

QDyLoRA: Quantized Dynamic Low-Rank Adaptation for Efficient Large Language Model Tuning

Hossein Rajabzadeh^{1,2}, Mojtaba Valipour¹, Tianshu Zhu², Marzieh Tahaei²,
Hyock Ju Kwon¹, Ali Ghodsi¹, Boxing Chen², Mehdi Rezagholizadeh²

¹ University of Waterloo, ² Huawei Noah's Ark Lab

{hossein.rajabzadeh, mojtaba.valipour, hjkwon, ali.ghodsi}@uwaterloo.ca

{mehdi.rezagholizadeh, tianshu.zhu, marzieh.tahaei, boxing.chen}@huawei.com

Abstract

Finetuning large language models requires huge GPU memory, restricting the choice to acquire larger models. While the quantized version of the Low-Rank Adaptation technique, named QLoRA, significantly alleviates this issue, finding the efficient LoRA rank is still challenging. Moreover, QLoRA is trained on a pre-defined rank and, therefore, cannot be reconfigured for its lower ranks without requiring further fine-tuning steps. This paper proposes QDyLoRA -Quantized Dynamic Low-Rank Adaptation-, as an efficient quantization approach for dynamic low-rank adaptation. Motivated by Dynamic LoRA, QDyLoRA is able to efficiently finetune LLMs on a set of pre-defined LoRA ranks. QDyLoRA enables fine-tuning Falcon-40b for ranks 1 to 64 on a single 32 GB V100-GPU through one round of fine-tuning. Experimental results show that QDyLoRA is competitive to QLoRA and outperforms when employing its optimal rank.

1 Introduction

The popularity of adopting Large Language Models (LLMs) across a diverse range of downstream tasks has rapidly increased over the past two years. Fine-tuning LLMs has become necessary to enhance their performance and introduce desired behaviors while preventing undesired outputs (Ding et al., 2023). However, as the size of these models increases, fine-tuning costs become more expensive. This has led to a large body of research that focuses on improving the efficiency of the fine-tuning stage (Liu et al., 2022; Mao et al., 2021; Hu et al., 2021; Edalati et al., 2022; Sung et al., 2022).

Low-rank adapter (LoRA) (Hu et al., 2021) is a well-known, parameter-efficient tuning (PEFT) method that reduces memory requirements during fine-tuning by freezing the base model and updating a small set of trainable parameters in form of low-rank matrix multiplication added to matrices in the base model. However, the memory demand

during fine-tuning remains substantial due to the necessity of a backward pass through the frozen base model during stochastic gradient descent.

Recent research has thus focused on further reducing memory usage by designing new parameter-efficient modules that can be tuned without necessitating gradients from the base models (Sung et al., 2022). Alternatively, researchers have explored combining other efficiency strategies with parameter-efficient tuning methods (Kwon et al., 2022; Dettmers et al., 2023).

Among these approaches, QLoRA (Dettmers et al., 2023) stands out as a recent and highly efficient fine-tuning method that dramatically decreases memory usage. It enables fine-tuning of a 65-billion-parameter model on a single 48GB GPU while maintaining full 16-bit fine-tuning performance. QLoRA achieves this by employing 4-bit NormalFloat (NF4), Double Quantization, and Paged Optimizers as well as LoRA modules.

However, another significant challenge when utilizing LoRA modules is the need to tune their rank as a hyperparameter. Different tasks may require LoRA modules of varying ranks. In fact, it is evident from the experimental results in the LoRA paper that the performance of models varies a lot with different ranks, and there is no clear trend indicating the optimal rank. On the other hand, any hyperparameter tuning for finding the optimal rank contradicts the primary objective of efficient tuning and is not feasible for very large models. Moreover, when deploying a neural network on diverse devices with varying configurations, the use of higher ranks can become problematic for highly sensitive devices due to the increased parameter count. To address this, one typically has to choose between training multiple models tailored to different device configurations or determining the optimal rank for each device and task. However, this process is costly and time-consuming, even when using techniques like LoRA.

