Summary of GPTQ, Inference Optimization, and LLM.int8()

Yuxuan Jiang (jyuxuan), Finn Roblin (finnrob), Marwa Houalla (mhoualla)

GPTQ

Problem and Motivation

We would like to use quantization to decrease the memory footprint and compute time of ML models. Quantization compresses each weight to a smaller data type. For instance the LLM.int8 quantization method converts 32 bit floats to 8 bits. Doing this represents a 4x decrease in memory use and possibly a 16x decrease in FLOPS required for a matrix multiplication.

A major goal of quantization methods is to make models as small as possible without a significant performance degradation. The main contribution of GPTQ is a model weight quantization method that is fast to compute, and shrinks floats to 3-4 bits without destroying model performance. They achieve 3-4x latency speedup with less perplexity increase than other quantization methods.

Related Works

GPTQ is inspired by several papers that describe pruning methods for neural networks. Optimal Brain Damage (LeCun et al. 1989) describes a method for pruning NN parameters where the deleted weights change the loss function the least. Its method is to use a Taylor series approximation of the change in training loss. It makes the simplifying assumption that the first and third order approximations in the series are unnecessary.

Optimal Brain Surgeon (Hassibi et al., 1993) adjusts the OBD algorithm by dropping some of its assumptions. In particular, it does not assume that weights are independent of each other. It also shows a general solution for minimizing the change in loss. The objective function is:

$$\min_{\mathbf{\Delta w},q} \frac{1}{2}\mathbf{\Delta w}^{\mathbf{T}}\mathbf{H}\mathbf{\Delta w} \quad s.t. \;\; \mathbf{e_q^T}\cdot\mathbf{\Delta w} + w_q = 0$$

$$\downarrow$$

$$\mathbf{\Delta w} = -\frac{w_q}{[\mathbf{H}^{-1}]_{qq}}\mathbf{H}^{-1}\cdot\mathbf{e_q} \quad and \quad L = \frac{1}{2}\frac{w_q^2}{[\mathbf{H}^{-1}]_{qq}}$$

A third paper, Optimal Brain Quantization (Frantar et al., 2022), adapts the above solution to work with the quantization paradigm. Instead of finding the best weights to delete/zero, OBQ

finds the best weights to quantize. It quantizes the row with the weights that minimize the error:

$$w_q = \underset{w_q}{\text{argmin}} \frac{(\text{quant}(w_q) - w_q)^2}{[\mathbf{H}_F^{-1}]_{qq}}, \quad \boldsymbol{\delta}_F = -\frac{w_q - \text{quant}(w_q)}{[\mathbf{H}_F^{-1}]_{qq}} \cdot (\mathbf{H}_F^{-1})_{:,q}.$$

Where the inverse Hessian is calculated as:

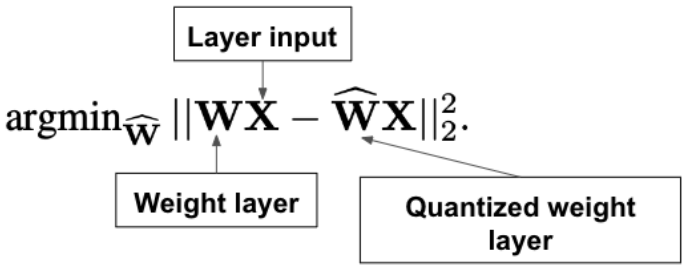
$$\mathbf{H}_{-q}^{-1} = \left(\mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}_{:,q}^{-1} \mathbf{H}_{q,:}^{-1}\right)_{-p}.$$

(the -q notation refers to deleting the qth row/col.)

OBTQ builds directly on OBQ.

