Model Quantization 1: Basic Concepts

Florian June · Follow

6 min read · Oct 24, 2023

Quantization of deep learning models is a memory optimization technique that reduces memory space by sacrificing some accuracy.

In the era of large language models, quantization is an essential technique during the training, finetuning and inference stages. For example, qlora achieves significant memory reduction by carefully designing 4-bit quantization, reducing the average memory requirements for finetuning a 65 billion parameter model from over 780GB of GPU memory to less than 48GB, without degrading runtime or predictive performance compared to a fully finetuned baseline using 16-bit precision.

Therefore, understanding the principles of quantification is crucial for in-depth research on large language models. This is also the main purpose of this series of articles.

This article mainly introduces and distinguishes some basic concepts of quantification technology.

What is Quantization

In mathematics and digital signal processing, quantization refers to the process of mapping input values from a large set (usually a continuous set) to a smaller set (usually with a finite number of elements), which is similar to the discretization technique in the field of algorithms.

The main task of model quantization in deep learning is to convert high-precision floating-point numbers in neural networks into low-precision numbers.

The essence of model quantization is function mapping.

By representing floating-point data with fewer bits, model quantization can reduce the size of the model, thereby reducing memory consumption during inference. It can also increase inference speed on processors that are capable of performing faster low-precision calculations.

For example, Figure 1 represents quantizing a 32-bit precision floating-point vector [0.34, 3.75, 5.64, 1.12, 2.7, -0.9, -4.7, 0.68, 1.43] into int8 fixed-point numbers. Using a function mapping, one possible quantized result is: [64, 134, 217, 76, 119, 21, 3, 81, 99]:

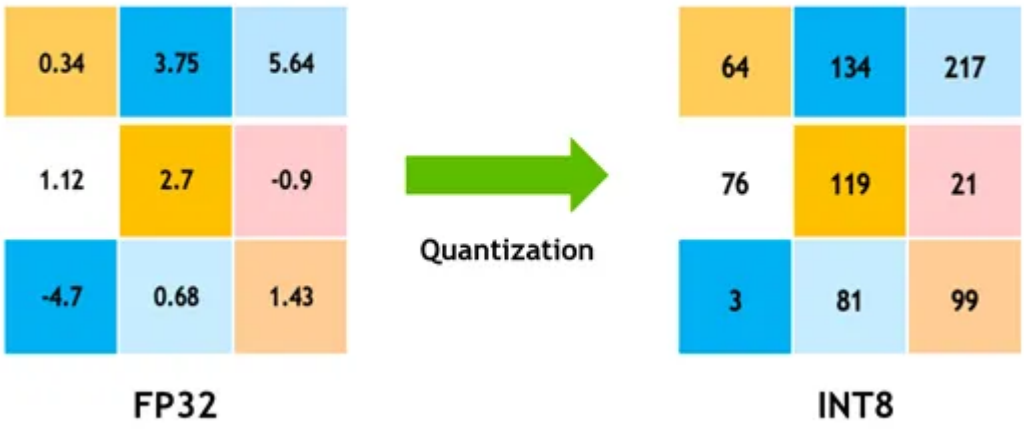


Figure 1

Floating Point Numbers and Fixed Point Numbers

Fixed Point Numbers

The position of the decimal point in fixed-point numbers is predetermined and fixed in computer storage. The integer and decimal parts of a decimal are converted separately into binary representation.

For example, decimal number 25.125

- Integer part: The decimal number 25 in binary is represented as: 11001
- Decimal part: The decimal number 0.125 in binary is represented as: .001
- Therefore, the decimal number 25.125 is represented as 11001.001 in binary.

In an 8-bit computer, the first 5 bits represent the integer part of a decimal number, and the last 3 bits represent the fractional part. The decimal point is assumed to be after the fifth bit .

Using 11001001 to represent the decimal number 25.125 seems perfect and easy to understand. However, the issue is that in an 8-bit computer, the maximum value that can be represented for the integer part is 31 (in decimal); and for the fractional part, the maximum value that can be represented is 0.875. **The range of data representation is too small.**

Of course, in a 16-bit computer, increasing the number of bits for the integer part can represent larger numbers, and increasing the number of bits for the fractional part can improve decimal precision. However, this approach is very costly for computers, so most computers do not choose to use fixed-point representation for decimals, but instead use floating-point representation.