Table 1: A comparison between QLoRA and QDyLoRA on the MMLU benchmark, reporting 5-shot test results for LLMs of varying sizes. QDyLoRA is evaluated on ranks [1,2,4,8,16,32,64] and the best rank is reported in brackets.

Dataset	LLaMA-7b		LLaMA-13b		Falcon-40b	
	QLoRA	QDyLoRA	QLoRA	QDyLoRA	QLoRA	QDyLoRA
Alpaca	38.8 [64]	39.7 [16]	47.8 [64]	47.6 [8]	55.2 [64]	57.1 [4]
OASST1	36.6 [64]	36.8 [16]	46.4 [64]	47.2 [8]	56.3 [64]	56.7 [4]
Self-Instruct	36.4 [64]	37.2 [8]	33.3 [64]	41.6 [4]	51.8 [64]	51.1 [4]
FLAN-v2	44.5 [64]	45.9 [4]	51.4 [64]	52.1 [8]	58.3 [64]	60.2 [4]

DyLoRA (Valipour et al., 2022), is a recent PEFT method that aims to address these challenges by employing dynamic Low-Rank Adapter (DyLoRA). Inspired by nested dropout, this method aims to order the representations of the bottleneck at low-rank adapter modules. Instead of training LoRA blocks with a fixed rank, DyLoRA extends training to encompass a spectrum of ranks in a sorted manner. The resulting low-rank PEFT modules not only provide increased flexibility during inference, allowing for the selection of different ranks depending on the context, but also demonstrate superior performance compared to LoRA, all without imposing any additional training time.

In this paper, we employ the DyLoRA PEFT method in conjunction with the quantization scheme utilized in the QLoRA work, resulting in QDyLoRA. QDyLoRA has all the aforementioned benefits of DyLoRA but with significant memory reduction both during training and at inference through 4-bit quantization. We utilize QDyLoRA for efficient fine-tuning of LLaMA-7b, LLaMA-13b, and Falcon-40b models across ranks ranging from 1 to 64, all on a single 32GB V100 GPU. Once tuned, we determine the optimal rank by inferring the model on the test set. Our results reveal that the optimal rank can be quite low, yet it outperforms QLoRA.

1.1 Related Work

Low-rank PEFT methods These methods aim to fine-tune pre-trained LLMs for specific tasks while minimizing computational and memory resources. Low-rank adaptation techniques were inspired by (Aghajanyan et al., 2020), demonstrating that pre-trained language models possess a low intrinsic dimension. Since then, several works have explored the incorporation of trainable parameters in the form of low-rank up-projection/down-projection during fine-tuning. In (Houlsby et al., 2019), the Adapter module includes a down projec-

tion, a non-linear function, an up projection, and a residual connection. These modules are sequentially inserted after the feed-forward network (FFN) or attention blocks.

Additionally, (He et al., 2021) extends the Adapter concept by introducing trainable modules that run in parallel (PA) with the original pre-trained language-model (PLM) module. As a result of this extension, PA has demonstrated improved performance compared to the original Adapter method. One notable approach among these techniques is LoRA (Hu et al., 2021), which introduces low-rank up-projection/down-projection into various matrices within a PLM. This method offers efficient inference by seamlessly integrating the adapter module into the original model’s weight matrices.

Quantization-aware PEFT methods Alpha-Tuning (Kwon et al., 2022), aims to combine parameter-efficient adaptation and model compression. Alpha-Tuning achieves this by employing post-training quantization, which involves converting the pre-trained language model’s full-precision parameters into binary parameters and separate scaling factors. During adaptation, the binary values remain fixed for all tasks, while the scaling factors are fine-tuned for the specific downstream task.

QLoRA (Dettmers et al., 2023) is a more recent quantization-aware PEFT that combines a low-rank adapter with 4-bit NormalFloat (NF4) quantization and Double Quantization (DQ) of the base model to optimize memory usage. NF4 ensures an optimal distribution of values in quantization bins, simplifying the process when input tensors have a fixed distribution. DQ further reduces memory overhead by quantizing quantization constants.

