# MATRYOSHKA QUANTIZATION

**Pranav Ajit Nair*** **Puranjay Datta*** **Jeff Dean** **Prateek Jain** **Aditya Kusupati**
Google DeepMind *Equal Contribution
{pranavajitnair, kusupati}@google.com

## ABSTRACT

Quantizing model weights is critical for reducing the communication and inference costs of large models. However, quantizing models – especially to low precisions like int4 or int2 – requires a trade-off in model quality; int2, in particular, is known to severely degrade model quality. Consequently, practitioners are often forced to maintain multiple models with different quantization levels or serve a single model that best satisfies the quality-latency trade-off. On the other hand, integer data types, such as int8, inherently possess a nested (Matryoshka) structure where smaller bit-width integers, like int4 or int2, are nested within the most significant bits. Leveraging this insight, in this paper, we propose Matryoshka Quantization (MatQuant), a novel multi-scale quantization technique that alleviates the aforementioned challenge. This technique allows us to train and maintain a single quantized model but serve it with the precision demanded by the deployment. Furthermore, leveraging MatQuant's co-training and co-distillation regularization, int2 precision models extracted by MatQuant outperform standard int2 quantization by up to to 4% and 7% with OmniQuant and QAT as base algorithms respectively.

## 1 INTRODUCTION

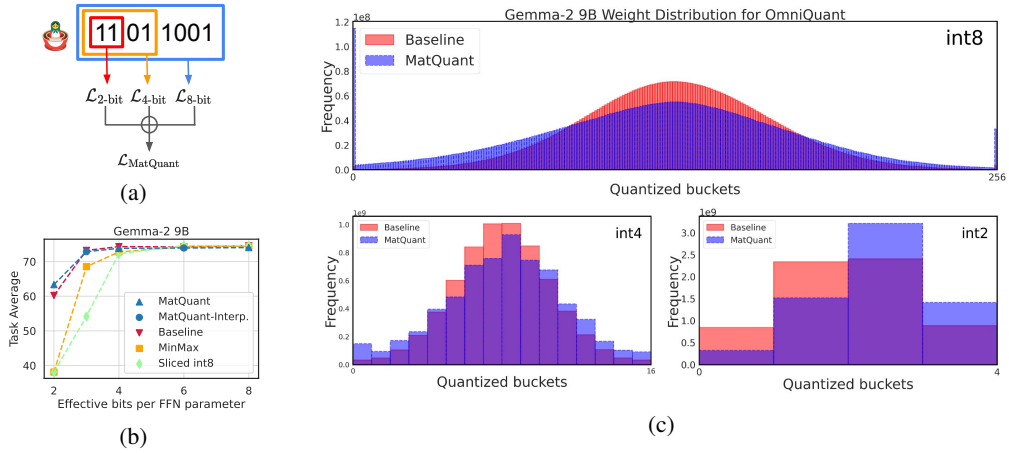


Figure 1: (a) MatQuant is a multi-scale quantization training technique using the inherent Matryoshka structure of int8 $\rightarrow$ int4 $\rightarrow$ int2. (b) Empirical gains of MatQuant on downstream tasks, especially $> 8\%$ for int2, on Gemma-2 9B with OmniQuant. (c) The right-shifted quantized weight distribution as a consequence of MatQuant's training mechanism that maximises accuracies across all precisions.

Due to their impressive performance, there is a strong push to deploy deep learning models, particularly large language models (LLMs) (G Team et al., 2024; Dubey et al., 2024; Achiam et al., 2023) in a large number of scenarios. Due to auto-regressive nature of LLMs, decode latency tends to dominate inference cost. Decode latency itself is dominated by communication cost of transferring model

weights from high-bandwidth memory (HBM) to the SRAM or due to transferring weights/activations in a distributed cluster.

Quantizing weights and/or activations can significantly reduce the overall communication load and is, therefore, one of the most popular techniques for reducing inference costs (Dettmers et al., 2022). While floating-point representations are standard for training, integer data types such as int8, int4, and int2 are appealing alternatives for inference. However, current methods for quantizing to these varying integer precisions typically treat each target precision as an independent optimization problem, leading to a collection of distinct models rather than a single, versatile one. Furthermore, quantizing to extremely low precisions like int2 is known to be highly inaccurate. In this work, we pose the question of whether both of the above challenges can be addressed; that is, can we train a single model from which we can extract multiple accurate lower-precision models? We answer this question in the affirmative by introducing Matryoshka Quantization ($\mathrm{MatQuant}$), a novel multi-scale training method that leverages the inherent nested (Matryoshka) structure (Kusupati et al., 2022) within integer data types (Figure 1a). Specifically, *slicing* the most significant bits (MSBs) of an int8-quantized weight can directly yield an int4 or int2 model. Existing quantization techniques often neglect this structure, which limits the potential for multi-scale adaptable models operating at various bit-widths with optimal performance.

Instead, $\mathrm{MatQuant}$ simultaneously optimizes model weights across multiple precision levels (e.g., int8, int4, int2). At a high level, we represent each model parameter at different precision levels using shared MSBs, and then jointly optimize the loss for each precision level. This allows us to develop a single quantized model that can effectively operate at any of the chosen bit-widths, offering a spectrum of accuracy-vs-cost options. $\mathrm{MatQuant}$ is a general-purpose technique, applicable to most learning-based quantization methods, such as Quantization Aware Training (QAT) (Jacob et al., 2018) and OmniQuant (Shao et al., 2023).

We demonstrate the efficacy of $\mathrm{MatQuant}$ when applied to quantizing the Feed-Forward Network (FFN) parameters of standard LLMs (Gemma-2 2B, 9B, and Mistral 7B) (Vaswani et al., 2017) – typically, FFN is the main latency block hence the focus on improving the most significant component's latency. Our results show that $\mathrm{MatQuant}$ produces int8 and int4 models with comparable accuracy to independently trained baselines, despite the benefit of shared model parameters. Critically, the int2 models generated by $\mathrm{MatQuant}$ significantly outperform their individually trained counterparts, with $4\%$ higher accuracy on downstream tasks (Figure 1b). We also extend $\mathrm{MatQuant}$ to quantize all weights of a Transformer layer. In Figure 1c, we find that quantizing with $\mathrm{MatQuant}$ shifts the quantized weight distribution toward higher values, contributing to improved int2 performance. Finally, in Section G, we also demonstrate that using an extra bit to represent outliers significantly boosts the performance for our sliced int2 models.

Beyond improving chosen precision performance, $\mathrm{MatQuant}$ allows for seamless extraction of interpolative bit-widths, such as int6 and int3. $\mathrm{MatQuant}$ also admits a dense accuracy-vs-cost trade-off by enabling layer-wise Mix'n'Match of different precisions. Therefore, even if the hardware only supports int4 and int2, it's possible to serve models at various effective precisions, tailored to the deployment environment. Overall, $\mathrm{MatQuant}$ and its variants present a significant step toward developing multi-scale models with high flexibility and performance, pushing the boundaries of low-bit quantization for efficient LLM inference.

## 2 MATRYOSHKA QUANTIZATION

In this section, we elaborate on our novel proposed approach, $\mathrm{MatQuant}$ with preliminaries behind QAT and OmniQuant covered in Appendix B.

$\mathrm{MatQuant}$ is a general purpose framework to develop a single model that can do well at any precision. It is a multi-scale training technique that works with most learning-based quantization schemes like QAT and OmniQuant discussed earlier. At its core, taking inspiration from Kusupati et al. (2022), $\mathrm{MatQuant}$ optimizes the quantization loss for several target bit-widths jointly.

To have a single model for various integer precisions, we nest smaller bit-widths into large ones – leveraging the inherent Matryoshka nature of the integer data type. So, if we want to extract a $r$-bit model from a $c$-bit model ($0 < r < c$), we can just *slice out* the $r$ most significant bits (MSBs) – using a right shift, followed by a left shift of the same order. Formally, the $S(q^c, r)$ operator slices

the most significant $r$ bits from a $c$-bit quantized vector $q^c$:

$$S(q^c, r) = \text{clamp}\left(\left\lfloor \frac{q^c}{2^{c-r}} \right\rceil, 0, 2^r - 1\right) * 2^{c-r} \tag{1}$$

Once we have this structure, we can optimize for several precisions by slicing the MSBs from the largest bit-width we are optimizing for. Let $R = \{r_1, r_2, ..., r_K\}$ be the bit-widths we want to optimize for, $Q(\cdot, )$ represent the quantization function of the base algorithm (i.e., any learning-based quantization scheme), $\mathcal{L}(\cdot)$ represent the loss function pertaining to the base algorithm, $F(\cdot)$ represent the forward pass required to compute the loss, $\theta$ represent the set of model/auxiliary parameters we are optimizing for and let $W_F$ represent the model parameters. $\mathrm{MatQuant}$'s overall objective can be formulated as follows:

$$\min_P \frac{1}{N} \sum_{i \in [N]} \sum_{r \in R} \lambda_r \cdot \mathcal{L}\left(F(S(Q(\theta, c), r), x_i'), y_i'\right) \tag{2}$$

where $y_i' = y_i$ for QAT and $y_i' = F_l(W_F^l, X_l^i)$ for OmniQuant, and $x_i' = x_i$ for QAT and $x_i' = X_l^i$ for OmniQuant. $\lambda_r$ is the loss reweighing factor for bit-width $r$.

In this work, we default to training $\mathrm{MatQuant}$ with three bit-widths, $R = \{8, 4, 2\}$, and subsequently perform a grid search over $\lambda_r$. This process aims to optimize performance such that the model performs well across all targeted precision levels. Further, while the focus of this paper is primarily on integer data types, we discuss the possibility of extending $\mathrm{MatQuant}$ to floating-point representations in Section D.5.

A key point to note is that $\mathrm{MatQuant}$ primarily alters the quantized weight distributions across precision levels compared to the base quantization algorithm (OmniQuant or QAT). Figure 1c illustrates the differences in the quantized weight histograms obtained with and without $\mathrm{MatQuant}$ on Gemma-2 9B using OmniQuant. Upon close observation, we find that all the distributions of $\mathrm{MatQuant}$ are shifted to the right; that is, weights quantized with $\mathrm{MatQuant}$ tend to use more higher-valued weights. While this might not significantly impact int8 or even int4 models, int2 models benefit from utilizing more of the possible quantized weights compared to the baseline. Because int2 favors higher-valued weights, this effect propagates to higher-valued weights for int4, and then to int8. This observation highlights the potential overparameterization and freedom in the int8 data type to accommodate the more stringent needs of int2 during joint training. We further explore the effects of this phenomenon in Section D.3 to develop a better standalone quantization technique for a single target precision.

#### 2.0.1 Interpolative Behavior

**Slicing.** Although we explicitly train $\mathrm{MatQuant}$ for three precisions (int8, int4, int2), we find that the resulting model, when quantized to interpolated bit-widths like int6 & int3 by slicing (Eq. 1) the int8 model, performs on par with a baseline trained explicitly for that precision. It is also significantly better than slicing an int8 quantized model. We attribute this strong interpolation in bit-width space to $\mathrm{MatQuant}$, and present more results in Sections 3.1 & C.1.

**Mix'n'Match.** $\mathrm{MatQuant}$ also enables the use of different precisions at different layers through layer-wise Mix'n'Match (Devvrit et al., 2023), even though we never trained for these combinatorial possibilities. These large number of models, obtained at no cost, densely span the accuracy-vs-memory trade-off. Section 3.2 for detailed experiments.

## 3 Experiments

In this section, we present an empirical evaluation of $\mathrm{MatQuant}$ working with two popular learning-based quantization methods: OmniQuant (Section 3.1) and QAT (Section C.1). We demonstrate $\mathrm{MatQuant}$'s efficiency on Transformer-based LLMs. Unless otherwise mentioned, our primary focus is on weight quantization within the parameter-intensive FFN blocks of the Transformer layer.

For our experiments, we chose the default target quantization precisions to be int8, int4, and int2. Furthermore, we showcase the interpolative nature of $\mathrm{MatQuant}$ through evaluations on int6 and int3, as well as its elastic ability to densely span the accuracy-vs-cost trade-off using layer-wise Mix'n'Match (Section 3.2). Finally, we ablate on improving the performance of $\mathrm{MatQuant}$ (Sections D.1 and D.2) and extend $\mathrm{MatQuant}$ to the quantization of FFN and Attention parameters. (Section D.3). During the process of extending $\mathrm{MatQuant}$ to all Transformer parameters, we uncovered an interesting

Table 1: MatQuant with OmniQuant across Gemma-2 2B, 9B and Mistral 7B models. MatQuant performs on par with the baseline for int4 and int8 while significantly outperforming it for int2. Even the int3, int6 models obtained for free through interpolation from MatQuant perform comparably to the explicitly trained baselines. Task Avg. is average accuracy on the evaluation tasks (↑) while log pplx (perplexity) is computed on C4 validation set (↓).

| Data type | Method | Gemma-2 2B | | Gemma-2 9B | | Mistral 7B | |
|---|---|---|---|---|---|---|---|
| | OmniQuant | Task Avg. | log pplx. | Task Avg. | log pplx. | Task Avg. | log pplx. |
| bfloat16 | | 68.21 | 2.551 | 74.38 | 2.418 | 73.99 | 2.110 |
| int8 | Baseline | 68.25 | 2.552 | 74.59 | 2.418 | 73.77 | 2.110 |
| | MatQuant | 68.02 | 2.570 | 74.05 | 2.438 | 73.65 | 2.125 |
| int4 | Sliced int8 | 62.87 | 2.730 | 72.26 | 2.480 | 38.51 | 4.681 |
| | Baseline | 67.03 | 2.598 | 74.33 | 2.451 | 73.62 | 2.136 |
| | MatQuant | 66.58 | 2.618 | 73.83 | 2.491 | 73.06 | 2.153 |
| int2 | Sliced int8 | 39.78 | 17.030 | 38.11 | 15.226 | 37.29 | 11.579 |
| | Baseline | 51.33 | 3.835 | 60.24 | 3.292 | 59.74 | 3.931 |
| | MatQuant | **52.37** | **3.800** | **63.35** | **3.187** | **62.75** | **3.153** |
| int6 | Sliced int8 | 67.72 | 2.497 | 74.64 | 2.353 | 73.00 | 2.071 |
| | Baseline | 68.06 | 2.554 | 74.23 | 2.420 | 74.10 | 2.112 |
| | MatQuant | 67.52 | 2.574 | 73.92 | 2.440 | 73.63 | 2.127 |
| int3 | Sliced int8 | 41.35 | 6.024 | 54.18 | 3.977 | 39.21 | 10.792 |
| | Baseline | 64.37 | 2.727 | 73.23 | 2.549 | 71.68 | 2.211 |
| | MatQuant | 64.47 | 2.618 | 72.87 | 2.607 | 71.16 | 2.238 |

hybrid quantization algorithm (between Baseline and MatQuant). Section D.3 further details this method, called Single Precison MatQuant, which stabilizes the otherwise QAT baseline for all the Transformer weights. Finally, we also discuss extending MatQuant beyond integer data types (Section D.5 and the considerations for effective deployment on current hardware (Section D.4). Further training and fine-grained evaluation details are in the Appendix.

