Vicuna-7B and 13B models, outperforming GPTQ with the same model size. In the case of 3-bit quantization, SqueezeLLM outperforms both GPTQ and the state-of-the-art AWQ method with a group size of 128 even without incorporating sparsity. Furthermore, the addition of a 0.45% sparsity level proves highly effective with the Vicuna-13B model, achieving a near-perfect 50/50 split for 3-bit quantization as well.

5.4 Hardware Deployment and Profiling

While grouping with permutation is an effective way to confine the quantization range, our Dense-and-Sparse scheme can achieve higher accuracy with simpler kernels. 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 (4th row) shows up to 2.3× 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 considerable perplexity gains it enables.

Additionally, when incorporating sparsity, we still observed latency gains relative to the FP16 baseline. As shown in Tab. 3, keeping 0.05% of sensitive values in FP16 only adds approximately 20% latency overhead across different model sizes, while still providing up to 1.9× speed up compared to the baseline. Keeping 0.45% of parameters in FP16 only adds 40-45% latency overhead relative to the dense-only implementation, while still resulting in 1.7× speed up compared to the FP16 baseline. In contrast, when accounting for permutation, the GPTQ runtime is degraded heavily. 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 heavily on coalesced memory accesses in order to optimally utilize memory bandwidth. This shows how our Dense-and-Sparse quantization methodology allows for both higher accuracy as well as better performance relative to GPTQ.

Table 3: *Hardware profiling of latency and memory usage for LLaMA 7B, 13B, and 30B quantized into 3-bit when generating 128 tokens on an A6000 GPU. The first row is the non-quantized FP16 baseline, and the second and third rows are non-grouped and grouped GPTQ, respectively. Note that all GPTQ results are with activation ordering. Rows four, five, and six show the performance of SqueezeLLM with different sparsity levels, with the fourth row indicating the dense-only SqueezeLLM. The same analysis with sequence length 1024 is provided in Tab. 6.*

Method	**Latency (Seconds)**			**Peak Memory (GB)**		
	LLaMA-7B	LLaMA-13B	LLaMA-30B	LLaMA-7B	LLaMA-13B	LLaMA-30B
FP16	3.2	5.6	OOM	12.7	24.6	OOM
GPTQ	1.4	2.1	4.1	2.9	5.3	12.4
GPTQ (g128)	13.7	24.2	61.9	3.0	5.6	12.9
SqueezeLLM	1.5	2.4	4.9	2.9	5.4	12.5
SqueezeLLM (0.05%)	1.8	2.9	5.9	3.0	5.5	12.7
SqueezeLLM (0.45%)	2.2	3.4	7.2	3.2	5.9	13.8

6 CONCLUSION AND LIMITATIONS

We have presented the SqueezeLLM quantization framework which attempts to address the Memory Wall problem associated with generative LLM inference. Our hardware profiling results clearly demonstrate that the main bottleneck for decoder model inference is memory bandwidth and not compute. Based on this observation, SqueezeLLM incorporates two novel ideas which allow ultra-low precision quantization of LLMs with negligible degradation in generation performance. The first idea is the sensitivity-based non-uniform quantization method; and the second idea is a simple, yet effective, Dense-and-Sparse decomposition designed to deal with outlier values that negatively impact quantization. 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. Additionally, we demonstrate that, under the same memory constraints, our quantized models can achieve performance improvements compared to full precision models. 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 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.

7 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. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

REFERENCES

[1] https://developer.nvidia.com/cusparse.

[2] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. BinaryBERT: Pushing the limit of BERT quantization. *arXiv preprint arXiv:2012.15701*, 2020.

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

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. ZeroQ: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[6] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[7] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. AdaBERT: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020.

[8] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[9] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Towards the limit of network quantization. *arXiv preprint arXiv:1612.01543*, 2016.

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PALM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[11] Insoo Chung, Byeongwook Kim, Yoonjung Choi, Se Jung Kwon, Yongkweon Jeon, Baeseong Park, Sangha Kim, and Dongsoo Lee. Extremely low bit transformer quantization for on-device neural machine translation. *arXiv preprint arXiv:2009.07453*, 2020.

