MXQ: A Mixed Quantization Approach for Large Language Model

No Author Given

No Institute Given

Abstract. Large Language Models (LLMs) have demonstrated significant capabilities in intelligent activities such as natural language comprehension, content generation, and knowledge retrieval. However, training and deploying these models require substantial computation resources, setting up a significant barrier for developing AI applications and conducting research. Various model compression techniques have been developed to address the demanding computational resource issue. Nonetheless, there has been limited exploration into high-level quantization strategy orthogonal to existing methods to offer better flexibility of balancing the trade-off between memory usage and accuracy. In this paper, we propose an effective mixed-quantization method named MXQ to bridge the research gap for a better memory-accuracy balance. Specifically, we observe that the weight distributions of LLMs vary considerably from layer to layer, resulting in different tolerances to quantization errors. Motivated by this, we derive a novel quantization optimisation formulation to solve for the layer-wise quantization parameters, while enforcing the overall quantization memory consumption budget into the constraints. The new formulation can be efficiently solved by converting to a mixed integer programming problem, which has efficient off-the-shell LP solvers. Experiments demonstrated that our method can reduce memory usage by approximately 8% while maintaining accuracy loss within 1%.

Keywords: Quantization · Large Language Model · Linear Programming · Transformer · PTQ · LLama-2

1 Introduction

When deploying or fine-tuning a pre-trained Large Language Model (LLM), GPUs are typically employed to accelerate the neural network's forward and backward propagation processes. These specialized processors excel at executing massive parallel operations, such as large-scale matrix multiplication[3]. To minimize latency, the model parameters, gradients, and associated optimizer states are stored in GPU memory. However, this approach often proves inadequate in addressing the increasing computational resource demands resulting from the exponential growth in parameters of pre-trained LLMs, driven by the scaling laws [13]. According to a recent study [7], the high computational demands have hindered research on large LLMs, as only very limited researches surveyed were

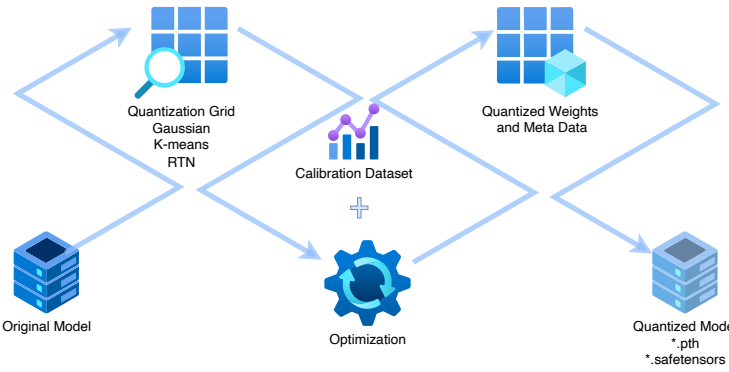

Fig. 1. Model quantization converts an original model to the quantized model with several steps: quantization grid, optimisation with calibration dataset, and quantized weights and metadata storage.

able to conduct practical experiments due to the prohibitive costs of deploying large pre-trained language models to validate their hypotheses experimentally.

Fortunately, an innovative model compression technique, known as *model quantization*, has been developed to tackle this challenge and significantly reduce storage requirements. Quantization is a technique used to convert the high-precision numeric representation of large pre-trained models' parameters into compact, low-bit equivalents. This process reduces memory consumption and boosts inference speed. Although current quantization methods strive to preserve model precision as much as possible, some loss of accuracy is inevitable. It is almost impossible for a quantization method to maintain the exact accuracy of the unquantized model. The quantization research community typically aims for near-lossless compression, commonly considered to be within 1% error relative to the uncompressed baseline, as defined in the MLCommons benchmark [20]. The key challenge of quantization lies in the trade-off between reducing the memory requirements and preserving the model performance.

