DeltaDQ: Distribution-Driven Delta Compression for Fine-tuned LLMs

Anonymous ACL submission

Abstract

Large language models have demonstrated remarkable success across a wide range of domains, with supervised fine-tuning being widely adapted to make them more suitable for real-world scenarios. Given the diversity of downstream tasks and varying demands, efficiently deploying multiple full-parameter fine-tuned models presents a significant challenge. To address this, we analyze *Balanced Intermediate Dropout*, a distribution-related phenomenon, whereby the matrix-computed intermediate results for the delta weight of each fine-tuned model have extremely small variance and min-max range. Leveraging this phenomenon, we propose a novel distribution-driven delta compression framework DeltaDQ, which employs *Group-wise Balanced Dropout* and *Delta Quantization* to efficiently compress the delta weight. *Group-wise Balanced Dropout* achieves a favorable trade-off with accuracy and performance, ensuring an N:M sparsity pattern. *Delta Quantization* further compresses the delta weight based on distribution characteristics. Experimental results show that the accuracy of our framework on WizardMath-7B,13B at 96.875% compress rate is improved by 4.47 and 4.70 compared with baseline, and we even improve the accuracy by 1.83 and 0.61 compared with the original model on WizardCoder-13B,34B.

1 Introduction

Large Language Models (LLMs) (Brown et al., 2020; Touvron et al., 2023) have achieved unprecedented advances in recent years, and to be able to utilize LLMs efficiently, most researchers and users have adopted the Supervised Fine-Tuning (SFT) (Ouyang et al., 2022) to emerge the capabilities of LLMs for a variety of different downstream tasks. SFT enables LLMs to achieve better quality in mathematical reasoning, code generation, and other tasks. Meanwhile, despite the existence

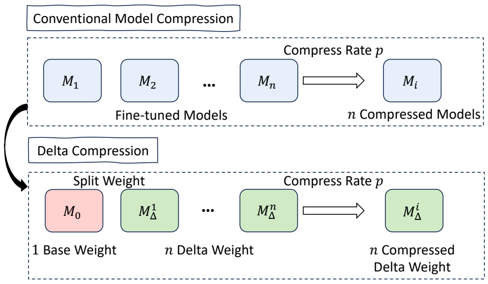


Figure 1: Overview of delta compression

of many Parameter Efficient Fine-Tuning (PEFT) (Ding et al., 2022) methods such as LoRA (Hu et al., 2021), full-parameter fine-tuning models still have higher accuracy under many complex downstream tasks (Chen et al., 2022).

However, how to deploy multiple full-parameter fine-tuning models efficiently in the inference stage becomes a new challenge. One difficulty is the large number of fine-tuned models due to the numerous downstream tasks in real-world scenarios. Another is the great variation of requests from different models, which leads to substantial resource demand and low utilization if all models are deployed simultaneously. Conversely, when the models are loaded according to the request demand, the duration of loading will be long due to the enormous number of parameters of the models, which leads to a sharp increase in the total request latency.

The recent study DELTAZIP (Yao and Klimovic, 2023) highlights the efficacy of delta compression, a novel technique that diverges from conventional methods. As shown in Figure 1, instead of compressing the entire fine-tuned model, delta compression targets the model-specific delta weight, offering a more focused and potentially efficient compression strategy. Although the memory requirement of n homologous fine-tuned models changes from the original $p * n$ to $1 + p * n$ for the same compress rate p, compressing the delta weight allows for a higher compress rate p compared to the original fine-tuned weight, resulting in an overall reduction in memory requirement. Nevertheless, since DELTAZIP uses SparseGPT (Frantar and Alistarh, 2023) to compress the delta weight, it does not fully utilize the properties of delta weight, resulting in a significant loss of accuracy at high compress rates. DARE (Yu et al., 2023a) discover that most of the parameters of delta weight can be randomly dropout (Srivastava et al., 2014), but the compressed delta weight is unstructured, leading to unfriendly deployment.

To address the above issues and achieve efficient deployment of multiple full-parameter fine-tuning models, we propose a novel distribution-driven delta compression framework DeltaDQ. We observe that the intermediate results of delta weight during matrix computation have extremely small variance and min-max range distribution properties, which we call *Balanced Intermediate Results*. Taking advantage of this phenomenon, we design *Group-wise Balanced Dropout*, which groups the elements of each row of delta weight with the size of M, randomly retains N elements, and dropout the rest of the elements, which maintains accuracy while avoids performance degradation by utilizing the N:M sparsity pattern (Mishra et al., 2021). We also incorporate the distributional properties of delta weight and employ *Delta Quantization* to further quantize the delta weight and improve the compress rate. DeltaDQ achieves a compress rate of 96.875% with little or no accuracy loss for WizardMath (Luo et al., 2023a), WizardCoder (Luo et al., 2023b), and MetaMath (Yu et al., 2023b).