Solution Overview

OBTQ builds directly on OBQ's objective function:



OBTQ makes three improvements to OBQ:

1. Quantize each row arbitrarily, instead of greedily.
    a. OBQ quantizes a row at a time, and it greedily picks the least costly (in terms of the above objective function) row to quantize. However, the authors found that picking some arbitrary fixed order to quantize rows works just as well.
    b. This allows us to update the inverse Hessian fewer times (once per column as opposed to once per weight).
2. Batch updates to the Hessian.
    a. The inverse Hessian computation has a low compute-to-memory access ratio, since all rows of the Hessian need to be updated.
    b. In OBTQ the algorithm works on smaller 128x128 blocks of the matrix at a time and in parallel, and flushes these updates to the global matrix once each update is finished.
3. Cholesky Reformulation
    a. There are numerical stability problems with the current implementation. In particular, the Hessian becomes indefinite (meaning eigenvalues neither all

positive nor all negative, which slows down convergence). Since there are so many computations, small floating point inaccuracies combine over time to degrade accuracy.

b. To account for these instability issues, the authors take the Cholesky decomposition of the inverse Hessian, which is a standard trick in numerical linear algebra.

The main contribution of this paper is to implement the OBQ findings into a usable quantization framework that can quantize floats down to 3-4 bits without large performance dropoffs.

Results

GPTQ can quickly quantize models down to 3-4 bits without major performance degradation.

GPTQ can quickly quantize large models. A different quantization framework, ZeroQuant-LKD, takes ~3 hours vs 20 min to quantize OPT-13B.

OPT	13B	30B	66B	175B
Runtime	20.9m	44.9m	1.6h	4.2h
BLOOM	1.7B	3B	7.1B	176B
Runtime	2.9m	5.2m	10.0m	3.8h

Table 2: OPTQ runtime for full quantization of the 4 largest OPT and BLOOM models.

A key metric to assess performance degradation due to quantization is to see if model perplexity increases. The “round-to-nearest” quantization strategy has higher (worse) perplexity than OPTQ over all benchmarks, but especially when quantizing to 3 bit floats.

OPT	Bits	125M	350M	1.3B	2.7B	6.7B	13B	30B	66B	175B
full	16	27.65	22.00	14.63	12.47	10.86	10.13	9.56	9.34	8.34
RTN	4	37.28	25.94	48.17	16.92	12.10	11.32	10.98	110	10.54
OPTQ	4	**31.12**	**24.24**	**15.47**	**12.87**	**11.39**	**10.31**	**9.63**	**9.55**	**8.37**
RTN	3	1.3e3	64.57	1.3e4	1.6e4	5.8e3	3.4e3	1.6e3	6.1e3	7.3e3
OPTQ	3	**53.85**	**33.79**	**20.97**	**16.88**	**14.86**	**11.61**	**10.27**	**14.16**	**8.68**

Table 3: OPT perplexity results on WikiText2.

BLOOM	Bits	560M	1.1B	1.7B	3B	7.1B	176B
full	16	22.42	17.69	15.39	13.48	11.37	8.11
RTN	4	25.90	22.00	16.97	14.76	12.10	8.37
OPTQ	4	**24.03**	**19.05**	**16.48**	**14.20**	**11.73**	**8.21**
RTN	3	57.08	50.19	63.59	39.36	17.38	571
OPTQ	3	**32.31**	**25.08**	**21.11**	**17.40**	**13.47**	**8.64**

Table 4: BLOOM perplexity results for WikiText2.

Finally, the inference latency decreases upon quantization with OPTQ. There is a 3-4x speedup in inference, allowing the number of GPUs to be reduced to maintain the same inference latency.

GPU	FP16	3bit	Speedup	GPU reduction
A6000 – 48GB	589ms	130ms	$4.53\times$	$8 \rightarrow 2$
A100 – 80GB	230ms	71ms	$3.24\times$	$5 \rightarrow 1$

Table 6: Average per-token latency (batch size 1) when generating sequences of length 128.

Limitations

One application of quantization is to allow GPU-poor users to run large parameter models. GPTQ is implemented in CUDA so it cannot easily be run on CPU (nor would it really make sense for most of the lin alg operations to be on CPU).

Future Research Directions

The inference speedups are from using less memory, not due to computational reductions. In particular, HW doesn't support mixed precision operations, FP16 x INT4, so the actual multiplications are the same speed or slower (due to casting back to FP16).

Quantization of activation functions (not just model weights) is another area of research (addressed in the last paper/blog post of this presentation).