[12] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*.

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

[14] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian Aware trace-Weighted Quantization of neural networks. *NeurIPS'19 workshop on Beyond First-Order Optimization Methods in Machine Learning.*, 2019.

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

[16] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. GLAM: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

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

[18] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

[19] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[20] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

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

[22] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. Koala: A dialogue model for academic research. Blog post, April 2023.

[23] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[24] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. AI and Memory Wall. *https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8*, 2021.

[25] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[26] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

[27] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[28] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

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

[30] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[31] Yafeng Huang, Huanrui Yang, Zhen Dong, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Yuan Du, Shanghang Zhang, and Kurt Keutzer. Output sensitivity-aware detr quantization. 2023.

[32] Forrest N Iandola, Albert E Shaw, Ravi Krishna, and Kurt W Keutzer. SqueezeBERT: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316*, 2020.

[33] Yongkweon Jeon, Chungman Lee, Eulrang Cho, and Yeonju Ro. 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*, pages 12329–12338, 2022.

[34] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[35] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-BERT: Integer-only bert quantization. *arXiv preprint arXiv:2101.01321*, 2021.

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

[37] Sehoon Kim, Karttikeya Mangalam, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Big little transformer decoder. *arXiv preprint arXiv:2302.07863*, 2023.

[38] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

[39] Olga Kovaleva, Saurabh Kulshreshtha, Anna Rogers, and Anna Rumshisky. Bert busters: Outlier dimensions that disrupt transformers. *arXiv preprint arXiv:2105.06990*, 2021.

[40] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. The optimal BERT surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

[41] Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. *arXiv preprint arXiv:2204.09656*, 2022.

[42] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[43] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[44] Xiuyu Li, Long Lian, Yijiang Liu, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. *arXiv preprint arXiv:2302.04304*, 2023.

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

[46] Yijiang Liu, Huanrui Yang, Zhen Dong, Kurt Keutzer, Li Du, and Shanghang Zhang. NoisyQuant: Noisy bias-enhanced post-training activation quantization for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20321–20330, 2023.

[47] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[48] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*, 2019.

[49] Sangyun Oh, Hyeonuk Sim, Jounghyun Kim, and Jongeun Lee. 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*, pages 658–673. Springer, 2022.

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

[51] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[52] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020.

[53] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[54] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *arXiv preprint arXiv:2207.07061*, 2022.

[55] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-BERT: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 2020.

[56] Gil Shomron, Freddy Gabbay, Samer Kurzum, and Uri Weiser. Post-training sparsity-aware quantization. *Advances in Neural Information Processing Systems*, 34:17737–17748, 2021.

[57] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[58] David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

[59] David R So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668*, 2021.

[60] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

[61] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[62] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[63] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

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

[65] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest

can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[66] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

[67] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

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

[69] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *arXiv preprint arXiv:2209.13325*, 2022.

[70] Sean Welleck, Kianté Brantley, Hal Daumé Iii, and Kyunghyun Cho. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pages 6716–6726. PMLR, 2019.

[71] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[72] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[73] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020.

[74] Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. NAS-BERT: task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1933–1943, 2021.

[75] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. *arXiv preprint arXiv:2206.01861*, 2022.

[76] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W Mahoney, and Kurt Keutzer. HAWQV3: Dyadic neural network quantization. *arXiv preprint arXiv:2011.10680*, 2020.

[77] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. *arXiv preprint arXiv:1912.07145*, 2019.

[78] Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. AutotinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models. *arXiv preprint arXiv:2107.13686*, 2021.

[79] Shixing Yu, Zhewei Yao, Amir Gholami, Zhen Dong, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. Hessian-aware pruning and optimal neural implant. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3880–3891, 2022.

[80] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. RPTQ: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023.

[81] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. GOBO: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824. IEEE, 2020.

[82] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8BERT: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

[83] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.

[84] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[85] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. TernaryBERT: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

[86] Yifan Zhang, Zhen Dong, Huanrui Yang, Ming Lu, Cheng-Ching Tseng, Yandong Guo, Kurt Keutzer, Li Du, and Shanghang Zhang. Qd-bev: Quantization-aware view-guided distillation for multi-view 3d object detection. 2023.

