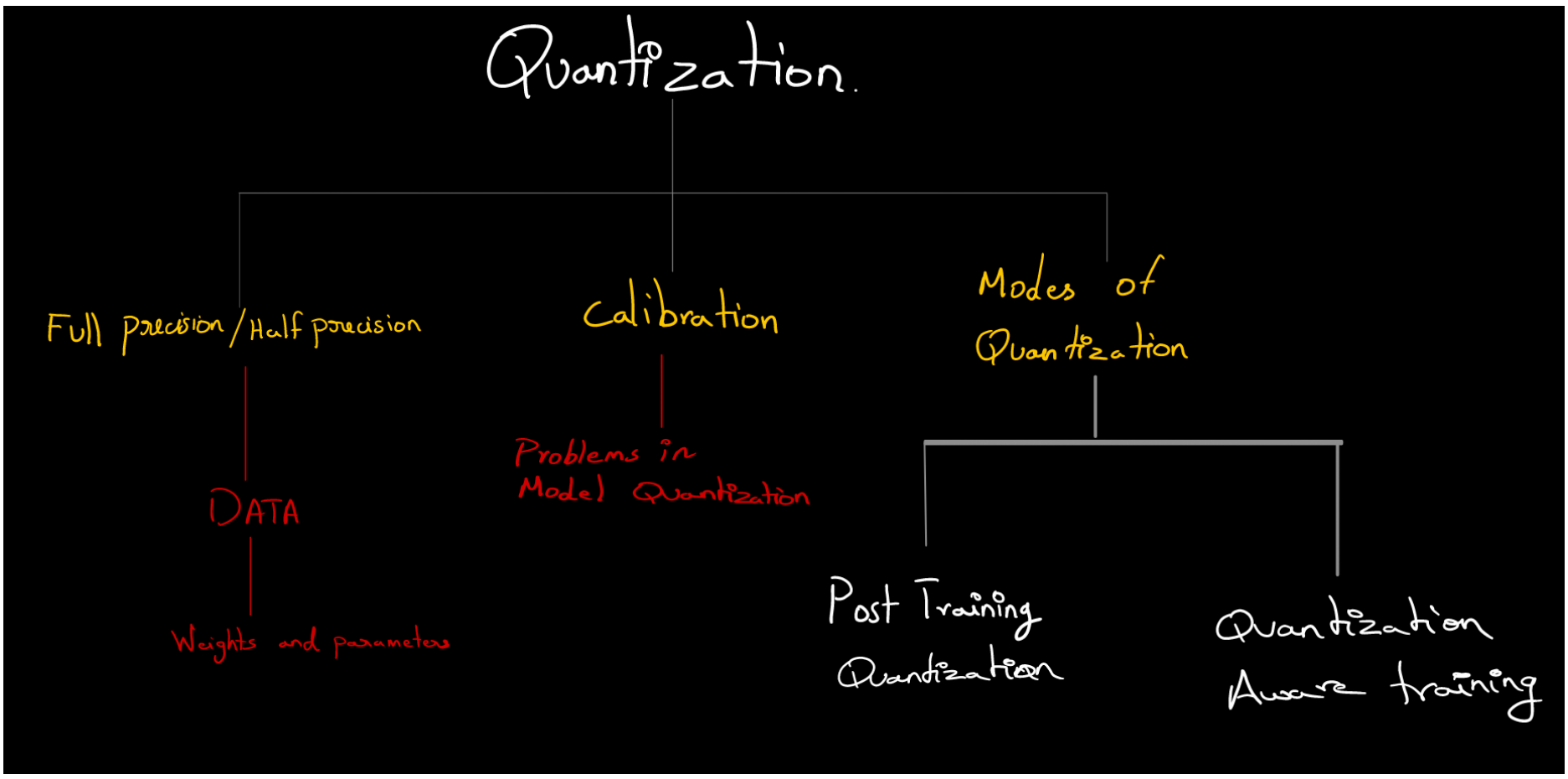
Fine Tuning Quantization InDepth Intuition.

Introduction

This document delves into the essential concept of quantization, a technique used to optimize the memory footprint and inference speed of large language models (LLMs) like ChatGPT. We'll explore the nuances[’it means the detailed and often delicate aspects of understandin’] of Full Precision, Half Precision, and how Quantization enables efficient deployment on diverse hardware. Additionally, we'll discuss the crucial role of Calibration and the two primary quantization modes: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT).

Quantization.



Detailed Examination :

QUANTIZATION : conversion from Higher memory formate to a lower memory formate.

Full Precision / Half Precision : is somthing related to data types , How data is been stored in the memory,

In LLM models its about weights and parameters, Bcz llm are deep learning neural networks in the form of transformers or bert.

Calibration : in Model Quantization their will be some problem how we can do calibration

Their are 2 types of **modes to quantize.**

---

1. Full Precision / Half Precision.

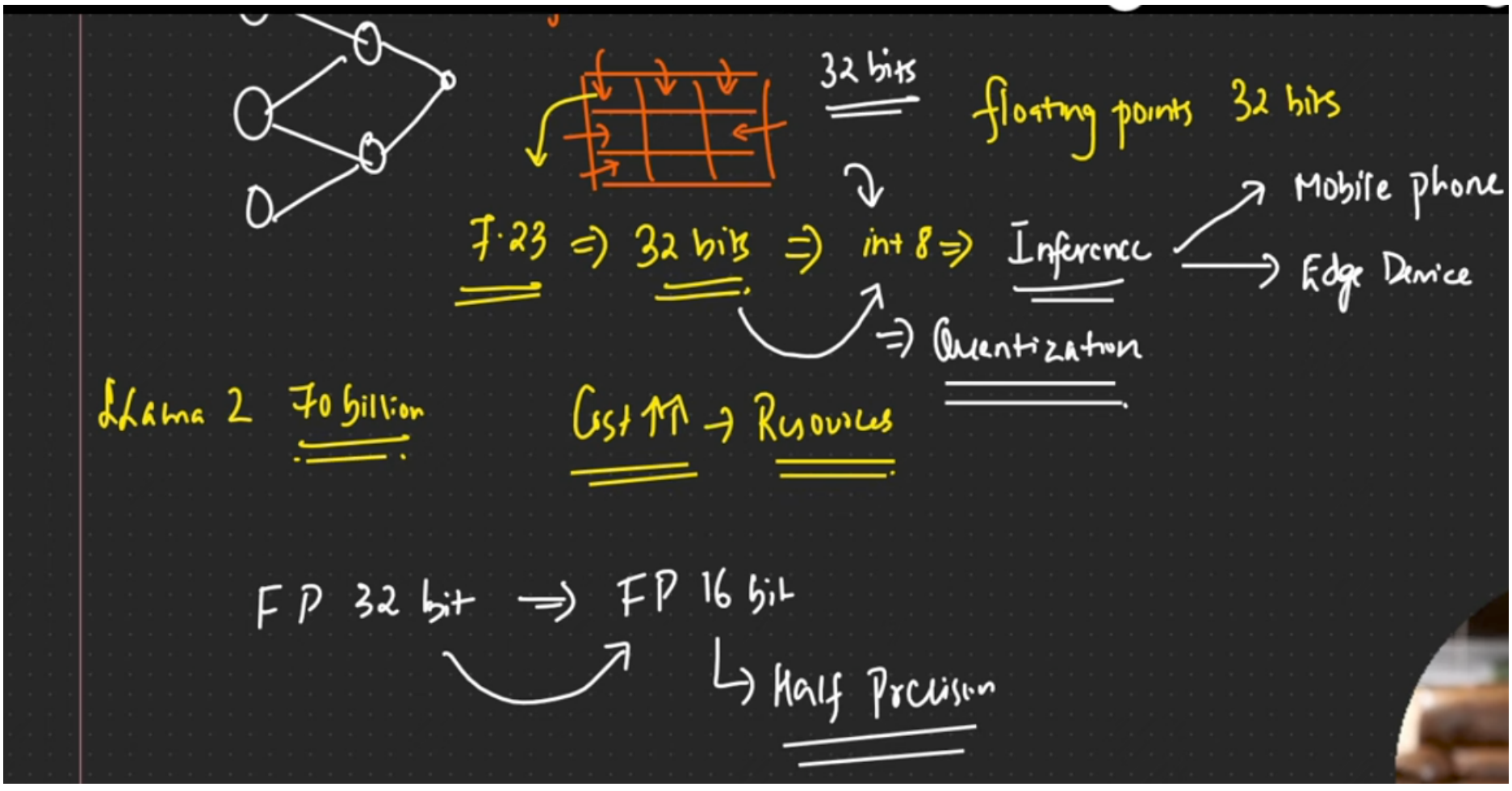
1.1 - Understanding Precision Levels

In the realm of deep learning, weights and parameters within neural networks are often represented using floating-point numbers. These numbers can be stored with varying levels of precision, which significantly impacts memory usage and computational efficiency.

Full Precision (FP32): This format utilizes 32 bits to store each value, providing the highest level of accuracy but also consuming the most memory.

Half Precision (FP16): This format uses 16 bits per value, offering a balance between accuracy and memory efficiency. Many GPUs today support FP16 operations, making it a popular choice for training and inference.

Detailed Examination :



Any neural network or any data, when train neural network what parameter probabilly involved over here is **Weights,** Now Weights are usually in the form of **Matrix.**

in that matrix eg, we have 3×3 matrix, every value is probabilly stored in the memory in the form of **32bits,** we can aslo say it as floating point 32 but **FP** as diff meaning

we also denote it as **FP32.** we can also consider it as **Full Precision / Single Precision.** (in Short it is like Folating Piont number)

1.2 - The Memory Challenge of Large LLMs

Large language models (LLMs) like LLaMa 2, with their billions of parameters, pose a considerable memory challenge. These parameters, encompassing weights and biases, must be stored in memory, leading to massive storage requirements.

Standard Hardware Limitations: Standard GPUs and systems with limited RAM struggle to accommodate these memory demands, rendering direct fine-tuning or inference on such systems impractical.

Cloud Solutions (but at a Cost): While cloud resources can address this limitation, they come with a significant cost associated with the computing power and storage needed.

Detailed Examination :

eg. i have **number 7.23** is stored based on **32bits** in memory.