To manage memory during gradient checkpointing, QLoRA employs Paged Optimizers, utilizing NVIDIA’s unified memory feature for effi-

Table 2: Comparing the performance of QLoRA and QDyLoRA across different evaluation ranks. Both models receives the same training settings. Maximum LoRA rank is set to 64. Falcon-40b is adopted as the base LLM. Exact matching and Bleu-score are used as evaluation measurements for GSM8k and Web-GLM, respectively.

Data	Method	Rank						
		1	2	4	8	16	32	64
Web-GLM	QLoRA	19.9	19.9	19.9	33.8	35.2	52.7	54.3
	QDyLoRA	43.3	56.0	54.9	53.3	53.3	50.5	50.2
GSM8k	QLoRA	8.9	8.91	8.9	15.1	20.5	22.6	28.1
	QDyLoRA	21.4	25.3	28.2	30.6	29.8	28.5	27.4

cient GPU memory management. These techniques collectively enable high-fidelity 4-bit fine-tuning while effectively handling memory constraints.

Dynamic PEFT methods DyLoRA paper (Valipour et al., 2022) introduces a novel approach for training low-rank modules to work effectively across a range of ranks simultaneously, eliminating the need to train separate models for each rank.

Inspired by the concept of nested dropout, the authors propose a method for organizing the representations within low-rank adapter modules. This approach aims to create dynamic low-rank adapters that can adapt well to various ranks, rather than being fixed to a single rank with a set training budget. This is achieved by dynamically selecting ranks during training, allowing for greater flexibility without the need for extensive rank searching and multiple model training sessions.

Algorithm 1 QDyLoRA - Training and Inference

Require: $r \in [r_{min}, r_{max}]$; i : the number of training iterations; α : a scaling factor; p_B : probability distribution function for rank selection; $X \in \mathbb{R}^{d \times n}$: all input features to LoRA; $W_0 \in \mathbb{R}^{m \times d}$ the original frozen pre-trained weight matrix, $W_{dw} \in \mathbb{R}^{r \times d}$; $W_{up} \in \mathbb{R}^{m \times r}$; Q : Quantizer; $L_{\downarrow b}^{DY}$: objective function given truncated weights

Initialization:
 $W_0^{NF4} = Q(W_0)$ // Quantize W_0 to NF4

Iterations:
while $t < i$ **do**
Forward:
 $b \sim p_B(\cdot)$ // sample a specific rank, during test is given
 $W_{dw\downarrow b} = W_{dw}[:, b]$ // truncate down-projection matrix
 $W_{up\downarrow b} = W_{up}[:, b]$ // truncate up-projection matrix
 $W_0^{DDequant-NF4} = \frac{W_0^{NF4}}{c_2^{FP8}/c_1^{FP32}}$ // dequantized the chunks of the parameters that are needed
 $h = W_0^{DDequant-NF4} X^{BF16} + \frac{\alpha}{b} W_{up\downarrow b}^{BF16} W_{dw\downarrow b}^{BF16} X^{BF16}$ // calculate the LoRA output
Backward:
 $W_{dw\downarrow b}^{BF16} \leftarrow W_{dw\downarrow b}^{BF16} - \eta \nabla_{W_{dw\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{DY}$
 $W_{up\downarrow b}^{BF16} \leftarrow W_{up\downarrow b}^{BF16} - \eta \nabla_{W_{up\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{DY}$
end while

2 Proposed Method: Quantized DyLoRA

Following DyLoRA notations (Valipour et al., 2022), we define a truncated weight $W_{\downarrow b} \in \mathbb{R}^{r \times d}$

as $W[:, b, :]$. Assume we have a set of input features $X \in \mathbb{R}^{d \times n}$, a set of pre-trained weights W_0 , and a given range of desired ranks represented by $r \in [r_{min}, r_{max}]$ that we want the model to operate with, and a dynamic objective function $L_{\downarrow b}^{DY}$ that can evaluate a truncated sub-model. Then we can use the following equation to calculate the forward pass of the model at each iteration.

$$h = W_0^{DDequant-NF4} x^{BF16} + \frac{\alpha}{b} W_{up\downarrow b}^{BF16} W_{dw\downarrow b}^{BF16} x^{BF16} \quad (1)$$

where α is the LoRA scalar, and b is the chosen rank by the $p_B(\cdot)$ during training stage.

