AdaQuantLM: LLM Quantization with Adaptive Bit-Widths

Shuangyi Chen, Ashish Khisti
Department of ECE, University of Toronto
shuangyi.chen@mail.utoronto.ca, akhisti@ece.utoronto.ca

Abstract

Current LLM quantization methods focus on single bitwidth quantization, requiring time-consuming finetuning and benchmarking for each bitwidth version, which limits their adaptability to different scenarios. To address these challenges, we propose AdaQuantLM, a method for LLM quantization with adaptive bit-width. Inspired by techniques such as AdaBits and Additive Quantization for Language Models (AQLM), AdaQuantLM leverages the additivity of codewords in quantized models. This allows for the efficient conversion between different bit-widths by adding or removing specific codewords, eliminating the need for storing full-precision weights. Our approach jointly quantizes and fine-tunes LLMs across multiple bit-widths, enabling the model to adapt to devices with varying computational resources while maintaining performance. We demonstrate the effectiveness of AdaQuantLM through experiments on the Gemma-2b model, highlighting its potential for broad applicability in the efficient deployment of LLMs.

1 Introduction

Model quantization is a well-established technique that has been thoroughly studied and refined over the years. It is widely applied to optimize neural networks, particularly in environments with limited computational power and memory, such as mobile devices and embedded systems. However, while quantization is well understood and broadly applied in traditional neural networks, the emergence of Large Language Models (LLMs) introduces new complexities. These models, with their immense scale and intricate architectures, present unique challenges in quantization that remain largely unexplored.

Recent quantization methods for LLMs are typically divided into two categories: zero-shot and optimization-based. Zero-shot techniques, such as Llm.int8() [3] and QLoRA [4], use scaling operations to normalize parameters before quantization, allowing users to perform the process locally with minimal computational resources. Optimization-based methods, such as those in [6, 7, 10, 5], focus on minimizing quantization error using a calibration dataset. These resource-intensive processes are typically done centrally, with the resulting quantized models distributed for use. All of these methods focus on single bitwidth quantization. However, this approach is inefficient because it requires time-consuming fine-tuning and benchmarking for each quantized version, making it difficult to quickly adapt to different scenarios. As illustrated in Figure 2, directly combining the adapter from 8-bit LLM fine-tuning with a 4-bit LLM results in a decline in model performance. While separate quantizing and finetuning process for the 4-bit LLM is required to achieve comparable performance, this approach is not economical.

In order to overcome such shortcomings, we aim to quantize a LLM and finetune it only once and use it on various bitwidth settings, as shown in the bottom of Figure 1. Many approaches addresses this common issue for traditional neural networks. AdaBits [8] was the first approach to introduce the idea of training a single model adaptable to multiple bit-widths. It achieves this by jointly training

