Towards End-to-end 4-Bit Inference on Generative Large Language Models

Saleh Ashkboos[*]
ETH Zurich
saleh.ashkboos@inf.ethz.ch

Ilia Markov[*]
IST Austria
ilia.markov@ist.ac.at

Elias Frantar
IST Austria
elias.frantar@ista.ac.at

Tingxuan Zhong
Xidian University
ztx12190033@gmail.com

Xincheng Wang
Xidian University
xcwang.post@gmail.com

Jie Ren
KAUST
jie.ren@kaust.edu.sa

Torsten Hoefler
ETH Zurich
htor@inf.ethz.ch

Dan Alistarh
IST Austria & Neural Magic
dan.alistarh@ista.ac.at

Abstract

We show that the majority of the inference computations for large generative models such as LLaMA and OPT can be performed with both weights and activations being cast to 4 bits, in a way that leads to practical speedups while at the same time maintaining good accuracy. We achieve this via a hybrid quantization strategy called QUIK, which compresses most of the weights and activations to 4-bit, while keeping some outlier weights and activations in higher-precision. Crucially, our scheme is designed with computational efficiency in mind: we provide GPU kernels with highly-efficient layer-wise runtimes, which lead to practical end-to-end throughput improvements of up to 3.1x relative to FP16 execution. Code and models are provided at https://github.com/IST-DASLab/QUIK.

1 Introduction

Large language models (LLMs) from the Generative Pretrained Transformer (GPT) [19] family have gained massive popularity. One key contributor to their adoption by enthusiasts has been the ability to compress them using advanced quantization techniques, e.g., [6, 9, 15, 28], enabling local storage and efficient generative inference for these models, even on personal computers. Yet, the vast majority of work on quantization can be categorized into two cases:

- *Weight-only quantization methods* [6, 7, 9, 12, 15, 15] that help reduce the massive memory-transfer costs of LLM execution, but do not reduce computation, and thus cannot provide significant speedup for computationally-bound settings, such as prompt processing or large-batch inference.
- *Joint weight-activation quantization methods*, which can provide computational improvements, but either focus exclusively on 8-bit weights and activations [6, 27], or execute with large amounts of accuracy loss relative to their uncompressed counterparts [22, 28].

Contribution. In this paper, we take a step towards bridging the precision gap between accurate weight-only (typically 4-bit) and weight-and-activation methods (typically 8-bit), and present preliminary results showing that a large fraction of the computation in modern LLMs such as OPT [30] and LLaMA-2 [24] can be performed accurately using 4-bit activations *and* weights.

[*]These authors contributed equally.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

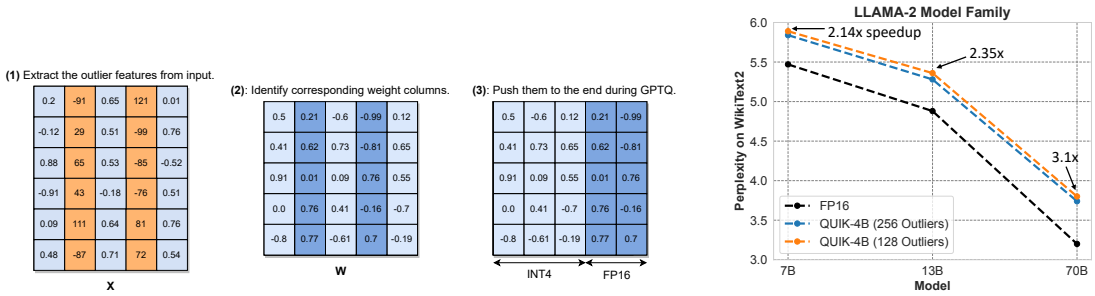


Figure 1: Outlier-aware quantization with QUIK. **Left:** The weight columns are extracted based on the outlier columns in the input. We permute the outlier columns toward the end of the matrix before applying GPTQ quantization to accumulate the quantization errors in the FP16 columns. **Right:** QUIK achieves up to 3.1× speedup with minor accuracy degradation on LLaMA-2 models.

On the accuracy side, we show significantly improved results relative to prior work, by introducing a hybrid selective quantization procedure, by which matrices are split into "base" weights and activations, which are processed exclusively at 4-bit precision, and "outlier" weights and activations, which are processed at higher precision. Using this approach as well as additional key insights on layer sensitivity, we can run the vast majority of the computation in highly-accurate models such as LLaMA-2 [24] in INT4, while dropping less than 0.5 perplexity across model sizes.