The general pipeline of a model quantization process is shown in Fig. 1. The goal is to convert a pre-trained original LLM model to a quantized model with reduced storage while ensuring the least performance drop. At first, quantization represents parameters in lower precision by mapping the original high-precision numeric representation of parameters into a narrower range of values, known as quantization grids or bins, allowing them to be packed into fewer bits. For example, using 4-bit quantization, a parameter occupies only 4 bits, resulting in a 75% reduction in memory usage compared to the unquantized float16 counterpart. Then, an optimization step is usually employed to compensate for quantization errors, which may require additional metrics, such as a calibration dataset, to help adjust the quantized weights and minimize activation errors. After that, adjacent weights are packed into the same byte to save space. The associated

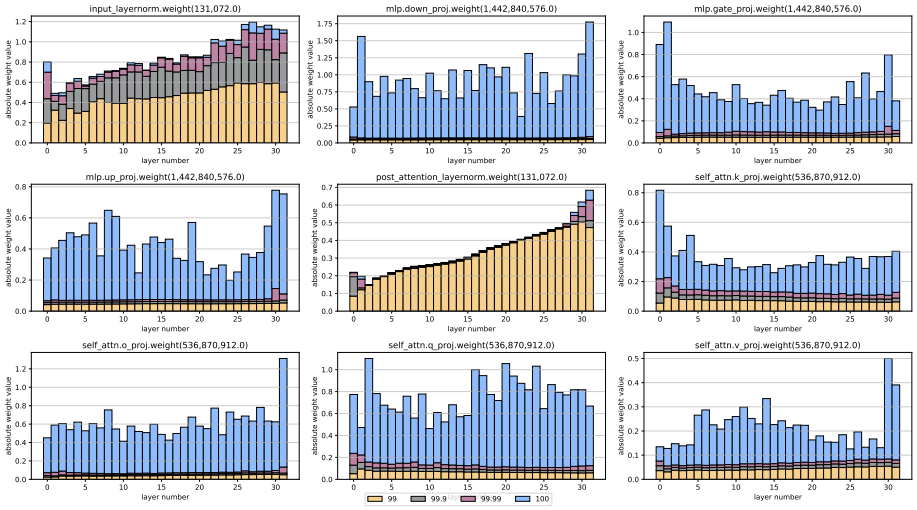

Fig. 2. Weight distributions of all layers in Llama-7b model, measured by various quantiles (99%, 99.9%, 99.99%, and 100%). Different layers exhibit significantly distinguished patterns, which motivates us to design a novel layer-wise quantization method. Best viewed in colour and zoom in.

metadata are usually quantized using the same process to further reduce storage. Finally, the quantized model is saved in persistent storage for distribution.

However, existing methods adopting the above quantization pipeline can not offer flexible control of the memory-accuracy balance since they usually adopt identical quantization configurations for each weight matrix. To tackle this issue, we propose a novel mixed-quantization method named MXQ, which can offer flexible control of the memory-accuracy trade-off and directly include the memory budget in the optimization. Specifically, we conducted a comprehensive investigation on the weight distribution of all network layers in Llama-7b model and observed an intriguing phenomenon: the normalization, MLP and self-attention modules exhibit significantly distinct weight distributions across various layers, as shown in Fig. 2. For example, `layernorm` has a wider weight band while `self_attn` owns a much narrower one. Motivated by this observation, we design a new quantization optimization formulation by introducing a set of layer-wise configurations to handle their distinctive distributions. Moreover, the memory budget is controlled by a well-devised storage cost function in the formulation constraints. For efficient optimization, we convert the quantization formulation to a mixed integer linear programming (LP) problem and solve it with efficient off-the-shell LP solvers. The quantization error matrices and storage cost matrices are pre-computed before solving the LP problem to accelerate the overall quantization process.

Comprehensive experiments are conducted to validate the effectiveness of MXQ on three benchmark datasets with three model architectures, showing a

good balance between memory storage and model performance. The contributions of this paper are summarized as follows:

- We propose an effective quantization method named MXQ, which offers a novel optimization formulation with layer-wise configurations in the objective and a storage cost function in the constraints.
- The new formulation is transformed into a mixed integer linear programming (LP) problem, which can be solved with efficient off-the-shell LP solvers.
- Experiments are performed on three standard benchmarks for the Llama family language models, showing flexible control as well as better memory-accuracy balance over state-of-the-art quantization methods.