Work Presented at the Machine Learning and Compression Workshop @ NeurIPS 2024

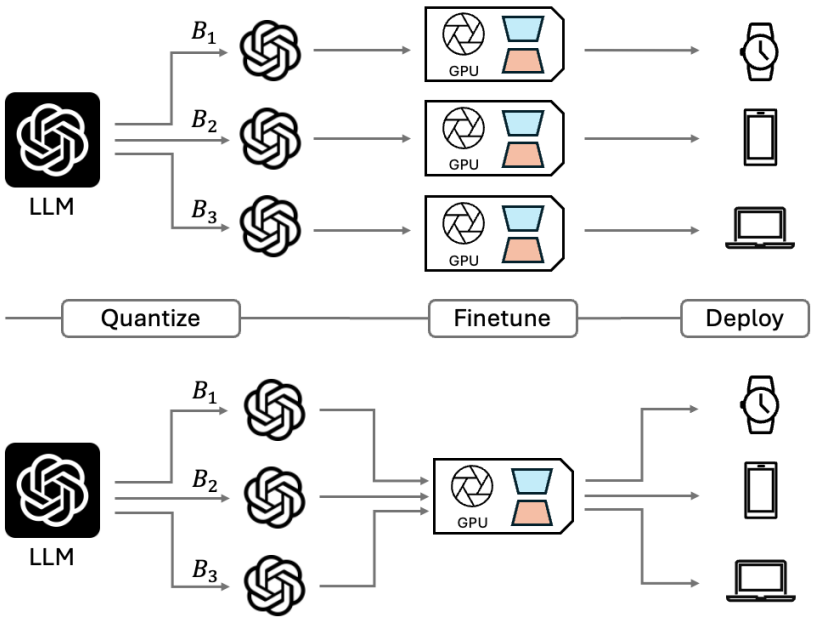

Figure 1: Deployment of LLMs with different bit-widths according to the computational budget. Above: Individually finetune several quantized LLMs with different bit-widths for each scenario. Bottom: Finetune a single LLM quantized with adaptive bit-widths and switch to the proper bit-width in real application based on the device condition.

multiple bit-width versions. Additionally, it introduces a nested quantization scheme where lower bitwidth model weights can be derived directly from higher bitwidth weights, rather than from the full-precision model. The direct conversion between quantized models of different bitwidths obviates the needs for the storage of the full-precision weights, which is especially beneficial for LLMs.

A recent work, AQLM [6], introduces Additive Quantization for Language Models, which compresses the weight matrices of LLMs while preserving model accuracy by using multiple codebooks to better represent the weights. Inspired by Adabits [8] and AQLM, we developed our method for LLM quantization with adaptive bit-width, called AdaQuantLM. Leveraging the inherent additivity of codewords in Additive Quantization, our approach represents the quantized weight vector using multiple codebooks corresponding to different bit-width codes. The transition between quantized LLMs of different bit-widths can be achieved by adding or dropping specific codewords. To ensure that the quantized model can adapt to devices with varying computational resources while maintaining comparable performance, we jointly quantize and finetune LLMs of different bit-widths, learning the codebooks and codes alternately following AQLM. We evaluate the effectiveness of our method on the Gemma-2b model [12].

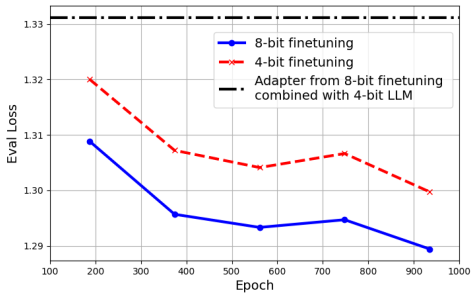

Figure 2: Comparison of finetuning 8-bit and 6-bit version Llama-7b[13] on OASST1[9], the black line denotes the loss level when we combine the adapter obtained from 8-bit finetuning with 6-bit version.

2 Related Works & Preliminaries

2.1 LLM Quantization

To achieve memory-efficient model inference, LLMs are often deployed with lower-precision quantized weights. Quantization methods for LLMs generally fall into two categories: zero-shot and optimization-based quantization. The first category includes techniques like Llm.int8() [3] and QLoRA [4], which rely on scaling operations to normalize parameters before mapping them into a predefined set of quantization levels. Zero-shot quantization methods are computationally lightweight, allowing users to download the full-precision model and perform quantization locally. However, this approach can lead to lower precision, potentially resulting in a degradation of model performance.

On the other hand, optimization-based methods, such as those in [6, 7, 10, 5], focus on minimizing quantization error, often using a calibration dataset. Due to the resource-intensive nature of these optimization processes, they are typically performed once by a designated entity, with the resulting quantized models distributed directly. However, these methods require significant computational resources, making them impractical for many non-tech companies. Additionally, current LLM quantization methods are limited in that a LLM is usually finetuned for a single dedicated bitwidth, showing significant performance degradation when quantized to other bitwidths. In other words, supporting multiple bitwidths would necessitate quantizing and fine-tuning multiple copies of the model for each specific bitwidth, as depicted in the above part of Figure 1.

In this work, we focus on optimization-based quantization methods and aim to address the resource-intensive challenges associated with multi-bit LLM quantization with adaptive configurations.