Importantly, our hybrid scheme, termed QUIK, is designed in a way that incurs minimal runtime overhead in a practical implementation. We demonstrate this by implementing efficient kernels, which show high per-layer speedups and yield up to 3.1× end-to-end throughput improvements relative to the FP16 baseline (Figure 1). Along the way, we also perform a study of the efficiency of different low-precision formats on current GPU architectures.

2 Method

2.1 Accurate Quantization via QUIK

We focus on the task of accelerating linear layers within Large Language Models (LLMs) by employing 4-bit quantization for both weight matrix $\mathbf{W}$ and input matrix $\mathbf{X}$. Following the PyTorch definition [18], a linear layer carries out a linear transformation along with a bias vector denoted as $\mathbf{b}$, taking the form of $\mathbf{X}\mathbf{W}^\mathbf{T} + \mathbf{b}$. We now describe the background and details of the technique.

Outliers in Input Quantization. While there are several methods for accurate weight quantization, it is known that the activation matrices are hard to quantize [5, 27, 28]. This is mainly because of the presence of *outlier features* in such matrices, where some of the columns have up to 100x larger magnitudes. LLM.int8 [5] identifies and extracts the outlier columns of $\mathbf{X}$ during the forward pass and quantizes the rest of the elements with 8-bit. However, this method is not efficient at runtime due to the added computational cost of determining outliers on-the-fly. Recent work [27] has shown that the outlier features are fixed for each layer across various datasets, which means *we can extract the outlier indices offline* using a small calibration set.

GPTQ Weight Quantization. GPTQ [9] involves the quantization of the weight matrix $\mathbf{W}$ while retaining the activations $\mathbf{X}$ in FP16. To achieve this goal, it iterates over the weight columns, and for each column, it proceeds to quantize all of its elements simultaneously. Following this, GPTQ adjusts the remaining unquantized columns by using second-order information to compensate for the introduced quantization error in the current step. This process *accumulates the quantization errors at the last columns*, making them more sensitive to the quantization.

QUIK: Activation Outlier-Aware GPTQ. During the linear transformation $\mathbf{X}\mathbf{W}^\mathbf{T}$, the outlier columns in $\mathbf{X}$ will always multiplied by certain columns in $\mathbf{W}^\mathbf{T}$ (see Figure 1). We can utilize this observation and improve the quality of GPTQ quantization, while quantizing (part of) the activations as well. To this end, since the outlier columns are fixed across datasets, we begin by extracting the

indices of the outlier columns by means of a calibration set. Then, we rearrange the weight columns (and their corresponding rows and columns in the Hessian matrix), to shift the outlier columns toward the end. Finally, we perform quantization on the weight columns up to the index of the outlier features. This circumvents quantization of these hard-to-quantize columns, and also helps to improve GPTQ quantization by aggregating the quantization errors to the columns we keep in FP16, and removing potential weight outliers from the 4bit weight quantization scale.

Weight Clipping. Weight clipping improves quantization by trimming the input distribution before rounding. This could be done by either training the whole network to find the optimal clipping thresholds [3, 8, 22]; or employing heuristic methods [12, 13, 15]. We found that applying linear search over the clipping thresholds for weight quantization improves final perplexity.

Partial Quantization. The described approach is sufficient for effective quantization of OPT models with minimal impact on accuracy (see Section 3). However, highly-accurate massive models such as LLaMA2-70B present a unique challenge due to their FeedForward layers, which involve three linear transformations along with element-wise multiplication, as well as their use of GeLU activations. More specifically, previous work [14] has shown that the $\text{Down}_{\text{proj}}$ layers have a wide input distribution, making their quantization hard. We found that we can recover the accuracy by quantizing the $\text{Down}_{\text{proj}}$ layers using only 8 bits, without any additional changes to our methods. We describe the selection procedure and provide additional discussion in Section **??**.

2.2 GPU Kernel Support for QUIK

We now provide a high-level description of models in the QUIK format can be executed efficiently on GPU. At the execution level, the quantized matrix multiplication consists of three parts: 1) quantization of activations, 2) matrix multiplication of quantized input and weights, and 3) dequantization of the result. Please see Algorithm 1.