2 Related Works

The Quantization techniques have been extensively explored, and numerous innovative methods were invented and applied in a variety of use cases, such as inference [9,14], fine-tuning [5,10] and optimizer state [4]. These techniques can be generally categorised into two categories: **(1)** Quantization Aware Training (QAT) [18], requiring backward propagation and being tightly coupled with model training, and **(2)** Post Training Quantization (PTQ) [17], which is a training-free methodology. PTQ has been recognised as the mainstream as there are a substantial number of pre-trained LLMs. For example, the HuggingFace model search page reports approximately 700,000 models published as of June 2024. Consequently, advancements in PTQ research provide greater practical value.

In this section, we focus on the weight-only quantization methods, particularly those closely associated with our proposed mixed-quantization strategy, which can serve as the underlying quantization implementation. In the following subsections, we first describe relevant quantization grid strategies and then survey two categories of weight-only quantization methods: calibration-based methods and calibration-free methods.

2.1 Calibration-based Approaches

The calibration-based approaches, based on advanced mathematical theory such as the Hessian matrix and Fisher information, usually produce better-quantized models. However, they tend to be slow and prone to overfitting the calibration dataset. The representative state-of-the-art implementations of calibration-based approaches include GPTQ and AWQ.

GPTQ [9] is based on Optimal Brain Quantizer (OBQ) [8], which quantized one weight at a time while constantly updating all not-yet-quantized weights to compensate for the error incurred by quantizing a single weight. GPTQ improves OBQ by quantizing weight column-wise to eliminate repeated calculation of the inverse of the Hessian Matrix, thus scaling to a larger model with parameters

as many as a few hundred billion. GPTQ has extensively optimized kernels to accelerate mixed-precision matrix multiplication. Thus, the GPTQ quantized models not only save memory but also run faster.

AWQ [14], based the observation that the importance of LLM's weights is non-uniform, proposes a quantization method to identify the minority "salient" weights by measuring activation magnitude and scaling the identified weights to minimize quantization errors. What makes AWQ unique is that rather than isolating salient weights into separate storage like sparse matrix, it utilizes the same quantized storage to preserve the salient weights, eliminating the need to develop specialised mixed-precision matrix multiplication kernel for fast inference.

2.2 Calibration-free Approaches

HQQ [2] approaches minimizing quantization errors by relying solely on the weight without considering the layer activation. Furthermore, it incorporates the $L_{p<1}$-norm loss function to effectively model outliers through a hyper-Laplacian distribution, which captures the long-tailed nature of outlier errors more accurately than the squared error, resulting in a better representation of error distribution. The outstanding feature of HQQ lies in its extraordinary quantization speed, which achieves a very close performance compared to the top quantization methods.
BnB [5] employs a novel high-precision technique to quantize pre-trained model weights to 4-bit NormalFloat, which employs the Gaussian distribution exhibited in model weights. The 4-bit NormalFloat datatype represents 16 values $(q1, q2, \cdots, q16)$ in the interval $[-1, 1]$. Each weight matrix is chunked into small groups for better quantization accuracy.

3 Mixed Quantization Method

Previous studies on quantizing LLMs[5], [9], [14], [2] often use identical quantization configurations for each matrix. This approach lacks flexibility in balancing the trade-off between memory consumption and model performance under various resource constraints and may be sub-optimal, especially in billion-scale LLMs. Not all weight matrices are equally quantizable. As demonstrated in Fig. 2, the 99, 99.9, 99.99 and 100 percentiles of the absolute weight values are extracted from the 32 self-attention and MLP linear layers of the Llama2-7b model. Some weight matrices have extremely narrow weight bands around zero and huge outliers, indicating varying weight distributions across layers and types of weights. This implies that the presence of large outliers in some layers makes quantization particularly challenging.