Our main contributions are as follows:

- We have discovered *Balanced Intermediate Results*, where delta weight has better compressibility with extremely small variance and min-max range during matrix computation.
- We propose a novel distribution-driven delta compression framework, DeltaDQ, which mainly consists of *Group-wise Balanced Dropout* and *Delta Quantization* to maximize the compress rate while maintaining accuracy and performance.
- Experimental results show that our framework can achieve a compress rate of 96.875%, and specifically the accuracy on WizardMath-7B,13B is improved by 4.47 and 4.70 compared to baseline, respectively. While on WizardCoder-13B,34B we even improve the accuracy by 1.83 and 0.61 compared to the original model.

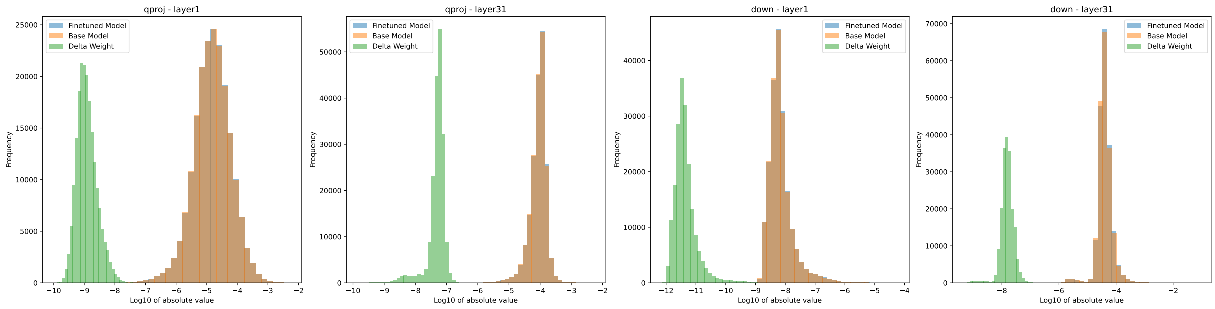
2 Related Work

2.1 Low-Rank Adaptation of Large Language Models

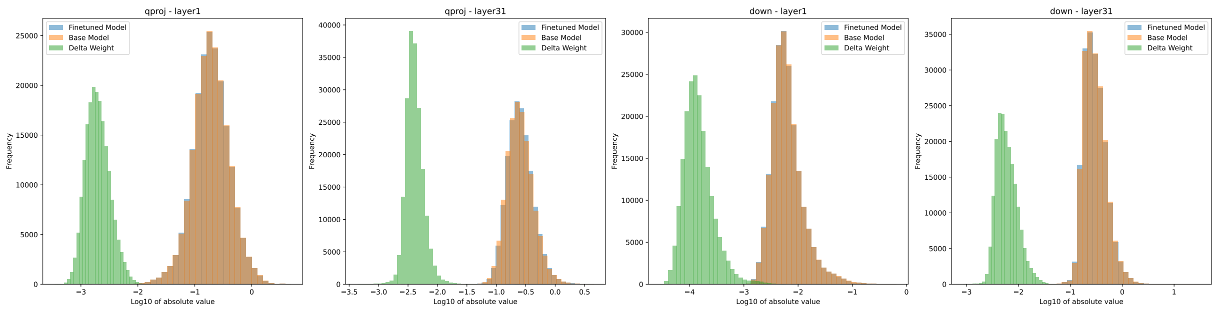
To reduce the resource requirement and cost of SFT, many PEFT methods have appeared, the most representative of which is the Low-Rank Adaptation (LoRA). LoRA (Hu et al., 2021) speeds up computation and reduces resource requirements by updating only low-rank weights. QLora (Dettmers et al., 2023) proposes NF4 quantization to quantize low-rank weights, which further reduces resource requirements. For efficiently serving thousands of lora adapters, there are many multi-tenant lora serving work such as Pets (Zhou et al., 2022), Punica (Chen et al., 2023), and S-Lora (Sheng et al., 2023) that enhance the inference performance. Despite PEFT having substantially reduced the cost of SFT, their accuracy still trails behind that of full-parameter fine-tuning methods on complex tasks (Chen et al., 2022). In this work, we focus on deploying the full-parameter fine-tuning model.