### 3.1 MatQuant WITH OMNIQUANT

Table 1 shows the efficacy of MatQuant when used with FFN-only OmniQuant and compared to explicitly trained OmniQuant baselines for the target precisions, i.e., int8, int4, and int2, across all the models. While the average downstream accuracy of MatQuant for int8 and int4 quantization is within 0.5% of the corresponding independently trained baselines, the int2 quantized models of MatQuant are 1.04%, 3.11%, and 3.01% more accurate for Gemma-2 2B, 9B, and Mistral 7B, respectively. Similar trends and improvements follow when measuring performance through validation log perplexity. Further, the quantized int4 and int2 models *sliced* from the int8 OmniQuant baseline suffer a significant drop in accuracy around int4, demonstrating that the nested structure of int8 is not well utilized.

**Sliced Interpolation.** Beyond the target quantization granularities, MatQuant allows for interpolation to bit-widths not optimized during training. We find that the accuracy of the int6 and int3 models obtained by slicing the MatQuant models is comparable to explicitly trained baselines for both precisions.

### 3.2 LAYERWISE MIX'N'MATCH

MatQuant enables another form of elastic and interpolative behavior through Mix'n'Match. Mix'n'Match provides a mechanism to obtain a combinatorial number of strong models by using different quantization granularities, from the training target bit-widths, across layers. Figure 2 shows the ability of Mix'n'Match to densely span the accuracy-vs-bits-per-FFN-parameter (memory/cost) trade-off for the Gemma-2 9B model trained using MatQuant with OmniQuant. While there are many more feasible models, we only showcase the best models obtained through the strategy described in Appendix F. Interestingly, the Mix'n'Match model, with a sub-4-bit effective width, is more accurate than the 4-bit sliced model. This opens up possibilities for effective serving depending on hardware support (Section D.4) .

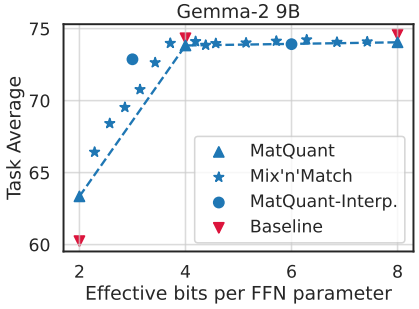


Figure 2: Mix'n'Match on Gemma-2 9B model trained using MatQuant with OmniQuant allows elastic accuracy-vs-cost model extraction for free during deployment.

## REFERENCES

AmirAli Abdolrashidi, Lisa Wang, Shivani Agrawal, Jonathan Malmaud, Oleg Rybakov, Chas Leichner, and Lukasz Lew. Pareto-optimal quantized resnet is mostly 4-bit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3091–3099, 2021.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

Harshavardhan Adepu, Zhanpeng Zeng, Li Zhang, and Vikas Singh. Framequant: Flexible low-bit quantization for transformers. *arXiv preprint arXiv:2403.06082*, 2024.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *CoRR*, abs/2404.00456, 2024. doi: 10.48550/ARXIV.2404.00456. URL https://doi.org/10.48550/arXiv.2404.00456.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6239. URL https://doi.org/10.1609/aaai.v34i05.6239.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36, 2024.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. Efficientqat: Efficient quantization-aware training for large language models. *CoRR*, abs/2407.11062, 2024. doi: 10.48550/ARXIV.2407.11062. URL https://doi.org/10.48550/arXiv.2407.11062.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 2924–2936. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1300. URL https://doi.org/10.18653/v1/n19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018. URL http://arxiv.org/abs/1803.05457.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. *Advances in neural information processing systems*, 28, 2015.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35: 30318–30332, 2022.

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.

F Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, Prateek Jain, et al. Matformer: Nested transformer for elastic inference. *arXiv preprint arXiv:2310.07707*, 2023.

Dayou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 102–116. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.7. URL https://doi.org/10.18653/v1/2024.acl-long.7.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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.

Gemini G Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Gemma-Team. Gemma 2: Improving open language models at a practical size. *ArXiv*, abs/2408.00118, 2024. URL https://api.semanticscholar.org/CorpusID:270843326.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL https://doi.org/10.48550/arXiv.2310.06825.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=0jpbpFia8m.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

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.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. LLM-QAT: data-free quantization aware training for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pp. 467–484. Association for Computational Linguistics, 2024a. doi: 10.18653/V1/2024.FINDINGS-ACL.26. URL https://doi.org/10.18653/v1/2024.findings-acl.26.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: LLM quantization with learned rotations. *CoRR*, abs/2405.16406, 2024b. doi: 10.48550/ARXIV.2405.16406. URL https://doi.org/10.48550/arXiv.2405.16406.

Yuexiao Ma, Huixia Li, Xiawu Zheng, Feng Ling, Xuefeng Xiao, Rui Wang, Shilei Wen, Fei Chao, and Rongrong Ji. Affinequant: Affine transformation quantization for large language models. *arXiv preprint arXiv:2403.12544*, 2024.

Pranav Ajit Nair and Arun Sai Suggala. Cdquant: Accurate post-training weight quantization of large pre-trained models using greedy coordinate descent. *CoRR*, abs/2406.17542, 2024. doi: 10.48550/ARXIV.2406.17542. URL https://doi.org/10.48550/arXiv.2406.17542.

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. *Journal of machine learning research*, 21(140):1–67, 2020.

Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pp. 1746–1754. PMLR, 2014.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8732–8740. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6399. URL https://doi.org/10.1609/aaai.v34i05.6399.

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. *arXiv preprint arXiv:2308.13137*, 2023.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=PxoFut3dWW.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. URL https://api.semanticscholar.org/CorpusID:13756489.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Haichao Yu, Haoxiang Li, Humphrey Shi, Thomas S. Huang, and Gang Hua. Any-precision deep neural networks. *ArXiv*, abs/1911.07346, 2019. URL https://api.semanticscholar.org/CorpusID:208138922.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1472. URL https://doi.org/10.18653/v1/p19-1472.

## A Related Work

Model weight quantization is an extremely powerful and prevalent technique for making resource-intensive neural networks suitable for deployment constraints – especially modern-day LLMs. Quantization algorithms can be categorized as either learning-free or learning-based. Learning-free methods use limited data to calibrate model parameters without relying on gradient descent. Learning-based methods, however, utilize gradient descent to update either model parameters or auxiliary parameters to aid in quantization.

**Learning-free Quantization Methods.** Naive quantization methods, such as MinMax, absmax, and zero-point quantization, aim to directly map the range of model weights to the target bit-width – see (Dettmers et al., 2022) for a detailed background. Dettmers et al. (2022) further improved this by identifying the need to handle outliers with higher precision than the rest of the model weights. The core principle of more recent learning-free quantization methods remains similar while improving various aspects of it and using small amounts of data for calibration. For example, GPTQ (Frantar et al., 2022) improves upon min-max quantization by iterating over all the coordinates, quantizing them one at a time, and updating the remaining full-precision coordinates to minimize the layer-wise activation reconstruction error. AWQ (Lin et al., 2023), SmoothQuant (Xiao et al., 2023), and AffineQuant (Ma et al., 2024) scale the weights and activations to reduce outliers, thus making them easier to quantize. QuIP (Chee et al., 2024), FrameQuant (Adepu et al., 2024), and QuaRoT (Ashkboos et al., 2024) multiply the weights and activations by orthonormal matrices before quantizing to reduce the number of outliers. SqueezeLLM (Kim et al., 2024) uses clustering to obtain the optimal buckets for quantization, and CDQuant (Nair & Suggala, 2024) improves upon GPTQ by greedily choosing the coordinates to descend along. While learning-free methods are inexpensive and work well at higher bit-widths, they are often suboptimal in the low-precision regime, which benefits greatly from learning-based techniques.

**Learning-based Quantization Methods.** Quantization Aware Training (QAT) (Jacob et al., 2018; Abdolrashidi et al., 2021) is a logical approach to ensure that models are easy to quantize during inference while retaining high accuracy. However, because QAT involves updating all the model parameters, its adoption for LLMs has been limited. Several recent works improve the performance and efficiency of QAT. LLM-QAT (Liu et al., 2024a) and BitDistiller (Du et al., 2024) enhance QAT with knowledge distillation from the full-precision model. EfficientQAT (Chen et al., 2024) minimizes the block-wise reconstruction error before performing end-to-end training. This significantly reduces the time it takes for QAT to converge. On the other hand, some techniques significantly reduce the overhead by learning only the auxiliary parameters, such as scaling factors and zero-points, that aid in quantization instead of updating the actual weight matrices. For example, OmniQuant (Shao et al., 2023) does not update the model parameters; instead, it learns additional scales and shifting parameters (that aid with quantization) through gradient descent over the block-wise reconstruction error and achieves better accuracy than most QAT techniques. Likewise, SpinQuant (Liu et al., 2024b) uses gradient descent to learn its rotation matrices. This class of learning-based quantization techniques (OmniQuant, SpinQuant, etc.) is widely adopted due to their appeal of achieving QAT-level accuracy at a fraction of the cost.

**Multi-scale Training.** Training across multiple data scales (resolutions) was heavily popularized in computer vision for both recognition and generation (Adelson et al., 1984; Lin et al., 2017; Denton et al., 2015). More recently, the paradigm of multi-scale training has shifted to models (Rippel et al., 2014; Yu et al., 2018; Kusupati et al., 2022; Devvrit et al., 2023), where the data remains the same, and models of varying capacity, all nested within one large model, are trained jointly. This joint, nested (Matryoshka-style) learning with varying model sizes results in a smooth accuracy-vs-compute trade-off and is beneficial in many downstream applications and real-world deployments. However, the most obvious structure with a nested nature is the bit structure of the integer data type. Given the success of multi-scale training for inputs, outputs, and model weights, it is imperative to explore it further for integer data types, especially in the context of quantization, which aids in the deployment of resource-intensive LLMs. Following this idea, Yu et al. (2019) have successfully trained a single model that can do well at any precision. However, the experiments were limited to ConvNets and small Neural Networks. In this paper, we extend the idea of nested precision to LLMs and show that it indeed works at scale. We also show that, for the first time, our models are quality neutral for intermediate precisions such as int3 and int6 that we never trained for, and densely span the accuracy-vs-bits trade-off. In Section D.3, we show that even to train models for a fixed target

precision, having loss over the sliced bits of an 8-bit model does better than training a model explicitly for that precision, indicating that $\mathrm{MatQuant}$ is a fundamentally better way to do low-bit quantization.

# B MATRYOSHKA QUANTIZATION

## B.1 PRELIMINARIES

### B.1.1 QUANTIZED AWARE TRAINING

Quantized Aware Training (QAT) learns a $c$-bit quantized model by optimizing for the end-to-end cross entropy loss using gradient descent. It uses the quantized weights for the forward pass and a straight through estimator (STE) (Bengio et al., 2013) to propagate gradients through the quantization operator during the backward pass.

To mathematically formulate QAT, we define MinMax quantization of a real-valued vector $w$ in $c$ bits as follows:

$$\begin{aligned} Q_{\text{MM}}(w, c) &= \text{clamp}\left(\left\lfloor \frac{w}{\alpha} + z \right\rceil, 0, 2^c - 1\right) \\ \alpha = \frac{\max(w) - \min(w)}{2^c - 1}, &\quad z = -\frac{\min(w)}{\alpha} \end{aligned} \tag{3}$$

where $Q_{\text{MM}}(w, c)$ is the $c$-bit quantized version of $w$, $\alpha$ is the scaling factor and $z$ is the zero point.

Let $W_F$ represent weights of a Transformer LLM and let $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ be a labelled dataset where $x_i$ and $y_i$ represent the input and output respectively. With $L_{\text{CE}}$ as the cross entropy loss, the optimization of QAT is:

$$\min_{W_F} \frac{1}{N} \sum_{i \in [N]} \mathcal{L}_{\text{CE}}\left(F(x_i; Q_{\text{MM}}\left(W_F, c\right)), y_i\right) \tag{4}$$

where $F(\cdot)$ represents the LLM's forward pass.

### B.1.2 OMNIQUANT

OmniQuant, unlike QAT, does not update the model parameters. Instead, it learns additional scaling and shifting parameters through gradient descent over layer-wise L2 error reconstruction. These auxiliary parameters aid with quantization. Similar to QAT, OmniQuant also uses a straight through estimator during optimization. However, unlike QAT, OmniQuant operates with limited data, making it much more attractive for resource-scarce settings.