To address this issue, we propose a mixed quantization strategy, referred to as MXQ, to allocate optimal quantization configurations to each weight matrix according to a user-specified overall bit per parameter budget. MXQ leverages the mixed integer linear programming [12]. In this section, we elaborate on the

Table 1. Valid values of the mixed quantization configuration

Parameter	b_1	g_1	b_2	g_2
Value	2, 3, 4, 8	32, 64, 128	8	128

proposed MXQ method and work out a way to identify ideal configurations that minimize the Frobenius norm of the difference between the original matrix and its quantized counterpart while confining the memory usage within the constraint of the target memory budget. Let's denote $c_i = (b_1, g_1, b_2, g_2)$ as the configuration parameters used to quantize the ith matrix of the LLM, where:

- b_1 denotes the first level bit width to represent the quantized weights.
- g_1 refers to the first level group size to break the parameters into granular chunks.
- b_2 describes the second level bit width to quantize zeros and scales.
- g_2 is the second level group size for quantization of zeros and scales.

Let C be the set of all possible configurations. In this paper, we limit the search space as listed in Table 1, leading to the cardinality of all possible configurations to be 9. Moreover, let $\{W^{(i)}\}_{i=1}^{N}$ be the set of N matrices in the LLM to be quantized. In Equation 1, we formulate the quantization problem as an optimization problem, as follows:

$$\begin{aligned} \underset{c_1,c_2,\cdots,c_N}{\arg\min} & \sum_{\substack{i\in\{1,\dots N\}\\ c_i\in C}} \left\| W^{(i)} - \hat{W}^{(i)}_{c_i} \right\|_F \\ \text{s.t.} & \sum_{\substack{i\in\{1,\dots N\}\\ c_i\in C}} \texttt{storage}(W^{(i)}, c_i) \le \beta, \end{aligned} \tag{1}$$

where parameter β denotes the overall memory budget for the quantized model in megabytes. The storage cost function `storage` is defined in Equation 2:

$$\texttt{storage}(W^{(i)}, c_i) = |W^{(i)}| \cdot \left(b_1 + \frac{2b_2}{g1} + \frac{32}{g_1 \cdot g_2} \right) \tag{2}$$

For simplicity, we use the same second-level bit width and group size for scales and zeroes, as variations in these configurations have minimal impact on overall memory consumption.

To minimize the objective function in Equation 1, we calculate Frobenius norms and storage costs with respect to all possible configurations ($N \times |C|$ in total) for a given LLM. To accelerate the process, the Frobenius norm and storage cost can be pre-computed and stored in two matrices beforehand, denoted as $F \in \mathbb{R}^{N\times|C|}$ and $S \in \mathbb{R}^{N\times|C|}$. With these pre-computed metrics in place, the optimal quantization configurations problem is re-formulated as a mixed integer linear programming problem by introducing a series of binary decision variables

x_j, where $j \in \{1, \ldots, M\}$ and $M = N \times |C|$. Specifically, the optimization problem is as follows:

$$\begin{aligned} &\arg\min_{X} \quad F \cdot X \\ &\text{s.t.} \quad S \cdot X \leq \beta\,, \\ &\qquad A \cdot X = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{N \times 1}\,, \end{aligned} \tag{3}$$

$$\text{where} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}_{M \times 1}\,,$$

$$x_j \in \{0, 1\} \quad \forall j \in \{1, \ldots, M\}\,,$$

$$F = \begin{bmatrix} f_1 \; f_2 \cdots f_M \end{bmatrix}\,,$$

$$S = \begin{bmatrix} s_1 \; s_2 \cdots s_M \end{bmatrix}\,,$$

$$A = \begin{bmatrix} \overbrace{1, \cdots, 1}^{|C|}, \overbrace{0, \cdots, 0}^{|C|}, \cdots, \overbrace{0, \cdots, 0}^{|C|} \\ \overbrace{0, \cdots, 0}^{|C|}, \overbrace{1, \cdots, 1}^{|C|}, \cdots, \overbrace{0, \cdots, 0}^{|C|} \\ \vdots \\ \overbrace{0, \cdots, 0}^{|C|}, \overbrace{0, \cdots, 0}^{|C|}, \cdots, \overbrace{1, \cdots, 1}^{|C|}, \end{bmatrix}_{N \times M}\,.$$

As we search for one optimal configuration out of $|C|$ for a total number of N matrices, we initialize $A \in \mathbb{R}^{N \times M}$ in Equation 3, *i.e.*, a matrix of N rows by M columns with only 0's and 1's. Each row contains $|C|$ consecutive ones corresponding to positions of the weight matrices encoded in A. Equation 3 can be solved efficiently by off-the-shelf LP solvers such as Gurobi [11] and HiGHS [12]. In the experiment, the scipy wrapper of HiGHS is adopted to solve Equation 3.