2.2 Model Compression of Large Language Models

Many model compression efforts have also emerged to reduce the deployment cost of LLMs. SparseGPT (Frantar and Alistarh, 2023) and Wanda (Sun et al., 2023) compress models by pruning and removing unimportant parameters. Methods like Smoothquant (Xiao et al., 2023), GPTQ (Frantar et al., 2022), LLM.int8() (Dettmers et al., 2022), and AWQ (Lin et al., 2023b) decrease the resource demands by quantizing the model's representation to a lower bit. In addition to these traditional model compression methods, delta compression shows excellent potential for compressing multiple full-parameter fine-tuning models. DELTAZIP (Yao and Klimovic, 2023) uses SparseGPT to compress the delta weight and builds an inference system, while DARE (Yu et al., 2023a) realizes a model merging method with higher accuracy by randomly dropout the delta weight. However, the former methods experience significant accuracy loss at high compress rates, while the latter is tailored for model merging scenarios where the delta weight eventually merges back into the original weights, leading to unstructured delta weight.



(a) Variance



(b) Min-Max Range

Figure 2: Comparison of the variance and min-max range distribution of the intermediate results of the delta weight, fine-tuned model, and base model for each output element matrix computation

3 Method

3.1 Preliminaries

Delta Compression. Given n full-parameter fine-tuning models $\{M_1, M_2, ..., M_n\}$, which are fine-tuned on the homologous base model, such as Llama. We can split the weights of each model into two parts, the weights of base model W_0 and the delta weight ΔW_i, in the following way:

$$W_i = W_0 + \Delta W_i \ . \tag{1}$$

With the delta weight ΔW_i obtained in the above way, we transform the compressed target into delta weights ΔW_i which is called delta compression.

Layer-Wise Compression. In model compression, the whole optimization problem is usually transformed into a layer-by-layer subproblem. Assume that the weights of l-th layer of model M_i is W_i^l and the compressed weights are $\hat{W}_i^l$, the optimization objective of model compression is to minimize the layer-wise $\mathcal{L}_2$-loss, defined as $\mathcal{L}_{layer}$:

$$\mathcal{L}_{layer} = ||X_i^l {W_i^l}^T - X_i^l {\hat{W}_i^l}^T||_2^2 \ , \tag{2}$$

where X_i^l is the inputs of the l-th layer.

3.2 Balanced Intermediate Results

For a better understanding of the characteristics of delta compression, further analysis of element-wise intermediate results of linear layers has been performed. Assuming $W_i^l, \hat{W}_i^l \in R^{d_1 \times k}$, $X_i^l \in R^{k \times d_2}$ and the outputs $A_i^l, \hat{A}_i^l \in R^{d_1 \times d_2}$, we can transform $\mathcal{L}_{layer}$ further:

$$\begin{aligned} \mathcal{L}_{layer} &= ||X_i^l {W_i^l}^T - X_i^l {\hat{W}_i^l}^T||_2^2 \\ &= ||A_i^l - \hat{A}_i^l||_2^2 \\ &= \sum_{p=1,q=1}^{d_1,d_2} (a_{p,q}|_i^l - \hat{a}_{p,q}|_i^l)^2 \ , \end{aligned} \tag{3}$$

where $a_{p,q}|_i^l$, $\hat{a}_{p,q}|_i^l$ are the elements on the outputs A_i^l and $\hat{A}_i^l$ locations (p, q) before and after compression:

$$a_{p,q}|_i^l = \overbrace{w_{p,0}x_{0,q} + ... + w_{p,k}x_{k,q}}^{e_{all}} \ , \tag{4}$$

and if we use the pruning method to compress the model, setting mask $m_{s_1}, ..., m_{s_1+k_1} \in \{1\}^{k_1}$ and $m_{s_2}, ..., m_{s_2+k_2} \in \{0\}^{k_2}$, there are:

$$\hat{a}_{p,q}|_i^l = \overbrace{w_{p,s_1}x_{s_1,q} + ... + w_{p,s_1+k_1}x_{s_1+k_1,q}}^{e_{stay}} \ . \tag{5}$$

As shown in Figure 2, we find that each element $\Delta a_{p,q}|_i^l$ of the delta weight outputs $X_i^l \Delta {W_i^l}^T$ has the following two properties compared to the original outputs $X_i^l {W_i^l}^T$:

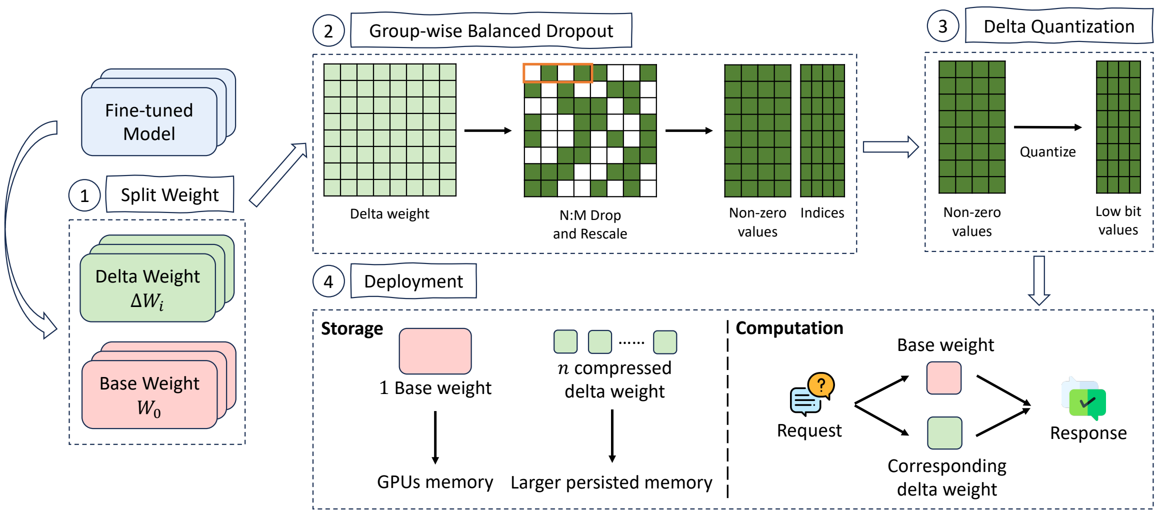


Figure 3: Overview of our DeltaDQ delta compression framework. Our framework is divided into four steps; Step1: Split Weight; Step2: *Group-wise Balanced Dropout*; Step3: *Delta Quantization*; Step4: Deployment.

- Small Variance: The intermediate results $\Delta w_{p,0}x_{0,q}, ..., \Delta w_{p,k}x_{k,q}$ of $\Delta a_{p,q}|_i^l$ have a small variance between them;
- Narrow Min-Max Range: The intermediate results $\Delta w_{p,0}x_{0,q}, ..., \Delta w_{p,k}x_{k,q}$ of $\Delta a_{p,q}|_i^l$ have a very small range between the maximum and minimum values.

We call the above phenomenon *Balanced Intermediate Results*, which explains that the distribution smoothing property of delta weight allows for better compressibility compared to the original fine-tuned weights.

3.3 Overview of DeltaDQ

Based on the above observation, we propose a novel distribution-driven delta compression framework DeltaDQ, which is shown in Figure 3. Firstly, the weights of the fine-tuned model are split into W_0 and ΔW_i. Leveraging *Balanced Intermediate Results* phenomenon, we introduce a simple-yet-effective method *Group-wise Balanced Dropout* to prune the delta weight, striking a favored trade-off between compress rate, accuracy, and performance. In addition, we utilize *Delta Quantization* to enhance the compress rate further. For the final deployment phase, we apply an independent computation strategy, enabling the deployment of as many fine-tuned models as possible.

3.4 Group-wise Balanced Dropout

Exploiting the distributional properties of *Balanced Intermediate Results*, we can randomly drop some of the weights according to the dimension of the matrix computation, i.e., the row dimension of the weights, then rescale the remaining weights, called Balanced Dropout. Through the above method, for any element $\Delta a_{p,q}|_i^l$, we can approximate such that $\Delta a_{p,q}|_i^l$ before compression and $\Delta \hat{a}_{p,q}|_i^l$ after compression are equal, such that $\Delta a_{p,q}|_i^l - \Delta \hat{a}_{p,q}|_i^l$ equals 0, which roughly makes $\mathcal{L}_{layer} = 0$.

However, since the delta weight is unstructured after dropping in this way, it leads to a degradation of the deployment performance. The traditional structured approach drops all on the dimension leading to $\mathcal{L}_{layer} \neq 0$ certainly. Modern GPUs are optimized for the N:M sparsity pattern, enabling them to deliver superior performance with models compressed accordingly. We can simply add N:M constraints to the original Balanced Dropout to *Group-wise Balanced Dropout*, this is done by adding $\Delta W_i^l \in \mathbb{R}^{d_1 \times d_2}$ for the l-th layer:

1. N:M Drop: In the row dimension, m consecutive elements are grouped into sets, from which n elements are randomly selected and the remaining $m - n$ elements are dropped;
2. Rescale: Multiply by $\frac{m}{n}$ for the remaining n elements of each group.

The original whole dimension dropout is similar to simple random sampling with T_1 distinct schemes, whereas *Group-wise Balanced Dropout* resembles grouped sampling, featuring T_2 distinct schemes:

$$T_1 = C(k, k_1) = \frac{k!}{k_1!(k - k_1)!}\ , \qquad (6)$$

$$T_2 = C(m, \frac{k_1}{k} * m)^{\frac{k}{m}} \quad . \tag{7}$$

Although T_1 offers a substantially larger solution set than T_2, the model weights' dimensions, commonly exceeding 1000, render the number of T_2 solutions adequate to maintain the $\mathcal{L}_{layer} = 0$ assumption with reasonable accuracy.

