LoRA, QLoRA: In-Depth Mathematical Intuition - Fine-Tuning LLM Models

Introduction

Fine-tuning large language models (LLMs) like GPT-4 and GPT-4 Turbo is a crucial aspect of adapting them to specific tasks and domains. However, traditional full-parameter fine-tuning methods present significant challenges due to memory constraints and computational resources. This is where techniques like LoRA (Low-Rank Adaptation) and QLoRA (Quantized LoRA) come into play. These techniques offer efficient ways to fine-tune LLMs by selectively updating parameters.

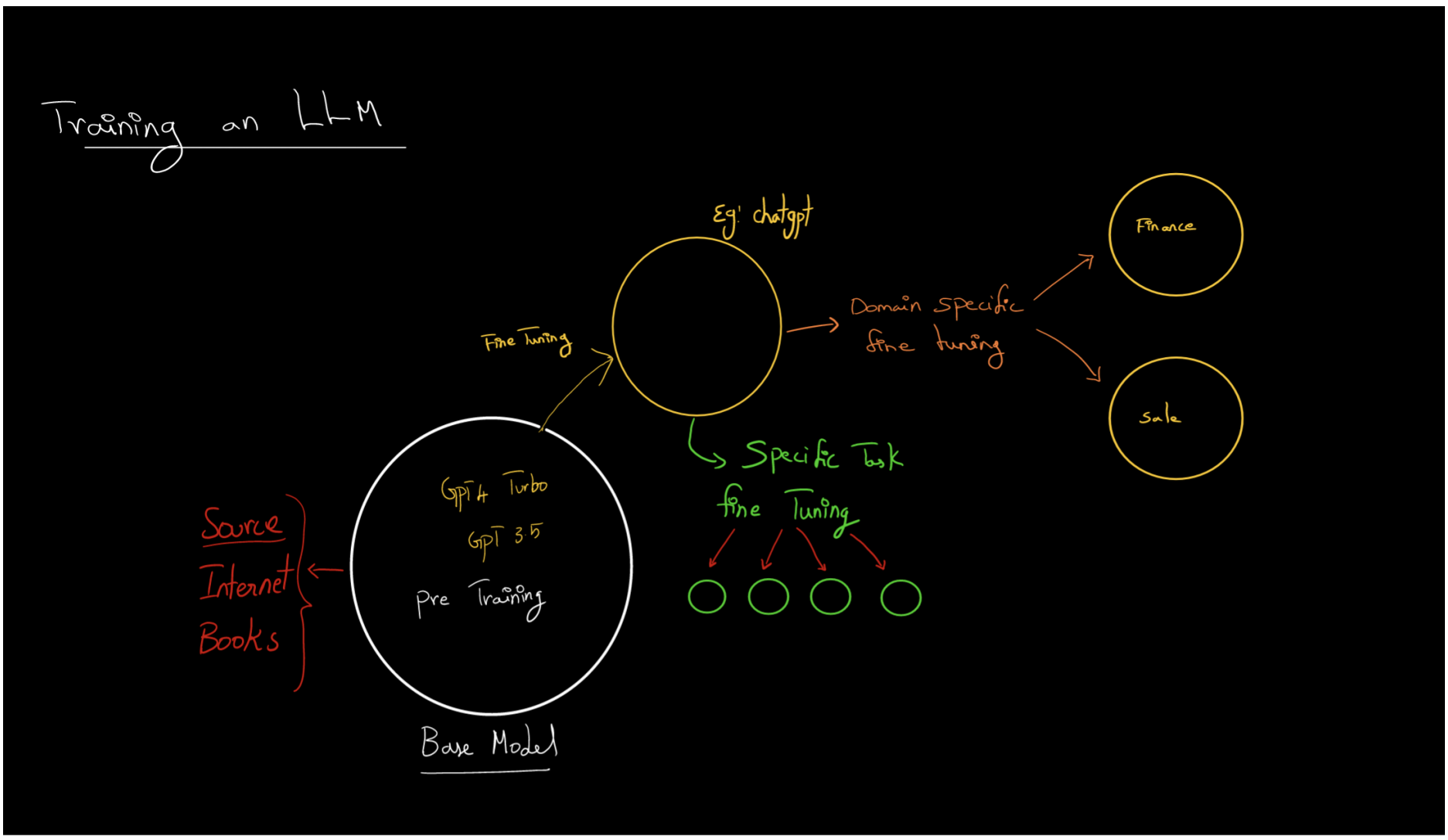
Why LoRA and QLoRA is used?

Low Rank Adaptation(LoRA) is specifically used in Fine Tuning LLM Models.

Initially when ever their is a pre-trained LLM model basically means Gpt4 or Gpt4-Turbo and this model is been created by OpenAi

We basically say this model as Base Model and this model is trained with huge amount of data, So the data sources can be internet, it can be book, can be multiple sources.

all this model they can be measured with token and words like it support 1.5 million token, its been trained with this many number of words. What happens is all this modles probably to predict the next word it will have the context of all those tokens and then it will be able to give response.



these all models are basically base model and we also say this as Pre-Trained Model