Following QLoRA (Dettmers et al., 2023), we used 4-bit Normal Float (NF4) for storing the double quantized pre-trained weights. As all the computations need to be calculated in BFloat16 precision, DDequant-NF4 will dequantize the stored data. Similar to (Dettmers et al., 2023), we have:

$$W_0^{DDequant-NF4} = \frac{W_0^{NF4}}{c_2^{FP8}/c_1^{FP32}} \quad (2)$$

where c_1^{FP32} and c_2^{FP8} are quantization constants introduced in (Dettmers et al., 2023). After this process, we can update the dynamic LoRA parameters using:

$$\begin{aligned} W_{dw\downarrow b}^{BF16} &\leftarrow W_{dw\downarrow b}^{BF16} - \eta \nabla_{W_{dw\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{DY} \\ W_{up\downarrow b}^{BF16} &\leftarrow W_{up\downarrow b}^{BF16} - \eta \nabla_{W_{up\downarrow b}^{BF16}} \mathcal{L}_{\downarrow b}^{DY} \end{aligned} \quad (3)$$

Algorithm 1 describes the workflow of our proposed QDyLoRA in detail.

3 Experiments and Evaluation

This section evaluates the efficiency and efficacy of QDyLoRA through several instruct-fine-tuning

Table 3: Comparing the performance of DyLoRA, QLoRA and QDyLoRA across different evaluation ranks. all models receives the same training settings. Maximum LoRA rank is set to 64. The results are reported in terms of exact matching.

Data;LLM	Method	Rank						
		1	2	4	8	16	32	64
GSM8K;LLaMA-7b	DyLoRA	12.96	16.91	17.06	19.94	18.50	18.35	14.94
	QLoRA	0.0	0.0	0.0	0.0	0.0	0.0	12.66
	QDyLoRA	12.59	15.09	18.50	16.76	16.91	18.65	14.71
TriviaQA;LLaMA-7b	DyLoRA	19.27	23.20	22.99	23.32	23.25	24.12	22.43
	QLoRA	0.0	0.0	0.0	0.0	0.0	0.0	15.52
	QDyLoRA	6.66	12.49	17.16	19.51	20.09	21.65	20.27
GSM8K;LLaMA2-13b	DyLoRA	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	QLoRA	0.0	0.0	0.0	0.0	0.0	0.0	21.08
	QDyLoRA	1.90	15.01	22.97	25.55	24.26	23.81	22.08

tasks. The first experiment compares QDyLoRA with QLoRA on Massively Multitask Language Understanding (MMLU) benchmark (Hendrycks et al., 2020), consisting of more than 50 different tasks, spanning from fundamental mathematics and U.S. history to computer science and law. As shown in Table 1¹, we finetune LLaMA-7b, LLaMA-13b, LLaMA2-13b, and Falcon40b on different datasets, Alpaca (Taori et al., 2023), OASST1 (Köpf et al., 2023), Self-Instruct (Wang et al., 2022), and FLAN-v2 (Chung et al., 2022), using QLoRA and QDyLoRA techniques. We use the same training budget and maximum LoRA rank² for each technique. The results consistently show that QDyLoRA achieves a superior performance by finding the optimal rank.