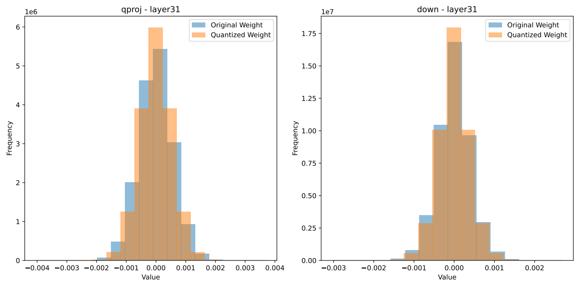


Figure 4: Distribution of weights before and after quantization

3.5 Delta Quantization

While the *Group-wise Balanced Dropout* enables performance to be maintained while reducing memory, M is usually not larger than 32 due to the N:M sparsity pattern limited by the design of current GPUs to avoid the bank conflict issue (Lin et al., 2023a), which limits the maximum compress rate to 96.875% (1:32). Fortunately, according to our insights, the delta weight has the property of being more tightly distributed with fewer outliers. This distributional property allows us to further compress the delta weight using quantization, called *Delta Quantization*.

To avoid inference speed degradation, we use the uniform quantization approach and min-max method to statistically obtain the quantization parameters:

$$X_Q = \mathcal{Q}(X_R) = \text{clamp}(0, 2^{k-1}, \lfloor \frac{X_R}{S} \rceil) \quad , \tag{8}$$

$$S = \frac{\max(|X_R|)}{2^{k-1} - 1} \quad , \tag{9}$$

$$\text{clamp}(l, u, x) = \begin{cases} l, & x \leq l \\ x, & l \leq x \leq u \\ u, & x \geq u \end{cases} \quad , \tag{10}$$

where X_R denotes the original float value, X_Q represents a quantized k-bit integer. l and u represent the lower and upper bounds of the quantization range, respectively. $\lfloor \cdot \rceil$ rounds to the nearest integer. S is the scaling factor.

Quantizing the weights ΔW_i to a lower bit representation can substantially decrease memory usage and bandwidth requirements, thereby accelerating computational speed. Figure 4 illustrates that the weight distribution remains closely aligned with the original distribution following quantization.

3.6 Deployment

In the deployment phase, we utilize the DELTAZIP framework to strategically segregate the storage of the base model weights, W_0, and the compressed delta weight, $\Delta\hat{W}_i$. Specifically, W_0 is maintained in the GPUs memory for quick access, while $\Delta\hat{W}_i$, which are significantly compressed relative to their original size, are stored on a persistent storage solution, such as a hard disk, which offers more expansive storage capabilities. This structure allows for efficient retrieval of $\Delta\hat{W}_i$ from the hard disk to the GPUs memory on an as-needed basis, triggered by incoming computational requests.

Computation processes are conducted in parallel, with the base model weights being processed simultaneously alongside the associated delta weight. After the computation within each linear layer is complete, the outputs are meticulously synchronized to ensure that the final outcome integrates both the foundational computations from the base model and the specific adjustments encoded in the delta weight. This method ensures a streamlined computation workflow that is both memory-efficient and capable of handling dynamic computational demands.

Fine-tuned Model	Base Model
WizardMath-7B	Llama2-7B
WizardMath-13B	Llama2-13B
WizardMath-70B	Llama2-70B
WizardCoder-7B	CodeLlama-7B
WizardCoder-13B	CodeLlama-13B
WizardCoder-34B	CodeLlama-34B
MetaMath-7B	Llama2-7B
MetaMath-13B	Llama2-13B

Table 1: Detailed information on the fine-tuned models selected for the evaluation, including parameter scales and their corresponding base models.