4 Experiments

4.1 Settings

Our experiments utilize HQQ [2] as the underlying quantization implementation due to its superior quantizing speed and accuracy. Theoretically, our method is orthogonal to other quantization methods that support custom quantization

Table 2. Comparison with state-of-the-art quantization methods. Perplexity is reported on various benchmarks and different Llama models.

PPL[1]↓		**Llama-2-7B**			**Llama-2-13B**			**Llama-3-8B**		
		WikiText2	C4[2]	PTB	WikiText2	C4	PTB	WikiText2	C4	PTB
FP16	-	5.18	6.95	22.19	4.63	6.45	31.85	5.81	8.98	9.27
b4 g32	HQQ	5.28	7.06	**22.20**	**4.69**	**6.51**	**32.21**	**6.07**	9.41	9.54
	GPTQ[3]	5.38	7.11	-	4.73	6.54	33.09	9.16	10.57	11.76
	AWQ	**5.26**	**7.04**	22.63	**4.69**	**6.51**	32.78	**6.07**	**9.40**	**9.50**
b4 g64	HQQ	5.30	7.11	**22.48**	**4.70**	6.54	**32.06**	6.19	9.60	9.64
	GPTQ	5.39	7.12	-	4.74	6.55	32.92	11.35	11.93	13.48
	AWQ	**5.28**	**7.07**	22.67	**4.70**	**6.53**	32.98	**6.15**	**9.52**	**9.58**
b4 g128	HQQ	5.35	7.16	22.61	4.74	6.57	**32.96**	6.38	9.94	9.87
	GPTQ	5.41	7.18	-	4.74	6.57	33.54	-	-	-
	AWQ	**5.31**	**7.10**	**22.41**	**4.71**	**6.55**	33.25	**6.21**	**9.67**	**9.70**
b3 g32	HQQ	**5.62**	**7.53**	**25.63**	**4.89**	**6.78**	35.24	**7.09**	**11.16**	**10.68**
	GPTQ	5.95	7.82	-	5.09	6.95	**34.18**	19.22	18.50	22.27
	AWQ[4]	-	-	-	-	-	-	-	-	-
b3 g64	HQQ	**5.82**	**7.80**	**28.73**	**4.98**	**6.94**	**35.15**	**7.80**	**12.35**	**11.52**
	GPTQ	6.11	8.08	-	5.15	7.07	37.50	-	-	-
	AWQ	-	-	-	-	-	-	-	-	-
b3 g128	HQQ	**6.20**	8.39	**42.25**	**5.15**	**7.14**	**35.84**	**9.31**	**14.90**	**13.61**
	GPTQ	6.26	**8.29**	-	5.25	7.19	39.87	32.52	28.01	39.92
	AWQ	-	-	-	-	-	-	-	-	-
MXQ	4.75	5.28	7.06	22.58	4.69	6.51	32.05	6.10	9.44	9.57
	4.50	5.29	7.08	22.46	4.69	6.51	32.25	6.11	9.47	9.59
	4.25	5.31	7.13	22.78	4.71	6.54	31.69	6.29	9.76	9.73
	4.01	5.40	7.26	23.71	4.77	6.62	33.61	6.52	10.19	9.86
	3.76	5.48	7.41	23.82	4.85	6.71	33.26	6.97	10.98	10.41
	3.50	5.67	7.65	27.20	4.93	6.84	34.92	7.31	11.71	10.88
	3.00	7.16	9.64	44.94	5.78	8.02	63.92	16.72	26.35	25.00
	2.75	10.38	13.81	101.20	6.73	9.23	81.59	35.98	54.73	50.49

[1] Perplexity evaluation is based HuggingFace [1].

[2] We picked the first 1,100 entries in the en subset, validation split for this experiment.

[3] We used github.com/AutoGPTQ/AutoGPTQ for this experiment. The Llama-2-7B's perplexity metrics on the PTB dataset were excluded as they were extremely high compared to the Llama-2-13B and Llama-3-8B.