Floating Point Numbers

Float numbers are numbers with a non-fixed decimal point, capable of representing a wide range of data, including integers and decimals.

Continuing from the previous example, we can use the IEEE-754 Floating Point Converter to convert 25.125 into a 32-bit floating point number:

0 10000011 10010010000000000000000, as shown in Figure 2:

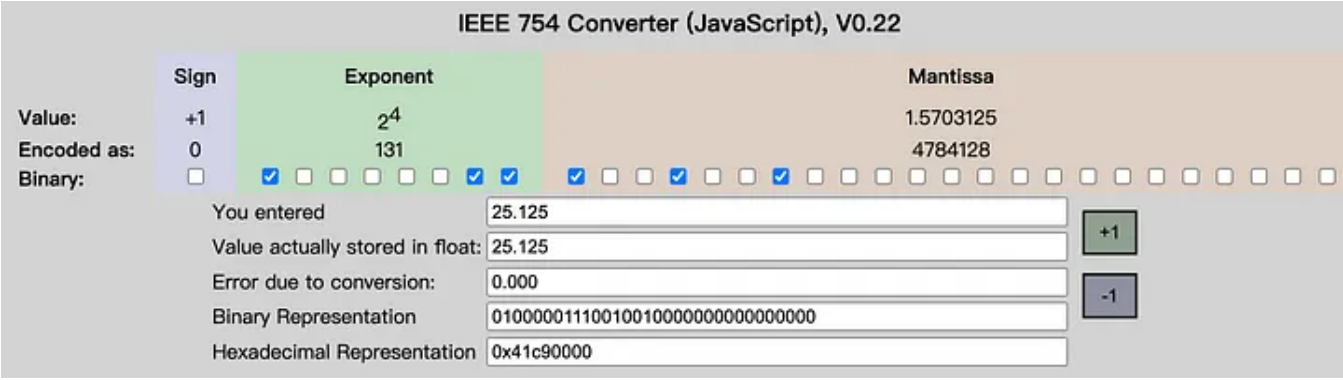
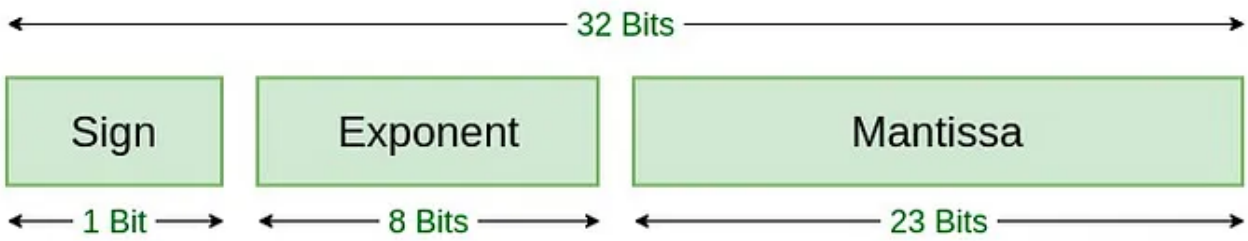


Figure 2

Let's take a look at the IEEE-754 Floating Point Standard, as shown in Figure 3:



Single Precision
IEEE 754 Floating-Point Standard

Figure 3

Specifically,IEEE 754 has 3 basic components:

1. The Sign of Mantissa: This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
2. The Biased exponent: The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
3. The Normalised Mantissa: The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

Now the problem is how to convert 25.125 to 0 10000011 10010010000000000000000? From the section on fixed-point numbers, we know that:

$$25.125_{10} = 11001.001_2$$

and

$$1001.001_2 = 1.1001001_2 \times 2^4$$

Therefore, we can append 0s after 1001001 to obtain a Normalised Mantissa of 10010010000000000000000.

Adding the exponent 4 to the bias 127 gives

$$131_{10} = 10000011_2$$

so the biased exponent part is 10000011.

Adding the sign of 0, the complete representation is 0 10000011 10010010000000000000000.