The second experiment provides a more in-depth comparison between QLoRA and QDyLoRA. In particular, we fairly finetuned Falcon-40b on WebGLM (Liu et al., 2023) and GSM8k (Cobbe et al., 2021) benchmarks, and compared their test performances across different ranks. As described in Table 2, QDyLoRA attains superior performance, notably when employing its optimal ranks (Rank 2 for Web-GLM and Rank 8 for GSM8k). Furthermore, QDyLoRA exhibits consistent superiority over QLoRA, particularly at lower ranks. These findings emphasize the adaptive nature of QDyLoRA in dynamically adjusting its focus during fine-tuning, leading to enhanced efficiency and efficacy compared to its static counterpart, QLoRA. The third experiment compares the performance of DyLoRA, QDyLoRA, and QLoRA on GSM8k

and TriviaQA (Joshi et al., 2017) while adopting LLaMA2-13b and LLaMA-7b as LLMs. Table 3 reports the results. As the table illustrates, for smaller-size models, i.e. LLaMA-7b, DyLoRA and QDyLoRA both perform superior than QLoRA. For larger models, i.e. LLaMA2-13b, DyLoRA fails due to the out-of-memory (OOM) error while QDyLoRA works the best in such situations.

4 On the semi-sorted behavior of QDyLoRA

As shown in Table 2, QDyLoRA reveals a semi-sorted performance across ranks. We justify this behavior by pointing out the limited finetuning budget. In a limited budget assumption, QDyLoRA updates its lower ranks more frequently than its higher ranks. That is because of the fact that lower ranks are also updated when higher ranks are selected. In other words, lower ranks have more chance to get updated than higher ranks. Hence, lower ranks are more tuned than higher ranks.

5 Conclusion

QDyLoRA offers an efficient and effective technique for LoRA-based fine-tuning LLMs on downstream tasks. Eliminating the need for fine-tuning multiple models to find the optimal LoRA rank and offering the possibility of fine-tuning larger LLMs are two main advantages of QDyLoRA. The experimental results demonstrated that the optimal rank for QDyLoRA can be surprisingly low, yet it consistently outperforms QLoRA. QDyLoRA provides greater flexibility for deploying LLMs in various contexts and represents a promising step towards making fine-tuning large language models more accessible and efficient.

¹The same settings as the original QLoRA work are applied here.

²The maximum LoRA rank is fixed to 64. While QLoRA’s rank is always fixed, QDyLoRA can split the training across ranks in range 1 to 64.

Limitations

While the 4-bit QDyLoRA exhibits notable performance, it falls short of achieving the performance levels of full precision fine-tuning. One possible solution could be dynamic quantized DyLoRA (DyQDyLoRA), in which the quantization level could also vary during finetuning. In particular, the finetuning strategy can dynamically switch between different quantization levels based on a predefined learning feedback. Additionally, further research is required to investigate the impact of LoRA's scalar and the range of underlying ranks in QDyLoRA.

References

- Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- 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. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.
- Ali Edalati, Marzieh Tahaei, Ivan Kobzyev, Vahid Par-tovi Nia, James J Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*.
- Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.
- 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*.
- Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.
- Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, et al. 2023. Openassistant conversations—democratizing large language model alignment. *arXiv preprint arXiv:2304.07327*.
- Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. 2022. Alphasoft: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. *arXiv preprint arXiv:2210.03858*.
- Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Moh-ta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.
- Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. Webglm: Towards an efficient web-enhanced question answering system with human preferences. *arXiv preprint arXiv:2306.07906*.
- Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*.
- Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. *Advances in Neural Information Processing Systems*, 35:12991–13005.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.
- Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobzyev, and Ali Ghodsi. 2022. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hananeh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

A Supplementary Material

A.1 Hyperparameters

Table 4 provides an overview of the hyperparameters and experimental configurations employed in this study, which are crucial configurations that determine various aspects of the training process and model behavior in this study. Common key parameters across the experiments include the choice of optimizer, Adam-Beta2 value, maximum gradient norm, and warmup ratio, which collectively influence how the model adjusts its weights during training. LoRA-specific parameters such as LoRA dropout probability, maximum LoRA rank, and alpha value control the behavior of LoRA layers. Additionally, double quantization and quantization type impact the precision of numerical representations within the model, which are considered the same as baselines. Learning rate scheduling and weight decay contribute to the optimization process, helping to prevent overfitting and stabilize training. Random seeds ensure reproducibility, while the specified GPU determines the hardware used for training. Each model configuration, whether for the Web-GLM, GSM8k, or the specific experiment outlined in Table 1 and Table 3, features parameters tailored to the characteristics of the dataset and the computational resources available. These hyperparameters collectively shape the training process, ultimately influencing the performance and effectiveness of the models in the study.