[4] The github.com/casper-hansen/AutoAWQ used by this experiment has no 3-bit quantization implementation.

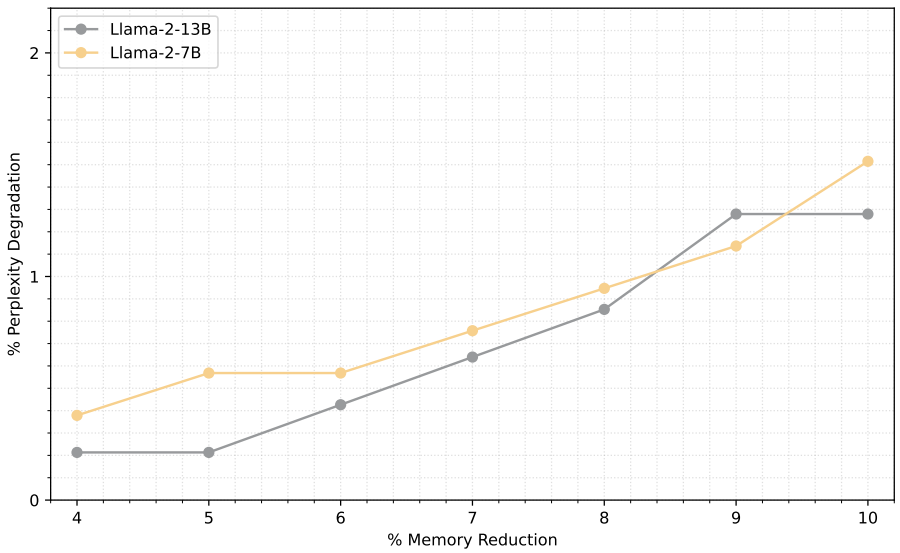

Fig. 3. The relationship between the Llama models' perplexity on WikiText2 and memory reduction. The tiny circles represent the 7-bit budgets (4.06, 4.10, 4.15, 4.19, 4.24, 4.28, 4.33), which correspond to the 7 memory reductions(from 4% to 10%) of the HHQ b4g32 baseline, equivalent of 4.51 in term of bit budget. All metrics are measured in percent. For the two Llama-2 models, MXQ achieves memory cutdown up to 8% while maintaining perplexity drop within 1%.

configurations. We focus on 3- and 4-bit quantization as these configurations better preserve the performance of LLMs [6]. For benchmarking, the proposed method is applied to the Llama [21] family models, including Llama-2-7B, Llama-2-13B, and Llama-3-8B.

Perplexity is utilised as the evaluation metric, *i.e.*, a stringent measure of LLM accuracy. Good results in perplexity evaluation generally reflect the true performance of the LLM [6]. To evaluate the effectiveness, we compare the perplexity of the proposed mixed quantization approach against other quantization implementations on WikiText-2 [16], C4 [19], and PTB [15]. The current state-of-the-art calibration-based and calibration-free quantization methods are adopted as the baselines, including HQQ [2], GPTQ [9] and AWQ [14]. These baselines are re-evaluated with 3- or 4-bit configurations according to Table 1, ensuring a fair comparison.

All the experiments were conducted on a Linux workstation with Nvidia RTX 4090 GPU and reproduced on another Linux server with Nvidia RTX 4080 GPU. The Llama models, as well as the three datasets, were fetched from HuggingFace.