Understand when you have very big neural network or LLM models parameters keeps on increasing.

consider LLama 2 model with 70 billion parameters. it as 70b parameters in term of weights and bias, now it’s not possible to around do a fine tunning w.r.t the normal GPU or very limited RAM in system

cannot directly download the specific model and put it in my ram or load it in Vram,

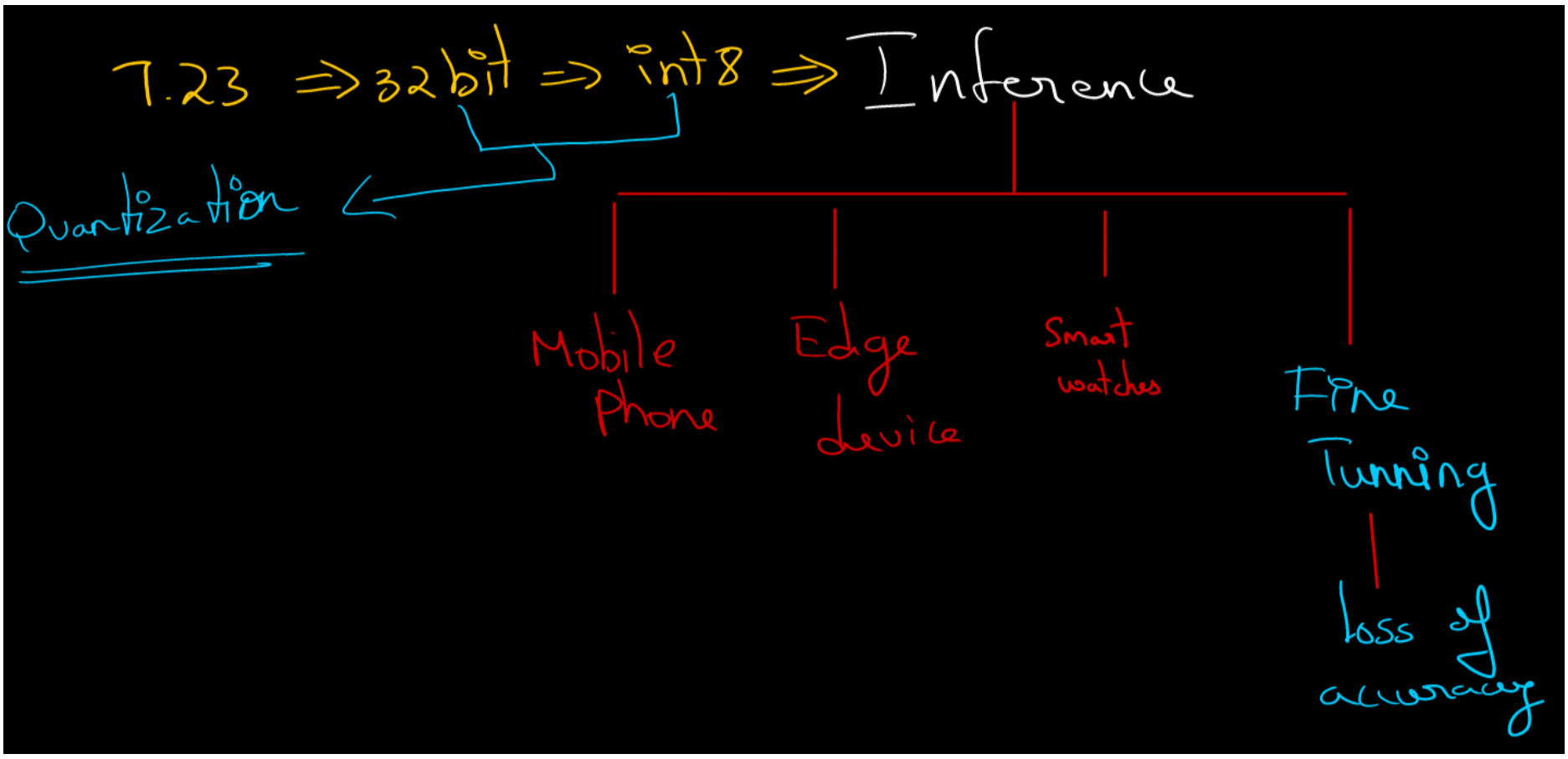
we can use cloud space but it will have lot of cost involved based on the resources, what we can do.. ?

1.3 - Quantization: Reducing Memory Footprint

Quantization provides a solution for overcoming the memory constraints associated with LLMs. It involves converting the high-precision floating-point numbers (FP32) used for weights and parameters to a lower-precision format, typically 8-bit integers (INT8).

Benefits: This conversion drastically reduces the memory footprint of the model, making it feasible to store and deploy it on less powerful devices like mobile phones, embedded systems, and even personal computers.

Trade-Off: Potential Accuracy Loss: However, this conversion can lead to some loss of accuracy due to the reduced precision. It's crucial to weigh the trade-off between memory savings and potential accuracy loss when choosing a quantization method.



we can covert this **32bit into int8 and then download and use the model**,with in my system i will be able to Inference it. for fine tunning a new data set i would need Gpu. If i consider w.r.t **Inferencing** it becomes quite easy all my value that are stored in **32bits it will be stored in the form of 8bits** what we are specifically doing is we are **converting from high memory formate to low memory format** and this is what called as **Quantization.**

a very important thing with quantization we will be able to inference it quickly.

▼ **1.4 - Inference**

1.4.1 - Inference in LLMs (Large Language Models):

Definition: Inference refers to running a trained language model on new data (usually in the form of a prompt,Question, input) and generating an output (such as a response, prediction, translation).

Explanation: Imagine you have a well-trained language model like ChatGPT. When you give it a sentence or a question, it processes that input and produces a meaningful answer. This process of generating a response based on the input is what we call inference.

This is how we leverage the knowledge encoded in LLMs for real-world tasks.

1.4.2 - Why Is Inference Important?

Inference allows LLMs to be useful in real-world applications. Once a model is trained, its true value lies in its ability to provide relevant and accurate outputs when faced with new data.

Inference allows us to apply the specialized knowledge acquired during fine-tuning to real-world tasks.

It ensures that our LLM performs well on specific problems, such as question answering, translation, or sentiment analysis.

Example : Suppose you input the prompt: “Translate the English sentence ‘Hello, how are you?’ into French.”The LLM would perform inference by processing this prompt and generating the translated output: “Bonjour, comment ça va ?”

1.4.3 - Why Quantization Helps with Inference?

Quantization significantly accelerates inference by reducing the amount of data that needs to be processed during calculations. This is particularly beneficial on devices with limited computational resources.

1.4.4 - Inference During Fine-Tuning:

Fine-Tuning

Fine-tuning is a crucial step in training large language models (LLMs) like ChatGPT. Here's how it works:

- First, a pre-trained LLM (such as GPT-3 or BERT) is created using a massive amount of text data.
- Then, this pre-trained model is fine-tuned on a smaller, domain-specific dataset to adapt it to a specific task or application.
- Fine-tuning involves training the model further on task-specific data, adjusting its weights and parameters to specialize it for a particular purpose.

When we fine-tune a language model, we use it for inference on the task-specific data.

During inference, the fine-tuned model processes input examples (such as sentences or prompts) and produces relevant outputs (such as predictions or answers).

For example, if we fine-tune a model for sentiment analysis, inference involves inputting a sentence and predicting whether it has a positive or negative sentiment.

1.4.5 - Fine-Tuning vs. Pre-training Inference:

Pre-training (the initial training on a large corpus) and fine-tuning (the specialized training) both involve inference.

However, **during pre-training, the model learns general language patterns**, **while during fine-tuning, it adapts to specific tasks or domains.**

1.4.6 - When to Use Inference?

- Use inference whenever you need your fine-tuned LLM to process new data and provide meaningful outputs.
- For example, if you've fine-tuned an LLM for sentiment analysis, inference helps classify new sentences as positive or negative.

1.4.7 - Uses of Inference in LoRA and QLoRA

- **Question Answering**: Given a question, the fine-tuned LLM can infer an accurate answer.
- **Text Generation**: Inference generates coherent text, summaries, or translations.
- **Classification**: It predicts labels (e.g., sentiment, topic) for input data.

In summary : inference in LoRA and QLoRA fine-tuning ensures that our LLM performs effectively on specific tasks by generating meaningful outputs based on new input data.

Detailed Examination **:**

Let's say i have a LLM model it i give any i/p to that i will be able to get the o/p (Response), now when i give any i/p all the calculation w.r.t different weight will happen, when i have bigger GPU this Inferencing will happen quickly. If i have GPU with less core's the calculation will take time

If i convert my 32bit to 8bit now every weights are basically converted to 8Bits

Now the calculation their will be difference it will happen much more quicker

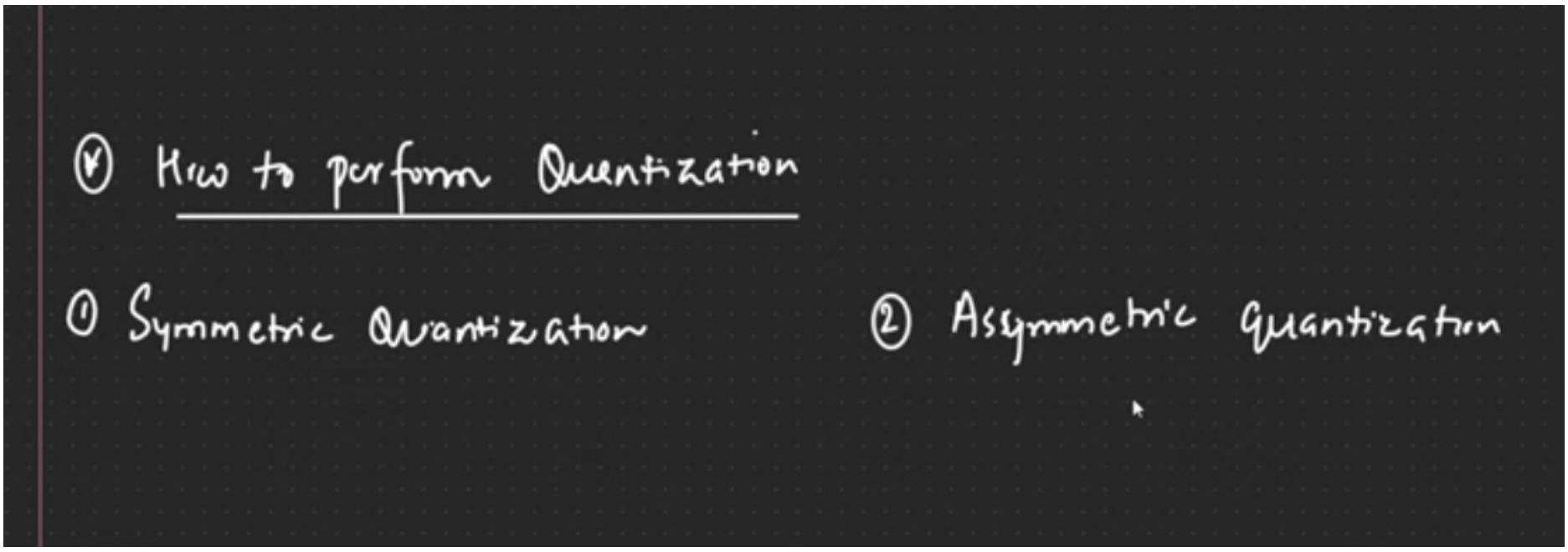
So Quantization is very much important for inferencing