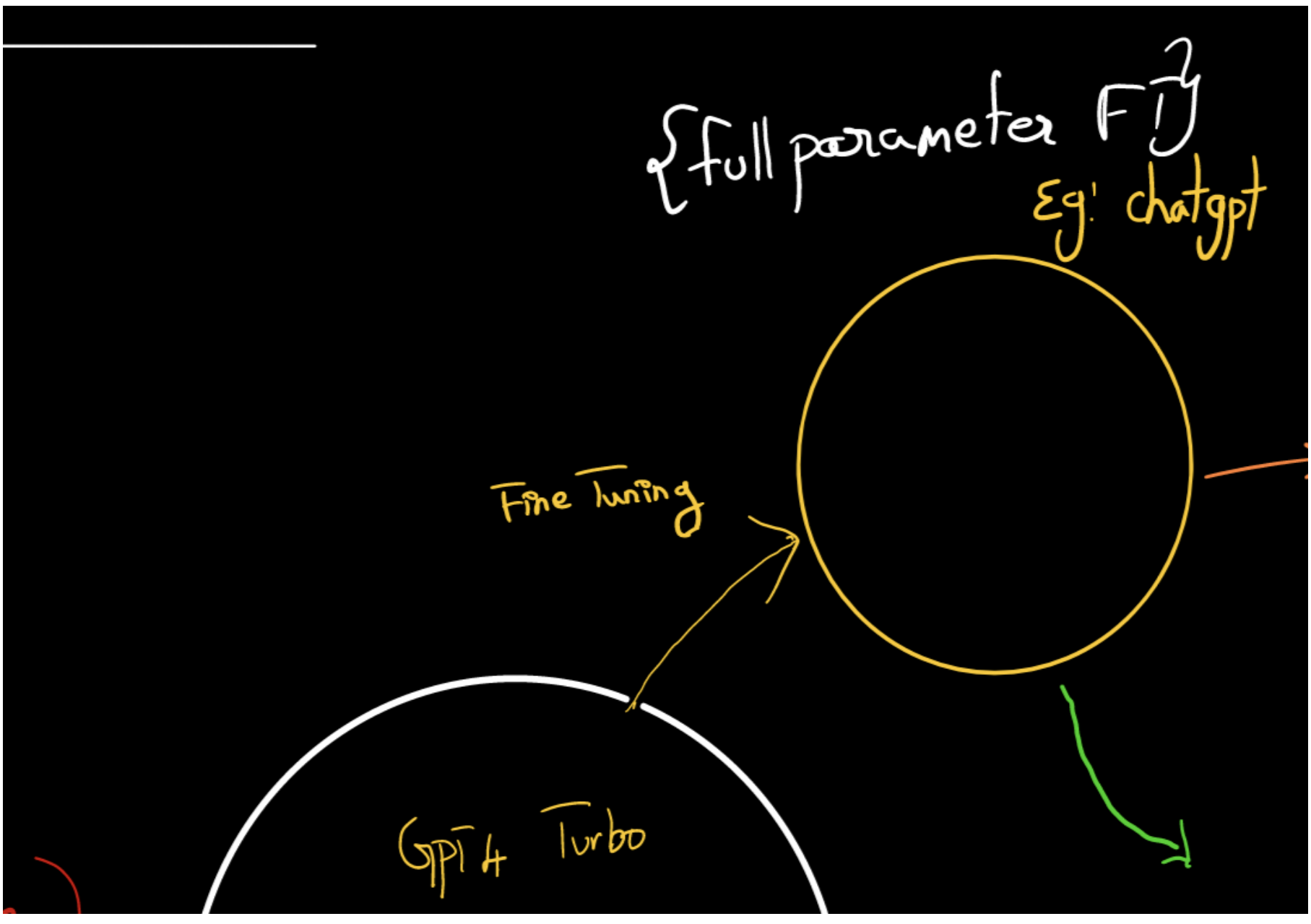
some of the E.g:- Gpt4, Gpt4 Turbo, Gpt 3.5, Gemini, Gemini 1.4.. all these models are specifically pre-trained models

1. The Fine-Tuning Landscape

👉🏾 Before diving into LoRA and QLoRA, let's understand the different approaches to fine-tuning LLMs.

Their are various way for fine tuning it

▼ Full Parameters Fine-Tuning



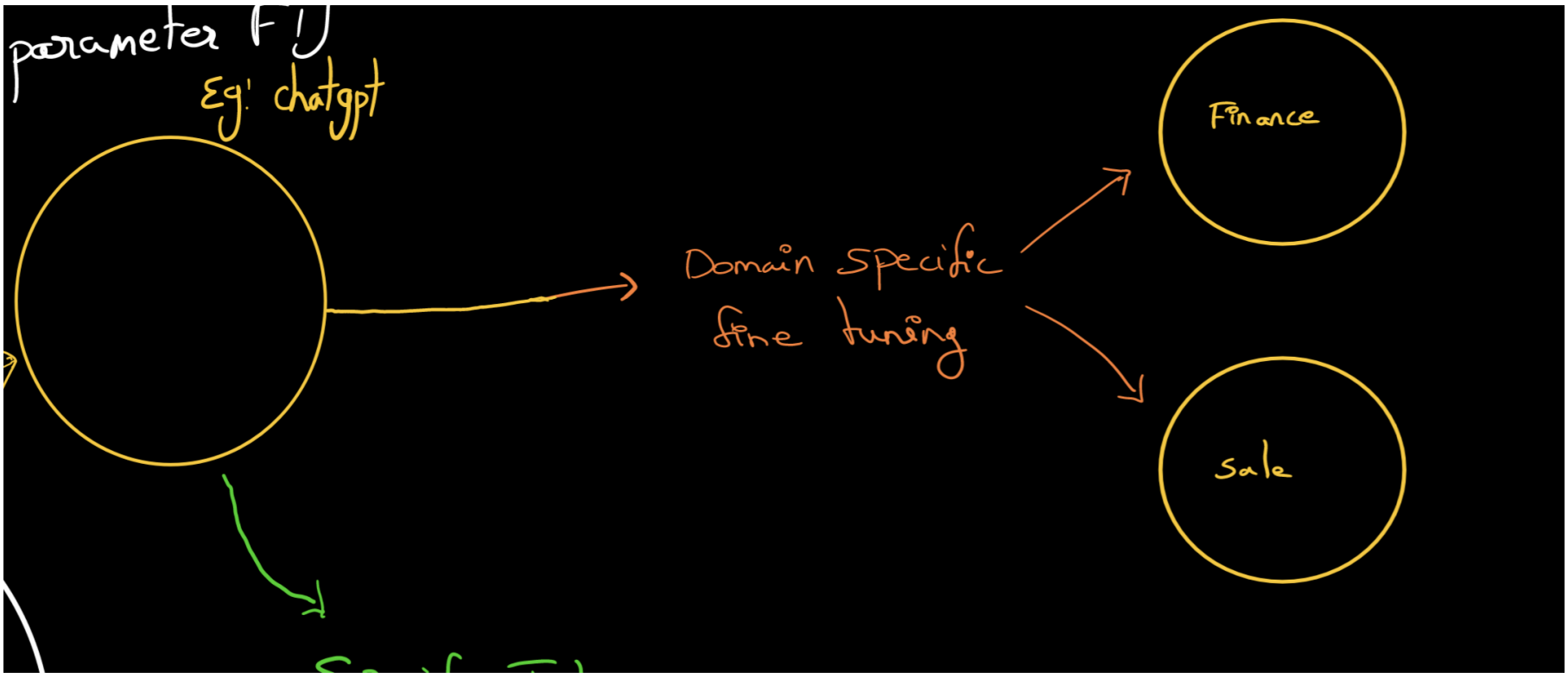
Full Parameter Fine-Tuning: This involves updating all weights within the pre-trained LLM model. This method achieves high accuracy but requires substantial computational resources and memory.