OmniQuant adds two learnable scales, $\gamma$ and $\beta$, to MinMax quantization as follows:

$$\begin{aligned} Q_{\text{Omni}}(w, c) &= \text{clamp}\left(\left\lfloor \frac{w}{\alpha} + z \right\rceil, 0, 2^c - 1\right) \\ \alpha = \frac{\gamma \cdot \max(w) - \beta \cdot \min(w)}{2^c - 1}, &\quad z = -\frac{\beta \cdot \min(w)}{\alpha} \end{aligned} \tag{5}$$

OmniQuant also adds another set of learnable shifting and scaling parameters to the FFN's affine projections as follows:

$$XW + b \rightarrow ((X - \delta) \oslash s) \cdot Q_{\text{Omni}}(W \odot s) + b + \delta \cdot W \tag{6}$$

where $X \in \mathbb{R}^{n \times d}$ is the input to the affine transformation, $W \in \mathbb{R}^{d \times d_o}$ is the linear projection associated with the affine transformation, $b \in \mathbb{R}^{d_o}$ is the bias vector, $\delta \in \mathbb{R}^d$ and $s \in \mathbb{R}^d$ are learnable shift and scale parameters respectively.

With the goal of optimizing the layer-wise L2 error (where a layer consists of an Attention block followed by an FFN block), OmniQuant's overall objective can be portrayed as follows:

$$\min_{\gamma, \beta, \delta, s} ||F_l(W_F^l), X_l) - F_l(Q_{\text{Omni}}(W_F^l), X_l)||_2^2 \tag{7}$$

where $F_l(\cdot)$ represents the forward pass for a single layer $l$, $W_F^l$ represents the layer parameters and $X_l$ represents the layer's input. Note that the above objective is optimized independently for each of the $L$ Transformer layers.

# C EXPERIMENTS

In this section, we present an empirical evaluation of MatQuant working with two popular learning-based quantization methods: OmniQuant (Section 3.1) and QAT (Section C.1). We demonstrate MatQuant's efficiency on Transformer-based LLMs. Unless otherwise mentioned, our primary focus is on weight only quantization within the parameter-intensive FFN blocks of the Transformer layer.

For our experiments, we chose the default target quantization precisions to be int8, int4, and int2. Furthermore, we showcase the interpolative nature of MatQuant through evaluations on int6 and int3, as well as its elastic ability to densely span the accuracy-vs-cost trade-off using layer-wise Mix'n'Match (Section 3.2). Finally, we ablate on improving the performance of MatQuant (Sections D.1 and D.2) and extend MatQuant to the quantization of FFN and Attention parameters. (Section D.3). Further training and fine-grained evaluation details are in the Appendix.

**Models and Data.** We experiment with Gemma-2 (Gemma-Team, 2024) 2B, 9B, and Mistral 7B (Jiang et al., 2023) models. For OmniQuant experiments, we sample 128 examples with a sequence length of 2048 from the C4 dataset (Raffel et al., 2020) and train using a batch size of 4. We train for a total of 10M tokens for all models except the int2 baseline, where we train the model for 20M tokens (Shao et al., 2023). For QAT experiments, we sample a fixed set of 100M tokens from the C4 dataset and train all our models using a batch size of 16 and a sequence length of 8192 for a single epoch.

**Baselines.** For OmniQuant and QAT, our primary baselines (referred to as "Baseline" in the tables and figures) are models trained explicitly for a given precision. When interpolating the models trained with MatQuant for int6 and int3, we do not perform any additional training. However, the baselines are trained explicitly for 6 and 3 bits respectively. We also compare against a sliced int8 OmniQuant/QAT baseline model to the corresponding precision (referred to as "Sliced int8" in the tables).

Table 2: MatQuant with QAT across Gemma-2 2B, 9B and Mistral 7B models. MatQuant performs on par with the baseline for int4 and int8 while significantly outperforming it for int2. Even the int3, int6 models obtained for free through interpolation from MatQuant perform comparably to the explicitly trained baselines. Task Avg. is average accuracy on the evaluation tasks (↑) while log pplx (perplexity) is computed on C4 validation set (↓).

| Data type | Method | Gemma-2 2B | | Gemma-2 9B | | Mistral 7B | |
|---|---|---|---|---|---|---|---|
| | QAT | Task Avg. | log pplx. | Task Avg. | log pplx. | Task Avg. | log pplx. |
| bfloat16 | | 68.21 | 2.551 | 74.38 | 2.418 | 73.99 | 2.110 |
| int8 | Baseline | 67.82 | 2.458 | 74.17 | 2.29 | 73.48 | 2.084 |
| | MatQuant | 67.44 | 2.449 | 74.52 | 2.262 | 72.58 | 2.104 |
| int4 | Sliced int8 | 67.13 | 2.483 | 73.36 | 2.276 | 71.76 | 2.18 |
| | Baseline | 67.03 | 2.512 | 73.26 | 2.324 | 72.13 | 2.105 |
| | MatQuant | 66.59 | 2.499 | 73.24 | 2.429 | 71.99 | 2.148 |
| int2 | Sliced int8 | 39.27 | 10.217 | 40.40 | 7.259 | 37.41 | 9.573 |
| | Baseline | 47.74 | 3.433 | 56.02 | 2.923 | 54.95 | 2.699 |
| | MatQuant | **52.20** | **3.055** | **62.29** | **2.265** | **61.97** | **2.524** |
| int6 | Sliced int8 | 67.53 | 2.401 | 74.15 | 2.232 | 73.35 | 2.097 |
| | Baseline | 67.75 | 2.460 | 74.31 | 2.293 | 72.71 | 2.077 |
| | MatQuant | 67.33 | 2.453 | 74.30 | 2.265 | 72.59 | 2.106 |
| int3 | Sliced int8 | 59.56 | 2.882 | 68.70 | 2.512 | 64.33 | 2.493 |
| | Baseline | 61.75 | 2.678 | 69.9 | 2.43 | 68.82 | 2.197 |
| | MatQuant | 60.76 | 2.734 | 70.41 | 2.429 | 67.16 | 2.324 |

**Evaluation Datasets.** Following recent work (Frantar et al., 2022; Ma et al., 2024), we evaluate all the methods based on log perplexity and average zero-shot accuracy across a collection of downstream tasks. We use C4's test set to calculate perplexity, and for downstream evaluations, we test on ARC-c, ARC-e (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and Winogrande (Sakaguchi et al., 2020).

Table 3: Design choice ablation for loss re-weighting of the 3 target bit-widths (int8, int4, int2) that MatQuant explicitly optimizes. Note that MatQuant $(0, 0, 1) \equiv$ Single Precison MatQuant.

| Data type | Weightings | Gemma-2 2B | Gemma-2 9B | Mistral 7B |
|---|---|---|---|---|
| | | Task Avg. | | |
| int8 | $(0.1, 0.1, 1)$ | **68.02** | **74.05** | 73.27 |
| | $(0.2, 0.2, 1)$ | 67.91 | 73.91 | 73.44 |
| | $(0.3, 0.3, 1)$ | 68.01 | 73.88 | 73.56 |
| | $(0.4, 0.4, 1)$ | 67.95 | 73.84 | **73.65** |
| int4 | $(0.1, 0.1, 1)$ | 66.58 | 73.83 | 72.76 |
| | $(0.2, 0.2, 1)$ | 67.47 | 73.8 | 73.16 |
| | $(0.3, 0.3, 1)$ | 66.97 | 73.25 | 73.47 |
| | $(0.4, 0.4, 1)$ | **67.48** | **74.32** | **73.66** |
| int2 | $(0.1, 0.1, 1)$ | **52.37** | 63.35 | 63.25 |
| | $(0.2, 0.2, 1)$ | 51.88 | 64.04 | **63.99** |
| | $(0.3, 0.3, 1)$ | 51.05 | **64.1** | 63.6 |
| | $(0.4, 0.4, 1)$ | 51.69 | 61.98 | 62.75 |

### C.1 MatQuant WITH QAT

To further demonstrate the generality of MatQuant, we experiment on the same models using the popular QAT technique. Following the trend of experimental results with OmniQuant, we show in Table 2 that the models trained using MatQuant with QAT are comparable to the explicitly trained baselines for all the targeted bit-widths of int8 and int4. However, int2 quantized models using MatQuant are $4.46\%$, $6.27\%$, and $7.02\%$ more accurate for Gemma-2 2B, 9B, and Mistral 7B, respectively.

**Sliced Interpolation.** Models trained using MatQuant with QAT exhibit strong interpolative performance similar to that of MatQuant with OmniQuant. We find that the accuracy of the int6 and int3 models obtained by *slicing* the MatQuant models is comparable to explicitly trained baselines for both interpolated bit-widths.

While OmniQuant only trains the auxiliary parameters needed for quantization, QAT also updates the weight parameters. This potentially results in severe overfitting to the C4 subset used in the experiments. We observe this overfitting in all the experiments presented in Table 2, where the log perplexities improve for QAT compared to OmniQuant, while the downstream accuracies suffer. This also highlights the need for high-quality data for QAT to realize its benefits; otherwise, users are better off using resource-friendly methods like OmniQuant.

## D ABLATIONS AND DISCUSSION

In this section, we present design ablations to improve MatQuant. Section D.1 discusses the effect of non-uniform weighting across target precisions (int8, int4, int2), and Section D.2 explores enabling co-distillation of lower precision levels (int4, int2) from the highest precision quantized model (int8). During the process of extending MatQuant to all Transformer parameters, not just the FFN block, we uncovered an interesting hybrid quantization algorithm (between Baseline and MatQuant). Section D.3 further details this method, called Single Precison MatQuant, which stabilizes the otherwise QAT baseline for all the Transformer weights. Finally, we also discuss extending MatQuant beyond integer data types and the considerations for effective deployment on current hardware.

### D.1 WEIGHTINGS ($\lambda_r$) FOR MatQuant

Depending on the constraints, we may wish to maximize the accuracy of one of the target bit-widths in MatQuant. Equation 2 provides a general formulation of MatQuant that supports searching over the weight $\lambda_r$ for bit-width $r$. The results in Section C are with the weights that have balanced

Table 4: Design choice ablations for co-distillation within MatQuant. x $\rightarrow$ y represents distilling the y-bit model from the x-bit model. We note that the accuracy for int2 has significantly improved while minimally impacting the other bit-widths.

| | Gemma-2 9B | OmniQuant | | QAT | |
|---|---|---|---|---|---|
| Data type | Config. | Task Avg. | log pplx. | Task Avg. | log pplx. |
| int8 | $[8, 4, 2]$ | **74.05** | 2.438 | 74.52 | 2.262 |
| | $[8, 4, 8 \rightarrow 2]$ | 72.76 | 2.473 | 74.75 | 2.242 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 73.99 | **2.435** | **74.87** | **2.240** |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 73.85 | 2.437 | 74.81 | 2.240 |
| int4 | $[8, 4, 2]$ | **73.83** | 2.491 | 73.24 | 2.295 |
| | $[8, 4, 8 \rightarrow 2]$ | 72.65 | 2.519 | 73.76 | 2.279 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 73.63 | 2.486 | 73.77 | **2.276** |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 73.55 | **2.478** | **73.93** | 2.277 |
| int2 | $[8, 4, 2]$ | 63.35 | **3.187** | 62.29 | **2.660** |
| | $[8, 4, 8 \rightarrow 2]$ | 62.64 | 3.289 | 62.31 | 2.670 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 62.91 | 3.138 | **62.70** | 2.673 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | **64.32** | 3.227 | 62.60 | 2.670 |

performance across target precisions. Table 3 shows the weight multiplier ablation results for Gemma-2 2B, 9B, and Mistral 7B. We find that a higher relative value for $\lambda_2$ is essential in attaining good int2 performance. Increasing $\lambda_4, \lambda_8$ to improve int8 and int4 models often results in accuracy drop for the int2 models. In general, we can see that a higher relative weight for a specific precision results in increased accuracy for that bit-width. We can consider re-weighting as scaling the importance of the bits during training, and finding an optimal re-weighting recipe is an interesting research question.

### D.2 Co-distillation for MatQuant

Given the nested nature of the models trained using MatQuant, we explored co-distillation, where the outputs from a higher-precision model are used as the target for the lower-precision nested model, either in a standalone fashion or alongside the ground truth target (weighted equally). Table 4 shows the effects of co-distillation applied to MatQuant with both OmniQuant and QAT on Gemma-2 9B. While int8 and int4 show no significant improvement, the nested int2 model benefits substantially from the int8 supervision, reaching $0.97\%$ higher accuracy than the non-co-distilled MatQuant with OmniQuant. Co-distillation in MatQuant opens up avenues for interesting design choices that can further leverage the inherent nested structure of integer data types.