2.2 Adaptive Quantization

In neural network quantization, many approaches aim to train a model once and enable its use across various bit-width settings. This strategy addresses the common issue where a model trained for a specific bit-width often suffers significant performance degradation when quantized to other bit-widths. AdaBits [8] was the first approach to introduce the idea of training a single model adaptable to multiple bit-widths. It achieves this by jointly training multiple bit-width versions and applying Switchable Clipping Levels to enhance the performance of quantized models at the lowest bit-width. In Any-precision [15], the authors propose a training framework based on knowledge distillation, allowing the model to be flexibly and directly adjusted to different bit-widths during runtime by truncating the least significant bits, facilitating a dynamic trade-off between speed and accuracy. Bit-Mixer [1] was introduced to train a meta-quantized network where any layer can change its bit-width during testing without compromising the overall network's performance. MEBQAT [14] achieves adaptive bit-width quantization-aware training (QAT) by incorporating bit-width settings and average gradients across different configurations, which are defined as meta-learning tasks. While these methods have proven effective for quantizing and training smaller neural networks, they remain impractical for large language models (LLMs).

2.3 Prior Work: AQLM

A recent work [6] introduces the Additive Quantization for Language Models (AQLM) method for the extreme compression of large language models. AQLM extends the classical Additive Quantization (AQ) approach, which has been primarily used in information retrieval, to compress the weight matrices of LLMs while maintaining model accuracy. The method involves two key innovations: first, it introduces learned additive quantization that adapts to the input distribution, and second, it jointly optimizes the quantization codebooks across different layers of the transformer model. These innovations allow AQLM to achieve Pareto-optimal performance in terms of the trade-off between accuracy and model size, especially in scenarios requiring compression to less than 3 bits per parameter.

Specifically, given a linear layer with weights $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ and calibration inputs $\mathbf{X} \in \mathbb{R}^{d_{in} \times n}$, the goal is to find quantized weights $\widehat{\mathbf{W}}$ that minimize the squared error between the original and compressed outputs:

$$\arg\min_{\widehat{\mathbf{W}}} ||\mathbf{W}\mathbf{X} - \widehat{\mathbf{W}}\mathbf{X}||_2^2. \tag{1}$$

In AQLM, $\widehat{\mathbf{W}}$ is quantized by splitting weight rows into groups of g elements, each represented as a sum of M vectors from learned codebooks $C_1, \ldots, C_M$, with each vector selected by a one-hot code b_m. The full weight matrix is then constructed by concatenating these groups:

$$\widehat{\mathbf{W}}_i = \sum_{m=1}^{M} C_m b_{i,1,m} \oplus \cdots \oplus \sum_{m=1}^{M} C_m b_{i,d_{in}/g,m}. \tag{2}$$

The algorithm learns the codebooks $C_m \in \mathbb{R}^{g \times 2^B}$ and codes b, and optimizes the error:

$$\arg\min_{C,b} ||\mathbf{W}\mathbf{X} - \left(\text{Concat}_{i,j} \sum_{m=1}^{M} C_m b_{i,j,m}\right) \mathbf{X}||_2^2. \tag{3}$$

The process starts with a residual K-means initialization of the codebooks and codes, followed by iterative alternating optimization of both until convergence. The method is detailed in the following three phases.

1. Code Optimization: The method first optimizes the codes (one-hot vectors) that represent the weight matrices by using a beam search algorithm, which iteratively refines these codes to minimize the error between the original and quantized model outputs.
2. Codebook Update: Next, it updates the vectors in the codebooks by solving a least squares problem, minimizing the difference between the quantized and original weights.
3. Intra-Layer Fine-Tuning: Finally, AQLM performs fine-tuning at the transformer block level to further reduce quantization errors. This involves adjusting the non-quantized parameters and codebooks together, ensuring the quantized model's outputs remain close to those of the original model.

3 AdaQuantLM

We aim to quantize a LLM and finetune it only once and use it on various bitwidth settings. In this section, we present our method for adaptive bit-width LLM quantization, termed AdaQuantLM. Our objective is to enable seamless conversion between different bitwidths, facilitating joint optimization of LLMs under various bitwidth settings, particularly in resource-constrained environments. We start by introducing the problem to be solved.