Detail Examination:

This fine tuning is done on all the weights of this specific base model

And, Some of the application we may generate is like Chatgpt, cloudy chatgpt, this is one way of fine tuning were we train all parameters

▼ Domain Specific Fine Tuning

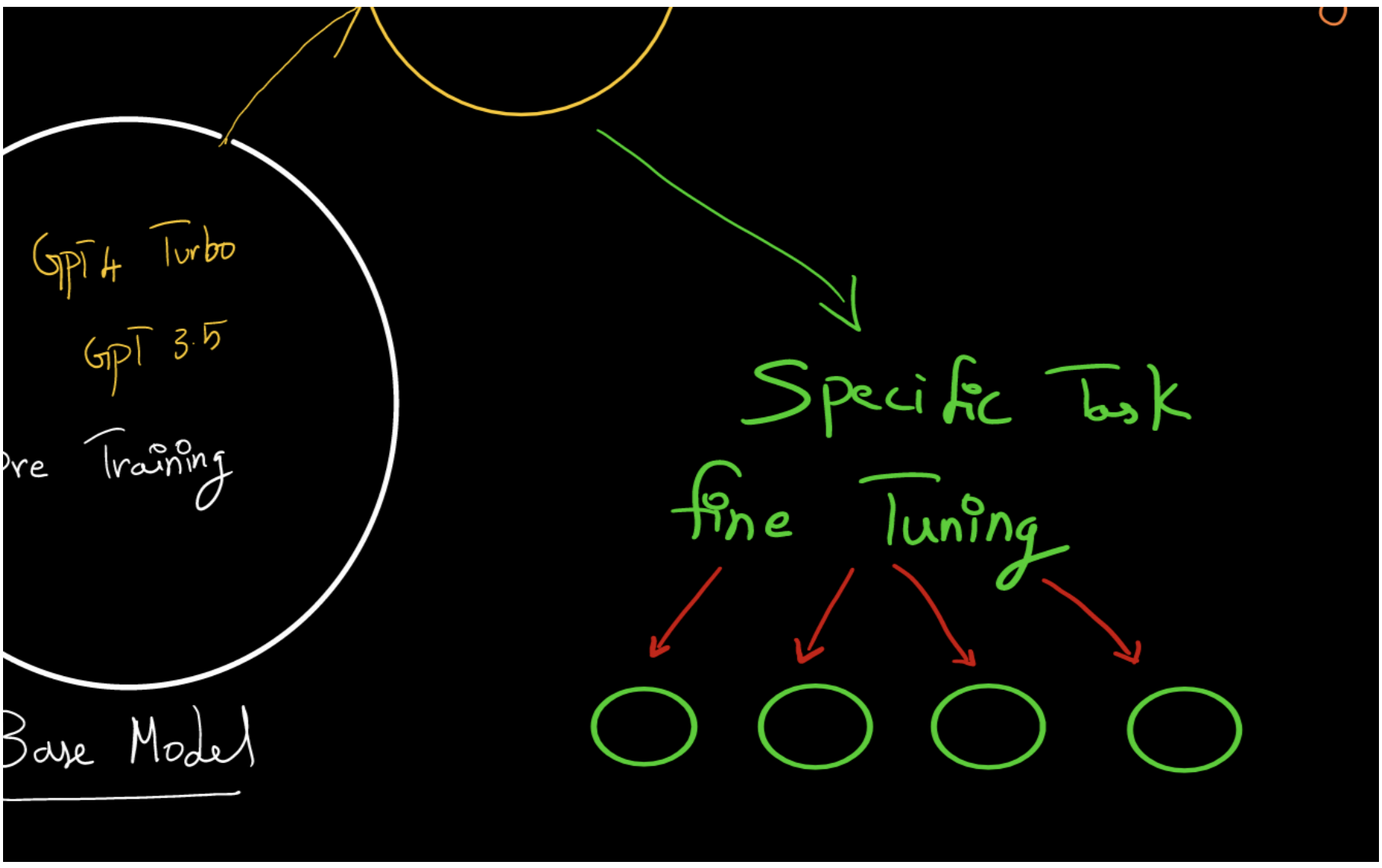


Domain-Specific Fine-Tuning: This approach tailors the pre-trained model to specific domains (e.g., finance, healthcare, legal) by fine-tuning it on a dataset relevant to that domain.

You can also take this model and perform domain specific fine tuning, lets say fine tuning a chatbot model for finance, sales, for different domain itself

---

▼ Specific Task fine Tuning



Specific Task Fine-Tuning: This focuses on training the model for specific tasks, such as question answering, text summarization, or translation.

in case of specific task fine tuning this are my different task , like task A, B, C, D.

A can something related to Q&A Chatbot, B can be document Q&A Chatbot.