This Inferencing we can use it in phone,watches etc. now with the help of quantization we can perform finetunnig their is a disadvantage

when we perform this quantization since we convert 32bits to int 8 their is some loss of info also and because of this we have loss of accuracy, their are diff techniques that we can overcome on it

---

Quantization Techniques: Symmetric vs. Asymmetric



Detailed Examination :

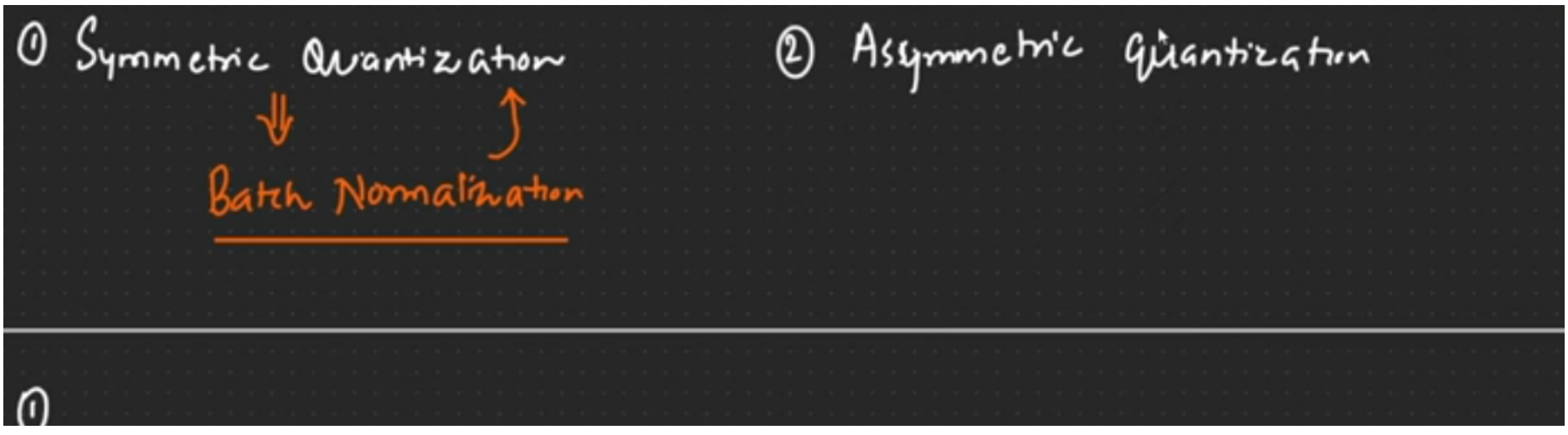
Lets go a head and try to understand how to perform quantization, Two primary techniques are used for quantization

1. Symmetric Quantization
2. Asymmetric Quantization

see, with the help of tensor flow and 4 lines of code we can do this

but its necessary to know that we can do it manually

1. Symmetric Quantization



Concept : In symmetric quantization, the range of values to be quantized is centered around zero, with equal positive and negative ranges.

Example : If we want to convert values in the range of -1000 to 1000 to 8-bit integers (0 to 255), the symmetric approach would be to map -1000 to 0, 0 to 127, and 1000 to 255.

Benefits : This method is often simpler to implement and can provide good results in situations where the data distribution is relatively centered.

Detailed Examination :

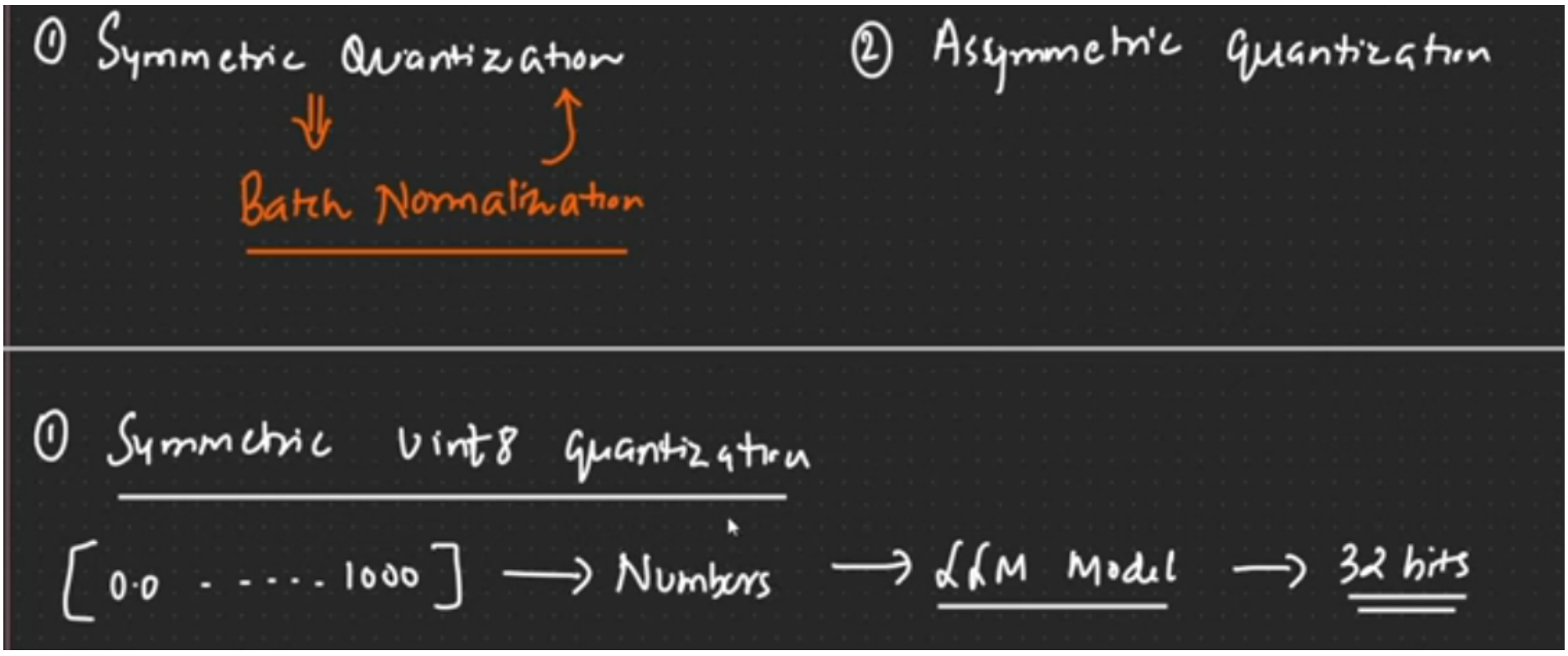
lets say their is a task in this 1st task is r.w Symmetric quantization.

In **Deep Learning** their is **Batch Normalization,** this Batch_normalizer is a technique of symmetric quantization

every time we will be able to see that during **forward and backward propagation** between all the layers we apply batch normalization, so that all our weight are **Zero Centered [Entier distribution of weight is centered near Zero]**

Symmetric Unsigned int8[Uint8] Quantization

Unsigned int8[Uint8]: Uint8 means it will not take any negative number it will be between 0 to 255.



Detailed Examination :

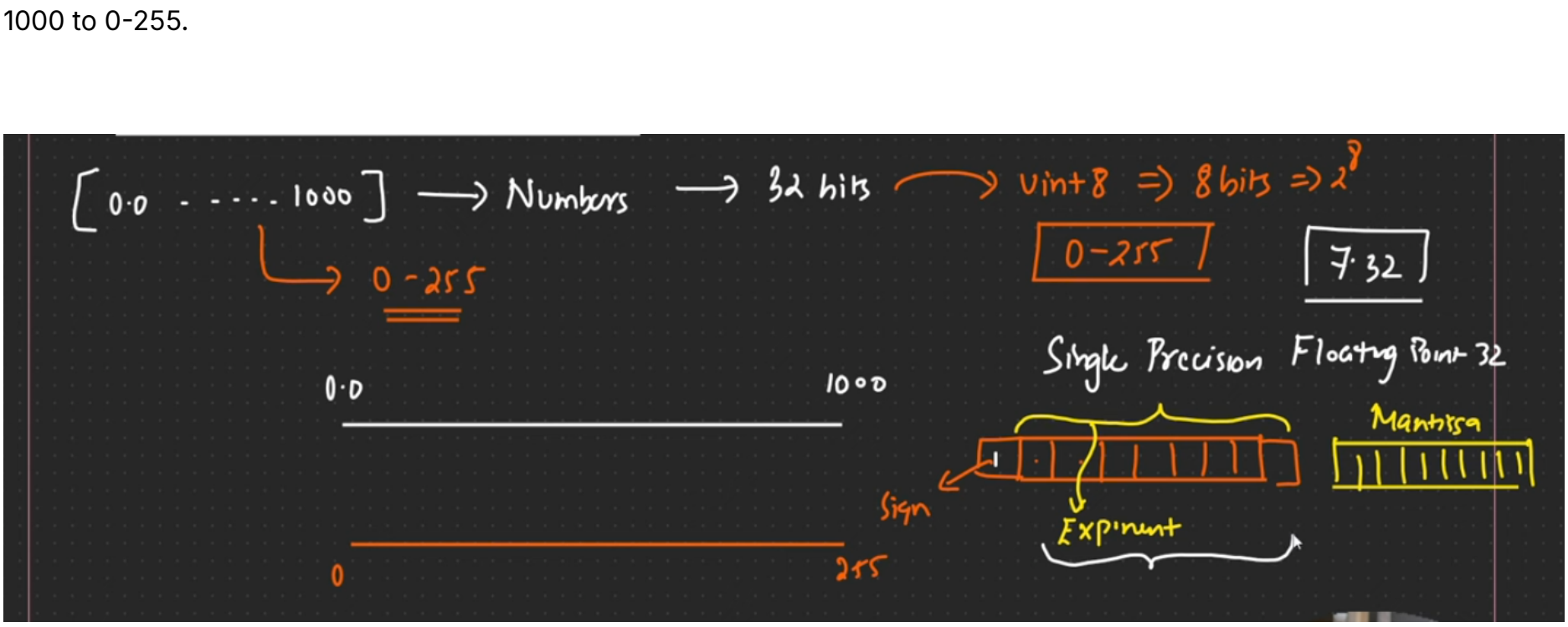
Lets say I have a floating point number between zero to 1000 → weights (Numbers)

and this are the weight for my larger model any LLM Model you have lot of parameters

and all the Numbers [weight] are stored in 32bits and the weights will not be in that range it will be in a minimalistic weights

This numbers are stored in form of 32 bits now main aim is to convert this into Unsigned int8 - [uint8] that basically means 8 bits which 2^8= 256

But when we say U - Unsigned that basically means my value will be ranging from 0-255. So i want to quantize from the 0-1000 to 0-255.



important thing. If we have this Single precision floating point 32. you have a number how that stored.

the first one bit is specifically used for sign or unsign value('+', '-')

then next 8 bit are stored for exponent and remaining 23 bits will basically will save for mantissa(this is specifically for fraction)

▼ **Mantissa** (also known as the **fractional part** or **significand**):

- In scientific notation, especially when representing floating-point numbers, the mantissa refers to the part after the decimal point.
- For example, in the number

  $$12345.6789.$$

  The mantissa is the part after the decimal point (in this case, 0.6789). It holds the significant digits that give the number its value.