Algorithm 1 Quantization and Dequantization kernels.

function QUANTIZATION($input$)
  $zeroAct, scaleAct \longleftarrow findZeroScale(input)$
  for $elem \in input, outElem \in output$ **do**
    $outElem \longleftarrow (elem - zeroAct)/scaleAct - halfRange$;
  end for
  return $output, zeroAct, scaleAct$
end function
function DEQUANTIZATION($input, zeroAct, scaleAct, scaleWeight, WeightsReduced$)
  for $elem \in input, outElem \in output$ **do**
    $x \longleftarrow elem * scaleAct * scaleWeight$;
    $shift \longleftarrow (zeroAct + halfRange * scaleAct) * WeightsReduced$;
    $outElem \longleftarrow x + shift$;
  end for
  return $output$;
end function

As long as activation quantization is asymmetric, we first find zero and scale of the vector and then perform element-wise quantization shifting the values to fit into `INT4` or `INT8` range (by 8 for `INT4`, by 128 for `INT8`, and $halfRange$ in Algorithm 1). The second operation leverages `CUTLASS` support of `INT4|INT8` types; it takes two quantized matrices and outputs the matrix multiplication result in `INT32` format. Dequantization of the matrix multiplication result involves rescaling of the input using scaling factors of the weights and activation quantizations. Due to the asymmetric nature of the activation quantization, we also need to perform shifting of the result using pre-computed row-wise aggregation of weights and zero factors of activations.

3 Experimental Validation

General setup. We evaluate our method on OPT [30] and LLaMA-2 [25] model, using HuggingFace [26] implementations of model definitions and datasets. Following SmoothQuant [27], we

extract outlier indices using 512 random sentences from the Pile [10] dataset. For GPTQ weight quantization, we randomly select 128 samples with sequence length 2048 from the C4 dataset [20]. We apply symmetric quantization to weights and asymmetric quantization to activations. Clipping thresholds for the weight quantization grid are found via a simple linear search over the squared error. Our scheme quantizes a 70B model in less than 2 hours on an NVIDIA A100 GPU.

Table 1: Accuracy results for 4bit models. (Left) Perplexity of 4-bit OPT models on WikiText2 dataset. SmoothQuant, RPTQ, and OmniQuant results are taken from [22], RPTQ denotes their improved numbers. (Right) QUIK ablation study results on 4bit LLaMA-2 family, also WikiText2 perplexity.

Model	OPT		
	13B	30B	66B
Baseline	10.13	9.56	9.34
SmoothQuant	7.4e3	1.2e4	2.2e5
RPTQ	17.83	11.50	11.16
OmniQuant	11.65	10.60	10.29
QUIK (ours)	**10.78**	**10.08**	**9.66**

Model	LLaMA-2		
	7B	13B	70B
Baseline	5.47	4.88	3.20
4-bit Down-Proj	8.87	7.78	6.91
256 Outliers	5.84	5.28	3.74

Accuracy Comparison on OPT. First, we compare the accuracy of QUIK against prior 4-bit activation quantization methods: SmoothQuant (applied for 4-bit) [27], RPTQ [28] and OmniQuant [22]. Table 1 (Left) shows the results of all methods for 4 larger OPT models on the WikiText2 task [17]. As can be seen, by effectively leveraging a small amount of full-precision outlier columns (here 256, which is $\approx 3\%$ of OPT-66B's hidden size), QUIK is able to significantly outperform prior 4-bit methods, dropping only 0.3 to 0.5 points in perplexity relative to the full precision baseline. We emphasize that, for a fair comparison, QUIK quantizes *all* linear backbone layers to 4-bit here.

LLaMA-2 Quantization. Next, we move to state-of-the-art LLaMA-2 models, where we ablate key parameters of QUIK: the 8-bit down-projection as well as the outlier count. (See Table 1 (Right) for the results.) Our main conclusion is that keeping the down-projection layers in 8-bit is critical to achieve high accuracy, as it improves perplexity by more than two points, across all models, and that increasing the number of outliers brings continuous small improvements. Adjusting the outlier count provides fine-grained control over the accuracy/compression trade-off. Additional details are presented in Appendix A.