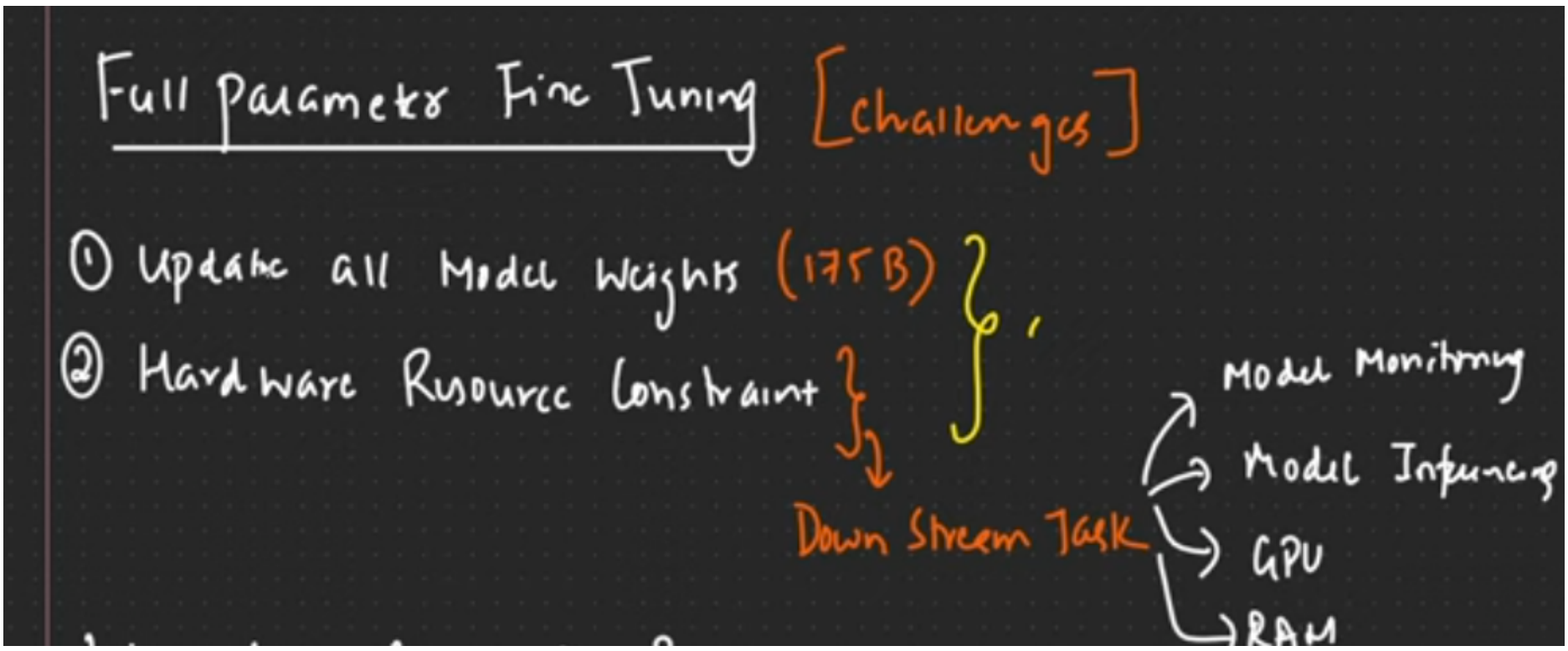
---

2. The Challenges of Full Parameter Fine-Tuning

Full parameter fine-tuning often presents significant challenges due to hardware resource constraints.

Memory Limitations: LLMs typically have billions of parameters, and updating all of them requires a massive amount of memory. Standard GPUs and systems with limited RAM may struggle to handle this.

Downstream Task Challenges: Fine-tuning can make deploying the model for downstream tasks (like model monitoring, inference, and real-time applications) more difficult due to the increased memory requirements and computational complexity.



Detail Examination :

let's see Full Parameter Fine Tuning and challenges with it. That is were LoRA is used.

In full parameter fine tuning the **major challenge is to update the model weight**.

Let's say i have a model which is some way around ***175B parameters that basically means 175 Billion weights.*** In this particular case when ever i fine tune the model i need to update all the weights.

"why it can be a challenge?.. Because their will be Hardware Resource Constraint".

w.r.t different task if really want to use this particular model i need more Gpu and Ram for inferencing purpose, so for **downstream task** it becomes very difficult

Downstream task E.g:- *Model Monitoring, Model Inferencing, similarly the Gpu & Ram Constraint we may have, so we may face multiple challenges when we have this full parameter fine tunning*

> 👉🏽 In order to overcome this fact we will be using LoRA and QLoRA.

3. Introducing LoRA: Efficient Fine-Tuning

LoRA (Low-Rank Adaptation) addresses these challenges by introducing a more efficient way to fine-tune LLMs. Instead of updating all parameters, LoRA selectively updates a smaller set of parameters that represent changes from the pre-trained model.

A. How LoRA Works / What Does LoRA do ?

Parameter Tracking : LoRA doesn't directly modify the original pre-trained weights. Instead, it tracks the changes in those weights that occur during fine-tuning.

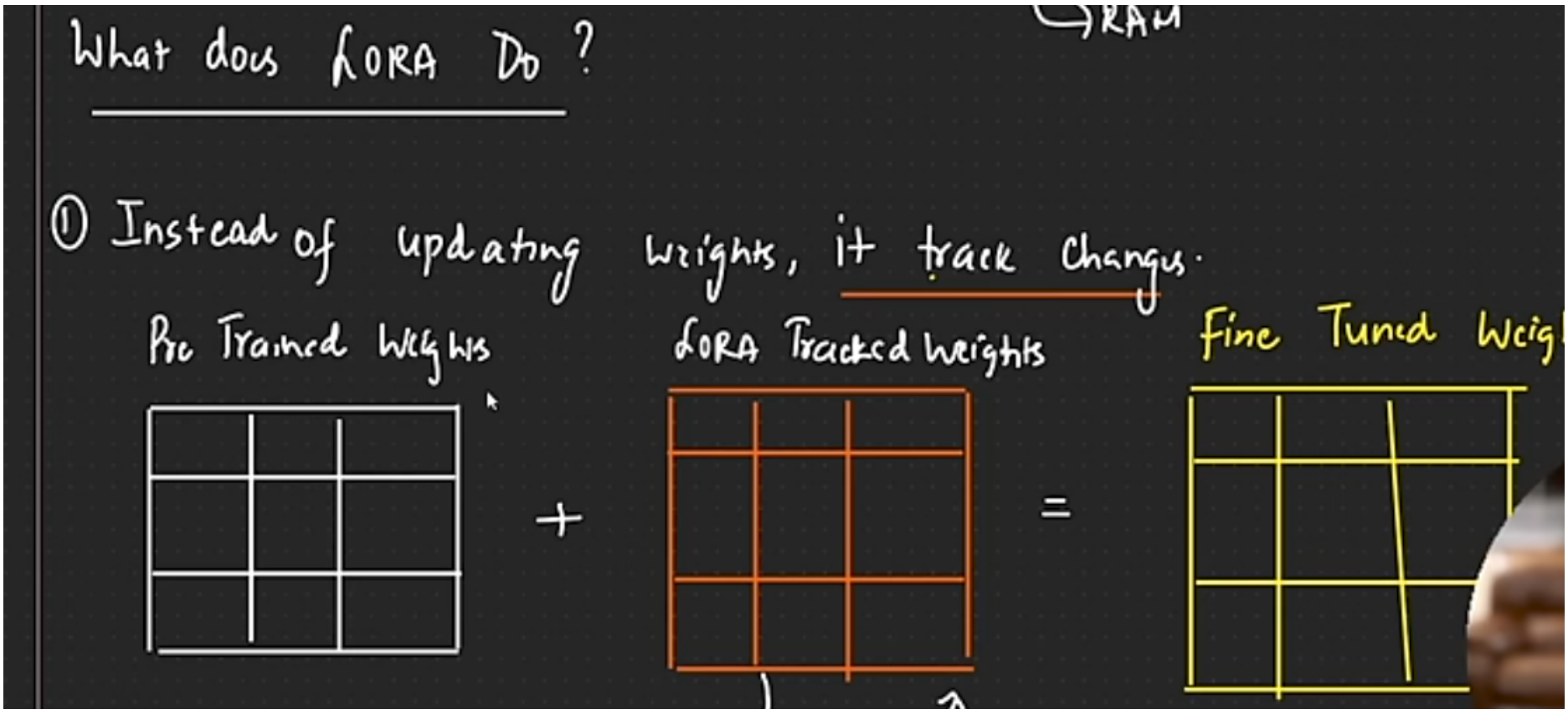
Matrix Decomposition : LoRA achieves this efficient tracking by utilizing matrix decomposition. A large weight matrix (representing the changes) is decomposed into two smaller matrices (often of a lower rank). This decomposition reduces the number of parameters that need to be stored and updated.