Comparison between Fixed-point Numbers and Floating-point Numbers

- When representing data on computers with the same number of digits, the range of data that floating-point numbers can represent is much larger than that of fixed-point numbers.
- When representing data on computers with the same number of digits, the relative precision of floating-point numbers is higher than that of fixed-point numbers.
- Floating-point numbers require calculations for both the exponent and the mantissa during computation, and the results need to be normalized. Therefore, floating-point operations involve more steps than fixed-point operations, resulting in slower computation speed.
- Currently, most computers use floating-point numbers to represent decimals.

Objects for Model Quantization

It mainly includes the following aspects:

- **Weights**: Quantizing weights is the most common and popular approach, it can reduce the model size, memory usage, and space occupation.
- **Activations**: In practice, activations often account for the majority of memory usage. Therefore, quantizing activations can not only greatly reduce memory usage but also, when combined with weight quantization, make full use of integer computation to achieve performance improvement.
- **KV cache**: Quantizing the KV cache is crucial for improving the throughput of long sequence generation.
- **Gradients**: Compared to the above, gradients are slightly less common because they are mainly used for training. When training deep learning models, gradients are usually floating-point numbers. They mainly serve to reduce communication overhead in distributed computing and can also reduce the cost during the backward pass.

Conclusion

This article mainly explains the concept of model quantization, fixed-point numbers and floating-point numbers, as well as the objects of model quantization.

The subsequent articles in this series mainly cover common quantization methods, stages of model quantization, granularity of model quantization, and the latest techniques in large language model quantization.

Furthermore, the latest AI-related content can be found in my newsletter.

Lastly, if there are any errors or omissions in this article, please kindly point them out.

References

https://en.wikipedia.org/wiki/Quantization

https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

https://www.h-schmidt.net/FloatConverter/IEEE754.html

https://en.wikipedia.org/wiki/Single-precision_floating-point_format

Llm | Model Quantization | Deep Learning | Machine Learning

Large Language Models

Follow

Written by Florian June

9.4K Followers · 111 Following

Most up-to-date and comprehensive: aiexpjourney.substack.com, youtube.com/@ai_exploration_journey

No responses yet

What are your thoughts?