Given a linear layer with weights $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ and calibration inputs $\mathbf{X} \in \mathbb{R}^{d_{in} \times n}$, our objective is to determine a set of quantized weights $\{\widehat{\mathbf{W}}_{B_i}\}$ that minimize the mean squared error between the outputs produced by the original and the quantized weights:

$$\arg\min_{\widehat{\mathbf{W}}_{B_1}, \ldots, \widehat{\mathbf{W}}_{B_N}} \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{W}\mathbf{X} - \widehat{\mathbf{W}}_{B_i}\mathbf{X}||_2^2. \tag{4}$$

where $\{B_1, \ldots, B_N\}$ are a set of bit-width candidates such that $B_i < B_{i+1}, 1 \leq i < N$. The set of quantized weights $\widehat{\mathbf{W}}_{B_i}$ is obtained using Additive Quantization. To make these weights adaptive—ensuring that lower bitwidth weights are derived from higher bitwidth weights rather than from the full-precision weights—we leverage the inherent additivity of codewords in Additive Quantization. For example, aiming to generate N quantized version of weights given bitwidth $\{B_i\}_{i=1}^N$ using codebooks $\{\{C_{i,j}\}_{j=1}^{M_i}\}_{i=1}^N$, we represent the quantized weights $\{\widehat{\mathbf{W}}_{B_i}\}_{i=1}^N$ as follows:

$$\widehat{\mathbf{W}}_{B_i} = \sum_{m=1}^{i} \sum_{j=1}^{M_i} C_{m,j} b_{i,m,1,j} \oplus \cdots \oplus \sum_{m=1}^{i} \sum_{j=1}^{M_i} C_{m,j} b_{i,m,d_{in}/g,j} \tag{5}$$

where g is the group size. Codebooks $\{C_{m,\cdot}\}_{m=1}^{i}$ are used to represent the quantized weights with bitwidth lower than B_i. Thus, the transformation from higher-bit quantized weights $\widehat{\mathbf{W}}_{B_i}$ to lower-bit quantized weights $\widehat{\mathbf{W}}_{B_{i-1}}$ simply requires dropping the representations obtained from the $\{C_{i,j}\}_{j=1}^{M_i}$ codebooks, expressed as $\sum_{j=1}^{M_i} C_{i,j} b_{i,i,1,j} \oplus \cdots \oplus \sum_{j=1}^{M_i} C_{i,j} b_{i,i,d_{in}/g,j}$. To elaborate further, the transformation between any two quantized weight versions is straightforward by using (5).

We consider two kinds of optimization of codebooks and codes:

- Parallel Optimization: We optimize codebooks and codes corresponding to different bitwidths in parallel.
- Sequential Optimization: We optimize codebooks and codes progressively, from lower to higher bitwidths.

Furthermore, we follow the algorithm introduced in AQLM, as described in Section 2.3, to alternatively optimize codebooks and codes. Also, there is finetuning for intra-layer cohesion after quantizing each layer.

4 Experiments

In this section, we evaluate AdaQuantLM through experiments.

Implementation & Configurations Our implementation is adapted from AQLM [6]. We quantize Gemma-2b [12] using one 6-bit codebook and one 8-bit codebook with group size 8. After quantizing each transformer layer by optimizing codes and codebooks alternatively, we finetune the remaining layers for 4 epoches to compensate for the quantization error. We use batch size 32 for calibration data of each layer. The calibration data is from RedPajama [2]. We use RTX6000 with 24GB GPU RAM for quantizing and finetuning.

Loss during Quantization and Finetuning We implement parallel optimization of codebooks and codes. Here we evaluate its effectiveness. After quantizing each transformer layer through alternating optimization of codes and codebooks, we finetune the remaining layers for 4 epochs to mitigate the quantization error. In Figure 3, we compare the finetuning loss of two quantization configurations, with two 8-bit codebooks, adapted from AQLM, and with one 6-bit codebook and one 8-bit codebook. In our method, despite using a lower-bitwidth codebook, the small gap between the two curves demonstrates a balanced trade-off between performance and model size.