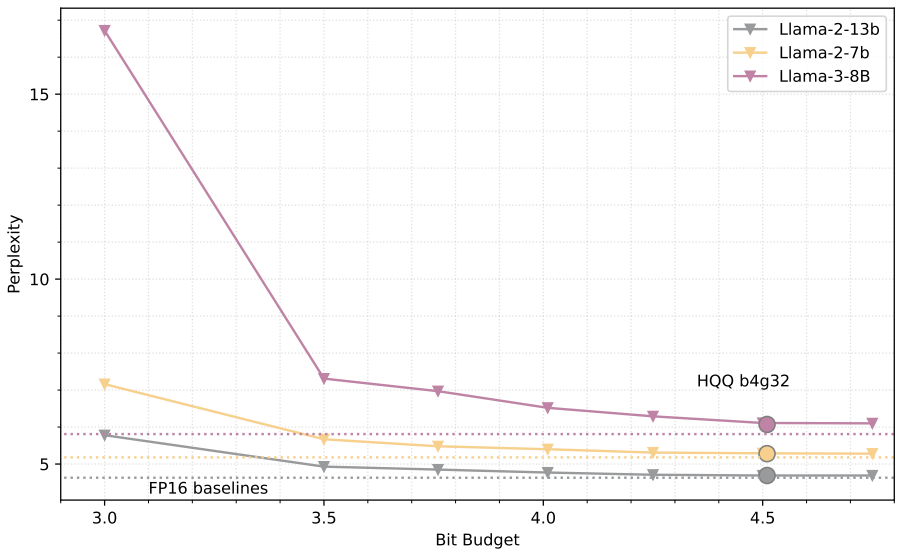

Fig. 4. This figure illustrates the relationship between the Llama models' perplexity on WikiText2 and bit budgets. The tiny triangles represent the 7-bit budgets 3.00, 3.50, 3.76, 4.01, 4.25, 4.50 and 4.75 respectively. The dotted lines indicate the FP16 baselines, while the circles denote the HQQ baselines at b4g32, corresponding to a bit budget of 4.50. The figure shows that the MXQ approaches the HQQ baselines as the bit budget increases but fails to surpass either the HQQ or FP16 baselines.

4.2 Experimental Results and Analysis

The perplexity metrics for various Llama models are presented in Table 2. The evaluation results for our mixed quantization method are shown as the "MXQ" rows at the bottom of the table. We tested eight-bit budgets, ranging from 2.75 to 4.75, as listed in the second column. The upper section of the table displays the results for HQQ, GPTQ, and AWQ under various settings. The first column indicates the combination of bit width and group size, while the second column denotes the quantization method.

As demonstrated in Table.2, MXQ offers a competitive trade-off between memory usage and accuracy. At a bit budget of 4.01, our method is very close to GPTQ 4-bit on WikiText2. Fig.4 illustrates the relationship between bit budget and WikiText2 perplexity on various Llama models, which indicates MXQ can achieve the same performance at bit budget equivalent to baseline. However, MXQ can't further boost quantization accuracy simply by increasing bit budget.

To investigate the relationship between perplexity and memory drop, we conducted additional experiments across seven-bit budgets (4.06, 4.10, 4.15, 4.19, 4.24, 4.28, 4.33), each corresponding to a 4–10% reduction in memory consumption relative to the best-performing quantization configuration, b4g32. As illus-

trated in Fig. 3, MXQ achieves up to an 8% reduction in memory usage for the Llama-2 models while keeping the WikiText2 perplexity loss within 1%.

5 Conclusion

The mixed quantization strategy is an effective approach to balancing model accuracy and memory consumption in an orthogonal manner. To address the vacancy of such a method, we proposed a novel method called MXQ, which can further reduce memory usage with minimal degradation in accuracy. This method offers a new technique for efficiently quantizing complex large language models(LLMs). Incorporating this strategy into the repository of LLM quantization utilities would be a valuable addition, enriching the tools available for optimizing large language models.