  In binary representation, the mantissa is the fraction part of a floating-point number, typically stored in a fixed number of bits (such as 23 bits for single-precision floating-point numbers).

Exponent:

- **Definition**: The exponent represents the power to which a base (usually 2 or 10) is raised.
- **Simple Words**: Think of it as a "zoom factor." If you have a small number like

  $$0.00123$$

  , the exponent (say,

  $$-3$$

) tells you how many times to multiply it by

  $$10$$

  . So,

  $$0.00123 \times 10^{-3}$$

  Putting it all together, when you write a number like

  $$3.14 \times 10^{2}$$

The **mantissa** is (the significant part).

$$3.14$$

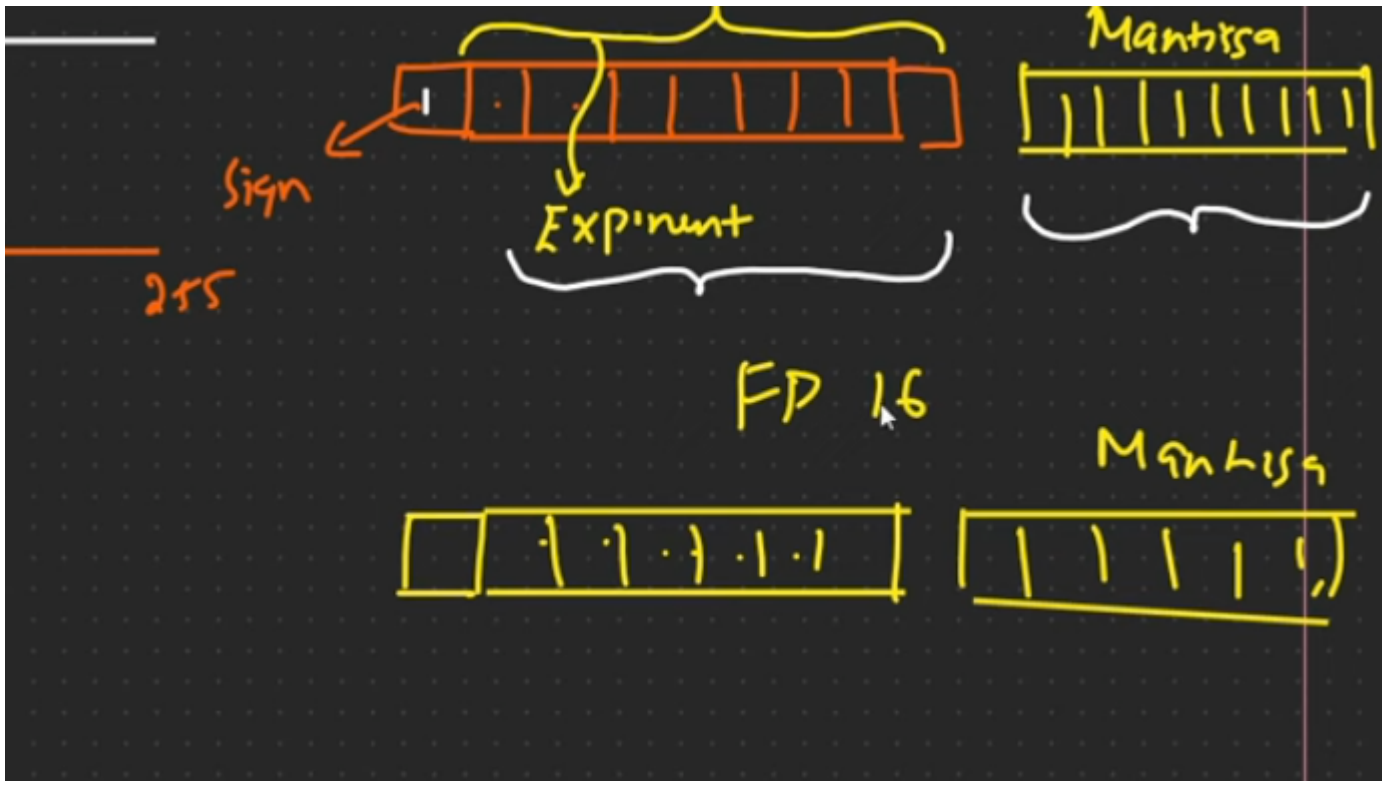
The **exponent** is (the zoom factor).

$$2$$

💡 mantissa holds the significant digits of a number, while the exponent determines its scale. Together, they form the scientific notation or floating-point representation. 🧱✨

e.g. if I have a number 7.32 it in a positive number, so the sign bit that will be a +ve value and then the 7 will probabilly stored in the 8 bit and remaining .32 will be put up in mantison **[we basically say this as fraction any number that comes after the decimal]** .

This is how the numbers are basically stored in memory.



lets see with FP16 → Half Precision floatingpoit 16bit

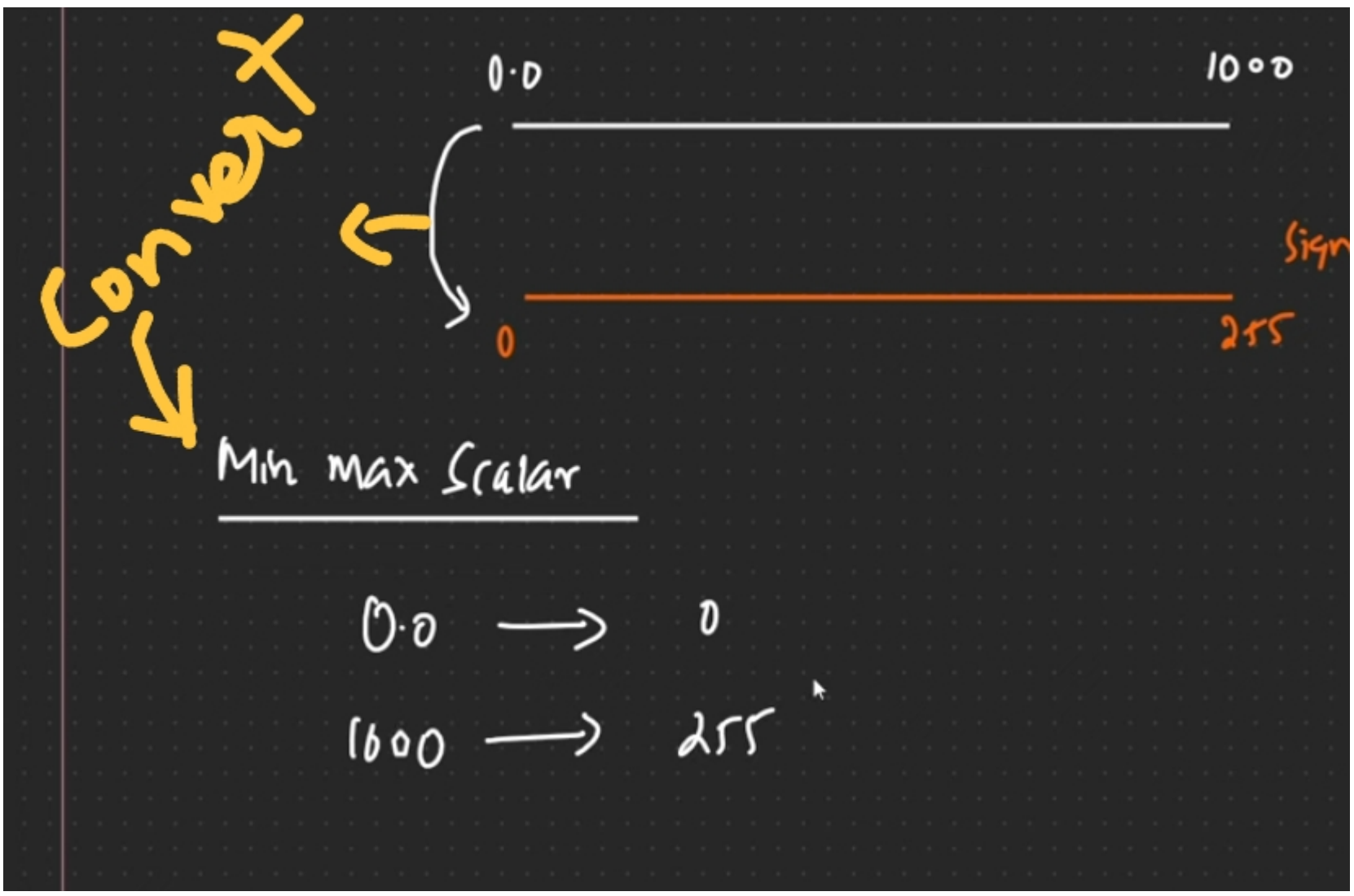
their will be 1 bit for Sign number, then 5 bit for exponent and remaining 10 bit will be saved w.r.t mantison.

know that FP16 takes less memory and FP32 has high memory

What is our main AIM here ?