Combining Changes : The decomposed matrices are then combined with the original pre-trained weights to create the final fine-tuned model.

Detail Examination :

LoRA says that Instead of updating all the weights in Full parameter fine tunning, **it will not update them Instead it will track the changes**

[Now what changes it will be tracking] == **It tracks the changes of the new weight based on fine tuning**



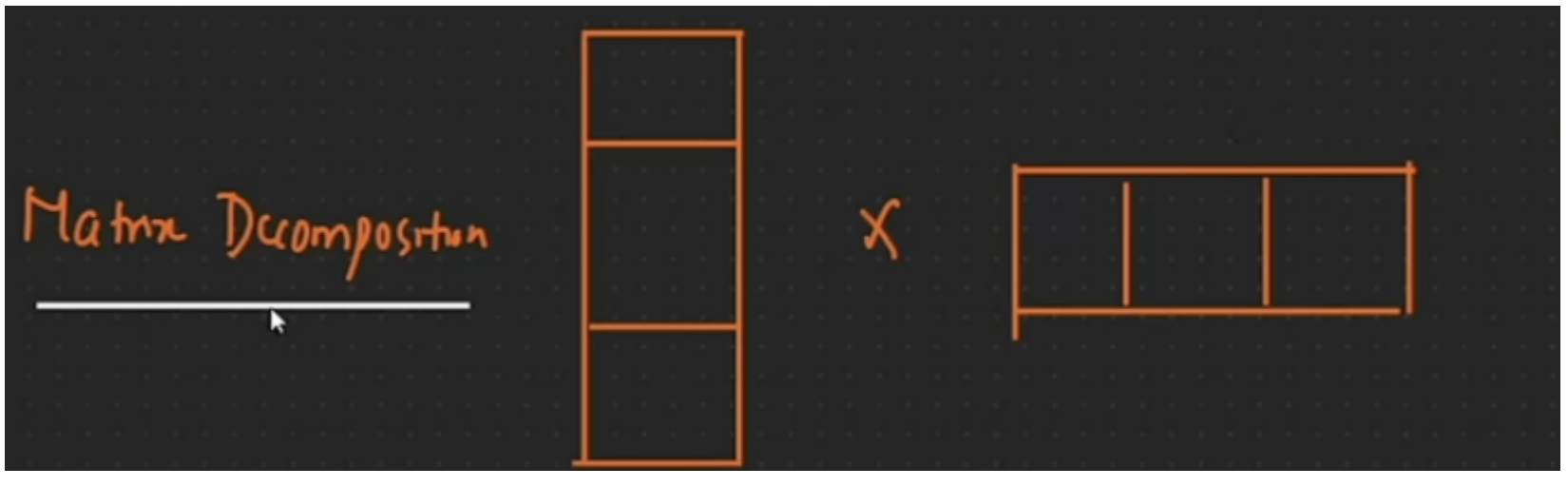
So, Based on the new weights how we will be combining these weights with pre-trained weights.

From above you can see the Pre-Trained Weights from base model(Like lama2). if we performe fine tune using LoRA , the **LoRA will Track the new Weights** from above image which will be of same size. lets say it's 3×3 then the new weight's when it probably do the forward and backward propagation those new weights will be tracked in a separate matrix **3×3** and then this 2 weights 3X3 and **3×3** will combine and we wil get the **Fine Tuned Weights.**

In this way what happens is that, this tracking will happen in a separate way

you can see here also we are Updating all the Weights here also the resource constraint will definitely happens. Yes right its about 3×3 just think of weights and parameters of 175 billion or 7 billion that time i will be having a huge matrix.

At this scenario we need to understand how LoRA works because these **Weights how it's getting Tracked it will just not get tracked in 3×3 matrix instead all this weights been tracked**