### D.3 Single Precison MatQuant

In Tables 1 and 2, MatQuant performs on par with the explicitly trained baselines for int4, int8, and the interpolated int3 and int6 precisions. However, the int2 models show a significant accuracy improvement. To investigate this, we conducted a simple ablation in MatQuant by removing the loss terms for int4 and int8 (i.e., $R = \{2\}$ in Equation 2 or setting $\lambda_4 = \lambda_8 = 0$) and present

Table 5: Single Precison MatQuant significantly improves upon the baseline for int2 and, at times, outperforms MatQuant. Crucially, int8 and int4 performances of Single Precison MatQuant experience a significant accuracy decrease (as shown in Tables 23 & 24) in Appendix L).

| int2 | Gemma-2 2B | | Gemma-2 9B | | Mistral 7B | |
|---|---|---|---|---|---|---|
| Method | Task Avg. | log pplx. | Task Avg. | log pplx. | Task Avg. | log pplx. |
| OmniQuant | 51.33 | 3.835 | 60.24 | 3.292 | 59.74 | 3.931 |
| S.P. MatQuant | **53.42** | **3.631** | **64.02** | **3.171** | **63.58** | **2.976** |
| MatQuant | 52.37 | 3.800 | 63.35 | 3.187 | 62.75 | 3.153 |
| QAT | 47.74 | 3.433 | 56.02 | 2.923 | 54.95 | 2.699 |
| S.P. MatQuant | 52.08 | **3.054** | **62.66** | **2.656** | 61.48 | **2.509** |
| MatQuant | **52.20** | 3.055 | 62.29 | 2.660 | **61.97** | 2.524 |

Table 6: Extending MatQuant with QAT to FFN + Attention parameters. Baseline QAT destabilizes for int2 and int3 but improves significantly through MatQuant & Single Precison MatQuant.

| Data type | Method | Gemma-2 9B | | Mistral 7B | |
|---|---|---|---|---|---|
| | QAT | Task Avg. | log pplx. | Task Avg. | log pplx. |
| bfloat16 | | 74.38 | 2.418 | 73.99 | 2.110 |
| int8 | Baseline | 74.61 | 2.353 | 73.73 | 2.091 |
| | MatQuant | 74.85 | 2.333 | 73.88 | 2.182 |
| int4 | Sliced int8 | 73.15 | 2.362 | 71.46 | 2.290 |
| | Baseline | 72.98 | 2.40 | 71.87 | 2.132 |
| | MatQuant | 74.01 | 2.396 | 71.44 | 2.441 |
| int2 | Sliced int8 | 38.97 | 23.467 | 35.06 | 10.640 |
| | Baseline | - | - | - | - |
| | S.P. MatQuant | **45.69** | **3.780** | 35.35 | 7.761 |
| | MatQuant | 44.19 | 3.826 | **38.36** | **10.971** |
| int6 | Sliced int8 | 74.49 | 2.290 | 73.61 | 2.104 |
| | Baseline | 74.65 | 2.357 | 73.72 | 2.093 |
| | MatQuant | 74.57 | 2.340 | 74.04 | 2.161 |
| int3 | Sliced int8 | 64.19 | 2.895 | 39.01 | 6.018 |
| | Baseline | - | - | - | - |
| | S.P. MatQuant | 67.68 | 2.520 | 67.59 | 2.335 |
| | MatQuant | 63.63 | 2.937 | 40.55 | 4.776 |

the results in Table 5. We call this version of MatQuant as Single Precison MatQuant. With Single Precison MatQuant, we observe a further boost of up to 1.05%, in the accuracy of int2 models at a $\sim$2% accuracy drop in the corresponding int4 and int8 models – int2 is still nested within int8. This improvement likely stems from the six additional bits available during MatQuant-style training to optimize the int2 representation.

In the case of Single Precison MatQuant, gradient descent is free to tune these six additional bits to improve the overall quality of the int2 model. In MatQuant, since we have additional losses to preserve the performance of the int4 and int8, the int2 performance is slightly worse than Single Precison MatQuant. However, since the int4 and int8 models are typically very close in accuracy to the bfloat16 model, MatQuant can shift some of the weights to improve the int2 model. As int4 and int8 models have substantially more quantized buckets than int2, we hypothesize that shifting some weights into adjacent buckets may not significantly affect their performance; however, it can significantly impact int2's performance. In fact, in the weight distributions presented in Fig 1c, we observe that MatQuant results in a model where larger number of weights are assigned to the higher-valued buckets. Conclusively, MatQuant and Single Precison MatQuant inherently seem to be a better way of performing low-bit quantization.

**FFN + Attention Weight Quantization.** We present results for FFN + Attention quantization for QAT in Table 6. For int8, int4 and the interpolated int6 model, MatQuant performs on par with the *Baseline*. However, we found int2 and int3 to be very unstable while quantizing both, the FFN and the Attention parameters. Most recent works that do QAT for both the blocks Chen et al. (2024); Liu et al. (2024a); Du et al. (2024) either do some form of warm starting for the quantized parameters, or have additional distillation and auxiliary loss functions. In the naive setup of minimizing the loss with respect to the ground truth, we find QAT to be very unstable at lower precisions. On the other hand, both MatQuant and Single Precison MatQuant are very stable further highlighting the benefits brought by MatQuant style training.

### D.4 Deployment Considerations

Current hardware accelerators have native support for serving int8 and int4 quantized models. Additionally, custom-implemented CUDA kernels can can support various low-precision bit-widths, like int2 and int3 (Chee et al., 2024; Frantar et al., 2022). MatQuant can generate a large number

of models at inference time. Depending on the serving environment, we can choose between Mix'n'Match models and homogeneous sliced models. For example, suppose the serving environment has a memory constraint equivalent to an int3 model but lacks optimized support for int3, while supporting int2. In this case, a Mix'n'Match model with a small performance drop when compared to the sliced int3 model could be deployed. More generally, as depicted in Figure 2, $\mathrm{MatQuant}$ densely spans the memory-versus-accuracy curve and can be leveraged to obtain performant model for several serving constraints. $\mathrm{MatQuant}$ can enable further research on hardware software co-design to effectively support elastic bit-widths on-the-fly during inference.

### D.5 Extension to Floating Point

Extending $\mathrm{MatQuant}$ to floating-point representations, such as FP8 and FP4, presents significant challenges. Given that the exponent is encoded within the bit representation and contributes to the value as a power of 2 (i.e., effectively $\log_2$), slicing it results in buckets whose sizes increase exponentially, unlike the integer case, where bucket sizes are constant. For example, slicing the first two bits from int8 yields buckets of $0$, $64$, $128$, $192$. Here, the bucket size $(64)$ is constant; however, this would not be the case when slicing two exponent bits from FP8. This is a promising avenue for future research that could further unlock the benefits of $\mathrm{MatQuant}$, even during large-scale pretraining.

## E Particulars of the Slicing Operation.

To extract a $r$-bit model from a $c$-bit model, we start by slicing out the most significant $r-1$ bits. We use $1$ for the $r^{\text{th}}$ bit if the $(r+1)^{\text{th}}$, else, we use $0$. This is captured by the round function in Equation 1 and is done to push values to higher buckets as we expect them to be more informative (Sun et al., 2024). For example, consider the the unsigned int8 value $53$. The first two MSBs are $0$s. Naively slicing them would round down $53$ to $0$, however, we want to round it up to $1$. Since the bit corresponding to $32$ is set, i.e., the $(r+1)^{\text{th}}$ MSB, instead of rounding $53$ down to $0$, we round it up to $1$.

The clamp$(\cdot)$ operation is also equally important. The rounding operation in Equation 1 will round $240$ down to $4$, however, unsigned int2 operates with only $0, 1, 2, 3$. clamp$(\cdot)$ here would make sure that $4$ is clamped down to $3$.

## F Addition Training Details

We run all our experiments on TPUv5e chips. For OmniQuant experiments, we use a constant learning rate of $1e-3$ and for QAT experiments, we linearly warmup the learning rate to $1e-5$ for 150 and use a consine decay schedule thereafter. For OmniQuant experiments, we sample 128 examples with a sequence length of 2048 from the C4 dataset (Raffel et al., 2020) and train using a batch size of 4. We train for a total of 10M tokens for all models except the int2 baseline, where we train the model for 20M tokens (Shao et al., 2023). For Co-distillation experiments where OmniQuant is the base algorithm, we train for a total of 8.3M tokens. For QAT experiments, we sample a fixed set of 100M tokens from the C4 dataset and train all our models using a batch size of 16 and a sequence length of 8192 for a single epoch. For Attn + FFN experiments with QAT, we sample a fixed set of 300M tokens from C4 and train with a batch size of 16 for a single epoch. We use $(\lambda_8, \lambda_4, \lambda_2) = (0.1, 0.1, 1.0)$ for all our Gemma experiments unless otherwise stated. In the case of Mistral 7B, for OmniQuant experiments, we use $(\lambda_8, \lambda_4, \lambda_2) = (0.4, 0.4, 1.0)$, and for QAT experiments we use $(\lambda_8, \lambda_4, \lambda_2) = (0.2, 0.2, 1.0)$. For all our $\mathrm{Extra\ Precison\ MatQuant}$ experiments, we use $(\lambda_8, \lambda_4, \lambda_2) = (1.0, 1.0, 1.0)$.

**Mix'n'Match** For a fixed effective bits-per-FFN layer, where each layer was quantized to either int2, int4, or int8, we explored four different quantization strategies: Pyramid, Reverse Pyramid, Increasing, and Decreasing. In the Pyramid strategy, the initial and final layers were quantized to int2, the central layers to int8, with int4 serving as an intermediate step. The Reverse Pyramid strategy followed the opposite approach, assigning int8 to the initial and final layers, int2 to the central layers, and int4 in between. The Increasing and Decreasing strategies assigned bit precision in ascending and descending order, respectively, across the layers. Our experimental results demonstrated that, for a given effective bits per FFN layer, the Pyramid strategy consistently outperformed the others.

Allocating higher precision (int8) to the middle layers helped preserve critical information, while the initial and final layers performed adequately with lower bit precision (int2 and int4), leading to a more efficient and effective quantization scheme.

Table 7: Results comparing MatQuant with Extra Precison MatQuant for Gemma-2 2B, 9B, and Mistral 7B, with OmniQuant as the base algorithm. We find that for the 2-bit model, having an extra bucket significantly boosts the performance, however, this is not the case with the higher precisions.

| Method | Gemma-2 2B | | | Gemma-2 9B | | | Mistral 7B | | |
|---|---|---|---|---|---|---|---|---|---|
| OmniQuant | Avg. Bits | Task Avg. | log pplx. | Avg. Bits | Task Avg. | log pplx. | Avg. Bits | Task Avg. | log pplx. |
| bfloat16 | | 68.21 | 2.551 | | 74.38 | 2.418 | | 73.99 | 2.110 |
| MatQuant | 8 | 68.02 | 2.570 | 8 | 74.05 | 2.438 | 8 | 73.65 | 2.125 |
| Extra Precison MatQuant | 8 | 67.85 | 2.580 | 8 | 74.33 | 2.446 | 8 | 73.46 | 2.132 |
| MatQuant | 4 | 66.58 | 2.618 | 4 | 73.83 | 2.491 | 4 | 73.06 | 2.153 |
| Extra Precison MatQuant | 4.023 | 66.54 | 2.617 | 4.022 | 74.26 | 2.470 | 4.022 | 73.13 | 2.155 |
| MatQuant | 2 | **52.37** | **3.800** | 2 | **63.35** | **3.187** | 2 | **62.75** | **3.153** |
| Extra Precison MatQuant | 2.052 | **55.70** | **3.355** | 2.050 | **68.25** | **2.823** | 2.051 | **65.99** | **2.569** |
| MatQuant | 6 | 67.52 | 2.574 | 6 | 73.92 | 2.440 | 6 | 73.63 | 2.127 |
| Extra Precison MatQuant | 6.018 | 68.01 | 2.582 | 6.018 | 74.50 | 2.446 | 6.018 | 73.59 | 2.139 |
| MatQuant | 3 | 64.47 | 2.618 | 3 | 72.87 | 2.607 | 3 | 71.16 | 2.238 |
| Extra Precison MatQuant | 3.031 | 63.24 | 2.757 | 3.029 | 73.25 | 2.535 | 3.030 | 71.55 | 2.228 |

## G Extra Precision MatQuant

Equation 8 clearly allows an extra bucket to be included into the quantization range, i.e, a $r$-bit model would have $2^r + 1$ possible values instead of $2^r$.

$$S(q^c, r) = \left( \left\lfloor \frac{q^c}{2^{c-r}} \right\rceil \right) * 2^{c-r} \tag{8}$$

For example, consider slicing the first two MSBs from an unsigned int8 value, $234$. As per Equation 1, $234$ first gets rounded to $4$, following which it gets clipped to 3, and finally is scaled up to $3*64 = 192$ (Note that MatQuant int2 allows for $0$, $64$, $128$, $192$). However, since the clipping operation is missing in Equation 8, $4$ is never clipped down to 3, and $S(q^c, r)$ is now $4 * 64 = 256$ Thus, for certain int2 values in our final quantized model, we will have to store an extra bit. This is the case with int3, int4 and int6 as well where an extra bit is required to represent certain values. In Table 7, we can see that the fraction of parameters that fall into this extra bucket is very small. However, for our 2-bit models, this additional bucket gives significant improvements in performance, for example, in Table 7 int2 Gemma-2 9B's average downstream accuracy goes up by $5\%$ when trained with an additional bucket (referred to as Extra Precison MatQuant in Table 7). This number is further boosted to $6\%$ with co-distillation, as evidenced by Table 8. We hypothesize that this additional bucket helps with capturing the outliers and thus leads to a significant performance boost. As highlighted by recent work (Dettmers et al., 2023; Kim et al., 2024), it is crucial to store certain outliers full precision. Interestingly, we show that even a single bit is enough to capture several of these outliers, especially for low bit quantization. Finally, note that this performance boost is not very evident in higher precisions where there are enough buckets to account for the outliers.

**Mix'n'Match** As shown in Figure 3 with a strong int2 model (i.e., 2.050 bits on average), Extra Precison MatQuant Mix'n'Match densely spans the Pareto-optimal accuracy-vs-bits-per-FFN-parameter (memory/cost) trade-off for Gemma-2 9B model trained using MatQuant with Omni-Quant – sometimes even improving on the bfloat16 model accuracy. Consequently, hardware supporting only int2 and int4 data types can still accommodate a model with a memory footprint similar to that of an int3 quantized model, and quality comparable or superior to int3; the additional bits required in the case of int2 can be packed into int2/int4. However, custom CUDA kernel would be required to enable sparse additions of these additional bits to the model weights.

## H Detailed Downstream Evaluations for OmniQuant and QAT

Tables 9, 10, 11, 12, 13, and 14 present downstream evaluation results on Gemma-2 2B, Gemma-2 9B and Mistral 7B with OmniQuant and QAT.