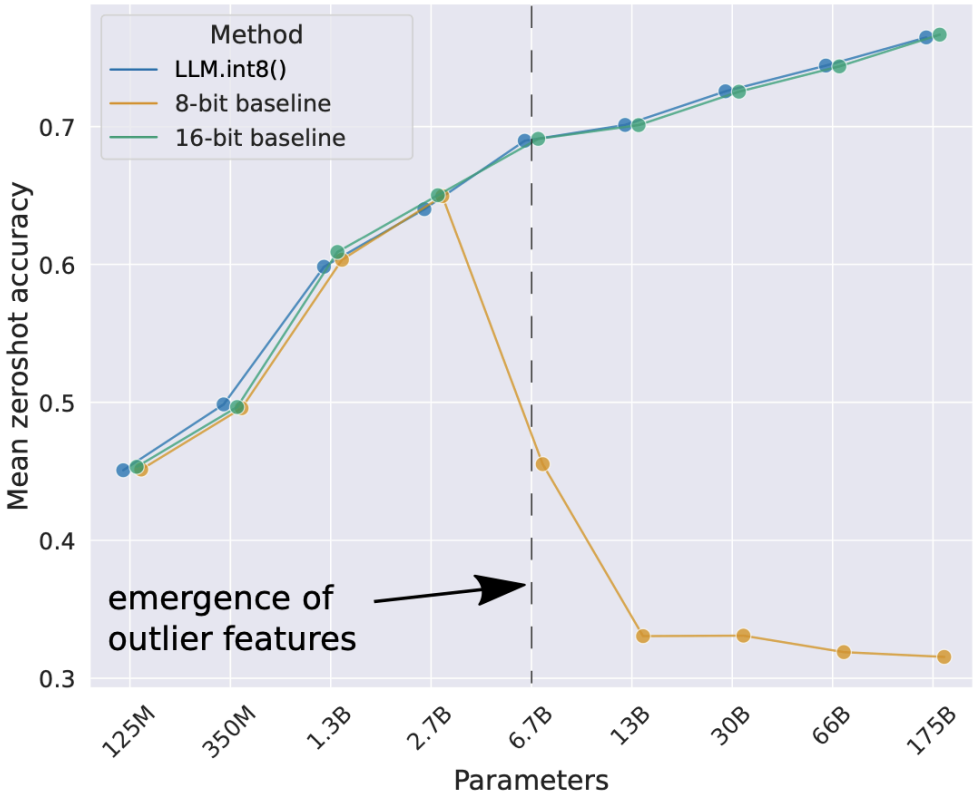
Summary of Class Discussion

There were a few clarification questions that we incorporated when writing the summary. Hopefully all is clear. Please see the bottom of the document for complete (all 3 papers) discussion.

LLM.int8()

Problem and Motivation

The exploding sizes of large transformer models poses unsustainable GPU memory pressures for both training and inference. Quantization techniques, which use data types with reduced precision to represent data, can reduce the memory footprint in half if int8 is used to replace FP16. However, in practice after quantization, model quality suffers severe deterioration due to reduced precision. The quality reduction worsens with larger model sizes.



This paper describes a quantization technique to achieve memory saving with nearly no accuracy drop. The motivation comes from the observation that the majority of model weights and activations fall in a small, predictable numerical range, and it is the existence of a few outliers that rendered the quantized model unusable. Based on this observation, it would be promising to have a technique that handles outliers and normal values separately. What's more, the study showed that as the model scales, the locations and number of outliers become fixed, making efficient identification of outliers possible.