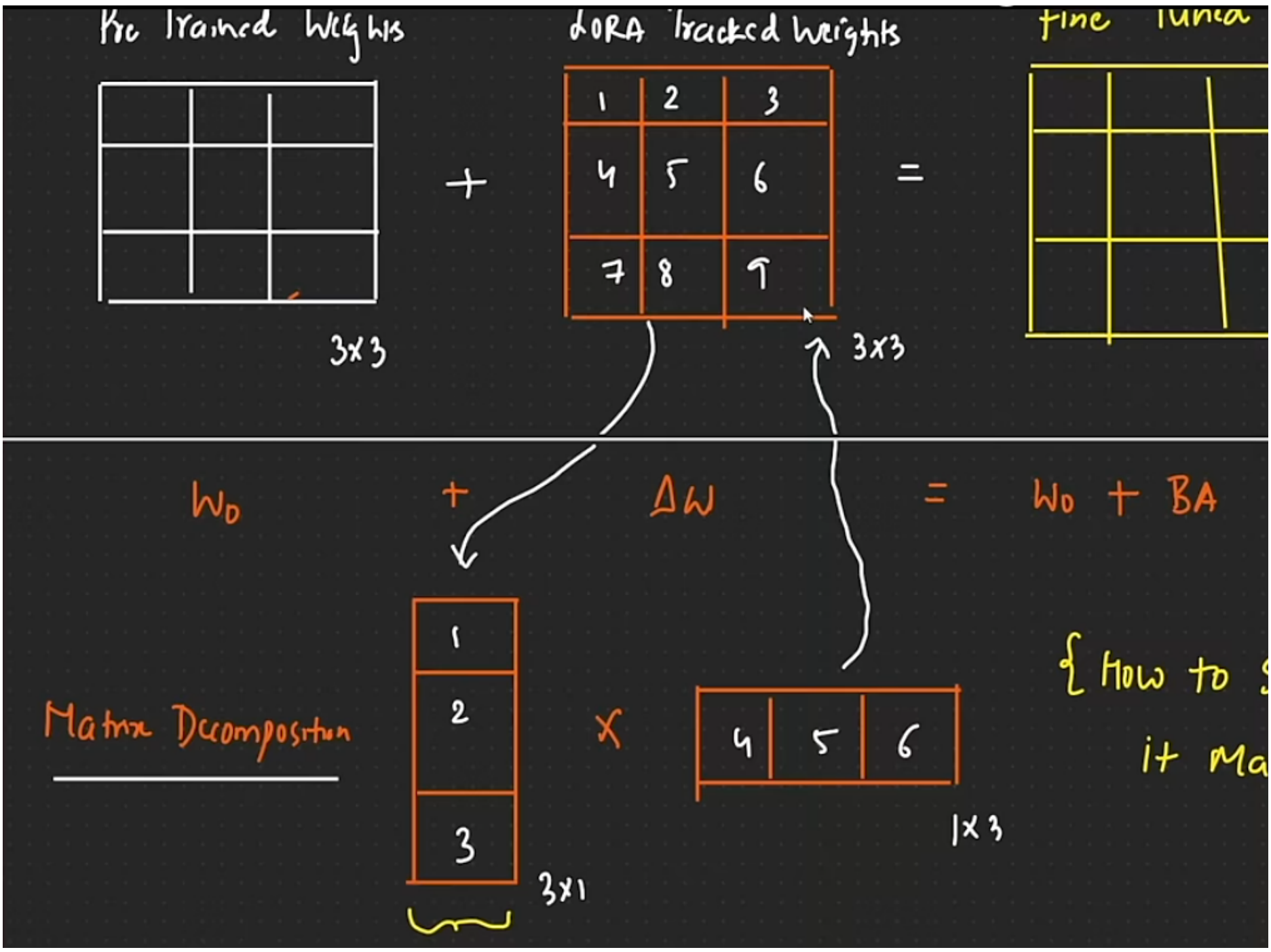
Their a simple mathematical equation will happen or technique called Matrix Decomposition

That means the same 3×3 matrix is saved in two smaller matrix, How . .?

Example : Imagine a 3×3 weight matrix representing changes. LoRA decomposes this into two matrices: a 3×1 matrix (B) and a 1×3 matrix (A). When multiplied, these smaller matrices reconstruct the original 3×3 change matrix.

Detail Examination :

we have a **3×3** we can save it as 3×1 —- 1×3 when we Multiply this two matrix we will be able to find that tracked weights 3×3



let’s consider around 9 weights you can see from above image we will be able to get those 9 weights from 6 number of parameters. When we Multiply this we will be able to get all these 9 parameters or weights

Inshort what LoRA is doing is that it is performing this Matrix Decomposition were the big matrix and this matrix can be of any size is decomposed in to 2 smaller matrix based on the parameter called as Rank.

B. Understanding Rank and its Impact :

Rank : The rank of the decomposed matrices influences the complexity of the changes that LoRA can capture. A higher rank allows LoRA to learn more complex relationships, but it also requires more parameters to represent those changes.

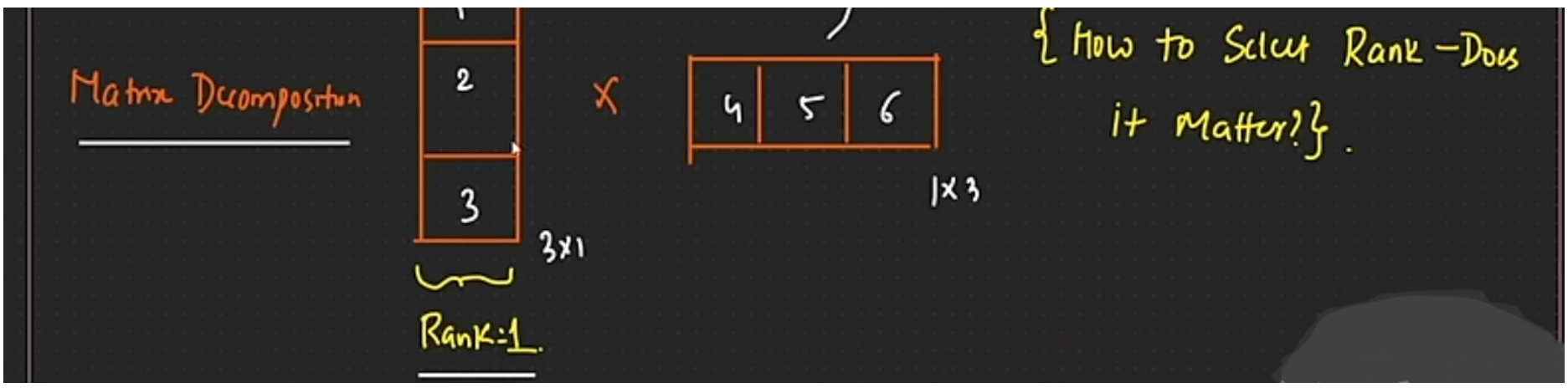
Balancing Efficiency and Complexity : LoRA aims to find the optimal rank that balances efficiency (reduced memory and computational cost) with the ability to learn the necessary changes during fine-tuning.

Research Findings : Research indicates that LoRA performs well with ranks typically ranging from 1 to 8.

Detail Examination :

How to calculate a rank of matrix?