Table 8: Design choice ablations for co-distillation within Extra Precison MatQuant. x $\rightarrow$ y represents distilling the y-bit model from the x-bit model. We note that the accuracy for 2.050 avg. bits has significantly improved while minimally impacting the other bit-widths.

| Gemma-2 9B | | OmniQuant | | | |
|---|---|---|---|---|---|
| | | MatQuant | | E.P. MatQuant | |
| Avg. Bits | Config. | Task Avg. | log pplx. | Task Avg. | log pplx. |
| (8, 8) | [8, 4, 2] | 74.05 | 2.438 | 73.97 | 2.451 |
| | [8, 4, 8 $\rightarrow$ 2] | 72.76 | 2.473 | 73.40 | 2.467 |
| | [8, 4, 2, 8 $\rightarrow$ 2] | 73.99 | 2.435 | 73.46 | 2.466 |
| | [8, 4, 2, 8 $\rightarrow$ 4; 2] | 73.85 | 2.437 | 73.32 | 2.466 |
| (4, 4.022) | [8, 4, 2] | 73.83 | 2.491 | 73.88 | 2.481 |
| | [8, 4, 8 $\rightarrow$ 2] | 72.65 | 2.519 | 73.84 | 2.488 |
| | [8, 4, 2, 8 $\rightarrow$ 2] | 73.63 | 2.486 | 73.01 | 2.495 |
| | [8, 4, 2, 8 $\rightarrow$ 4; 2] | 73.55 | 2.478 | 73.12 | 2.518 |
| (2, 2.050) | [8, 4, 2] | 63.35 | **3.187** | 68.52 | 2.809 |
| | [8, 4, 8 $\rightarrow$ 2] | 62.64 | 3.289 | 69.2 | 2.796 |
| | [8, 4, 2, 8 $\rightarrow$ 2] | 62.91 | 3.138 | **70.17** | **2.778** |
| | [8, 4, 2, 8 $\rightarrow$ 4; 2] | **64.32** | 3.227 | 69.72 | 2.804 |

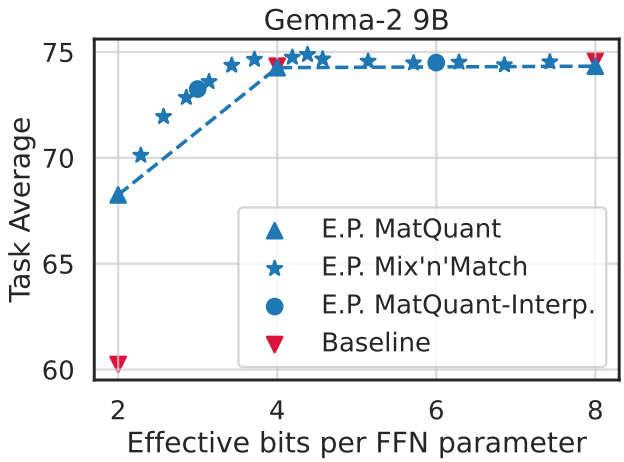


Figure 3: Mix'n'Match on Gemma-2 9B model trained using Extra Precison MatQuant with OmniQuant as the base algorithm allows elastic pareto-optimal accuracy-vs-cost model extraction for free during deployment.

## I Detailed Downstream Evaluations for MatQuant Re-weighting

Tables 15, 17, and 16 present downstream evaluation results for OmniQuant reweighting experiments on Gemma-2 2B, Gemma-2 9B and Mistral 7B.

## J Detailed Downstream Evaluations for Co-Distillation

Tables 18 and 19 present the downstream evaluation and perplexity results for MatQuant with co-distillation on Gemma-2 9B. We present results with both, OmniQuant and QAT as the base algorithms.

## K Detailed Evaluations for FFN + Attention Quantization

Tables 20 and 21 present the downstream evaluation and perplexity results for FFN + Attention quantization on Gemma-2 9B and Mistral 7B with OmniQuant and QAT.

## L Detailed Evaluation for Single Precison MatQuant

Tables 22, 23, 24, and 25 present the downstream evaluation results comparing Single Precison MatQuant to MatQuant and the *Baseline* for int2 quantization of Gemma-2 2B, Gemma-2 9B and Mistral 7B with OmniQuant and QAT. Since Single Precison MatQuant slices 2 bits from an 8-bit model and computes loss only over the first two bits, we can evaluate the Single Precison MatQuant model trained for 2-bits on int4 and int8. Downstream evaluation and perplexity results for this are presented in Tables 23 and 24. We also plot the weight distribution for Single Precison MatQuant in Figure 4.

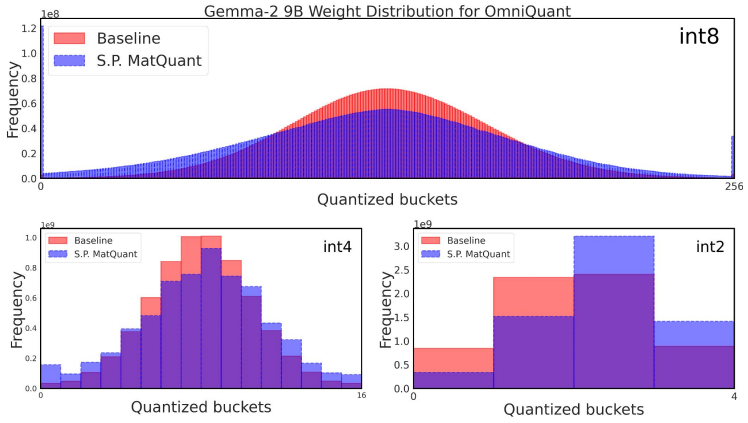


Figure 4: The Figure presents the weight distribution for Gemma-2 9B when trained with Single Precison MatQuant for int2 quantization. The right-shifted quantized weight distribution is a consequence of Single Precison MatQuant's training mechanism that heavily optimizes for the first 2 MSBs of the int8 representation.

## M Detailed Evaluation for Extra Precison MatQuant

Tables 26, 27, and 28 present downstream evaluation results for Extra Precison MatQuant when applied to Gemma-2 2B, 9B, and Mistral 7B with OmniQuant as the base algorithm. Table 29 presents downstream evaluation and perplexity results for our Extra Precison MatQuant co-distillation experiments on Gemma-2 9B with OmniQuant as the base algorithm.

Table 9: Table presents the downstream evaluation results for MatQuant when applied to OmniQuant on Gemma-2 2B.

| Data type | Method | Gemma-2 2B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | OmniQuant | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 50.09 | 71.59 | 76.45 | 69.69 | 78.29 | 63.14 | 68.21 |
| int8 | Baseline | 50 | 71.46 | 76.36 | 69.76 | 78.24 | 63.69 | 68.25 |
| | MatQuant | 49.66 | 71.00 | 76.73 | 68.85 | 78.56 | 63.30 | 68.02 |
| int4 | Sliced int8 | 41.55 | 66.12 | 72.02 | 62.34 | 75.79 | 59.43 | 62.87 |
| | Baseline | 48.46 | 70.96 | 74.22 | 67.66 | 77.26 | 63.61 | 67.03 |
| | MatQuant | 47.27 | 70.79 | 73.76 | 66.85 | 78.07 | 62.75 | 66.58 |
| int2 | Sliced int8 | 23.55 | 27.65 | 59.63 | 24.09 | 51.58 | 52.17 | 39.78 |
| | Baseline | 31.31 | 53.58 | 62.2 | 40.78 | 66.05 | 54.06 | 51.33 |
| | MatQuant | 29.95 | 54.21 | 64.40 | 44.37 | 66.81 | 54.46 | 52.37 |
| int6 | Sliced int8 | 48.72 | 71.13 | 76.06 | 69.12 | 78.45 | 62.83 | 67.72 |
| | Baseline | 49.32 | 71.76 | 76.48 | 69.52 | 78.56 | 62.75 | 68.06 |
| | MatQuant | 48.89 | 70.50 | 75.69 | 68.89 | 78.40 | 62.75 | 67.52 |
| int3 | Sliced int8 | 22.35 | 34.97 | 56.94 | 29.49 | 55.44 | 48.93 | 41.35 |
| | Baseline | 46.25 | 68.64 | 72.97 | 62.24 | 76.06 | 60.06 | 64.37 |
| | MatQuant | 44.03 | 67.09 | 74.25 | 62.78 | 77.26 | 61.40 | 64.47 |

Table 10: Table presents the downstream evaluation results for MatQuant when applied to OmniQuant on Gemma-2 9B.

| Data type | Method | Gemma-2 9B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | OmniQuant | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 58.96 | 77.57 | 83.33 | 77.31 | 81.12 | 67.96 | 74.38 |
| int8 | Baseline | 59.47 | 77.31 | 83.94 | 77.35 | 81.39 | 68.11 | 74.59 |
| | MatQuant | 57.59 | 77.02 | 84.01 | 76.61 | 81.18 | 67.88 | 74.05 |
| int4 | Sliced int8 | 55.80 | 75.04 | 82.32 | 73.56 | 80.47 | 66.38 | 72.26 |
| | Baseline | 58.79 | 78.37 | 83.55 | 76.71 | 81.45 | 67.09 | 74.33 |
| | MatQuant | 58.02 | 78.11 | 83.24 | 76.08 | 80.96 | 66.54 | 73.83 |
| int2 | Sliced int8 | 24.57 | 26.43 | 52.97 | 24.67 | 50.16 | 49.88 | 38.11 |
| | Baseline | 39.16 | 63.43 | 72.11 | 52.24 | 72.63 | 61.88 | 60.24 |
| | MatQuant | 40.78 | 67.85 | 73.64 | 60.56 | 72.09 | 65.19 | 63.35 |
| int6 | Sliced int8 | 59.04 | 77.61 | 84.62 | 77.10 | 81.18 | 68.27 | 74.64 |
| | Baseline | 59.22 | 77.27 | 83.21 | 77.1 | 81.12 | 67.48 | 74.23 |
| | MatQuant | 57.25 | 76.94 | 84.04 | 76.63 | 81.34 | 67.32 | 73.92 |
| int3 | Sliced int8 | 34.30 | 55.47 | 66.36 | 46.91 | 67.19 | 54.85 | 54.18 |
| | Baseline | 57.17 | 77.06 | 83.79 | 74.45 | 80.36 | 66.54 | 73.23 |
| | MatQuant | 55.80 | 76.89 | 81.99 | 74.27 | 80.14 | 68.11 | 72.87 |

Table 11: Table presents the downstream evaluation results for MatQuant when applied to OmniQuant on Mistral 7B.

| Data type | Method | Mistral 7B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | OmniQuant | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 49.57 | 73.74 | 84.4 | 80.61 | 81.18 | 74.43 | 73.99 |
| int8 | Baseline | 49.23 | 73.19 | 83.88 | 80.41 | 81.39 | 74.51 | 73.77 |
| | MatQuant | 49.06 | 72.52 | 84.74 | 79.21 | 81.45 | 74.90 | 73.65 |
| int4 | Sliced int8 | 21.33 | 33.67 | 42.08 | 28.62 | 55.66 | 49.72 | 38.51 |
| | Baseline | 49.23 | 73.23 | 83.94 | 79.9 | 81.34 | 74.11 | 73.62 |
| | MatQuant | 47.87 | 71.55 | 83.88 | 78.85 | 81.34 | 74.90 | 73.06 |
| int2 | Sliced int8 | 24.32 | 23.44 | 49.72 | 24.71 | 51.74 | 49.80 | 37.29 |
| | Baseline | 36.69 | 61.36 | 70.06 | 57.47 | 70.67 | 62.19 | 59.74 |
| | MatQuant | 37.88 | 62.58 | 73.15 | 65.89 | 73.88 | 63.14 | 62.75 |
| int6 | Sliced int8 | 48.21 | 71.09 | 83.21 | 79.93 | 81.28 | 74.27 | 73.00 |
| | Baseline | 50.26 | 73.65 | 84.04 | 80.55 | 81.66 | 74.43 | 74.1 |
| | MatQuant | 49.40 | 72.47 | 84.68 | 79.52 | 81.34 | 74.35 | 73.63 |
| int3 | Sliced int8 | 25.26 | 25.76 | 61.99 | 24.67 | 48.31 | 49.25 | 39.21 |
| | Baseline | 46.33 | 70.71 | 82.72 | 77.74 | 80.74 | 71.82 | 71.68 |
| | MatQuant | 47.35 | 71.00 | 80.00 | 76.96 | 80.30 | 71.35 | 71.16 |

Table 12: Table presents the downstream evaluation results for MatQuant when applied to QAT on Gemma-2 2B.

| Data type | Method | Gemma-2 2B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | QAT | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 50.09 | 71.59 | 76.45 | 69.69 | 78.29 | 63.14 | 68.21 |
| int8 | Baseline | 47.78 | 70.66 | 75.08 | 69.92 | 78.35 | 65.11 | 67.82 |
| | MatQuant | 45.39 | 71.21 | 75.99 | 68.74 | 78.40 | 64.88 | 67.44 |
| int4 | Sliced int8 | 46.16 | 69.53 | 75.35 | 68.49 | 78.18 | 65.04 | 67.13 |
| | Baseline | 46.16 | 71.59 | 73.73 | 68.72 | 78.62 | 63.38 | 67.03 |
| | MatQuant | 44.03 | 69.53 | 75.84 | 68.03 | 77.80 | 64.33 | 66.59 |
| int2 | Sliced int8 | 24.06 | 26.94 | 59.05 | 25.57 | 51.85 | 48.15 | 39.27 |
| | Baseline | 24.66 | 43.22 | 62.17 | 38.39 | 64.42 | 53.59 | 47.74 |
| | MatQuant | 28.33 | 51.85 | 63.64 | 46.94 | 68.28 | 54.14 | 52.20 |
| int6 | Sliced int8 | 47.87 | 70.83 | 74.25 | 69.80 | 77.86 | 64.56 | 67.53 |
| | Baseline | 47.7 | 70.88 | 74.92 | 69.72 | 78.07 | 65.19 | 67.75 |
| | MatQuant | 45.39 | 71.17 | 76.15 | 68.33 | 78.13 | 64.80 | 67.33 |
| int3 | Sliced int8 | 37.97 | 62.67 | 64.71 | 58.01 | 74.27 | 59.75 | 59.56 |
| | Baseline | 39.68 | 65.28 | 67.03 | 62.68 | 77.04 | 58.8 | 61.75 |
| | MatQuant | 36.95 | 66.20 | 64.25 | 61.03 | 75.19 | 60.93 | 60.76 |