Method	Structured	Quantization	Compress Rate	WizardMath			WizardCoder		
				7B	13B	70B	7B	13B	34B
Dense	✓	✗	0%	55.49	63.83	81.80	55.48	64.02	73.17
Magnitude	✗	✗	50%	52.00	**65.04**	74.29	46.95	63.41	69.51
DELTAZIP	✗	✗	50%	53.60	64.59	**81.65**	53.05	62.80	71.95
DARE	✗	✗	50%	**53.67**	64.36	80.89	57.31	62.80	73.17
DeltaDQ	✓	✗	50%	53.14	63.92	**81.65**	**58.32**	**65.24**	**75.00**
Magnitude	✗	✗	75%	45.71	61.71	44.2	10.36	12.19	35.36
DELTAZIP	✗	✗	75%	50.87	62.17	**81.96**	55.49	64.63	68.29
DARE	✗	✗	75%	51.70	63.52	80.21	**57.92**	61.58	73.17
DeltaDQ	✓	✗	75%	**53.22**	**64.51**	81.95	55.48	**65.24**	**74.39**
Magnitude	✗	✗	87.5%	32.37	55.99	30.47	6.70	2.43	0.00
DELTAZIP	✗	✗	87.5%	46.63	59.29	**80.82**	42.68	59.15	69.51
DARE	✗	✗	87.5%	50.11	63.07	80.28	**56.70**	62.19	71.95
DeltaDQ	✓	✗	87.5%	**53.14**	**63.76**	80.74	**56.70**	**65.24**	**72.56**
Magnitude	✗	✗	96.875%	2.27	30.70	47.83	0.00	0.00	0.00
DELTAZIP	✗	✓	96.875%	46.47	58.83	**80.82**	38.41	53.05	68.29
DARE	✗	✗	96.875%	46.09	58.75	79.90	53.65	65.24	71.34
DeltaDQ	✓	✓	96.875%	**50.94**	**63.53**	80.74	**53.68**	**65.85**	**73.78**

Table 2: Accuracy comparison of WizardMath and WizardCoder at various compress rates, with bold highlighting to indicate the top-performing method for each case. "Structured" represents whether the weights are structured after compression; "Quantization" indicates whether quantization is utilized; "Dense" denotes the original model.

4 Experiment

4.1 Setup

Baseline. We compare DeltaDQ with three prior compression methods. Magnitude (Han et al., 2015) is a classical numerical magnitude-based pruning method and strong baseline. DELTAZIP (Yao and Klimovic, 2023) is the current optimal delta compression framework. And DARE (Yu et al., 2023a) randomly dropout in the entire weights. DeltaDQ and all three methods compress for the delta weight.

Models, Datasets and Evaluation. We evaluate three types of fine-tuned models: WizardMath (Luo et al., 2023a), WizardCoder (Luo et al., 2023b), and MetaMath (Yu et al., 2023b). We choose three parameter sizes for the first two models and two for MetaMath. We exclude MetaMath-70B because it uses LoRA fine-tuning, while we focus on compressing full-parameter fine-tuning models. The detailed information is shown in Table 1. Our evaluation primarily uses two datasets: GSM8k (Cobbe et al., 2021) for assessing WizardMath and MetaMath, and HumanEval (Chen et al., 2021) for WizardCoder. The main metric we look at is the accuracy of these datasets. We evaluate the impact of various compression techniques at four distinct rates: 50%, 75%, 87.5%, and 96.875%

Implementation Details. DeltaDQ is built with PyTorch (Paszke et al., 2017) and utilizes models and datasets from Huggingface Transformers (Wolf et al., 2019), with accuracy assessments conducted through the vLLM framework (Kwon et al., 2023). Other methods use open-source implementations to measure accuracy. All our experiments are conducted on 8 NVIDIA V100 GPUs with 32G of memory and 8 NVIDIA A100 GPUs with 80G of memory.

4.2 Comparison with Baseline

As shown in Table 2, our framework achieves better accuracy for WizardMath and WizardCoder models compared to the remaining there methods in most cases. Our framework, excluding the WizardMath-70B model, attains top accuracy at 87.5% and 96.875% compress rates, with better GPUs efficiency than the former due to the characteristics of its structured compression. Especially, at a 96.875% compress rate, our framework exceeds the state-of-the-art accuracy for WizardMath-

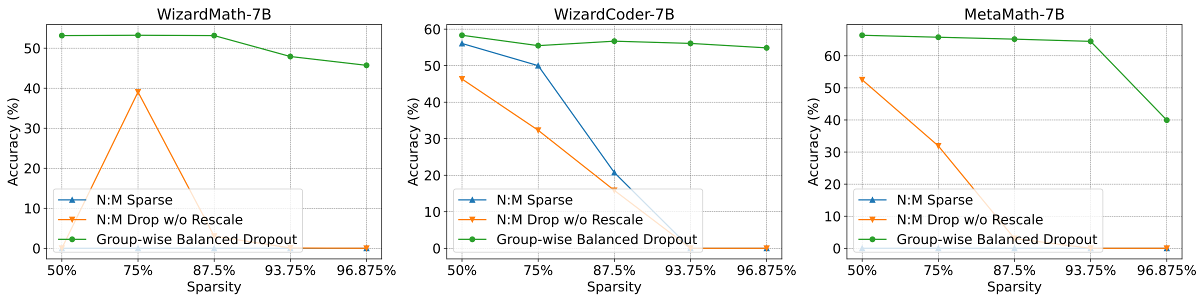


Figure 5: Accuracy comparison of *Group-wise Balanced Dropout* components across various compress rates.