It is a simple Algebraic equation based on Transpose of a matrix how we calculate the Rank

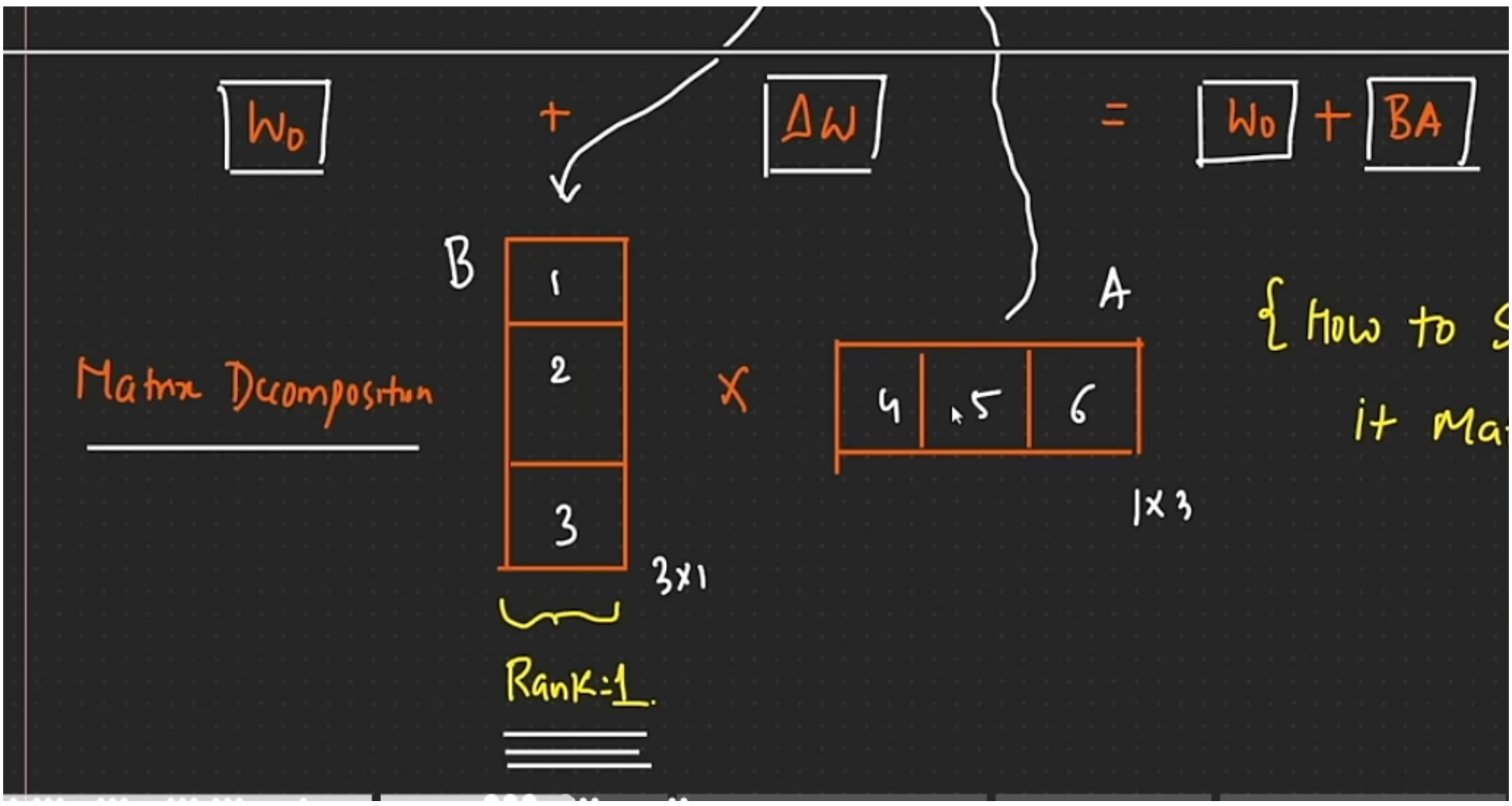


let's say that this matrix that i have which is a 3×3 one over here the rank of this particular matrix is 1 and if i use this 2 matrix you can see the number of parameters that im storing over here is less when compared to this 3×3

"Yes, their will some loss in precision but it's making sure that when we combine both this matrix we will be able to get the entier updated weight"

just imagine lets say i have 7 billion parameters now i am trying to perform fine tuning on those parameters when ever i track those weights this huge matrix which we saw(eg 3×3) will be decomposed into 2 smaller matrix

when we are decomposing this Updated Tracked Weights matrix into 2 smaller matrix obviously we will be needed less parameters to store those values, and by this way your fine tuning will become very much efficient and this really solve Resource constraint



and obviously this required less parameters if you decomposing our bigger matrix in to 2 smaller matrix less parameters is required

C. When to Use High Rank :

Complex Relationships : In cases where the fine-tuned model needs to learn more complex relationships, a higher rank might be necessary to represent those changes effectively.

Beyond the Base Model's Capabilities : If the base model has limited capacity to learn specific tasks or behaviors, a higher rank can help the fine-tuned model overcome these limitations.

IF THE MODEL WANTS TO LEARN COMPLEX THINGS THE WE CAN SPECIFICALLY USE HIGH RANK ***[let's say some of the model is not trained to interact or perform some of the behaviour at that point of time those complex things can be handled when you are probably increasing the number of ranks] .***