Table 13: Table presents the downstream evaluation results for MatQuant when applied to QAT on Gemma-2 9B.

| Data type | Method | Gemma-2 9B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | QAT | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 58.96 | 77.57 | 83.33 | 77.31 | 81.12 | 67.96 | 74.38 |
| int8 | Baseline | 58.11 | 75.38 | 80.12 | 78.7 | 81.5 | 71.19 | 74.17 |
| | MatQuant | 57.68 | 76.09 | 82.23 | 78.41 | 82.26 | 70.48 | 74.52 |
| int4 | Sliced int8 | 56.91 | 75.17 | 78.78 | 77.02 | 81.18 | 71.11 | 73.36 |
| | Baseline | 56.91 | 75.42 | 75.38 | 78.06 | 81.39 | 72.38 | 73.26 |
| | MatQuant | 56.66 | 75.72 | 77.55 | 77.30 | 81.23 | 70.96 | 73.24 |
| int2 | Sliced int8 | 23.46 | 28.28 | 57.09 | 29.76 | 53.48 | 50.36 | 40.40 |
| | Baseline | 33.45 | 55.43 | 62.26 | 54.8 | 70.51 | 59.67 | 56.02 |
| | MatQuant | 41.21 | 66.84 | 65.41 | 63.61 | 75.41 | 61.25 | 62.29 |
| int6 | Sliced int8 | 57.68 | 75.17 | 80.73 | 78.66 | 81.77 | 70.88 | 74.15 |
| | Baseline | 57.94 | 76.14 | 79.63 | 78.93 | 82.1 | 71.11 | 74.31 |
| | MatQuant | 57.25 | 76.01 | 81.83 | 78.25 | 81.77 | 70.72 | 74.30 |
| int3 | Sliced int8 | 50.60 | 67.85 | 75.54 | 71.07 | 79.11 | 68.03 | 68.70 |
| | Baseline | 53.07 | 75.04 | 66.61 | 74.94 | 80.03 | 69.69 | 69.9 |
| | MatQuant | 51.19 | 71.80 | 78.69 | 73.18 | 79.49 | 68.11 | 70.41 |

Table 14: Table presents the downstream evaluation results for MatQuant when applied to QAT on Mistral 7B.

| Data type | Method | Mistral 7B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | QAT | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 49.57 | 73.74 | 84.4 | 80.61 | 81.18 | 74.43 | 73.99 |
| int8 | Baseline | 48.89 | 71.63 | 82.42 | 81.69 | 81.18 | 75.06 | 73.48 |
| | MatQuant | 47.44 | 71.21 | 82.08 | 80.31 | 80.74 | 73.72 | 72.58 |
| int4 | Sliced int8 | 47.61 | 70.41 | 80.21 | 79.74 | 79.98 | 72.61 | 71.76 |
| | Baseline | 47.27 | 70.62 | 81.28 | 78.95 | 81.12 | 73.56 | 72.13 |
| | MatQuant | 45.99 | 72.22 | 81.90 | 79.08 | 80.36 | 72.38 | 71.99 |
| int2 | Sliced int8 | 24.40 | 25.97 | 47.52 | 24.66 | 50.27 | 51.62 | 37.41 |
| | Baseline | 29.78 | 48.23 | 64.5 | 55.11 | 70.84 | 61.25 | 54.95 |
| | MatQuant | 35.58 | 56.36 | 72.66 | 66.68 | 74.32 | 66.22 | 61.97 |
| int6 | Sliced int8 | 48.55 | 71.76 | 82.57 | 81.67 | 81.39 | 74.19 | 73.35 |
| | Baseline | 47.7 | 71.3 | 82.23 | 79.84 | 80.79 | 74.43 | 72.71 |
| | MatQuant | 46.93 | 71.34 | 81.96 | 80.27 | 80.52 | 74.51 | 72.59 |
| int3 | Sliced int8 | 38.99 | 61.11 | 72.54 | 65.65 | 77.48 | 70.24 | 64.33 |
| | Baseline | 44.54 | 67.97 | 73.98 | 76.31 | 79.65 | 70.48 | 68.82 |
| | MatQuant | 40.10 | 62.42 | 79.05 | 73.82 | 77.31 | 70.24 | 67.16 |

Table 15: Tables presents the downstream evaluation results on Gemma-2 2B for $\mathrm{MatQuant}$ loss reweighting when applied to OmniQuant. Weightings: $(x, y, z) \rightarrow (\lambda_8, \lambda_4, \lambda_2)$ (from Equation 2).

| | | Gemma-2 2B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Data type | Weightings | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| int8 | $(0.1, 0.1, 1)$ | 49.66 | 71 | 76.73 | 68.85 | 78.56 | 63.3 | 68.02 |
| | $(0.2, 0.2, 1)$ | 49.4 | 71.3 | 76.21 | 68.97 | 78.29 | 63.3 | 67.91 |
| | $(0.3, 0.3, 1)$ | 48.81 | 71.72 | 76.57 | 68.95 | 78.4 | 63.61 | 68.01 |
| | $(0.4, 0.4, 1)$ | 48.72 | 71.72 | 76.61 | 68.92 | 78.73 | 62.98 | 67.95 |
| | $(0.5, 0.5, 1)$ | 49.06 | 71.34 | 76.15 | 68.86 | 78.45 | 62.98 | 67.81 |
| int4 | $(0.1, 0.1, 1)$ | 47.27 | 70.79 | 73.76 | 66.85 | 78.07 | 62.75 | 66.58 |
| | $(0.2, 0.2, 1)$ | 48.63 | 71 | 76.06 | 68.11 | 77.97 | 63.06 | 67.47 |
| | $(0.3, 0.3, 1)$ | 47.7 | 71.17 | 75.08 | 67.57 | 77.69 | 62.59 | 66.97 |
| | $(0.4, 0.4, 1)$ | 48.29 | 71.25 | 76.76 | 67.46 | 77.58 | 63.54 | 67.48 |
| | $(0.5, 0.5, 1)$ | 48.04 | 70.66 | 75.9 | 67.57 | 78.4 | 64.01 | 67.43 |
| int2 | $(0.1, 0.1, 1)$ | 29.95 | 54.21 | 64.4 | 44.37 | 66.81 | 54.46 | 52.37 |
| | $(0.2, 0.2, 1)$ | 30.03 | 52.78 | 62.39 | 44.66 | 66.81 | 54.62 | 51.88 |
| | $(0.3, 0.3, 1)$ | 29.18 | 52.61 | 62.57 | 41.41 | 65.94 | 54.62 | 51.05 |
| | $(0.4, 0.4, 1)$ | 28.75 | 54.88 | 62.17 | 42.53 | 66.16 | 55.64 | 51.69 |
| | $(0.5, 0.5, 1)$ | 27.13 | 51.05 | 60.95 | 39.94 | 65.56 | 54.3 | 49.82 |
| int6 | $(0.1, 0.1, 1)$ | 48.89 | 70.5 | 75.69 | 68.89 | 78.4 | 62.75 | 67.52 |
| | $(0.2, 0.2, 1)$ | 49.32 | 70.96 | 75.87 | 68.93 | 78.29 | 62.67 | 67.67 |
| | $(0.3, 0.3, 1)$ | 48.98 | 71.63 | 76.21 | 68.68 | 78.73 | 63.46 | 67.95 |
| | $(0.4, 0.4, 1)$ | 48.98 | 71.72 | 75.75 | 68.83 | 78.67 | 63.61 | 67.93 |
| | $(0.5, 0.5, 1)$ | 49.4 | 71.59 | 76.21 | 68.63 | 78.29 | 63.85 | 67.99 |
| int3 | $(0.1, 0.1, 1)$ | 44.03 | 67.09 | 74.25 | 62.78 | 77.26 | 61.4 | 64.47 |
| | $(0.2, 0.2, 1)$ | 43.09 | 65.7 | 67.19 | 59.57 | 75.3 | 60.38 | 61.87 |
| | $(0.3, 0.3, 1)$ | 43.94 | 68.35 | 71.87 | 59.54 | 75.79 | 59.98 | 63.24 |
| | $(0.4, 0.4, 1)$ | 41.81 | 65.53 | 72.91 | 61.42 | 75.03 | 61.88 | 63.1 |
| | $(0.5, 0.5, 1)$ | 41.64 | 67.34 | 71.87 | 61.15 | 74.54 | 61.64 | 63.03 |

Table 16: Tables presents the downstream evaluation results on Gemma-2 9B for $\mathrm{MatQuant}$ loss reweighting when applied to OmniQuant. Weightings: $(x, y, z) \rightarrow (\lambda_8, \lambda_4, \lambda_2)$ (from Equation 2).

| | | Gemma-2 9B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Data type | Weightings | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| int8 | $(0.1, 0.1, 1)$ | 57.59 | 77.02 | 84.01 | 76.61 | 81.18 | 67.88 | 74.05 |
| | $(0.2, 0.2, 1)$ | 57.76 | 76.73 | 83.73 | 76.5 | 81.34 | 67.4 | 73.91 |
| | $(0.3, 0.3, 1)$ | 57.94 | 76.64 | 83.36 | 76.56 | 81.01 | 67.8 | 73.88 |
| | $(0.4, 0.4, 1)$ | 58.28 | 76.52 | 83.15 | 76.74 | 80.96 | 67.4 | 73.84 |
| | $(0.5, 0.5, 1)$ | 57.68 | 76.68 | 83.39 | 76.62 | 81.07 | 67.09 | 73.75 |
| int4 | $(0.1, 0.1, 1)$ | 58.02 | 78.11 | 83.24 | 76.08 | 80.96 | 66.54 | 73.83 |
| | $(0.2, 0.2, 1)$ | 58.96 | 77.9 | 82.57 | 76.14 | 81.07 | 66.14 | 73.8 |
| | $(0.3, 0.3, 1)$ | 57.42 | 77.23 | 81.62 | 75.72 | 80.85 | 66.69 | 73.25 |
| | $(0.4, 0.4, 1)$ | 58.96 | 78.32 | 84.53 | 76.17 | 81.45 | 66.46 | 74.32 |
| | $(0.5, 0.5, 1)$ | 57.08 | 77.02 | 84.65 | 76.11 | 81.56 | 66.06 | 73.75 |
| int2 | $(0.1, 0.1, 1)$ | 40.78 | 67.85 | 73.64 | 60.56 | 72.09 | 65.19 | 63.35 |
| | $(0.2, 0.2, 1)$ | 40.53 | 67.97 | 75.57 | 60.83 | 72.25 | 67.09 | 64.04 |
| | $(0.3, 0.3, 1)$ | 39.42 | 67.68 | 79.08 | 60.79 | 72.47 | 65.19 | 64.1 |
| | $(0.4, 0.4, 1)$ | 39.68 | 66.54 | 66.24 | 61.08 | 73.07 | 65.27 | 61.98 |
| | $(0.5, 0.5, 1)$ | 40.02 | 66.16 | 69.08 | 60.54 | 73.23 | 64.88 | 62.32 |
| int6 | $(0.1, 0.1, 1)$ | 57.25 | 76.94 | 84.04 | 76.63 | 81.34 | 67.32 | 73.92 |
| | $(0.2, 0.2, 1)$ | 57.25 | 76.6 | 83.79 | 76.46 | 81.12 | 67.64 | 73.81 |
| | $(0.3, 0.3, 1)$ | 58.7 | 76.98 | 83.09 | 76.63 | 80.69 | 67.32 | 73.9 |
| | $(0.4, 0.4, 1)$ | 58.28 | 76.43 | 83.15 | 76.76 | 81.18 | 67.09 | 73.81 |
| | $(0.5, 0.5, 1)$ | 58.28 | 76.3 | 83.33 | 76.68 | 81.18 | 66.93 | 73.78 |
| int3 | $(0.1, 0.1, 1)$ | 55.8 | 76.89 | 81.99 | 74.27 | 80.14 | 68.11 | 72.87 |
| | $(0.2, 0.2, 1)$ | 54.69 | 76.56 | 79.79 | 73.92 | 79.92 | 66.77 | 71.94 |
| | $(0.3, 0.3, 1)$ | 56.48 | 77.53 | 83.09 | 73.71 | 80.69 | 67.32 | 73.14 |
| | $(0.4, 0.4, 1)$ | 56.23 | 77.86 | 83.79 | 74.12 | 80.69 | 68.98 | 73.61 |
| | $(0.5, 0.5, 1)$ | 54.35 | 76.3 | 83.67 | 74.21 | 80.09 | 68.03 | 72.77 |

Table 17: Tables presents the downstream evaluation results on Mistral 7B for $\mathrm{MatQuant}$ loss reweighting when applied to OmniQuant. Weightings: $(x, y, z) \rightarrow (\lambda_8, \lambda_4, \lambda_2)$ (from Equation 2).