- We have 32bit number
- It need to convert to range of Unsigned int8 [U int8 - means it will not take any negative numbers, it will be between 0 - 255 (256)]
- This is what we really want to do

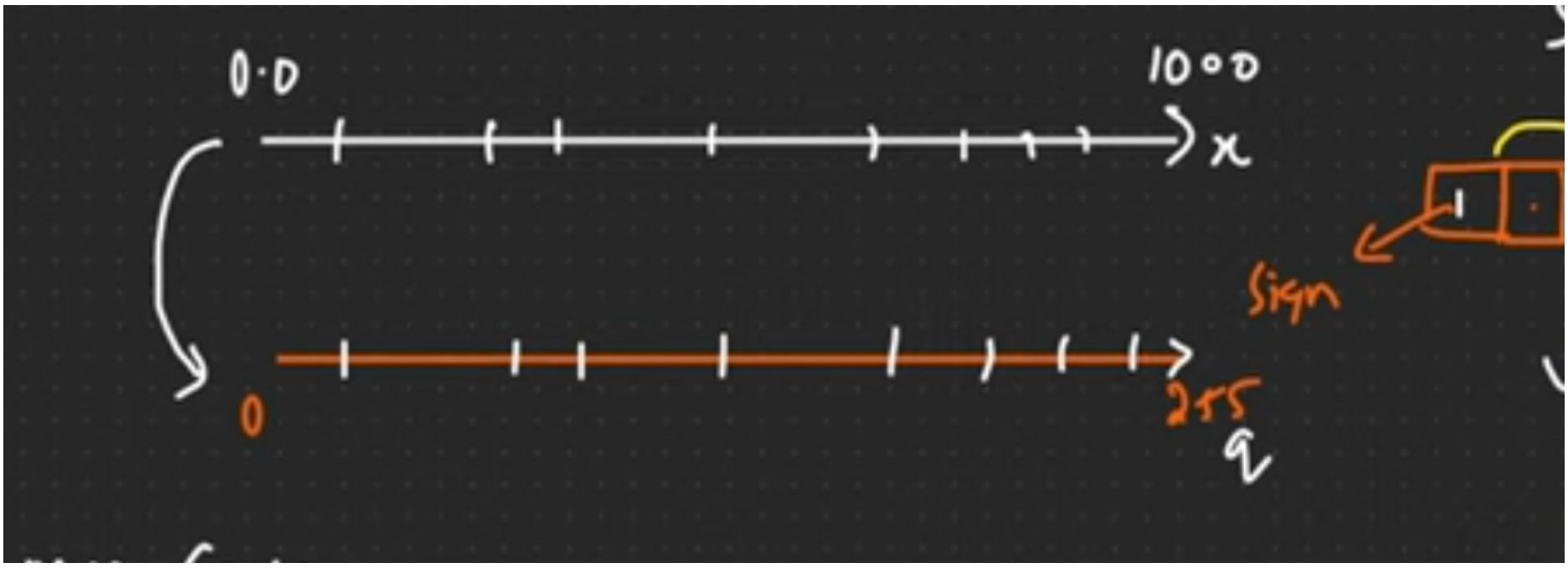
Now for this what will be the equation.



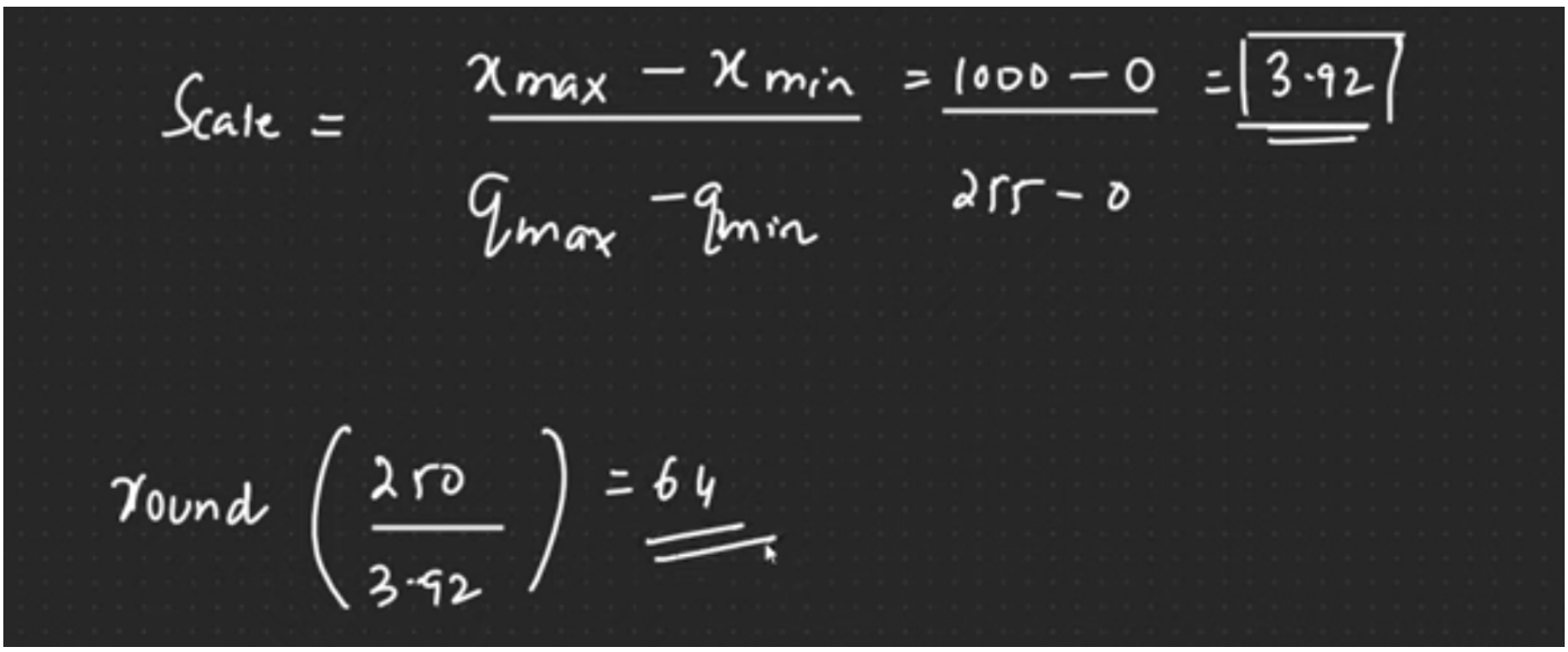
MIN Max scalar from any number from 1000fp to 255fp

The bits are decreasing from 1000-255 **Quantization basically happening**

we probabilly have to come up with the scale factor



here 1000 is my Xmax and 255 is my Qmax, showing **Quantization happens in symmetric distribution [Symmetrics basically means all the data are evenly distributed]** .



Here scale formula will be that scale

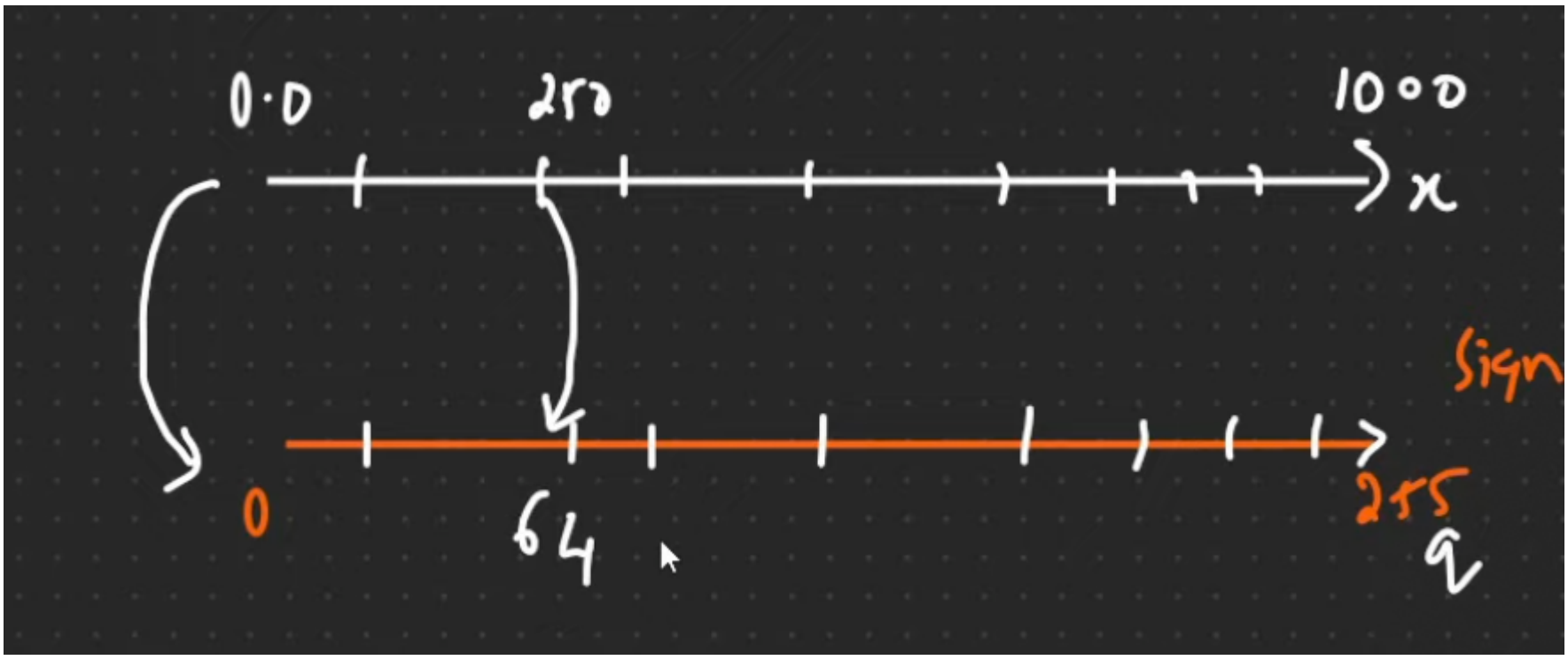
if i go with this specific division the value that i will be having is 3.92 which is

Scale Factor.