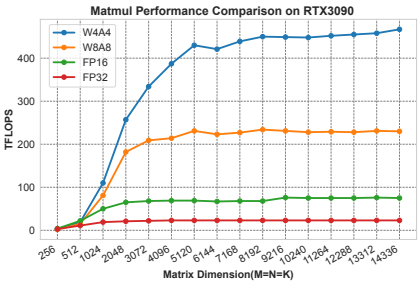


Figure 2: Ideal speedups for different layer sizes and compression types.

Zero Outliers Setting. Next, we study the results of keeping some of the layers with small outliers in the "zero outliers" setting. Table 2 shows how the accuracy of LLaMA-2 models changes when we use different threshold values, extracted using a linear search, for the outliers. The results show that there isn't any universal threshold that optimizes both accuracy and performance across all models. For example, 70B model cannot tolerate a threshold larger than 3.0, while the large perplexity jump occurs at 8.0 in the 13B model.

Additional evaluations. In the Appendix, we perform additional evaluations of QUIK models on various other tasks, including: PTB [16] and C4 [20] datasets. Overall, they show similar patterns to the WikiText results above. Also, we show at most $\approx 1\%$ degradation in famous Zero-Shot tasks.

Ideal and Layer-wise Speedups. The results in Figure 2 depict "ideal" computational power for layer-wise matrix multiplications at different precision levels, without taking into account any quantization/dequantization overheads. The results motivate our approach, as they show that significant speedups could be achieved via lower-precision specifically for these operations. Motivated by this

Model	T	LLaMA-2 7B	13B	70B
FP16	-	5.47	4.88	3.2
QUIK-4B	0	5.84 (0)	5.28 (0)	3.74 (0)
	2.0	5.91 (5)	5.33 (3)	3.75 (10)
	3.0	6.09 (11)	5.34 (8)	3.85 (30)
	4.0	6.13 (21)	5.36 (17)	5.15 (58)
	8.0	12.93 (55)	21.85 (66)	5.92 (219)

Table 2: Study of zero outlier setting on WikiText2 using 256 outliers. We use zero outliers when the maximum of scale is less than threshold **T**. For each experiment, the number of linear layers with zero outliers is written in parentheses.

observation, we turn our attention to realizable speedups when executing Algorithm 1, which includes the compression and decompression operations required for obtaining high practical performance.

In Figure 3, we compare layer-wise performance of quantized linear layers (QUIK-4B uses 256 outliers per layer) relative to FP16, for a full implementation of our algorithm. The matrix sizes correspond to layers in LLaMA models. We observe that QUIK-4B can achieve around $4\times$ speedup on large layers and more than $2\times$ on smaller ones. Thus, the raw low-precision matmul speedups can partially "hide" the overheads of QUIK.

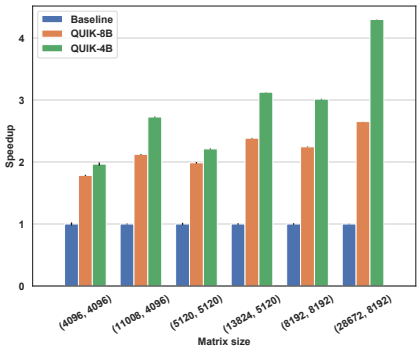


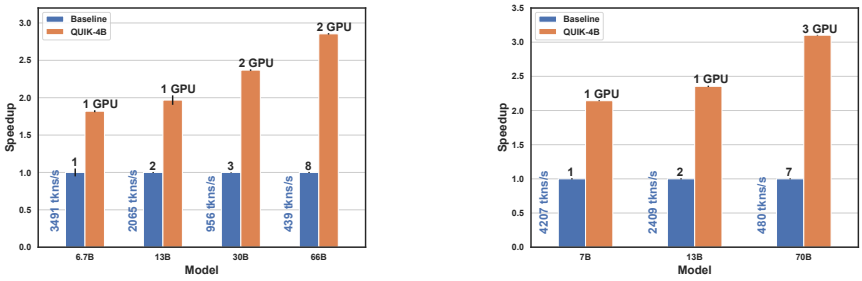
Figure 3: Layer-wise speedups on a single RTX3090 for different layer sizes and compression types. QUIK-4B with 256 outliers, QUIK-8B without outliers.