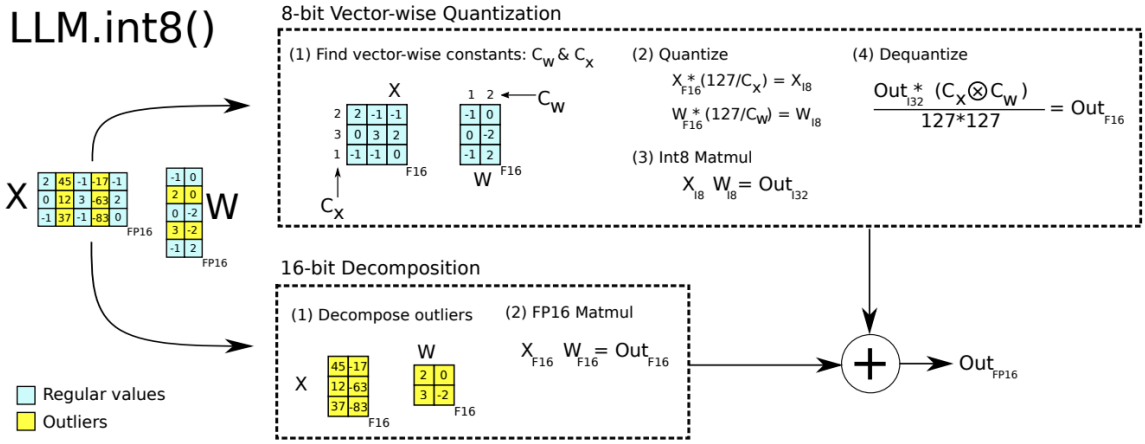
Related Works

There are three branches of related works for this paper:

1. 8-bit Data Types: This work focuses on Int8, as it is the only data type supported by GPUs and TPUs. Other data types like fixed-point or floating-point 8-bit data types can offer superior performance, as supported by analysis from existing literature.
2. Outlier Features in Language Models: The research extends previous studies on large magnitude outlier features in language models. It explores the theoretical relationship between outlier appearance and aspects such as layer normalization and token frequency distribution. The paper builds on this by demonstrating how the scale of autoregressive models relates to these outlier features and emphasizes the importance of accurately modeling outliers for effective quantization.
3. Multi-billion Scale Transformer Quantization: This topic discusses two contemporary quantization methods: nuQmm and ZeroQuant. Both employ group-wise quantization for greater precision and focus on accelerating inference and reducing memory footprint. In contrast, the authors' work prioritizes preserving predictive performance under an 8-bit memory footprint, achieving zero-degradation quantization for models up to 176 billion parameters. They acknowledge that methods like nuQmm and ZeroQuant are complementary to their approach. Additionally, the GLM-130B model, which incorporates insights from their work, achieves effective 8-bit quantization while maintaining 16-bit precision in matrix multiplication.

Solution Overview

This paper describes a quantization technique, **Mixed-Precision Decomposition**, to achieve memory saving with nearly no accuracy loss. It does so by separating the model weights into two groups, normal values that can be accurately represented with int8, and outliers (>6 standard deviations from mean). When performing matrix multiplication, the normal value group is performed in int8 for efficiency, and the outliers will be calculated in FP16, which has higher precision.



In this way, LLM.int8 can achieve nearly in-half memory reduction while having nearly no effect on model accuracy.

Limitations

The limitations of this work being mentioned in the paper and presentation were related to the authors not being able to do certain additional studies with the tool. A few examples include:

1. The technique was not experimented with the superior FP8 data type and model sizes larger than 175B.
2. The technique solely focused on reducing memory footprint, instead of promoting the calculation speed. Thus, this technique did not quantize attention computation.
3. The technique is for inference only and cannot be naively applied to training as it requires complex trade-offs between quantization precision, training speed and engineering complexity.

A few other (not listed) limitations of the techniques is:

1. The technique, at least as per its current implementation, can only work for un-quanitized models. Quantization has to be dynamically determined due to this mixed-precision decomposition technique.
2. The performance impact of this technique is huge. The outlier extraction technique is inefficient and has to be redone in every computation. The performance burden becomes significant when you are using a much smaller model than 175B, such as 7B, due to less predictable distribution of outliers. The execution time of a 7B model with LLM.Int8 can be doubled or even tripled.

Future Research Directions

1. Extending the Technique for Different Data Types: Future research can explore efficient yet accurate quantizations for data types like FP8 and INT4.
2. Addressing the Technique's Performance Overhead: Future research can explore how the outlier extraction algorithm can be made more efficient for smaller models.
3. Understanding the Theory Around Outliers: This paper made significant contributions to the patterns which outliers occur in transformer models. The outlier extraction idea was based on the pattern observed. Future research can strive for deeper insights of why outlier occurs and thus leading to fundamentally more efficient quantization techniques.