[87] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR, 2019.

8 APPENDIX

8.1 Data Skew in Per-channel Sparsity Pattern

Fig. 7 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. 4, we observed over 100% added runtime overhead when employing a standard CSR-based kernel. However, when processing the top 10 rows separately using a dense matrix-vector operation, we were able to drastically reduce the runtime overhead to 20% with sensitive values and 40-45% with both sensitive values and outliers.

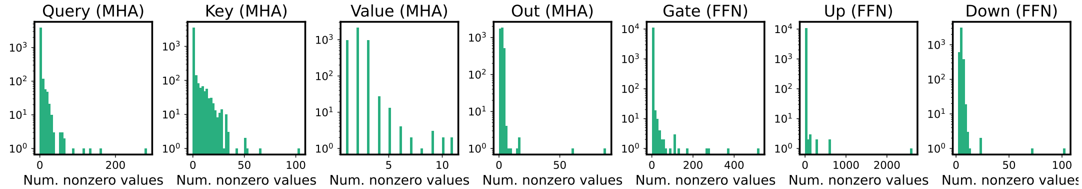

Figure 7: *Histograms of the number of non-zero entries per output channel in 7 different linear layers in the first LLaMA-7B block. The histograms reveal the presence of a few channels that contain significantly more non-zero entries than others, highlighting the skew in the sparsity patterns across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

Table 4: *Hardware profiling of latency and memory usage for LLaMA 7B, 13B, and 30B quantized into 3-bit when generating 128 tokens on an A6000 GPU. The first two rows show the performance of SqueezeLLM with different sparsity levels using a standard kernel for processing a CSR matrix. The last two rows show the performance of SqueezeLLM with different sparsity levels using a hybrid kernel for processing a CSR matrix that isolates out the top 10 channels with the most nonzeros and processing these separately.*

Sparse Kernel	Method	Latency (Seconds)			Peak Memory (GB)		
		LLaMA-7B	LLaMA-13B	LLaMA-30B	LLaMA-7B	LLaMA-13B	LLaMA-30B
	SqueezeLLM	1.5	2.4	4.9	2.9	5.4	12.5
Standard	SqueezeLLM (0.05%)	3.4	5.6	12.0	3.0	5.4	12.6
	SqueezeLLM (0.45%)	3.9	6.2	12.5	3.2	5.8	13.7
Hybrid	SqueezeLLM (0.05%)	1.8	2.9	5.9	3.0	5.5	12.7
	SqueezeLLM (0.45%)	2.2	3.4	7.2	3.2	5.9	13.8

8.2 Ablation Studies

8.2.1 ***Sensitivity-Based Quantization.*** In our ablation study, we investigate the impact of sensitivity-aware weighted clustering on the performance of non-uniform quantization. In Tab. 5, we compared the performance of sensitivity-aware 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-aware clustering is critical in reducing the perplexity to 7.75. This improvement is consistent across all sparsity levels.

Table 5: *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**

8.2.2 ***Impact of Sparsity Levels on SqueezeLLM.*** In Fig. 8 (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 8 (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.

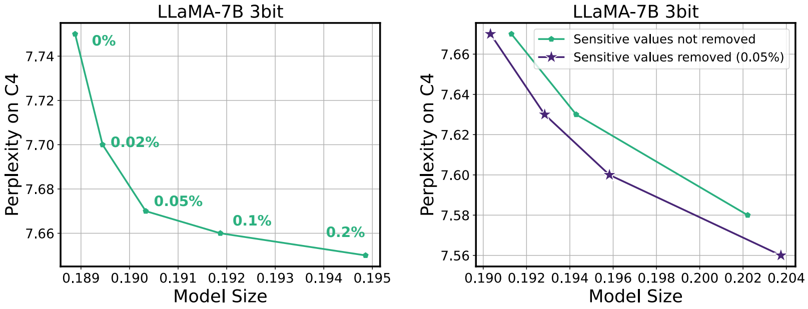

Figure 8: *(Left) Model size (normalized by the size of the FP16 model) and perplexity trade-off with different percentages of sensitive values included in the sparse matrix. Here, no outlier values are included in the sparse matrix. (Right) Comparison of the performance when the sensitive values are 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.*

8.2.3 ***Impact of Grouping on SqueezeLLM.*** In Fig. 10, 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. In addition, 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.