End-to-end speedups. Finally, we also demonstrate the end-to-end speedup benefits of QUIK models. For this purpose, we integrate QUIK into the widely used HuggingFace PyTorch implementations of OPT and LLaMA-2, by replacing linear layers with 4-bit (and 8-bit) QUIK re-implementations. For the LLaMA model, we use the standard FlashAttention [4] implementation for all the models (including the FP16 model). In Figure 4, we compare the throughput improvements of prefill passes (for single batches with 2048 tokens) for quantized models with the corresponding FP16 version. The bar plot shows throughput improvements of QUIK-4B compared to FP16, annotations to baseline represent the actual values of the baseline througput in our experiments. For instance, OPT-66B using FP16 linear layers occupied 8 GPUs, achieved 439 tokens/s whereas the same model inference with QUIK-4b linear layers occupied only 2 GPUs and resulted in 1253 tokens/s. In addition to a close to $4\times$ memory reduction, which reduces the number of required GPUs for inference, QUIK also achieves up to $3.1\times$ higher throughput relative to FP16, with the biggest improvements attained on the largest models (LLaMA2-70B), where the relative impact of overheads is lowest. We present a more advanced kernel in [1].

We emphasize that the speedups in these end-to-end experiments are exclusively through QUIK accelerated linear layers, as all other overheads are precisely the same. We observe that overheads from attention, softmax, or layernorm operations become significant once a large fraction of the computation occurs in 4-bit, which could be optimized further in our implementation.

4 Conclusion and Future Work

We have presented a new hybrid quantization scheme called QUIK, which allows us to execute the vast majority of inference computation using weights and activations quantized to 4 bits. QUIK minimizes accuracy loss by keeping the hard-to-quantize outlier layers in higher precision. Importantly, QUIK can be efficiently supported on GPU hardware, presenting significant speedup potential, especially for inferencing on large models.



(a) OPT Performance.

(b) LLaMA-2 performance.

Figure 4: End-to-end inference speedups for QUIK-4B with 256 outliers relative to the FP16 baseline, on NVIDIA RTX 3090 GPUs. Notes over the bars state the minimal number of GPUs required to run the inference.

Acknowledgement

This project received EuroHPC-JU funding under grant MAELSTROM, No. 955513 and the UrbanTwin project, a project in the strategic area Energy, Climate and Environmental Sustainability. We thank the Swiss National Supercomputing Center (CSCS) for providing computing resources.

References

[1] Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefler, and Dan Alistarh. Quik: Towards end-to-end 4-bit inference on generative large language models. *arXiv preprint arXiv:2310.09259*, 2023.

[2] Michael Boratko, Harshit Padigela, Divyendra Mikkilineni, Pritish Yuvraj, Rajarshi Das, Andrew McCallum, Maria Chang, Achille Fokoue-Nkoutche, Pavan Kapanipathi, Nicholas Mattei, et al. A systematic classification of knowledge, reasoning, and context within the ARC dataset. *arXiv preprint arXiv:1806.00358*, 2018.

[3] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[4] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with io-awareness. *arXiv preprint arXiv:2205.14135*, 2022.

[5] Tim Dettmers. 8-bit approximations for parallelism in deep learning. *International Conference on Learning Representations (ICLR)*, 2016.

[6] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

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

[8] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[9] 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.

[10] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[11] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 2021.

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

[13] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Lessons learned from activation outliers for weight quantization in large language models. *arXiv preprint arXiv:2306.02272*, 2023.

[14] Qingyuan Li, Yifan Zhang, Liang Li, Peng Yao, Bo Zhang, Xiangxiang Chu, Yerui Sun, Li Du, and Yuchen Xie. Fptq: Fine-grained post-training quantization for large language models. *arXiv preprint arXiv:2308.15987*, 2023.

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

[16] Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

[17] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[21] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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

[23] Sandeep Tata and Jignesh M Patel. PiQA: An algebra for querying protein data sets. In *International Conference on Scientific and Statistical Database Management*, 2003.

[24] 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.

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

[26] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[27] Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2022.

[28] 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.

[29] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[30] 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.

A Full Accuracy Results

In this section, we present the detailed accuracy results for OPT and LLaMA-2 families across different tasks and parameters.

Table 3 shows the perplexity results of OPT models. We use symmetric quantization for the weights in all our experiments. The results suggest that in a 4-bit setting, considering outlier features is crucial to preserve the accuracy even in small models (like OPT-1.3b).