Summary of Class Discussion

Please see bottom of document.

Large Transformer Model Inference Optimization

Problem and Motivation

The high computational and memory demands of large transformer models pose significant challenges for their efficient inference, especially in environments with limited resources. These models, essential for state-of-the-art achievements in various fields, require significant memory for parameters and intermediate computations, with KV cache needs escalating to 3TB for specific batch and context sizes, tripling the model size. Furthermore, their quadratic scaling of

inference costs with sequence length and autoregressive decoding limits parallel processing, increasing the complexity of their deployment in practical settings.

Addressing these challenges is crucial for making transformer models more accessible and usable. Therefore, research is focused on developing optimization techniques that reduce the computation load and memory usage of these models. The goal is to enable their effective deployment across different computing environments without sacrificing performance, ensuring that the benefits of transformer models can be fully realized in real-world applications.

Related Works

In recent literature, efforts to optimize large transformer model inference focus on reducing memory and computational requirements without sacrificing performance. Techniques such as knowledge distillation, as described by Hinton et al. (2015) and Gou et al. (2020), involve training a smaller, faster "student model" to mimic a larger "teacher model," with DistilBERT (Sanh et al., 2019) being a prime example of achieving substantial parameter reduction and speed improvement.

Network compression strategies, including pruning, quantization, and sparsification, aim to decrease the model's memory footprint and computational load. For instance, post-training quantization (PTQ) and quantization-aware training (QAT) adjust model precision post-training or during training, respectively, to balance performance and resource use effectively.

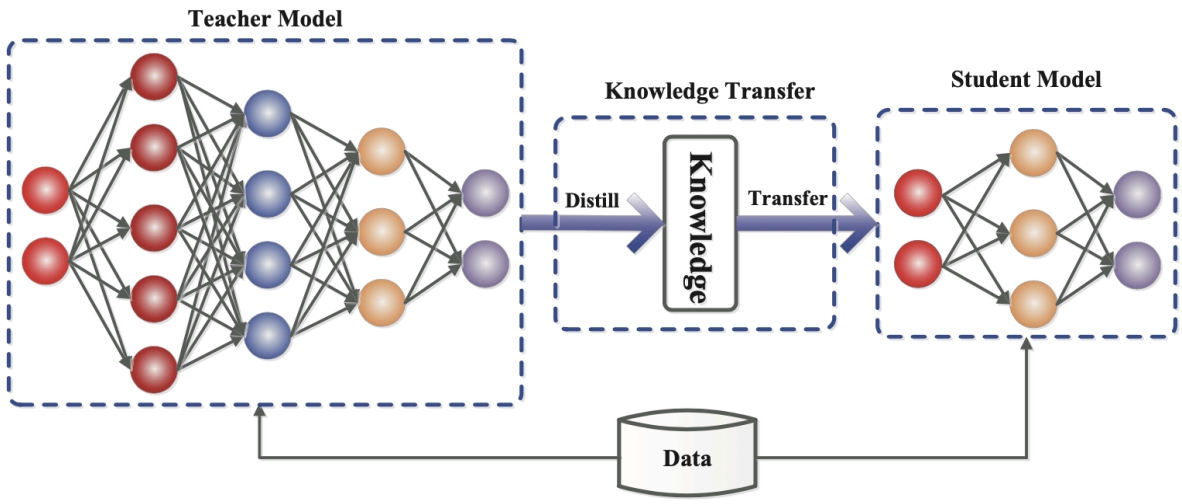


Fig. 1. The generic framework of teacher-student knowledge distillation training (Gou et al. 2020)

Moreover, structured pruning methods, such as magnitude pruning and gradual magnitude pruning (GMP), have been explored to eliminate unimportant weights and maintain

computational efficiency, as discussed by Zhu & Gupta (2017) and Renda et al. (2020). These approaches strive to identify and retain only the most critical parameters of the model, thereby enabling efficient inference on hardware with limited resources.