Respond

More from Florian June

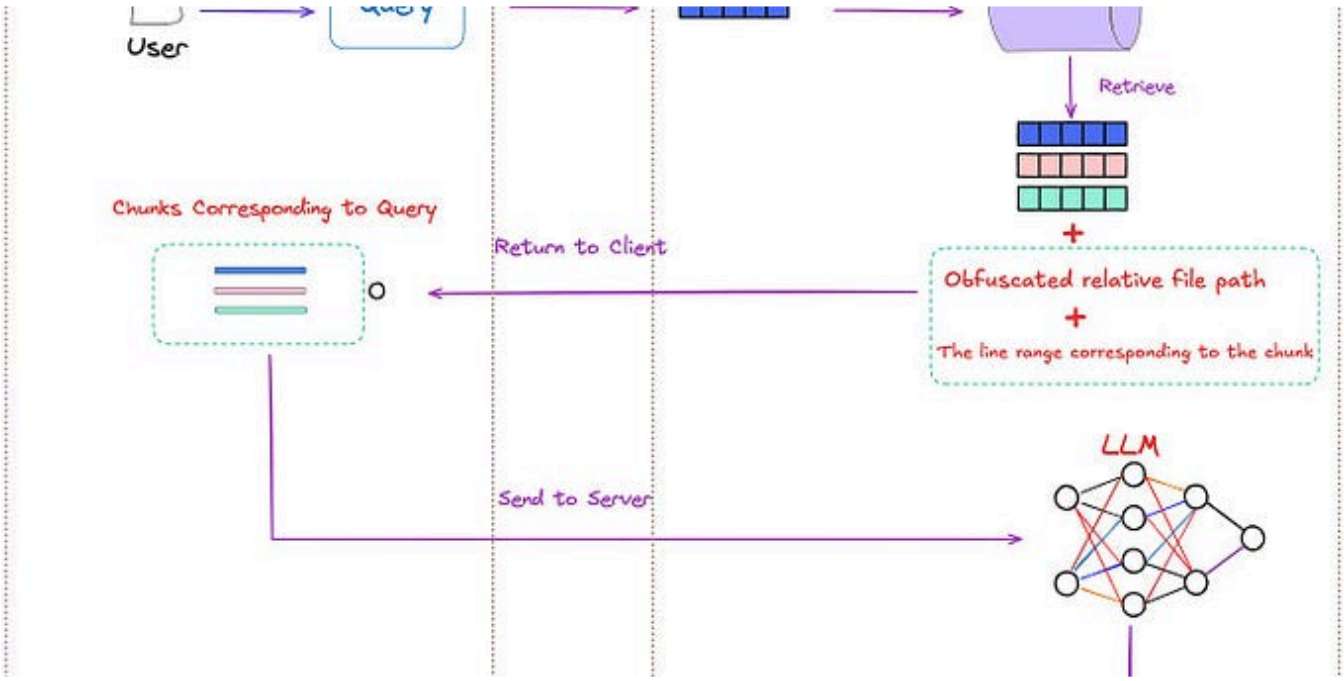
Summarize index	Uses an LLM to summarize entities and relationships in each community	None – the "lazy" approach defers all LLM use until query time
Refine query	None – the original query is used throughout	Uses an LLM to a) identify relevant subqueries and recombine them into a single expanded query, b) refine subqueries with matching concepts from the concept graph
Match query	None – all queries are answered using all community summaries (*breadth first*)	For each of *q* subqueries [3-5]: – Uses text chunk embeddings and chunk-community relationships to first rank text chunks by similarity to the query, then rank communities by the rank of their top-*k* text chunks (*best first*) – Uses an LLM-based sentence-level relevance assessor to rate the relevance of the top-*k* untested text chunks from communities in rank order (*breadth first*) – Recurses into relevant sub-communities after *z* successive communities yield zero relevant text chunks (*iterative deepening*) – Terminates when no relevant communities remain or *relevance test budget* / *q* is reached
Map answers	Uses an LLM to answer the original query over random batches of community summaries in parallel	For each of *q* subqueries [3-5]: – Builds a subgraph of concepts from the relevant text chunks – Uses the community assignments of concepts to group related chunks together – Uses an LLM to extract subquery-relevant claims from groups

Florian June

AI Innovations and Trends 10: LazyGraphRAG, Zerox, and Mindful-RAG

This article is the 10th in this promising series. Today, we will explore three exciting topics in AI:

Dec 13, 2024 161

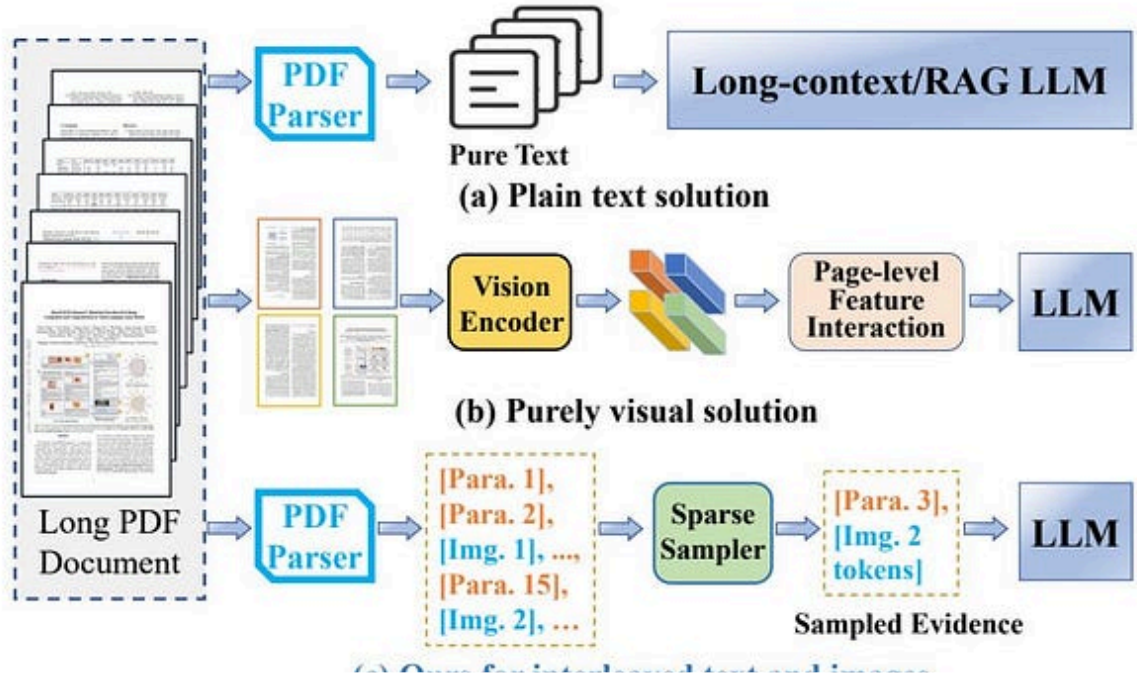


In Generative AI by Florian June

AI Innovations and Trends 09: Cursor Tool's RAG Features, TableRAG, and Llama OCR

You can watch the video:

Dec 10, 2024 214 2

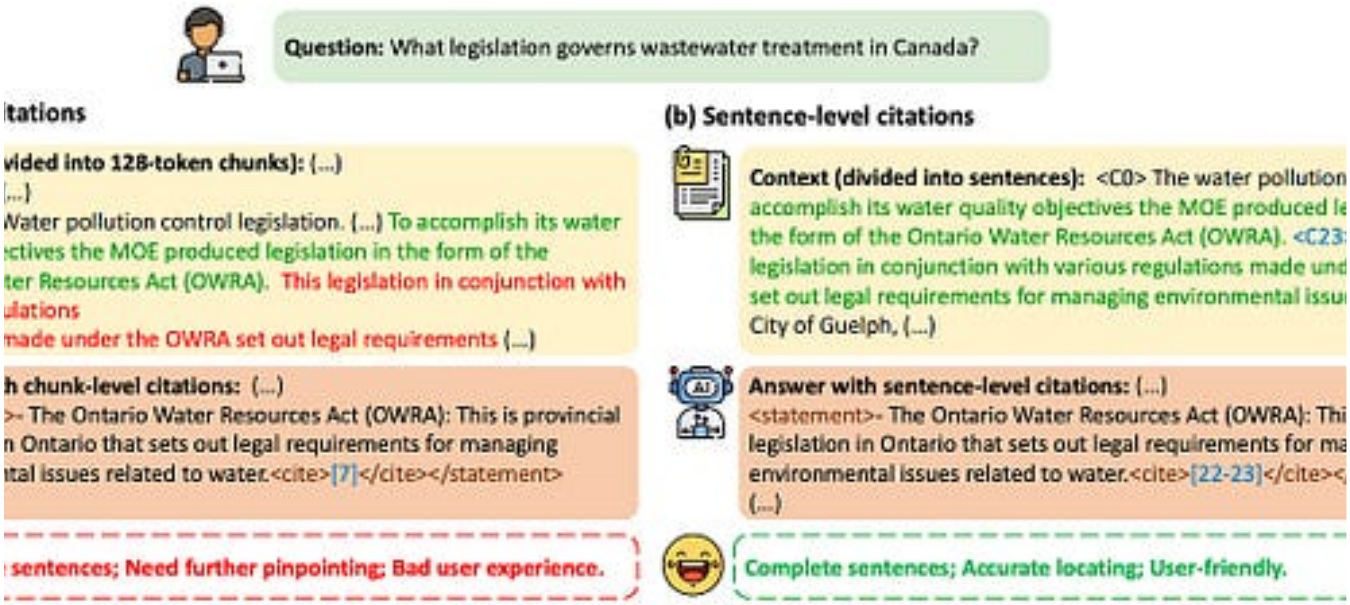


Florian June

AI Innovations and Insights 11: PDF-WuKong and PersonaRAG

This article is the 11th in this exciting series.