Acknowledgments. This study utilizes HiGHS Linear Programming Solver to find optimal quantization configurations and HQQ to quantize Llama family models.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Perplexity of fixed-length models, https://huggingface.co/docs/transformers/main/en/perplexity
2. Badri, H., Shaji, A.: Half-quadratic quantization of large machine learning models (November 2023), https://mobiusml.github.io/hqq_blog/
3. Baji, T.: Evolution of the GPU Device widely used in AI and Massive Parallel Processing. In: 2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM). pp. 7–9 (Mar 2018). https://doi.org/10.1109/EDTM.2018.8421507, https://ieeexplore-ieee-org.ezproxy.aut.ac.nz/abstract/document/8421507?casa_token=qbiJv_aI6WUAAAAA:kFUjaPaXJTBM1GE5l4jthJFI0a5Jgp7-cSqYXj5UFLiiJTMraVaudflGOpPSYbcy63preas0JTSU
4. Dettmers, T., Lewis, M., Shleifer, S., Zettlemoyer, L.: 8-bit Optimizers via Block-wise Quantization (Jun 2022). https://doi.org/10.48550/arXiv.2110.02861, http://arxiv.org/abs/2110.02861, arXiv:2110.02861 [cs]
5. Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: QLoRA: Efficient Finetuning of Quantized LLMs (May 2023). https://doi.org/10.48550/arXiv.2305.14314, http://arxiv.org/abs/2305.14314, arXiv:2305.14314 [cs]
6. Dettmers, T., Zettlemoyer, L.: The case for 4-bit precision: k-bit Inference Scaling Laws. In: Proceedings of the 40th International Conference on Machine Learning. pp. 7750–7774. PMLR (Jul 2023), https://proceedings.mlr.press/v202/dettmers23a.html, iSSN: 2640-3498
7. Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.M., Chen, W., Yi, J., Zhao, W., Wang, X., Liu, Z., Zheng, H.T., Chen, J., Liu, Y., Tang, J., Li, J., Sun, M.: Parameter-efficient fine-tuning of large-scale pre-trained language models. Nature Machine Intelligence **5**(3), 220–235 (Mar 2023). https://doi.org/10.1038/s42256-023-00626-4, https://www.nature.com/articles/s42256-023-00626-4, publisher: Nature Publishing Group

8. Frantar, E., Alistarh, D.: Optimal brain compression: A framework for accurate post-training quantization and pruning. Advances in Neural Information Processing Systems **35**, 4475–4488 (2022)
9. Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D.: GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers (Mar 2023). https://doi.org/10.48550/arXiv.2210.17323, http://arxiv.org/abs/2210.17323, arXiv:2210.17323 [cs]
10. Guo, H., Greengard, P., Xing, E.P., Kim, Y.: LQ-LoRA: Low-rank Plus Quantized Matrix Decomposition for Efficient Language Model Finetuning (Jan 2024). https://doi.org/10.48550/arXiv.2311.12023, http://arxiv.org/abs/2311.12023, arXiv:2311.12023 [cs]
11. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
12. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Mathematical Programming Computation **10**(1), 119–142 (Mar 2018). https://doi.org/10.1007/s12532-017-0130-5, https://doi.org/10.1007/s12532-017-0130-5
13. Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling Laws for Neural Language Models (Jan 2020). https://doi.org/10.48550/arXiv.2001.08361, http://arxiv.org/abs/2001.08361, arXiv:2001.08361 [cs, stat]
14. Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., Han, S.: AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv preprint arXiv:2306.00978 (2023), https://arxiv.org/abs/2306.00978
15. Marcus, M., Kim, G., Marcinkiewicz, M.A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., Schasberger, B.: 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), https://aclanthology.org/H94-1020
16. Merity, S., Xiong, C., Bradbury, J., Socher, R.: Pointer Sentinel Mixture Models (Sep 2016). https://doi.org/10.48550/arXiv.1609.07843, http://arxiv.org/abs/1609.07843, arXiv:1609.07843 [cs]
17. Nagel, M., Baalen, M.v., Blankevoort, T., Welling, M.: Data-Free Quantization Through Weight Equalization and Bias Correction. pp. 1325–1334 (2019)
18. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T.: A White Paper on Neural Network Quantization (Jun 2021). https://doi.org/10.48550/arXiv.2106.08295, http://arxiv.org/abs/2106.08295, arXiv:2106.08295 [cs]
19. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research **21**(140), 1–67 (2020), http://jmlr.org/papers/v21/20-074.html
20. Reddi, V.J., Cheng, C., Kanter, D., Mattson, P.a.e.: MLPerf Inference Benchmark. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). pp. 446–459 (May 2020). https://doi.org/10.1109/ISCA45697.2020.00045
21. Touvron, H., Martin, L., Stone, K.e.a.: Llama 2: Open Foundation and Fine-Tuned Chat Models (Jul 2023). https://doi.org/10.48550/arXiv.2307.09288, http://arxiv.org/abs/2307.09288, arXiv:2307.09288 [cs]