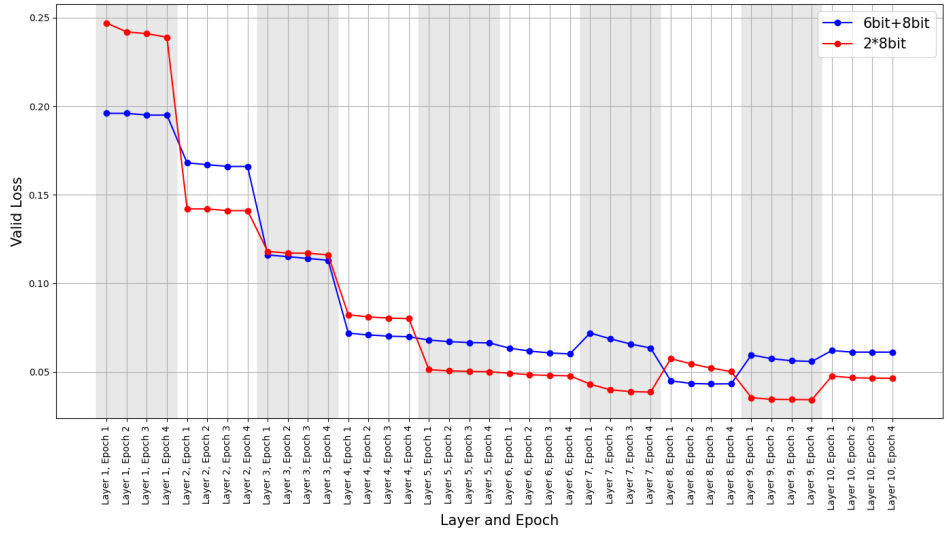

Figure 3: Comparison of the finetuning loss during quantization between two quantization setups: one using two 8-bit codebooks (adapted from AQLM) and the other using one 6-bit and one 8-bit codebook.

Evaluation We evaluate the quantized models with the Wikitext-2 dataset [11]. Table 1 presents a comparison of three quantized models: model-A, quantized with two learned 8-bit codebooks; model-B, quantized with a combination of learned 6-bit and 8-bit codebooks; and model-C, represented using only the learned 6-bit codebook. Our results show that while the performance gap between model-A and model-B is minimal, model-B has a smaller size. However, relying solely on a 6-bit codebook significantly degrades performance. In future work, we aim to optimize codebooks and codes progressively, from lower to higher bitwidths, rather than optimizing codebooks and codes of all bitwidths in parallel.

Model	model-A	model-B	model-C
Avg bits	2.002	1.753	0.752
Perplexity	10.635	13.087	25.842

Table 1: Comparison of bitwidth and performance of three quantized models.

5 Conclusions & Future Work

We propose AdaQuantLM, a method for LLM quantization with adaptive bit-width. AdaQuantLM addresses the limitations of existing LLM quantization methods, which require separate fine-tuning and benchmarking for each bitwidth configuration. By leveraging the additivity of codewords in quantized models, AdaQuantLM enables efficient conversion between different bitwidths without the need to store full-precision weights. This adaptive approach allows for joint quantization and fine-tuning across multiple bitwidths, making LLMs more versatile and suitable for deployment on devices with varying computational capabilities.

We consider two kinds of optimization of codebooks and codes, and only implement and evaluate the effectiveness of the parallel optimization on Gemma-2b model. In the future work, we will implement sequential optimization method and compare two optimization methods on more large models.

Acknowledgements

The research was financially supported by Hitachi Solutions, Ltd.

References

[1] Adrian Bulat and Georgios Tzimiropoulos. Bit-mixer: Mixed-precision networks with runtime bit-width selection, 2021.

[2] Together Computer. Redpajama: an open dataset for training large language models, 2023.

[3] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[5] 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, 2023.

[6] Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization, 2024.

[7] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023.

[8] Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths, 2020.

[9] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. Openassistant conversations – democratizing large language model alignment, 2023.

[10] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024.

[11] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[12] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024.

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[14] Jiseok Youn, Jaehun Song, Hyung-Sin Kim, and Saewoong Bahk. Bitwidth-adaptive quantization-aware neural network training: A meta-learning approach, 2022.

[15] Haichao Yu, Haoxiang Li, Honghui Shi, Thomas S. Huang, and Gang Hua. Any-precision deep neural networks, 2021.