Solution Overview

Optimization of large transformer models for inference involves several key strategies to balance computational efficiency and model performance. Knowledge distillation, as highlighted by Hinton et al. (2015) and Gou et al. (2020), enables the creation of smaller models that replicate the functionality of larger counterparts, exemplified by DistilBERT's (Sanh et al., 2019) reduction in size while retaining performance.

Concurrently, network compression methods, including pruning, quantization, and sparsification, aims to decrease memory footprint and computational load. Techniques such as post-training quantization (PTQ) and quantization-aware training (QAT) adjust model precision to optimize performance-resource trade-offs.

Furthermore, structured pruning and innovative approaches like N:M sparsity (Nvidia, 2020) and Layer-by-Layer Knowledge Distillation (LKD; Yao et al., 2020) specifically tailor model architecture to enhance decoding speed and efficiency, addressing the inherent challenges of transformer model scalability and resource consumption.

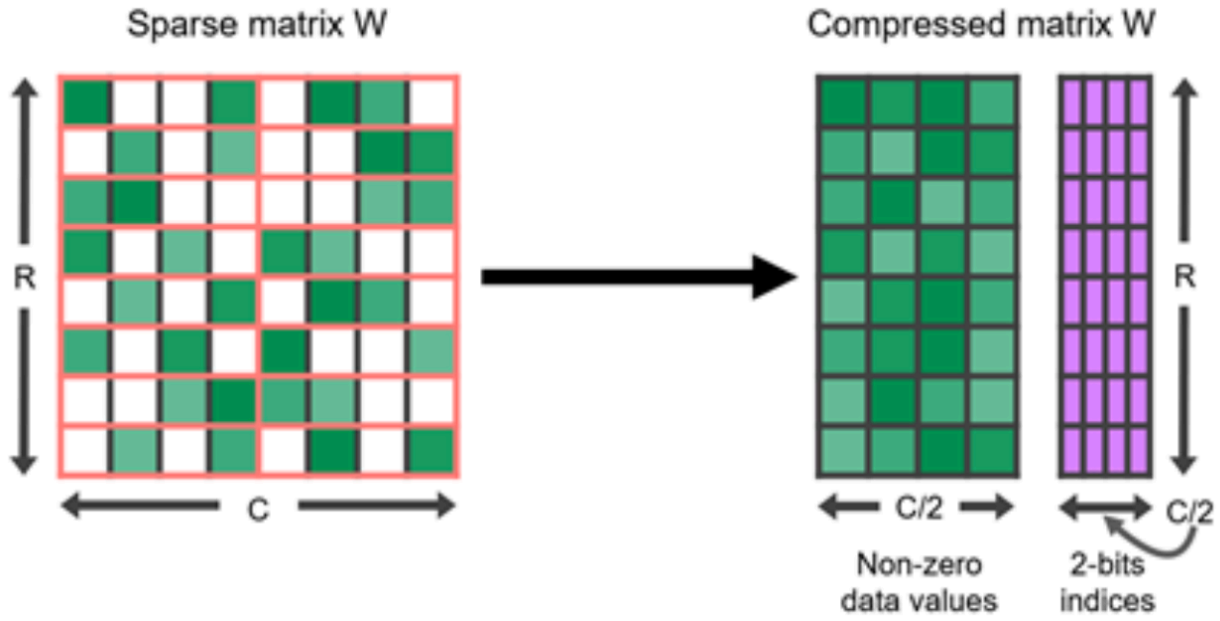


Fig. 2. A matrix of 2:4 structured sparsity and its compressed representation (Nvidia, 2020)

Limitations

Limitations in optimizing large transformer models include significant performance drops with simple-low precision quantization due to dynamic range issues and the computational expense of complex quantization strategies. Additionally, the effectiveness of network pruning and sparsity techniques can be hardware-dependent, potentially leading to suboptimal speedups. Moreover, sophisticated routing in Mixture-of-Experts (MoE) architectures can introduce complexity and computational overhead, challenging the balance between model efficiency and performance maintenance.