Method	Compress Rate	MetaMath	
		7B	13B
Dense	0%	66.79	71.03
Magnitude	50%	65.73	71.03
DELTAZIP	50%	65.88	71.49
DARE	50%	66.11	70.96
DeltaDQ	50%	**66.41**	**71.56**
Magnitude	75%	61.10	71.03
DELTAZIP	75%	65.13	71.87
DARE	75%	**67.09**	**71.94**
DeltaDQ	75%	65.80	71.11
Magnitude	87.5%	52.69	64.67
DELTAZIP	87.5%	61.26	69.07
DAREDARE	87.5%	**65.88**	69.74
DeltaDQ	87.5%	65.20	**70.58**
Magnitude	96.875%	13.57	42.30
DELTAZIP	96.875%	60.20	68.08
DARE	96.875%	53.29	60.80
DeltaDQ	96.875%	**64.67**	**68.53**

Table 3: Accuracy of MetaMath models at various compress rates, with the optimal method denoted in bold.

7B and 13B models by 4.47 and 4.70, respectively, and outperforms the original WizardCoder-13B and 34B models by 1.83 and 0.61. Table 3 illustrates that, in most cases, DeltaDQ consistently achieves optimal accuracy with the MetaMath model. The comparison results indicate that DeltaDQ is effective and broadly applicable in balancing trade-offs between compress rate, accuracy, and performance.

4.3 Analysis

Evaluation of Group-wise Balanced Dropout. We conduct a detailed analysis of the individual effects exerted by the components within *Group-wise Balanced Dropout*, as illustrated in Figure 5.

N:M	2:4	4:8	8:16	16:32
Acc	51.32	53.07	53.14	52.38
N:M	1:4	2:8	4:16	8:32
Acc	51.93	49.35	53.22	52.46
N:M	1:8	2:16	4:32	8:64
Acc	50.18	52.99	53.14	50.49
N:M	1:16	2:32	4:64	8:128
Acc	47.3	47.91	48.21	49.96

Table 4: Accuracy comparison of WizardMath-7B model across various N:M sparsity patterns.

N:M	2:4	4:8	8:16	16:32
Acc	54.87	56.09	54.87	58.53
N:M	1:4	2:8	4:16	8:32
Acc	54.87	55.48	54.87	54.26
N:M	1:8	2:16	4:32	8:64
Acc	56.70	56.09	54.87	55.48
N:M	1:16	2:32	4:64	8:128
Acc	53.65	56.09	56.09	58.53

Table 5: Accuracy comparison of WizardCoder-7B model across various N:M sparsity patterns.

N:M	2:4	4:8	8:16	16:32
Acc	65.65	66.33	66.26	66.41
N:M	1:4	2:8	4:16	8:32
Acc	65.35	65.65	65.80	65.57
N:M	1:8	2:16	4:32	8:64
Acc	65.20	64.97	64.89	65.27
N:M	1:16	2:32	4:64	8:128
Acc	64.51	61.94	62.24	63.83

Table 6: Accuracy comparison of MetaMath-7B model across various N:M sparsity patterns.

Our findings indicate that the *Rescale* component is critical for the *Group-wise Balanced Dropout*. Omitting *Rescale* leads to a drastic drop in model accuracy, with scores plunging to zero at 93.75% and 96.875% compress rates. This significant decrease can be primarily attributed to the ability of the *Rescale* operation to facilitate an approximation $\mathcal{L}_{layer}$ close to zero under delta compression. Conversely, the conventional N:M Sparse approach fails to accommodate the specialized distribution characteristics of the delta weight, often culminating in a substantial decline in accuracy. Combining *N:M Drop* and *Rescale*, *Group-wise Balanced Dropout* achieves superior accuracy.

Our analysis of accuracy variations for different N:M sparsity patterns is presented in Tables 4, 5, and 6. Given the GPUs bank conflict constraints, we limit M to a maximum of 32 for practical applications. However, for this evaluation, we allow a larger M to explore its potential impact. The experimental results reveal that at the same compress rate, varying N:M sparsity patterns impact accuracy. For example, the WizardMath-7B model at a 75% compress rate demonstrates that the 4:16 pattern yields an accuracy improvement of 3.87 over the 2:8 pattern. Moreover, there is no obvious relationship between the accuracy and the N:M sparsity pattern and a larger M does not necessarily result in higher accuracy.