Dec 18, 2024 157 1



In Level Up Coding by Florian June

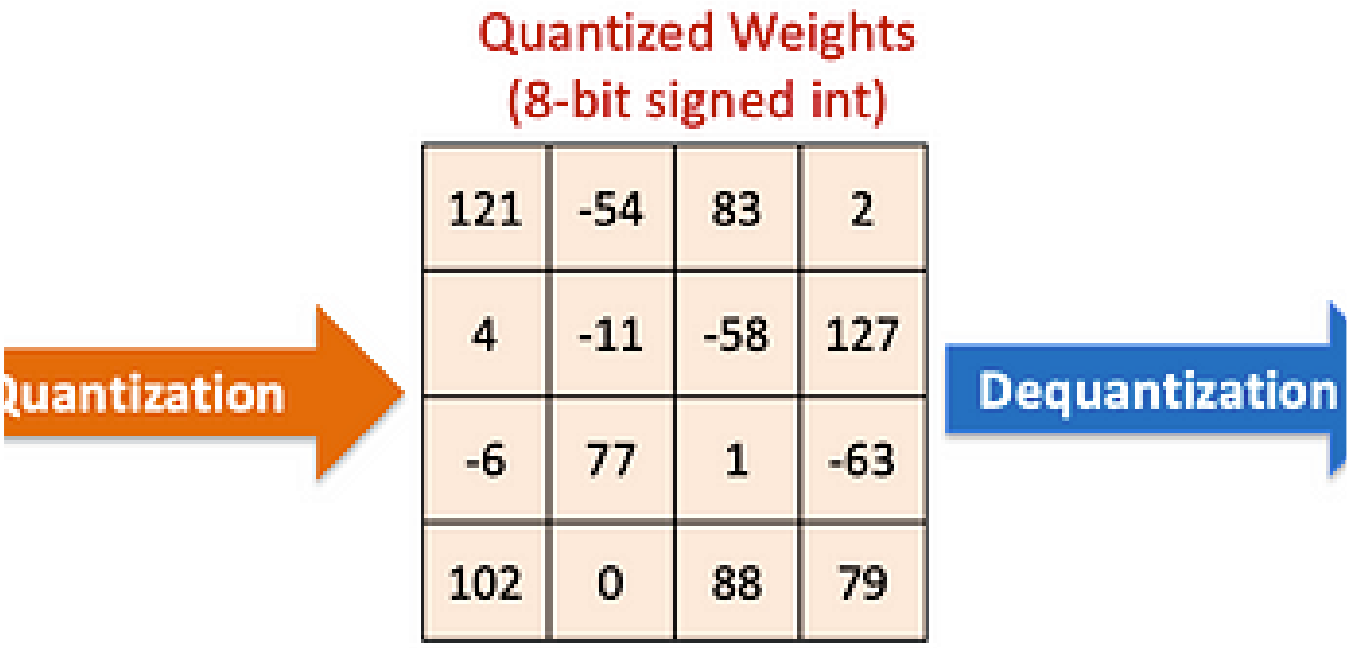
AI Innovations and Insights 12: LongCite and RAG Optimization

This article is the 12th in this promising series. Today, we will explore two exciting topics in AI, which are:

Dec 23, 2024 146

See all from Florian June

Recommended from Medium

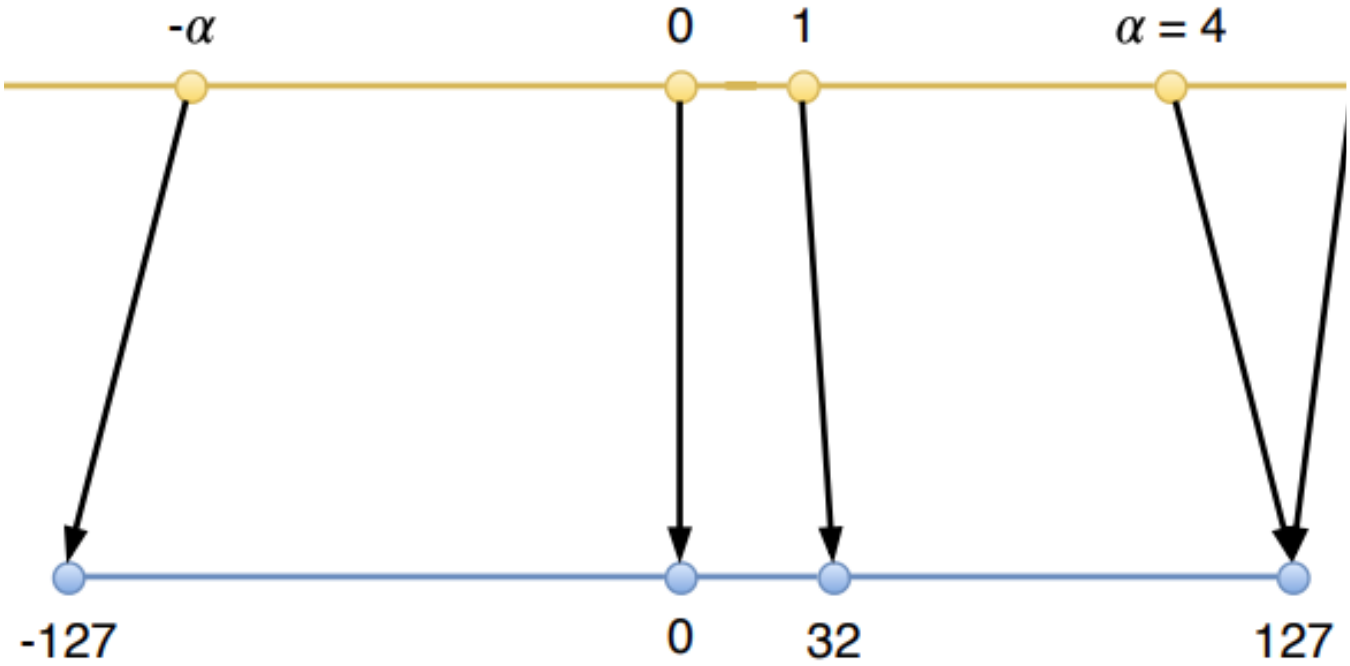


LM Po

Understanding Quantization for LLMs

As large language models (LLMs) continue to grow in size and complexity, the need for efficient deployment and inference becomes...

Jul 23, 2024 90



DeeperAndCheaper

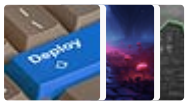
Quantization Basics

Introduction

Aug 26, 2024 20

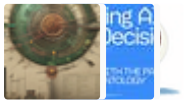
Lists

Natural Language Processing
1884 stories · 1533 saves

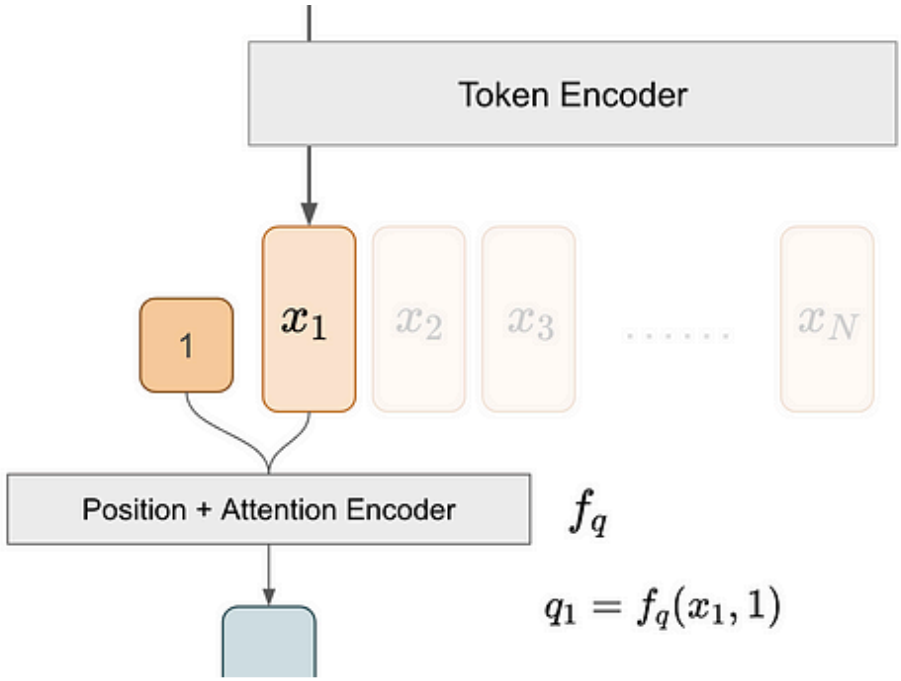


Predictive Modeling w/ Python
20 stories · 1768 saves

Practical Guides to Machine Learning
10 stories · 2139 saves



data science and AI
40 stories · 313 saves



Allen Liang

RoPE: A Detailed Guide to Rotary Position Embedding in Modern LLMs

Rotary Position Embedding (RoPE) has been widely applied in recent large language models (LLMs) to encode positional information, including...