Future Research Directions

Future research in transformer model optimization should concentrate on enhancing quantization strategies to balance performance with reduced precision, as highlighted by Bondarenko et al. (2021) and Dettmers et al. (2022). Additionally, advancements in pruning and structured sparsity methods, as discussed by Zhu & Gupta (2017) and Pool & Yu (2021), are essential for efficient resource use. Developing more effective knowledge distillation techniques, as exemplified by Sanh et al. (2019), and exploring architectural innovations like Mixture-of-Experts (MoE) will also be key to improving parallel processing and reducing computational demands.

Overall Q&A

1. **Q:** What are you losing when distilling from the teacher model to the student.
    a. **A:** When distilling a large teacher model into a smaller student model, there is an inherent loss of parameters, which can lead to a reduction in the model's capacity to learn and represent complex patterns comparable to the teacher model.
    b. At the same time, distillation attempts to retain as much of the teacher's performance as possible by learning to mimic its output distribution.
2. **Q:** What are the architecture constraints on the student model?
    a. **A:** The architecture constraints on the student model primarily revolve around the output layer, which needs to match the teacher model's output space to ensure that the student can learn the same task as the teacher.
    b. Beyond the output layer, the architecture of the student model can be flexible.
3. **Q:** Are the zeros in a sparse matrix inherently zero, or are they made zero intentionally?
    a. **A:** Elements in a matrix are forced to become zeros based on a predefined sparsity pattern or threshold.
    b. **Q (Follow-up):** And does the matrix need to adhere to a specific sparsity pattern, where elements are selectively zeroed?
        i. The sparsity pattern follows a consistent ratio (N:M 2:4).
4. **Q:** Does employing random swaps in a greedy permutation algorithm affect the quality of the data outcomes?

    a. **A:** No. They are employed as part of bounded regressions which do not inherently compromise data quality. Instead, it serves as a strategy to explore the solution space more comprehensively.
5. **Q:** What are the possibilities and challenges of integrating various compression methods, such as merging quantization with memory offloading or MoE?
    a. **A:** In principle, integrating different compression methods like quantization with memory offloading or MoE is possible and could lead to more efficient model execution.
    b. However, this integration must be handled carefully to avoid performance degradation. Imaginably because of the loss of precision from quantization and potential latency increases from offloading processes.
6. **Q:** What compression methods can be combined?
    a. **A:** Combining techniques like knowledge distillation and quantization is possible, but concrete studies are limited. There might be performance trade-offs.
    b. **Q (Follow-up):** Is the limitation more about compute time or accuracy?
        i. Layering optimizations could increase perplexity. The combination of methods is unlikely to yield exponential benefits.
    c. **Questioner note:** Activation quantization and layer quantization are orthogonal. For image generation, activation quantization is less effective. Combining smooth quantization with techniques from OPTQ might be feasible through outlier smoothing and iterative quantization.
7. **Q:** Can quantization be automated in processing, and how do frameworks handle it?
    a. Hugging Face supports quantization. Automatic quantization isn't well-developed; research could focus on heuristics for selecting optimal quantization strategies.
8. **Q:** What about "extreme quantization" and the use of RTN and BOOM?
    a. **A:** RTN gained popularity due to its efficacy. Different parameter sizes need specific quantization rules to manage perplexity sensitivity.
    b. **Q (Adjacent):** In mixed-precision quantization, how do weights and activations differ in scale?
        i. **A:** Weights are typically clustered around zero, excluding outliers, while activation ranges are broader. Multiple precisions in hardware could accommodate this variance.
9. **Q (Future Research):** Is exploring multiple precisions in quantization a promising research area?
    a. **A:** It's worth investigating, especially if it shows significant benefits.
10. **Q:** Can post-training and quant-aware training be merged for separate quantization of weights and activations?
    a. A: Combining them is possible, especially with techniques like Q lora for fine-tuning quantized models.

11. **Q:** Why is there less focus on quantizing activations?
    a. **A:** Activations are more challenging to quantize effectively compared to weights.
12. **Q (Sparsity):** Do we need to repeat the N:M sparsity algorithm regularly?
    a. **A:** The N:M algorithm might need periodic application to maintain sparsity.
    b. **Q (Follow-up):** Can applying sparsity algorithms twice double the sparsity and speed?
        i. A: This requires careful implementation to ensure effectiveness and efficiency.