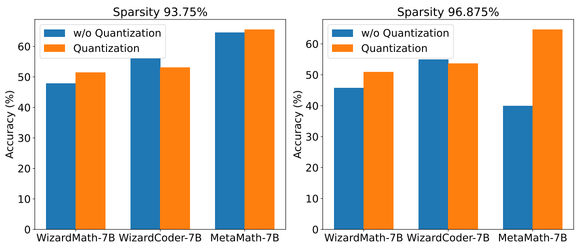


Figure 6: The impact of *Delta Quantization* on accuracy.

Evaluation of Delta Quantization. Our analysis investigates the impact of *Delta Quantization* on model accuracy. As reflected in Figure 6, implementing *Delta Quantization* post *Group-wise Balanced Dropout* enhances model accuracy. Specifically, at a 93.75% compress rate, this approach boosts accuracy by 3.49 for WizardMath-7B and by 1.06 for MetaMath-7B relative to their individual applications. Moreover, at a higher compress rate of 96.875%, the accuracy enhancements are notably pronounced, with WizardMath-7B and MetaMath-7B showing improvements of 5.23 and 24.72, respectively. Although WizardCoder-7B has a slight accuracy degradation using quantization at both compress rates, greater compress rates can be achieved due to the orthogonality of *Delta Quantization* and *Group-wise Balanced Dropout*. The evaluation results demonstrate that *Delta Quantization* is effective in substantially enhancing the compress rate while still preserving both the accuracy and performance of the model.

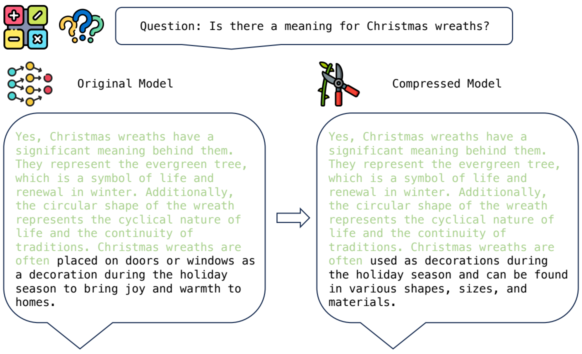


Figure 7: Comparing responses of the WizardLM-7B model before and after DeltaDQ.

Case Study. We additionally evaluate the change in response of WizardLM-7B, a chat model based on Llama fine-tuning, before and after DeltaDQ. As seen in Figure 7, the responses generated by the model before and after applying DeltaDQ exhibit a high degree of similarity for the identical question, even at a high compress rate of 96.875%. This illustrates the generalization of our framework to different types of fine-tuned models and its non-awareness to practical users.

5 Conclusion

In this paper, we introduce DeltaDQ, an innovative framework designed for delta compression. DeltaDQ is primarily composed of two cutting-edge techniques: *Group-wise Balanced Dropout* and *Delta Quantization*. *Group-wise Balanced Dropout* leverages the inherent properties of delta weights to selectively dropout weights in a stochastic manner, while *Delta Quantization* applies additional compression to the compressed weights. Impressively, our framework manages to accomplish lossless compression for the majority of models at an astounding compress rate of 96.875%.

Limitations

The effectiveness of deploying our framework in a real-world setting is dependent on the current state of N:M sparsity software tools and the level of support for such sparsity provided by GPUs hardware. Despite the potential benefits of our framework, the actual deployment performance is currently constrained by the absence of optimized libraries tailored for accelerating operations with low-bit N:M sparse weights. These specialized libraries would be necessary to fully exploit the efficiency gains promised by our framework, as they would enable faster computation and memory access patterns suited to the sparse structure of the weights.

Ethical Impact

Our framework's compression may lead to variations in the model's outputs, as the process can modify the exact values of the model weights, potentially influencing the inference results. However, the framework is optimized to minimize the impact on performance, striving to preserve the quality and consistency of the outputs.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022. Revisiting parameter-efficient tuning: Are we really there yet? *arXiv preprint arXiv:2202.07962*.

Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2023. Punica: Multi-tenant lora serving. *arXiv preprint arXiv:2310.18547*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR.

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

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Bin Lin, Ningxin Zheng, Lei Wang, Shijie Cao, Lingxiao Ma, Quanlu Zhang, Yi Zhu, Ting Cao, Jilong Xue, Yuqing Yang, et al. 2023a. Efficient gpu kernels for n: M-sparse weights in deep learning. *Proceedings of Machine Learning and Systems*, 5.

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

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023a. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023b. Wizardcoder: Empowering code large language models with evol-instruct.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

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

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

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

Xiaozhe Yao and Ana Klimovic. 2023. Deltazip: Multi-tenant language model serving via delta compression. *arXiv preprint arXiv:2312.05215*.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023a. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models.

Zhe Zhou, Xuechao Wei, Jiejing Zhang, and Guangyu Sun. 2022. {PetS}: A unified framework for {Parameter-Efficient} transformers serving. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 489–504.