Model	OPT-1.3b			OPT-6.7b			OPT-13b			OPT-30b			OPT-66b		
Task	WIKI	PT	C4	WIKI	PT	C4	WIKI	PT	C4	WIKI	PT	C4	WIKI	PT	C4
Baseline	14.63	16.96	14.72	10.86	13.09	11.74	10.13	12.34	11.20	9.56	11.84	10.69	9.34	11.36	10.28
GPTQ-4B	15.89	18.83	15.90	11.43	13.81	12.21	10.38	12.65	11.41	9.60	12.02	10.83	9.65	11.63	10.56
0 Outliers	15k	9k	10k	10k	9k	9k	9k	12k	9k	12k	13k	17k	12k	13k	10k
64 Outliers	26.259	27.143	22.981	11.473	13.888	12.348	11.031	13.305	11.971	10.283	12.557	11.267	9.851	11.965	10.742
128 Outliers	17.638	19.709	16.799	11.671	13.809	12.314	10.964	13.241	11.894	10.339	12.564	11.279	9.805	11.842	10.653
256 Outliers	17.358	19.525	16.607	11.184	13.811	12.262	10.779	13.175	11.847	10.078	12.465	11.226	9.662	11.793	10.635

Table 3: Perplexity scores of QUIK-4B over various OPT models with different outliers on three datasets: WikiText2 (WIKI), Pen Treebank (PT), and C4.

Table 4 shows the perplexity of QUIK on LLAMA-2 models. We provide a list of tricks to improve the quality of the model without too much overhead. We found that keeping the down-proj layer in 8 bits can improve the perplexity by about 3 points. Also, we found weight clipping as a cheap and efficient trick for improving the accuracy of QUIK-4B.

LLaMA-2	Down-Proj	Clipping	7B	13B	70B
FP16	W16A16	-	5.47	4.88	3.2
GPTQ-4B	W4A16	-	6.24	5.25	3.68
QUIK-4B	W4A4	-	8.78	7.78	6.91
QUIK-4B	W4A16	-	6.09	5.49	3.98
QUIK-4B	W4A8	-	6.11	5.5	4.0
QUIK-4B	W8A8	-	5.98	5.37	3.87
QUIK-4B	W8A8	✓	5.84	5.28	3.74

Table 4: LLaMA-2 perplexity results on WikiText2 using 256 outliers. We apply clipping only during the weight quantization.

B INT-8 Accuracy Results

In this section, we compare applying QUIK-8B against SmoothQuant, which is the SoTA INT-8 quantization, on the WikiText2 dataset. Table 5 shows our results. We use per-token/per-channel quantization for activations/weights in SmoothQuant and only apply the quantization on the linear layers (which is the case for QUIK also).

Model	**OPT**					**LLaMA-2**		
	1.3b	6.7B	13B	30B	66B	7B	13B	70B
FP16	14.63	10.84	10.13	9.56	9.34	5.47	4.88	3.20
SmoothQuant	14.70	10.89	10.37	9.59	9.80	5.58	4.94	3.48
QUIK-8B	**14.62**	**10.84**	**10.13**	**9.51**	**9.29**	**5.48**	**4.89**	**3.33**

Table 5: Accuracy results for 8bit models on WikiText2. We use 256 outliers in QUIK experiments. Following the SmoothQuant paper, we use $\alpha = 0.8$ hyperparameter for LLaMA-2 models.

C ZeroShot Evaluation

In this section, we evaluate QUIK-4B on five zero-shot tasks: PIQA [23], WinoGrande [21], HellaSwag [29], and Arc (Easy and Challenge) [2]. We use LM Evaluation Harness [11] for all our experiments.

Model	Bits	Arc Challenge	Arc Easy	HellaSwag	PIQA	WinoGrande	Avg. Score
OPT-30B	FP16	38.05	65.36	72.28	78.13	68.43	64.45
	QUIK-4B	36.69	64.39	70.84	77.75	67.01	63.34
OPT-66B	FP16	40.02	67.26	74.87	79.82	68.82	66.16
	QUIK-4B	38.82	64.73	73.68	79.43	68.82	65.10
LLaMA2-13B	FP16	48.98	77.44	79.38	80.52	72.22	71.70
	QUIK-4B	48.04	74.92	78.36	79.22	71.90	70.49
LLaMA2-70B	FP16	57.34	80.98	83.81	82.75	77.98	76.57
	QUIK-4B	56.14	79.00	81.57	81.56	76.56	74.97

Table 6: LM eval harness results of QUIK on OPT, LLaMA-2, using 256 outliers.