Any number that i have over xmax if i want to convert it from FP32bit to Uint8 i just need to use this scalling along with formula round

When i round with any member between xmax lets consider 250

250/3.92 = 63.77 if i do rounding it will be 64



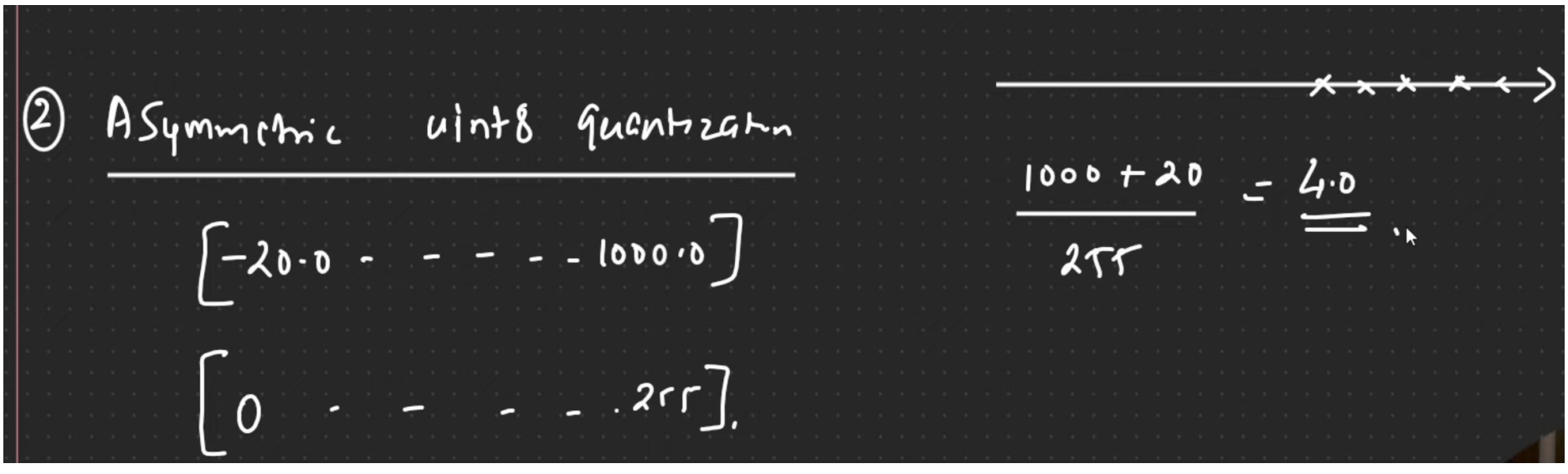
in short any number over Xmax let say 250 this will get convert to quantize value of 64. The same thing the code will be also doing and this for symmetric Uint 8 quantization

2. Asymmetric Quantization

Concept : Asymmetric quantization allows for a more flexible mapping of values, especially when the data distribution is skewed (not centered around zero).

Example : If we have values in the range of -20 to 1000, we might map -20 to 0, 0 to 127, and 1000 to 255. This means the positive range is expanded to cover more values than the negative range.

Benefits : This method can be more effective when dealing with data that has a clear bias towards positive or negative values.

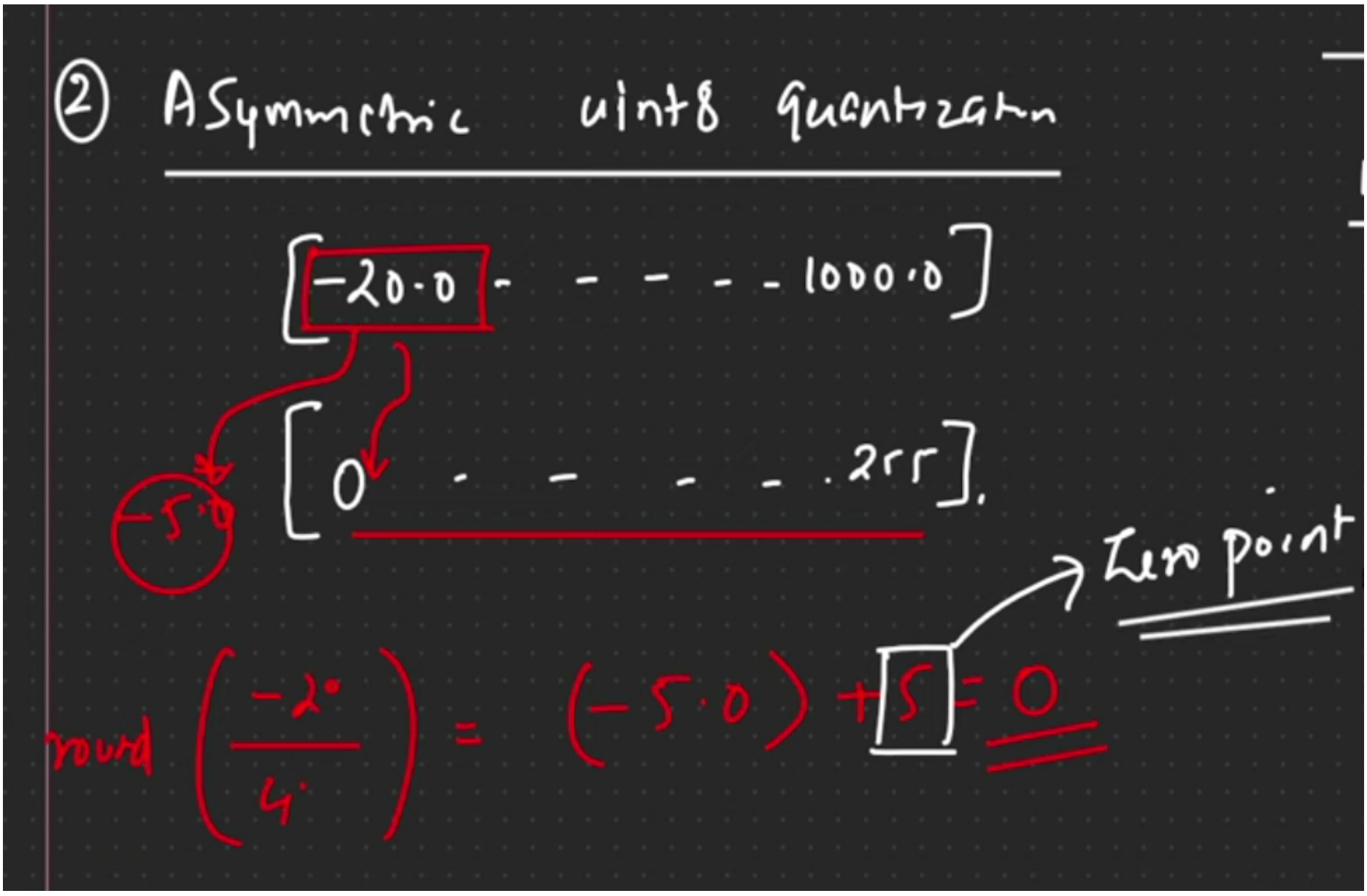


Detailed Examination :

If i want to perform this quantization lets say value lies between -20.0 to 1000 these are my floating point. now i have to perform quantization and convert this in to 0 to 255

in case of asymmetric, what happen in real number section the numbers are not really distributed it may be right or left sqewed

in this scenario my value ranges between -20 to 1000 and i need to convert this from this 0 to 225. here if same formula is been applied xmax - x min i will get scale factor 4.0.

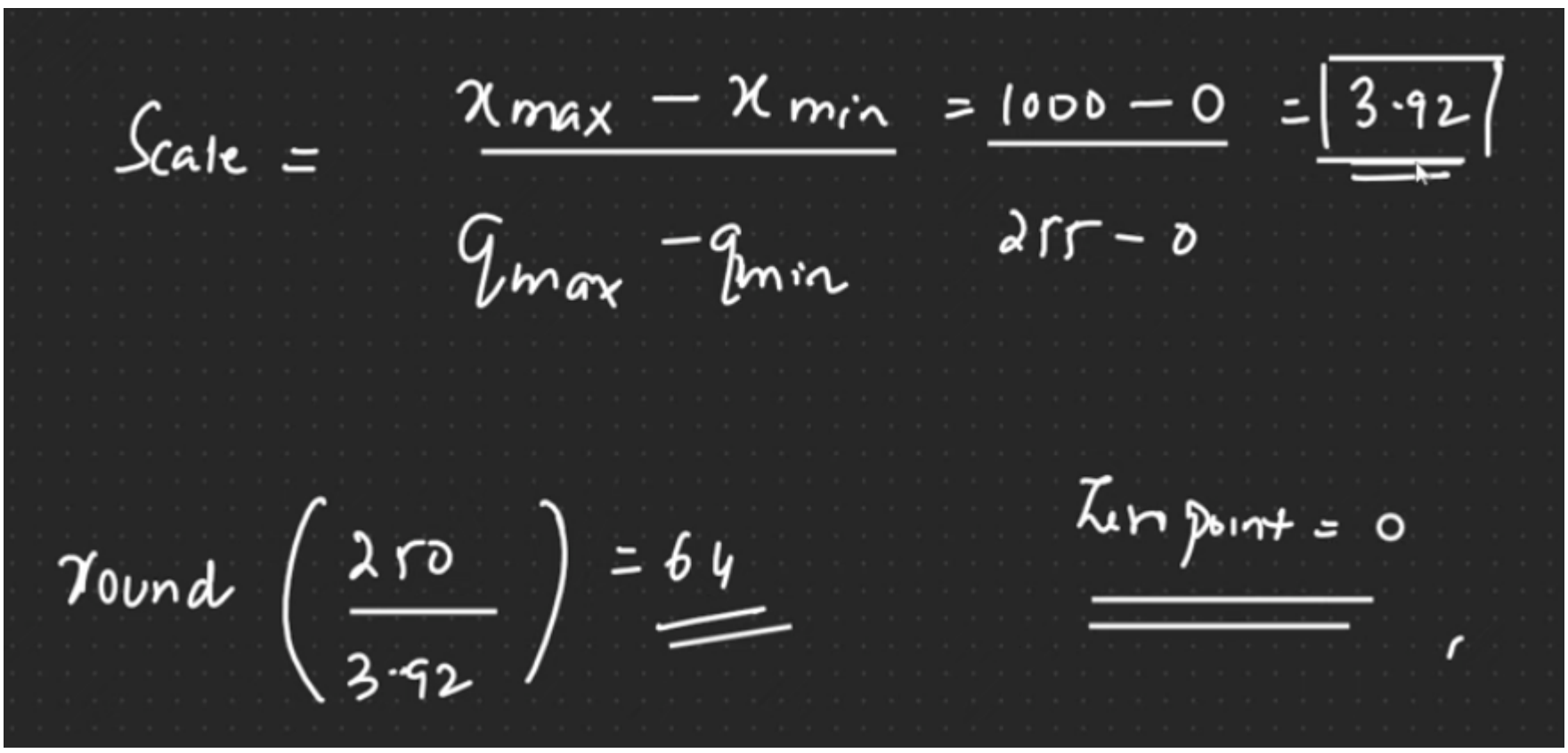


this -20.0 if i quantize it i will get -5 now you can understand this -20.0 is converted to -5.0 but you can see the distribution starts with 0 - 255