| | | Mistral 7B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Data type | Weightings | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| int8 | $(0.1, 0.1, 1)$ | 49.23 | 71.84 | 83.94 | 78.9 | 81.39 | 74.35 | 73.27 |
| | $(0.2, 0.2, 1)$ | 49.23 | 71.97 | 83.91 | 79.04 | 81.5 | 74.98 | 73.44 |
| | $(0.3, 0.3, 1)$ | 49.32 | 72.39 | 84.43 | 79.24 | 81.23 | 74.74 | 73.56 |
| | $(0.4, 0.4, 1)$ | 49.06 | 72.52 | 84.74 | 79.21 | 81.45 | 74.9 | 73.65 |
| | $(0.5, 0.5, 1)$ | 49.15 | 72.64 | 84.65 | 79.37 | 81.72 | 74.82 | 73.72 |
| int4 | $(0.1, 0.1, 1)$ | 47.61 | 71.59 | 83.3 | 78.32 | 81.61 | 74.11 | 72.76 |
| | $(0.2, 0.2, 1)$ | 48.12 | 72.14 | 84.07 | 78.72 | 81.45 | 74.43 | 73.16 |
| | $(0.3, 0.3, 1)$ | 48.21 | 72.81 | 84.4 | 79.02 | 81.18 | 75.22 | 73.47 |
| | $(0.4, 0.4, 1)$ | 47.87 | 71.55 | 83.88 | 78.85 | 81.34 | 74.9 | 73.06 |
| | $(0.5, 0.5, 1)$ | 48.21 | 71.97 | 83.82 | 79.03 | 81.39 | 74.35 | 73.13 |
| int2 | $(0.1, 0.1, 1)$ | 37.46 | 63.43 | 71.53 | 66.22 | 75.24 | 65.59 | 63.25 |
| | $(0.2, 0.2, 1)$ | 37.54 | 64.81 | 71.8 | 66.57 | 74.37 | 65.27 | 63.39 |
| | $(0.3, 0.3, 1)$ | 37.46 | 62.92 | 75.35 | 67.2 | 74.43 | 64.25 | 63.6 |
| | $(0.4, 0.4, 1)$ | 37.88 | 62.58 | 73.15 | 65.89 | 73.88 | 63.14 | 62.75 |
| | $(0.5, 0.5, 1)$ | 37.29 | 62.75 | 69.36 | 64.99 | 72.36 | 64.25 | 61.83 |
| int6 | $(0.1, 0.1, 1)$ | 49.57 | 71.72 | 83.76 | 78.87 | 81.28 | 74.03 | 73.2 |
| | $(0.2, 0.2, 1)$ | 49.49 | 72.52 | 84.22 | 79.08 | 81.39 | 74.19 | 73.48 |
| | $(0.3, 0.3, 1)$ | 48.89 | 72.01 | 83.85 | 79.2 | 81.39 | 74.35 | 73.28 |
| | $(0.4, 0.4, 1)$ | 49.4 | 72.47 | 84.68 | 79.52 | 81.34 | 74.35 | 73.63 |
| | $(0.5, 0.5, 1)$ | 49.4 | 72.39 | 84.31 | 79.5 | 81.28 | 74.27 | 73.52 |
| int3 | $(0.1, 0.1, 1)$ | 44.88 | 68.22 | 81.96 | 76.13 | 80.69 | 71.35 | 70.54 |
| | $(0.2, 0.2, 1)$ | 43.94 | 67.85 | 81.56 | 76.55 | 79.76 | 72.61 | 70.38 |
| | $(0.3, 0.3, 1)$ | 45.39 | 67.89 | 80.92 | 77.13 | 80.47 | 72.06 | 70.64 |
| | $(0.4, 0.4, 1)$ | 47.35 | 71 | 80 | 76.96 | 80.3 | 71.35 | 71.16 |
| | $(0.5, 0.5, 1)$ | 46.76 | 70.29 | 82.17 | 77.32 | 80.9 | 71.11 | 71.43 |

Table 18: Table presents the downstream evaluation and perplexity results for our $\mathrm{MatQuant}$ co-distillation experiments on Gemma-2 9B with OmniQuant.

| | OmniQuant | Gemma-2 9B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Data type | Config. | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average | log pplx. |
| int8 | $[8, 4, 8 \rightarrow 2]$ | 57.51 | 76.26 | 83.30 | 73.35 | 80.74 | 65.43 | 72.76 | 2.473 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 58.19 | 76.89 | 83.73 | 76.75 | 81.39 | 67.01 | 73.99 | 2.435 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 57.68 | 77.06 | 83.00 | 76.76 | 81.45 | 67.17 | 73.85 | 2.437 |
| int4 | $[8, 4, 8 \rightarrow 2]$ | 56.23 | 76.47 | 82.63 | 73.03 | 80.69 | 66.85 | 72.65 | 2.519 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 57.51 | 76.73 | 83.36 | 76.23 | 80.85 | 67.09 | 73.63 | 2.486 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 57.51 | 76.68 | 83.27 | 75.85 | 81.61 | 66.38 | 73.55 | 2.478 |
| int2 | $[8, 4, 8 \rightarrow 2]$ | 38.14 | 66.50 | 76.73 | 59.70 | 71.11 | 63.69 | 62.64 | 3.289 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 40.61 | 67.55 | 71.07 | 60.80 | 72.96 | 64.48 | 62.91 | 3.138 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 42.75 | 69.65 | 74.40 | 60.53 | 72.42 | 66.14 | 64.32 | 3.227 |
| int6 | $[8, 4, 8 \rightarrow 2]$ | 57.59 | 76.30 | 83.55 | 73.41 | 80.85 | 65.51 | 72.87 | 2.469 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 58.28 | 76.85 | 83.43 | 76.91 | 81.18 | 67.01 | 73.94 | 2.438 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 58.11 | 76.98 | 83.33 | 76.70 | 81.45 | 67.48 | 74.01 | 2.439 |
| int3 | $[8, 4, 8 \rightarrow 2]$ | 52.30 | 75.25 | 78.26 | 71.08 | 79.49 | 65.35 | 70.29 | 2.651 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 54.44 | 75.97 | 82.20 | 73.84 | 80.20 | 66.46 | 72.19 | 2.603 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 54.44 | 76.26 | 81.90 | 73.89 | 79.92 | 65.75 | 72.03 | 2.604 |

Table 19: Table presents the downstream evaluation and perplexity results for our MatQuant co-distillation experiments on Gemma-2 9B with QAT.

| QAT | | Gemma-2 9B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Data type | Config. | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average | log pplx. |
| int8 | $[8, 4, 8 \rightarrow 2]$ | 57.68 | 76.09 | 82.60 | 78.75 | 82.48 | 70.88 | 74.75 | 2.242 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 57.76 | 76.35 | 81.50 | 79.13 | 82.43 | 72.06 | 74.87 | 2.240 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 58.19 | 76.05 | 81.62 | 78.92 | 82.21 | 71.90 | 74.81 | 2.240 |
| int4 | $[8, 4, 8 \rightarrow 2]$ | 57.85 | 76.81 | 78.47 | 77.62 | 80.96 | 70.88 | 73.76 | 2.279 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 57.08 | 75.88 | 78.47 | 77.65 | 81.34 | 72.22 | 73.77 | 2.276 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 57.34 | 75.80 | 78.99 | 77.67 | 81.50 | 72.30 | 73.93 | 2.277 |
| int2 | $[8, 4, 8 \rightarrow 2]$ | 40.61 | 67.17 | 67.37 | 63.10 | 75.24 | 60.38 | 62.31 | 2.670 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 40.53 | 66.71 | 67.89 | 63.29 | 75.46 | 62.35 | 62.70 | 2.673 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 40.10 | 66.37 | 67.86 | 63.14 | 75.08 | 63.06 | 62.60 | 2.670 |
| int6 | $[8, 4, 8 \rightarrow 2]$ | 57.85 | 76.05 | 82.23 | 78.70 | 82.10 | 71.43 | 74.73 | 2.245 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 58.11 | 75.93 | 82.14 | 79.10 | 82.26 | 71.19 | 74.79 | 2.243 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 58.19 | 75.67 | 81.31 | 78.80 | 82.15 | 71.27 | 74.56 | 2.243 |
| int3 | $[8, 4, 8 \rightarrow 2]$ | 51.19 | 71.00 | 76.67 | 73.07 | 79.54 | 68.03 | 69.92 | 2.441 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 51.71 | 71.46 | 76.85 | 73.00 | 79.00 | 67.88 | 69.98 | 2.437 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 51.28 | 71.34 | 76.12 | 72.96 | 79.33 | 68.98 | 70.00 | 2.435 |

Table 20: Table presents the downstream evaluation results for MatQuant FFN + Attention quantization on Gemma-2 9B with QAT.

| Data type | Method | Gemma-2 9B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 58.96 | 77.57 | 83.33 | 77.31 | 81.12 | 67.96 | 74.38 |
| int8 | Baseline | 58.62 | 77.02 | 83.43 | 79.01 | 81.34 | 68.27 | 74.61 |
| | MatQuant | 59.47 | 77.99 | 84.13 | 77.85 | 81.23 | 68.43 | 74.85 |
| int4 | Sliced int8 | 57.42 | 76.01 | 80.86 | 76.34 | 80.03 | 68.27 | 73.15 |
| | Baseline | 56.06 | 74.96 | 79.27 | 77.83 | 80.25 | 69.53 | 72.98 |
| | MatQuant | 58.79 | 75.80 | 84.89 | 76.26 | 81.23 | 67.09 | 74.01 |
| int2 | Sliced int8 | 26.37 | 25.34 | 58.10 | 25.60 | 49.08 | 49.33 | 38.97 |
| | Baseline | - | - | - | - | - | - | - |
| | S.P. MatQuant | 25.26 | 38.47 | 62.14 | 35.09 | 61.70 | 51.46 | 45.69 |
| | MatQuant | 23.72 | 36.62 | 62.17 | 33.72 | 59.36 | 49.57 | 44.19 |
| int6 | Sliced int8 | 58.53 | 77.10 | 83.00 | 78.81 | 81.07 | 68.43 | 74.49 |
| | Baseline | 58.87 | 77.06 | 83.12 | 78.81 | 81.23 | 68.82 | 74.65 |
| | MatQuant | 58.96 | 78.03 | 83.30 | 77.72 | 80.96 | 68.43 | 74.57 |
| int3 | Sliced int8 | 44.71 | 65.28 | 71.56 | 65.25 | 75.84 | 62.51 | 64.19 |
| | Baseline | - | - | - | - | - | - | - |
| | S.P. MatQuant | 48.55 | 71.25 | 68.38 | 72.12 | 79.00 | 66.77 | 67.68 |
| | MatQuant | 43.34 | 61.91 | 75.96 | 65.20 | 75.46 | 59.91 | 63.63 |

Table 21: Table presents the downstream evaluation results for MatQuant FFN + Attention quantization on Mistral 7B with QAT.

| Data type | Method | Mistral 7B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 49.57 | 73.74 | 84.4 | 80.61 | 81.18 | 74.43 | 73.99 |
| int8 | Baseline | 49.23 | 72.9 | 83.49 | 80.26 | 81.28 | 75.22 | 73.73 |
| | MatQuant | 50.09 | 73.44 | 83.73 | 80.73 | 81.39 | 73.88 | 73.88 |
| int4 | Sliced int8 | 45.99 | 71.55 | 81.19 | 76.90 | 80.58 | 72.53 | 71.46 |
| | Baseline | 48.04 | 71.72 | 78.87 | 78.93 | 80.36 | 73.32 | 71.87 |
| | MatQuant | 46.59 | 70.29 | 81.65 | 77.34 | 80.25 | 72.53 | 71.44 |
| int2 | Sliced int8 | 22.61 | 25.38 | 37.86 | 24.40 | 49.13 | 50.99 | 35.06 |
| | Baseline | - | - | - | - | - | - | - |
| | S.P. MatQuant | 22.53 | 25.51 | 38.90 | 24.13 | 50.92 | 50.12 | 35.35 |
| | MatQuant | 21.33 | 25.59 | 57.37 | 24.85 | 50.92 | 50.12 | 38.36 |
| int6 | Sliced int8 | 49.32 | 73.53 | 82.60 | 80.28 | 80.96 | 74.98 | 73.61 |
| | Baseline | 49.32 | 73.4 | 82.48 | 80.24 | 81.28 | 75.61 | 73.72 |
| | MatQuant | 50.00 | 73.78 | 83.55 | 80.74 | 81.66 | 74.51 | 74.04 |
| int3 | Sliced int8 | 19.97 | 30.72 | 46.79 | 27.22 | 58.43 | 50.91 | 39.01 |
| | Baseline | - | - | - | - | - | - | - |
| | S.P. MatQuant | 43.86 | 67.51 | 70.43 | 73.97 | 80.36 | 69.38 | 67.59 |
| | MatQuant | 20.82 | 33.42 | 53.30 | 27.77 | 58.76 | 49.25 | 40.55 |

Table 22: Table presents downstream evaluation and perplexity results for Single Precison MatQuant, comparing it with MatQuant and the *Baseline* for int2 quatization of Gemma-2 2B with OmniQuant and QAT.

| int2 | | Gemma2-2B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Method | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Task Avg. | log pplx. |
| OmniQuant | S.P. MatQuant | 29.78 | 57.70 | 63.39 | 44.32 | 68.66 | 56.67 | 53.42 | 3.631 |
| | Baseline | 31.31 | 53.58 | 62.2 | 40.78 | 66.05 | 54.06 | 51.33 | 3.835 |
| | MatQuant | 29.95 | 54.21 | 64.40 | 44.37 | 66.81 | 54.46 | 52.37 | 3.800 |
| QAT | S.P. MatQuant | 28.07 | 52.36 | 62.87 | 46.80 | 68.88 | 53.51 | 52.08 | 3.054 |
| | Baseline | 24.66 | 43.22 | 62.17 | 38.39 | 64.42 | 53.59 | 47.74 | 3.433 |
| | MatQuant | 28.33 | 51.85 | 63.64 | 46.94 | 68.28 | 54.14 | 52.20 | 3.055 |