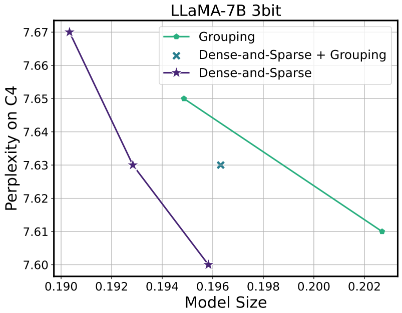

Figure 9: *Model size (normalized by the size of the FP16 model) and perplexity trade-offs of grouping and the Dense-and-Sparse decomposition on 3-bit quantization of the LLaMA-7B model. Here, we compare SqueezeLLM with (i) grouping using group sizes of 1024 and 512 (green), (ii) a hybrid approach that 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 achieves better size-perplexity trade-offs than both grouping and the hybrid approach.*

8.2.4 *Comparison of the OBD Framework versus the OBS Framework for Non-uniform Quantization*.

While our method adopts the Optimal Brain Damage (OBD) framework to minimize the perturbation of the final output of the model during quantization, it is worth noting that the Optimal Brain Surgeon (OBS) framework can also be considered as an alternative. Most existing solutions for LLM quantization including GPTQ [19], AWQ [45], and SpQR [13] have utilized the OBS framework, which aims to minimize the perturbation of output activations in individual layers. In this ablation study, we demonstrate that the OBD framework is superior to the OBS framework.

Under the OBD framework, 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 [45].

In Fig. 8.2.4, we compare the perplexity on the C4 dataset for 3-bit quantization of the LLaMA-7B model using the OBS framework versus the OBD framework. Across all sparsity levels obtained by adjusting the number of outliers being extracted, SqueezeLLM based on the OBD framework outperforms the alternative of using the OBS framework by a large margin of up to around 0.3 perplexity points.

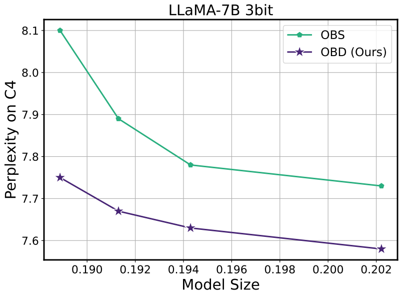

Figure 10: *Model size (normalized by the size of the FP16 model) and perplexity trade-offs for 3-bit quantization of the LLaMA-7B model using the Optimal Brain Surgeon (OBS) framework versus the Optimal Brain Damage (OBD) framework for determining the non-uniform quantization configuration. The trade-off 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.*

8.3 Additional Hardware Profiling Results

In Tab. 6, 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 6: *Hardware profiling of latency and memory usage for LLaMA 7B, 13B, and 30B quantized into 3-bit when generating 1024 tokens on an A6000 GPU. The first row is the non-quantized FP16 baseline, and the second and third rows are non-grouped and grouped GPTQ, respectively. Note that all GPTQ results are with activation ordering. Rows four, five, and six show the performance of SqueezeLLM with different sparsity levels, with the fourth row indicating the dense-only SqueezeLLM.*

Method	**Latency (Seconds)**			**Peak Memory (GB)**		
	LLaMA-7B	LLaMA-13B	LLaMA-30B	LLaMA-7B	LLaMA-13B	LLaMA-30B
FP16	26.5	47.0	OOM	13.1	25.2	OOM
GPT-Q	12.6	19.0	36.8	3.3	6.0	13.8
GPT-Q (g128)	110.7	176.1	500.8	3.4	6.2	14.3
SqueezeLLM	13.6	21.2	42.6	3.4	6.1	13.9
SqueezeLLM (0.05%)	16.1	24.9	49.3	3.4	6.2	14.1
SqueezeLLM (0.45%)	19.1	29.0	58.9	3.6	6.6	15.1