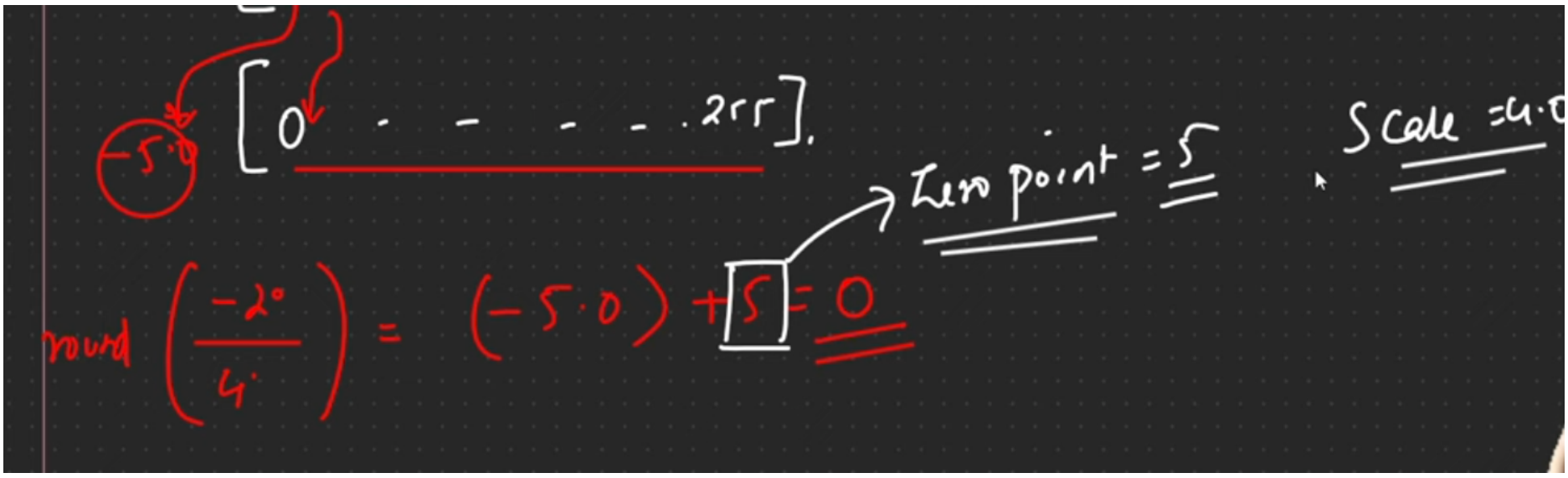
so, the thing is how can the -20.0 can forcefully make it to 0.0

we can add the same number as the answer came +5 by this we will get zero

In this case the number that we added +5 is zero point



for the above 1 which is the symmetrical distribution here the zero point was 0 only and scale was 3.92.

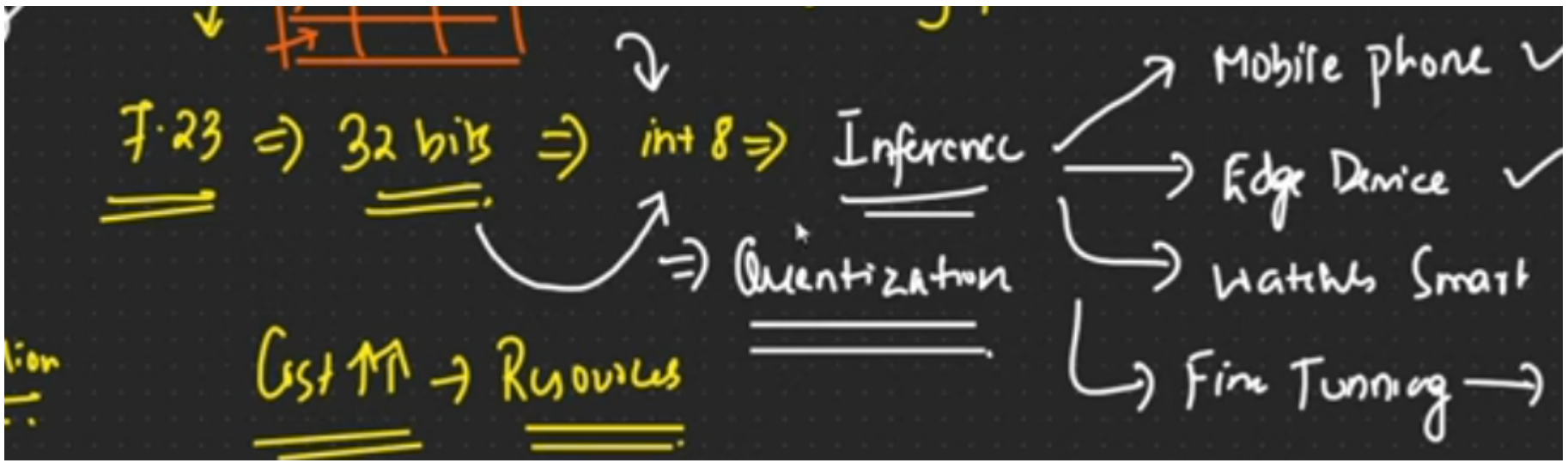


In this particular case Asymmetrical distribution here we have zero point = 5, scale = 4.0

👉🏾 **So, this 2 parameters Zero Point, Scale. we specially required for Quantization.**

2. Calibration

Calibration: Ensuring Accuracy - A Calibration is basically means how we will be able to **convert this 32bit into int8**



Calibration plays a vital role in minimizing accuracy loss during quantization. It involves adjusting the quantization process to optimize for the specific characteristics of the model and data.

The "Squeezing" Analogy: Imagine you have a range of values from 0 to 1000, and you want to squeeze them into the range of 0 to 255. Calibration helps determine the best way to "squeeze" these values while preserving as much accuracy as possible.

Mathematical Intuition: Calibration involves calculating a scale factor and a zero point that determine the mapping between the original floating-point values and their quantized integer counterparts.

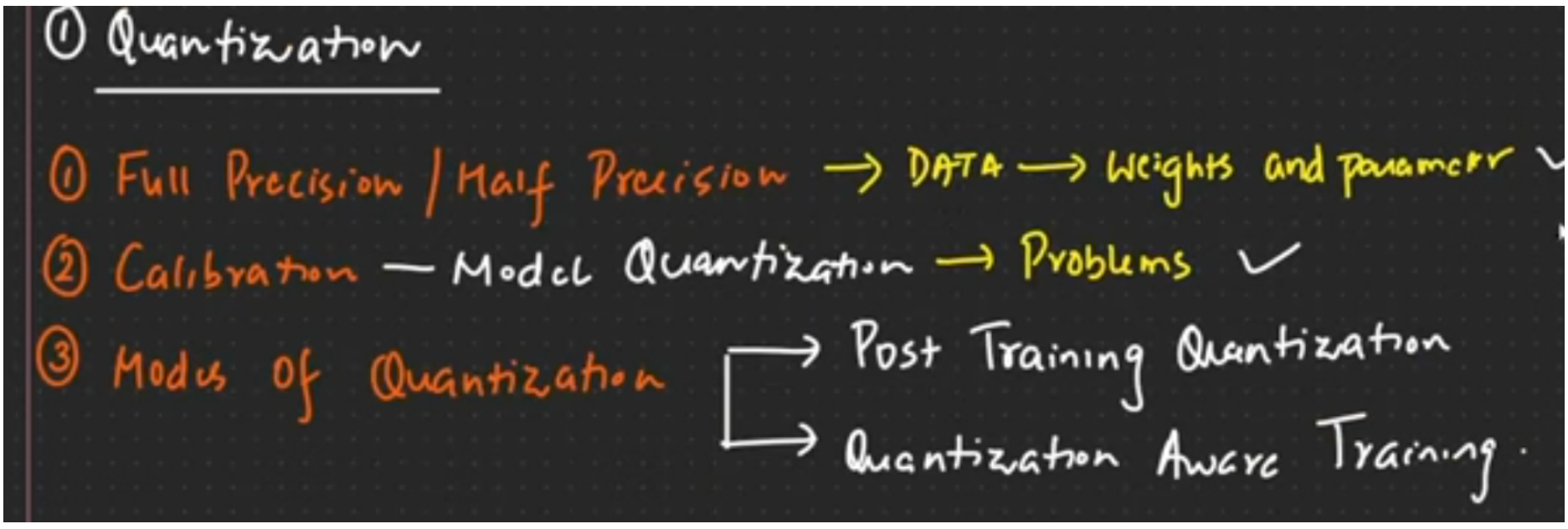
Quantization in Practice

TensorFlow: TensorFlow provides convenient functions and tools for performing quantization, simplifying the process.

Manual Quantization: While TensorFlow simplifies the process, understanding the underlying concepts and calculations is essential for fine-tuning and optimizing the quantization process for your specific use cases.

---

3. Modes of Quantization



1. Post Training Quantization
2. Quantization Aware Training

1. Post Training Quantization (PTQ).

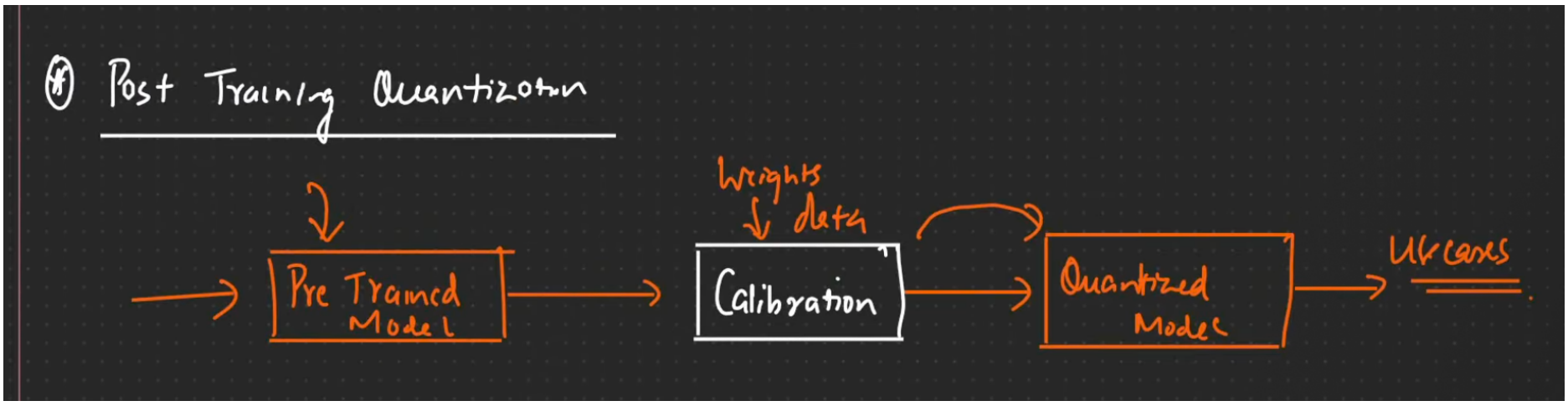
Concept : This method quantizes the model after it has been fully trained.