Table 23: Table presents downstream evaluation and perplexity results for Single Precison MatQuant, comparing it with MatQuant and the *Baseline* for int2, int4, int8 quatization of Gemma-2 9B with Baseline. Note that the model was trained with Single Precison MatQuant for int2; the int4 and int8 model were sliced post training.

| | | Gemma-2 9B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Data type | Method | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average | log pplx. |
| int8 | S.P. MatQuant | 57.94 | 76.64 | 82.66 | 76.98 | 81.01 | 67.56 | 73.80 | 2.372 |
| | Baseline | 59.47 | 77.31 | 83.94 | 77.35 | 81.39 | 68.11 | 74.59 | 2.418 |
| | MatQuant | 57.59 | 77.02 | 84.01 | 76.61 | 81.18 | 67.88 | 74.05 | 2.438 |
| int4 | S.P. MatQuant | 57.17 | 76.39 | 81.47 | 75.81 | 80.85 | 66.38 | 73.01 | 2.420 |
| | Baseline | 58.79 | 78.37 | 83.55 | 76.71 | 81.45 | 67.09 | 74.33 | 2.451 |
| | MatQuant | 58.02 | 78.11 | 83.24 | 76.08 | 80.96 | 66.54 | 73.83 | 2.491 |
| int2 | S.P. MatQuant | 40.44 | 66.75 | 77.92 | 60.42 | 72.52 | 66.06 | 64.02 | 3.171 |
| | Baseline | 39.16 | 63.43 | 72.11 | 52.24 | 72.63 | 61.88 | 60.24 | 3.292 |
| | MatQuant | 40.78 | 67.85 | 73.64 | 60.56 | 72.09 | 65.19 | 63.35 | 3.187 |

Table 24: Table presents downstream evaluation and perplexity results for Single Precison MatQuant, comparing it with MatQuant and the *Baseline* for int2, int4, int8 quatization of Gemma-2 9B with Baseline. Note that the model was trained with Single Precison MatQuant for int2; the int4 and int8 model were sliced post training.

| | | Gemma-2 9B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Data type | Method | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average | log pplx. |
| int8 | S.P. MatQuant | 55.89 | 75.84 | 79.57 | 75.47 | 81.07 | 68.43 | 72.71 | 2.363 |
| | Baseline | 58.11 | 75.38 | 80.12 | 78.7 | 81.5 | 71.19 | 74.17 | 2.29 |
| | MatQuant | 57.68 | 76.09 | 82.23 | 78.41 | 82.26 | 70.48 | 74.52 | 2.262 |
| int4 | S.P. MatQuant | 54.95 | 75.59 | 75.05 | 74.60 | 80.79 | 69.06 | 71.67 | 2.394 |
| | Baseline | 56.91 | 75.42 | 75.38 | 78.06 | 81.39 | 72.38 | 73.26 | 2.324 |
| | MatQuant | 56.66 | 75.72 | 77.55 | 77.30 | 81.23 | 70.96 | 73.24 | 2.295 |
| int2 | S.P. MatQuant | 40.53 | 67.38 | 66.91 | 63.62 | 75.63 | 61.88 | 62.66 | 2.656 |
| | Baseline | 33.45 | 55.43 | 62.26 | 54.8 | 70.51 | 59.67 | 56.02 | 2.923 |
| | MatQuant | 41.21 | 66.84 | 65.41 | 63.61 | 75.41 | 61.25 | 62.29 | 2.660 |

Table 25: Table presents downstream evaluation and perplexity results for Single Precison MatQuant, comparing it with MatQuant, and the *Baseline* for int2 quatization of Mistral 7B. Results are presented for both, OmniQuant and QAT as the base algorithms.

| int2 | | Mistral 7B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Method | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Task Avg. | log pplx. |
| OmniQuant | S.P. MatQuant | 37.63 | 64.14 | 72.45 | 67.47 | 74.81 | 64.96 | 63.58 | 2.976 |
| | Baseline | 36.69 | 61.36 | 70.06 | 57.47 | 70.67 | 62.19 | 59.74 | 3.931 |
| | MatQuant | 37.88 | 62.58 | 73.15 | 65.89 | 73.88 | 63.14 | 62.75 | 3.153 |
| QAT | S.P. MatQuant | 35.24 | 57.15 | 69.88 | 66.02 | 75.41 | 65.19 | 61.48 | 2.509 |
| | Baseline | 29.78 | 48.23 | 64.5 | 55.11 | 70.84 | 61.25 | 54.95 | 2.694 |
| | MatQuant | 35.58 | 56.36 | 72.66 | 66.68 | 74.32 | 66.22 | 61.97 | 2.524 |

Table 26: Table presents the downstream evaluation results for Extra Precison MatQuant when applied to OmniQuant on Gemma-2 2B.

| Avg. Bits | Method | Gemma-2 2B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | OmniQuant | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 50.09 | 71.59 | 76.45 | 69.69 | 78.29 | 63.14 | 68.21 |
| 8 | MatQuant | 49.66 | 71.00 | 76.73 | 68.85 | 78.56 | 63.30 | 68.02 |
| 8 | Extra Precison MatQuant | 48.04 | 71.8 | 75.78 | 67.64 | 78.07 | 63.22 | 67.42 |
| 4 | MatQuant | 47.27 | 70.79 | 73.76 | 66.85 | 78.07 | 62.75 | 66.58 |
| 4.023 | Extra Precison MatQuant | 45.65 | 70.29 | 74.8 | 66.07 | 77.58 | 62.27 | 66.11 |
| 2 | MatQuant | 29.95 | 54.21 | 64.40 | 44.37 | 66.81 | 54.46 | 52.37 |
| 2.052 | Extra Precison MatQuant | 34.39 | 59.64 | 62.69 | 52.11 | 69.86 | 55.56 | 55.71 |
| 6 | MatQuant | 48.89 | 70.50 | 75.69 | 68.89 | 78.40 | 62.75 | 67.52 |
| 6.018 | Extra Precison MatQuant | 47.1 | 71.46 | 76.02 | 67.47 | 77.91 | 63.61 | 67.26 |
| 3 | MatQuant | 44.03 | 67.09 | 74.25 | 62.78 | 77.26 | 61.40 | 64.47 |
| 3.031 | Extra Precison MatQuant | 44.45 | 68.56 | 69.11 | 62.28 | 75.95 | 62.59 | 63.82 |

Table 27: Table presents the downstream evaluation results for Extra Precison MatQuant when applied to OmniQuant on Gemma-2 9B.

| Avg. Bits | Method | Gemma-2 9B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | OmniQuant | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 58.96 | 77.57 | 83.33 | 77.31 | 81.12 | 67.96 | 74.38 |
| 8 | MatQuant | 57.59 | 77.02 | 84.01 | 76.61 | 81.18 | 67.88 | 74.05 |
| 8 | Extra Precison MatQuant | 58.11 | 78.03 | 83.27 | 76.17 | 81.18 | 67.09 | 73.97 |
| 4 | MatQuant | 58.02 | 78.11 | 83.24 | 76.08 | 80.96 | 66.54 | 73.83 |
| 4.022 | Extra Precison MatQuant | 57.25 | 77.36 | 84.86 | 75.52 | 81.5 | 66.77 | 73.88 |
| 2 | MatQuant | 40.78 | 67.85 | 73.64 | 60.56 | 72.09 | 65.19 | 63.35 |
| 2.050 | Extra Precison MatQuant | 48.72 | 72.18 | 79.2 | 68.11 | 76.17 | 66.77 | 68.52 |
| 6 | MatQuant | 57.25 | 76.94 | 84.04 | 76.63 | 81.34 | 67.32 | 73.92 |
| 6.018 | Extra Precison MatQuant | 58.87 | 78.03 | 83.61 | 76.18 | 81.45 | 67.09 | 74.21 |
| 3 | MatQuant | 55.80 | 76.89 | 81.99 | 74.27 | 80.14 | 68.11 | 72.87 |
| 3.029 | Extra Precison MatQuant | 55.46 | 76.14 | 84.04 | 74.49 | 80.14 | 67.32 | 72.93 |

Table 28: Table presents the downstream evaluation results for Extra Precison MatQuant when applied to OmniQuant on Mistral 7B.

| Data type | Method | Mistral 7B | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | OmniQuant | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average |
| bfloat16 | | 49.57 | 73.74 | 84.4 | 80.61 | 81.18 | 74.43 | 73.99 |
| 8 | MatQuant | 49.06 | 72.52 | 84.74 | 79.21 | 81.45 | 74.90 | 73.65 |
| 8 | Extra Precison MatQuant | 48.04 | 73.44 | 84.13 | 79.37 | 81.12 | 74.66 | 73.46 |
| 4 | MatQuant | 47.87 | 71.55 | 83.88 | 78.85 | 81.34 | 74.90 | 73.06 |
| 4.022 | Extra Precison MatQuant | 48.21 | 72.69 | 83.49 | 78.82 | 81.12 | 74.43 | 73.13 |
| 2 | MatQuant | 37.88 | 62.58 | 73.15 | 65.89 | 73.88 | 63.14 | 62.75 |
| 2.051 | Extra Precison MatQuant | 41.38 | 67.42 | 71.62 | 71.98 | 77.86 | 65.67 | 65.99 |
| 6 | MatQuant | 49.40 | 72.47 | 84.68 | 79.52 | 81.34 | 74.35 | 73.63 |
| 6.018 | Extra Precison MatQuant | 48.46 | 72.98 | 84.07 | 79.64 | 81.18 | 75.22 | 73.59 |
| 3 | MatQuant | 47.35 | 71.00 | 80.00 | 76.96 | 80.30 | 71.35 | 71.16 |
| 3.030 | Extra Precison MatQuant | 45.65 | 71.21 | 80.43 | 78.31 | 81.07 | 72.61 | 71.55 |

Table 29: Table presents the downstream evaluation and perplexity results for our Extra Precison MatQuant co-distillation experiments on Gemma-2 9B with OmniQuant.

| | OmniQuant | Gemma-2 9B | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Avg. Bits | Config. | ARC-c | ARC-e | BoolQ | HellaSwag | PIQA | Winogrande | Average | log pplx. |
| 8 | $[8, 4, 8 \rightarrow 2]$ | 57.59 | 77.27 | 81.83 | 75.48 | 81.01 | 67.25 | 73.4 | 2.467 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 57.17 | 77.36 | 82.2 | 75.82 | 80.96 | 67.25 | 73.46 | 2.466 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 56.4 | 77.82 | 82.32 | 75.02 | 80.63 | 67.72 | 73.32 | 2.466 |
| 4.022 | $[8, 4, 8 \rightarrow 2]$ | 57.68 | 78.45 | 82.97 | 75.5 | 80.85 | 67.56 | 73.84 | 2.488 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 57.51 | 77.61 | 80.46 | 74.74 | 81.12 | 66.61 | 73.01 | 2.495 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 56.57 | 77.99 | 82.54 | 74.77 | 80.58 | 66.3 | 73.12 | 2.518 |
| 2.050 | $[8, 4, 8 \rightarrow 2]$ | 48.81 | 74.03 | 81.65 | 68.1 | 77.48 | 65.11 | 69.2 | 2.796 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 49.15 | 75.34 | 83.12 | 68.79 | 77.64 | 67.01 | 70.17 | 2.778 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 49.83 | 75.04 | 79.79 | 68.38 | 77.86 | 67.4 | 69.72 | 2.804 |
| 6.018 | $[8, 4, 8 \rightarrow 2]$ | 57.42 | 77.19 | 81.87 | 75.42 | 81.01 | 67.8 | 73.45 | 2.468 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 57.51 | 77.48 | 82.32 | 75.88 | 81.07 | 66.61 | 73.48 | 2.467 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 56.4 | 78.03 | 82.63 | 75.14 | 80.79 | 67.4 | 73.4 | 2.498 |
| 3.029 | $[8, 4, 8 \rightarrow 2]$ | 55.63 | 75.88 | 80.12 | 74.01 | 80.36 | 67.96 | 72.33 | 2.549 |
| | $[8, 4, 2, 8 \rightarrow 2]$ | 54.35 | 76.85 | 79.33 | 74.6 | 80.47 | 67.4 | 72.17 | 2.543 |
| | $[8, 4, 2, 8 \rightarrow 4; 2]$ | 55.2 | 76.98 | 82.45 | 73.59 | 80.41 | 68.43 | 72.84 | 2.58 |

Table 30: Table presents downstream task average and log pplx (perplexity) when applied to OmniQuant and QAT on Gemma-2 2B, 9B and Mistral 7B models.

| int2 | Gemma-2 2B | | Gemma-2 9B | | Mistral 7B | |
|---|---|---|---|---|---|---|
| Method | Task Avg. | log pplx. | Task Avg. | log pplx. | Task Avg. | log pplx. |
| OmniQuant | 51.33 | 3.835 | 60.24 | 3.292 | 59.74 | 3.931 |
| S.P. MatQuant | 53.42 | 3.631 | 64.02 | 3.171 | 63.58 | 2.976 |
| MatQuant | 52.37 | 3.800 | 63.35 | 3.187 | 62.75 | 3.153 |
| S.P. E.P. MatQuant | 57.38 | 3.185 | 68.58 | 2.857 | 67.36 | 2.464 |
| E.P. MatQuant | 55.71 | 3.292 | 68.52 | 2.809 | 65.99 | 2.569 |
| QAT | 47.74 | 3.433 | 56.02 | 2.923 | 54.95 | 2.699 |
| S.P. MatQuant | 52.08 | 3.054 | 62.66 | 2.656 | 61.48 | 2.509 |
| MatQuant | 52.20 | 3.055 | 62.29 | 2.660 | 61.97 | 2.524 |
| S.P. E.P. MatQuant | 53.18 | 3.090 | 62.53 | 2.706 | 61.55 | 2.435 |
| E.P. MatQuant | 52.43 | 3.153 | 62.32 | 2.756 | 61.29 | 2.474 |