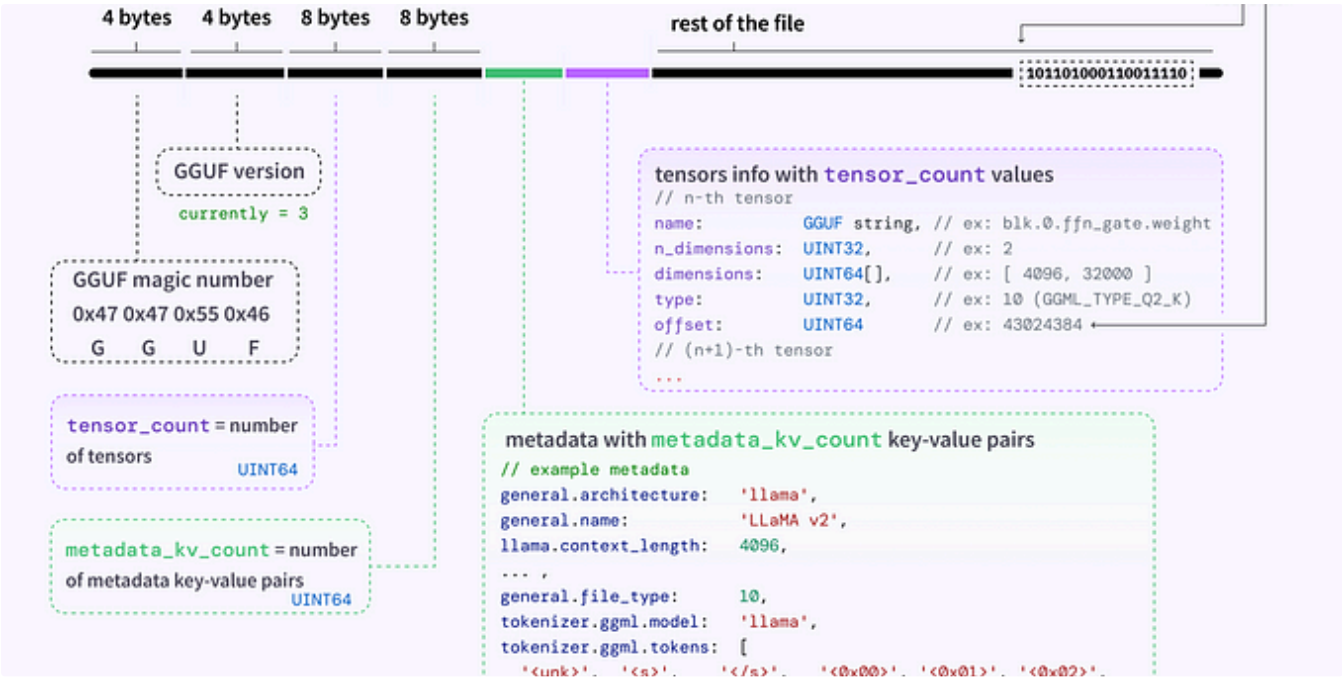
Dec 10, 2024 10

Benjamin Marie

Fine-tune Gemma 2 on Your Computer with LoRA and QLoRA

Using Hugging Face libraries and Unsloth

Jul 17, 2024 100 1



In GoPenAI by kirouane Ayoub

Exploring Bits-and-Bytes, AWQ, GPTQ, EXL2, and GGUF Quantization Techniques with Practical Examples

1. Bits-and-Bytes Quantization

Aug 22, 2024 58

In Towards AI by Serj Smorodinsky

Building Confidence in LLM Evaluation: My Experience Testing DeepEval on an Open Dataset

deepeval helped me uncover what is the real source of Beyonce's depression

Oct 26, 2024 221