Process : PTQ involves applying calibration to a pre-trained model to determine the optimal mapping for quantization. This calibrated model is then converted into a quantized model, which can be used for inference.

Benefits : PTQ is a simpler approach, as it doesn't require retraining the model.

Limitations : PTQ can lead to greater accuracy loss compared to QAT because it doesn't account for quantization during the training process.

Detailed Examination :



Here we have a pre trained model , if i want to use it, yes the weights will be high.

we apply Calibration (Squeezing the value) from higher to lower formate and then after performing the calibration, we take the weights data. what ever data is their in the pre-trained model we convert this in to a quantized model by applying this calibration process

then we can use this entier model for any use cases. This is a simple mechanism w.r.t Post Training quantization

whats happening : (Have a pre trained model were weights are fixed. i don’t need to change those weights, i will take this weights data, apply calibration and convert this in to a quantized model)

2. Quantization Aware Training

Concept : This method involves training the model with quantization in mind. It adjusts the weights during training to optimize for the quantized format.

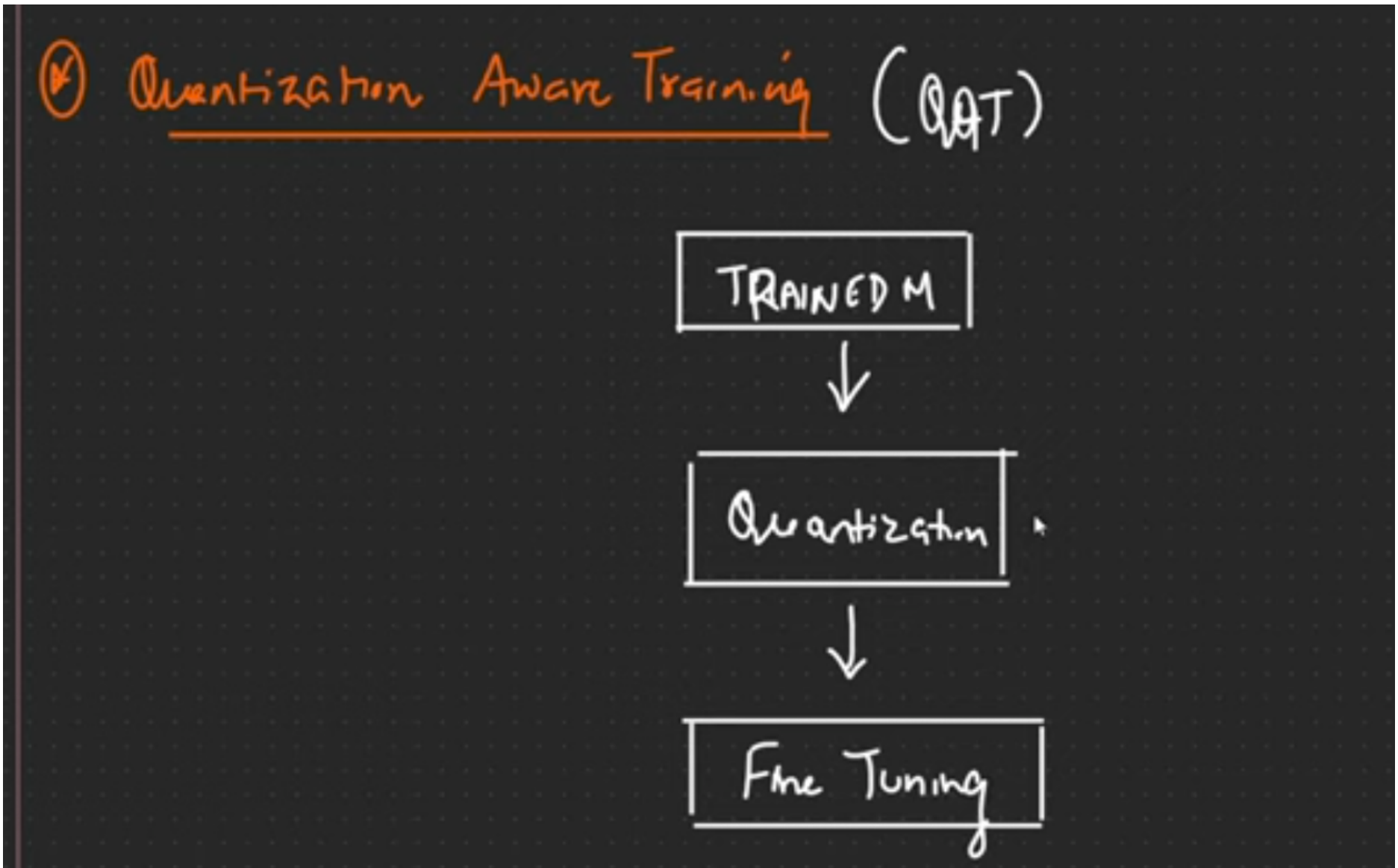
Process : QAT involves introducing quantized operations during the training process, allowing the model to adapt to the lower precision. This results in a quantized model that has been trained to minimize accuracy loss.

Benefits : QAT can lead to better accuracy than PTQ because it accounts for quantization during the training phase.

Considerations : QAT requires additional training time and resources.

Detailed Examination :

over there, the problem is if i perform calibration and create a quantized model their is a **loss of data because of this the accuracy will also decrease** for any use cases



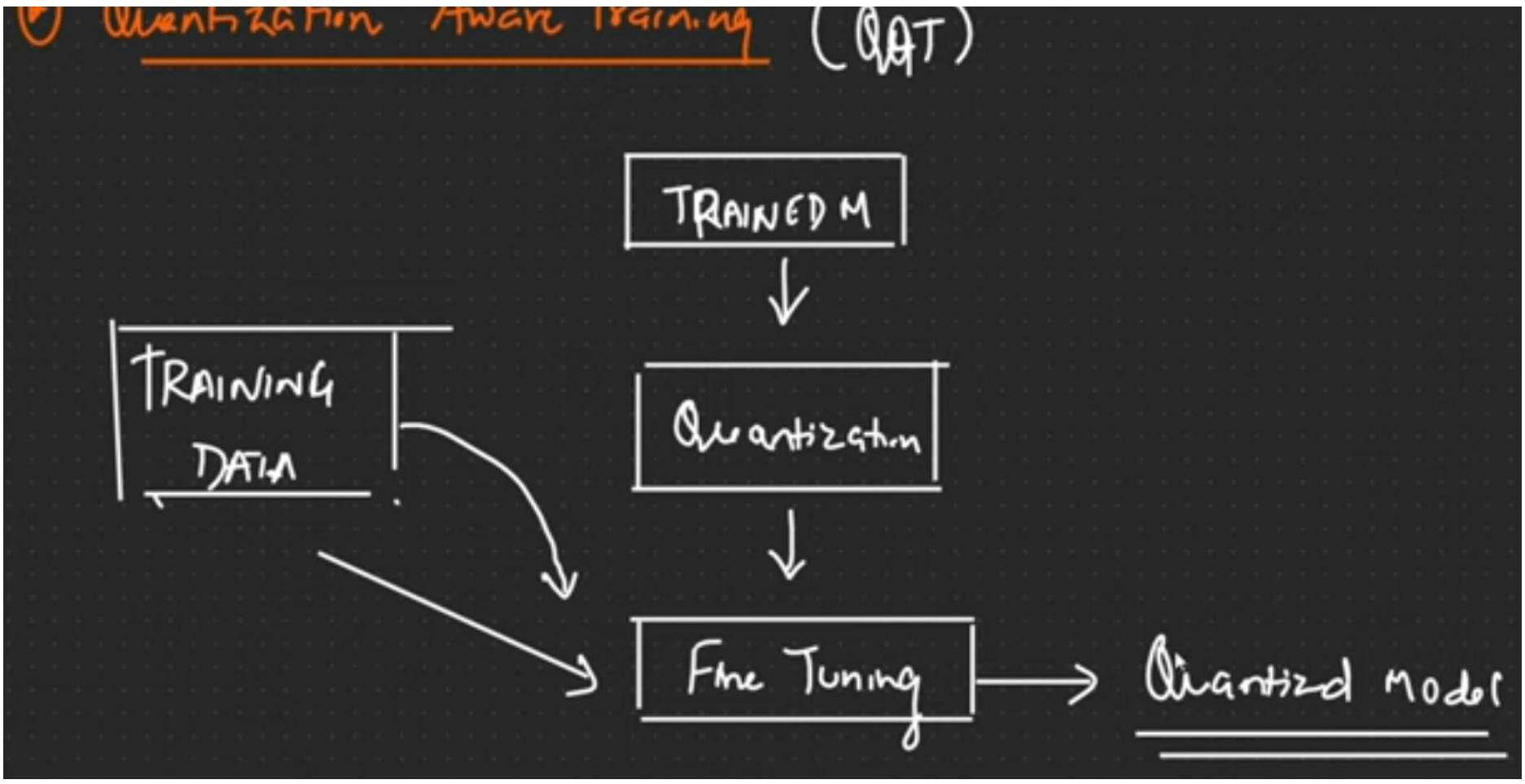
In case of QAT:

We will be taking our train model what ever the train model we use

Perform quantization = same calibration process apply.

Next step we will perform fine tuning.

We know that w.r.t PTQ some loss of data and accuracy will be thier



Here, w.r.t fine tunning we will take new training data. once we specifically take new training data we will be able to fine tune this model and then we create a quantized model

W.r.t to any fine tunning we dont use Post Training Quantization PTQ, we specifically use QAT_Technic.

That basically means we are just not losing data over here we are inturn adding more data for the training purpose through this we will fine tunning our data and then we create our quantized models.

This is the basics difference, all the fine tunning will be this type QAt so that we will not loss much data and accuracy

> 👉🏽 Their are 2 important technique we need to understand QLoRA, LoRA. We nee to understand specifically w.r.t Fine Tunning.

Summary

Quantization is a powerful technique for optimizing the memory efficiency and inference speed of LLMs. By understanding the different methods, calibration techniques, and the trade-offs involved, you can effectively apply quantization to deploy LLMs on various platforms and enhance their performance.

Important Considerations

Accuracy Trade-offs : Quantization always involves a trade-off between memory savings and accuracy. You need to evaluate the impact of quantization on your model's performance and select the appropriate method for your specific needs.

Hardware Compatibility : Ensure that the hardware you're targeting supports the chosen quantization format (e.g., INT8) to avoid compatibility issues.

Fine-Tuning Strategies : For fine-tuning LLMs after quantization, it's generally recommended to use QAT to minimize accuracy loss.This revised document offers a clearer and more structured explanation of quantization.