A.2 Generated Text Quality

To describe the quality of texts generated by QDy-LoRA, a sample query taken from GSM8k is fed to Falcon-40b trained by QDyLoRA. Below, we report the generated answers for different LoRA ranks.

Query: Janet’s ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers’ market daily for \$2 per fresh duck egg. How much in dollars does she make every day at the farmers’ market?

Rank=64: Janet has $16 - 3 - 4 = \ll 16-3-4=7 \gg 7$ eggs left. She sells $7 - 16 = \ll 7-16=9 \gg 9$ eggs at the

Model	Parameter	Value
Common settings	Optimizer	paged-adamw-32bit
	Adam-Beta2	0.999
	Max-Grad-Norm	0.3
	Warmup Ratio	0.03
	LoRA-Dropout	0.05
	LoRA-r	64
	LoRA-Alpha	16
	LoRA-Module	all
	Double-Quant	True
	Quant-Type	nf4
	LR Scheduler	Constant
	Learning Rate (LR)	1e-5
	Weight Decay	0.0
Seeds	[42]	
GPU	Tesla V100-PCIE-32GB	
Web-GLM	Max-Steps	2700
	Gradient-Accumulation-Step	2
	Source-Max-Len	2048
	Target-Max-Len	2048
	Batch Size	16
GSM8k	Max-Steps	300
	Gradient-Accumulation-Step	2
	Source-Max-Len	2048
	Target-Max-Len	2048
	Batch Size	16
Table 1	Max-Steps	5000
	Gradient-Accumulation-Step	16
	Source-Max-Len	16
	Target-Max-Len	512
	Batch Size	32
Table 3 - GSM8k	Save-Steps	30
	Early-Stopping-Patience	5
	Source-Max-Len	1024
	Target-Max-Len	1024
	Batch Size	2 for DyLoRA; 8 for QLoRA/QDyLoRA
Gradient-Accumulation-Step	8 for DyLoRA; 2 for QLoRA/QDyLoRA	
Table 3 - TriviaQA	Save-Steps	200
	Early-Stopping-Patience	5
	Source-Max-Len	512
	Target-Max-Len	512
	Batch Size	4 for DyLoRA; 4 for QLoRA/QDyLoRA
Gradient-Accumulation-Step	16 for DyLoRA; 1 for QLoRA/QDyLoRA	

Table 4: The list of hyperparameters employed across various experiments in our study. The common settings are applied to each experiments. If a parameter is not explicitly mentioned, it means that we use default values from HuggingFace Transformers v’4.28.0.dev0’ (source: <https://huggingface.co/docs/transformers>).

farmers’ market. She makes $9 * 2 = \ll 9*2=18 \gg 18$ at the farmers’ market every day.

Rank=32: She eats 3 eggs for breakfast and bakes 4 muffins per day, so she uses $3 + 4 = \ll 3+4=7 \gg 7$ eggs per day. She has $16 - 7 = \ll 16-7=9 \gg 9$ eggs left over each day. She sells these eggs at the farmers’ market for \$2 per egg, so she makes $2 * 9 = \ll 2*9=18 \gg 18$ per day at the market.

Rank=28: She eats 3 eggs for breakfast and bakes 4 muffins per day, so she uses $3+4 = \ll 3+4=7 \gg 7$ eggs per day. She sells $16-7 = \ll 16-7=9 \gg 9$ eggs per day for \$2 per egg. She makes $9*2 = \ll 9*2=18 \gg 18$ per day at the market.

Rank=16: Janet’s ducks lay 16 eggs per day and she eats 3 for breakfast every morning so she has $16-3=13$ eggs left. Janet bakes 4 muffins every day and she uses 4 eggs so she has $13-4=9$ eggs left. Janet sells the remaining eggs at the farmers’ market for \$2 each so she makes $9*2=\ll 9*2=18 \gg 18$

dollars a day. Janet makes 18 dollars a day at the farmers' market.