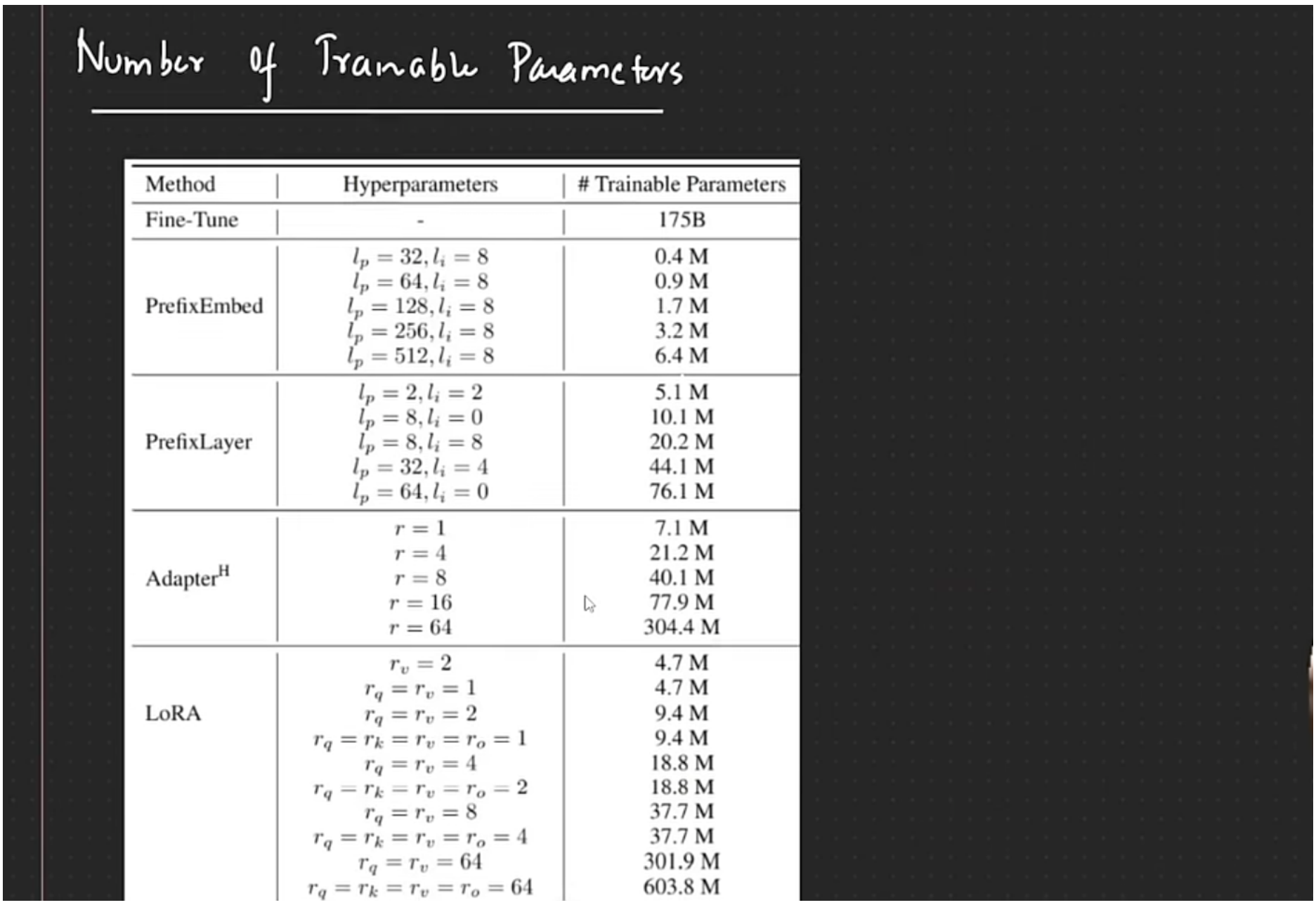
Now what will happen if we keep on increasing the Rank?

Detail Examination :

if you keep on increasing the rank, A&B parameters will also get Increasing but it will always be less than **LoRA Tracked Weights**.

> 👉🏾 *eg. IF I HAVE 7_BILLION PARAMETERS IF DECOMPOSED IN TO 2 SMALLER MATRIX WITH INCREASING RANK THEN ALSO THE PARAMETERS REQUIRED WILL BE LESS..*

How i am saying this because in research paper they have tried with multiple trainable parameters



Method	Hyperparameters	# Trainable Parameters
Fine-Tune	-	175B
PrefixEmbed	$l_p = 32, l_i = 8$	0.4 M
	$l_p = 64, l_i = 8$	0.9 M
	$l_p = 128, l_i = 8$	1.7 M
	$l_p = 256, l_i = 8$	3.2 M
	$l_p = 512, l_i = 8$	6.4 M
PrefixLayer	$l_p = 2, l_i = 2$	5.1 M
	$l_p = 8, l_i = 0$	10.1 M
	$l_p = 8, l_i = 8$	20.2 M
	$l_p = 32, l_i = 4$	44.1 M
	$l_p = 64, l_i = 0$	76.1 M
Adapter^{H}	$r = 1$	7.1 M
	$r = 4$	21.2 M
	$r = 8$	40.1 M
	$r = 16$	77.9 M
	$r = 64$	304.4 M
LoRA	$r_v = 2$	4.7 M
	$r_q = r_v = 1$	4.7 M
	$r_q = r_v = 2$	9.4 M
	$r_q = r_k = r_v = r_o = 1$	9.4 M
	$r_q = r_v = 4$	18.8 M
	$r_q = r_k = r_v = r_o = 2$	18.8 M
	$r_q = r_v = 8$	37.7 M
	$r_q = r_k = r_v = r_o = 4$	37.7 M
	$r_q = r_v = 64$	301.9 M
	$r_q = r_k = r_v = r_o = 64$	603.8 M

they are multiple techniques of fine tuning some of the technique that are their in column method fine tune in above tabular image here Adapter is very famous that is probabilly used befor LoRA

In Hyperparameter & Trainable Parameters we can find that as the Rank is increasing the parameters are decreases compared with Billions of parameters

Initially the trainable parameters are 175B but when i use technique like Adaptor it decreases to Million, initially it is 7.1M with Rank = 1 as i keep on Increasing the Rank the T_parameters will also Increase

we can compare 175B to 7.1M Weights the Percentage is very less

ll’y in LoRA because of Matrix Decomposition we can find as the Rank increases weights will also increase with less percentage compared with 175B

LoRA	$r_v = 2$	4.7 M
	$r_q = r_v = 1$	4.7 M
	$r_q = r_v = 2$	9.4 M
	$r_q = r_k = r_v = r_o = 1$	9.4 M
	$r_q = r_v = 4$	18.8 M
	$r_q = r_k = r_v = r_o = 2$	18.8 M
	$r_q = r_v = 8$	37.7 M
	$r_q = r_k = r_v = r_o = 4$	37.7 M
	$r_q = r_v = 64$	301.9 M
	$r_q = r_k = r_v = r_o = 64$	603.8 M

The Rank q, k, v. This 3 parameters Q, K, V

there only the all multiplication matrix happen w.r.with 3 parameters

> 👉🏽 *Compare from 175B to 4.7M parameters how it was possible ?*
>
> *Ans:- Matrix Decomposition, it's making sure that parameters are not much compared with 175B*

As we keep on increasing the rank from above image take 4, 8, 64 the parameters are obviously increasing but comparing with 175B it is very less 18.8M, 37.7M, 603.8M

▼ *Note* :

- MicroSoft came up with this LoRA technique in one of the Research paper and the have used Rank = 8 to do the Fine Tuning and it as performed absolutely well.
- Most of the time we select Rank 1 - 8 but at E.O.D how to select this Rank ?
- It won't matter because the parameters are increasing very less number over here as we go ahead so usually we can select Rank 1 - 8 to perform fine tuning their will be a scenario were when should we use High Rank

---

At EOD this equation will bee seen in most of the research paper

what LoRA is doing is, All the Track Weights are Decomposed into 2 smaller matrix with different Ranks

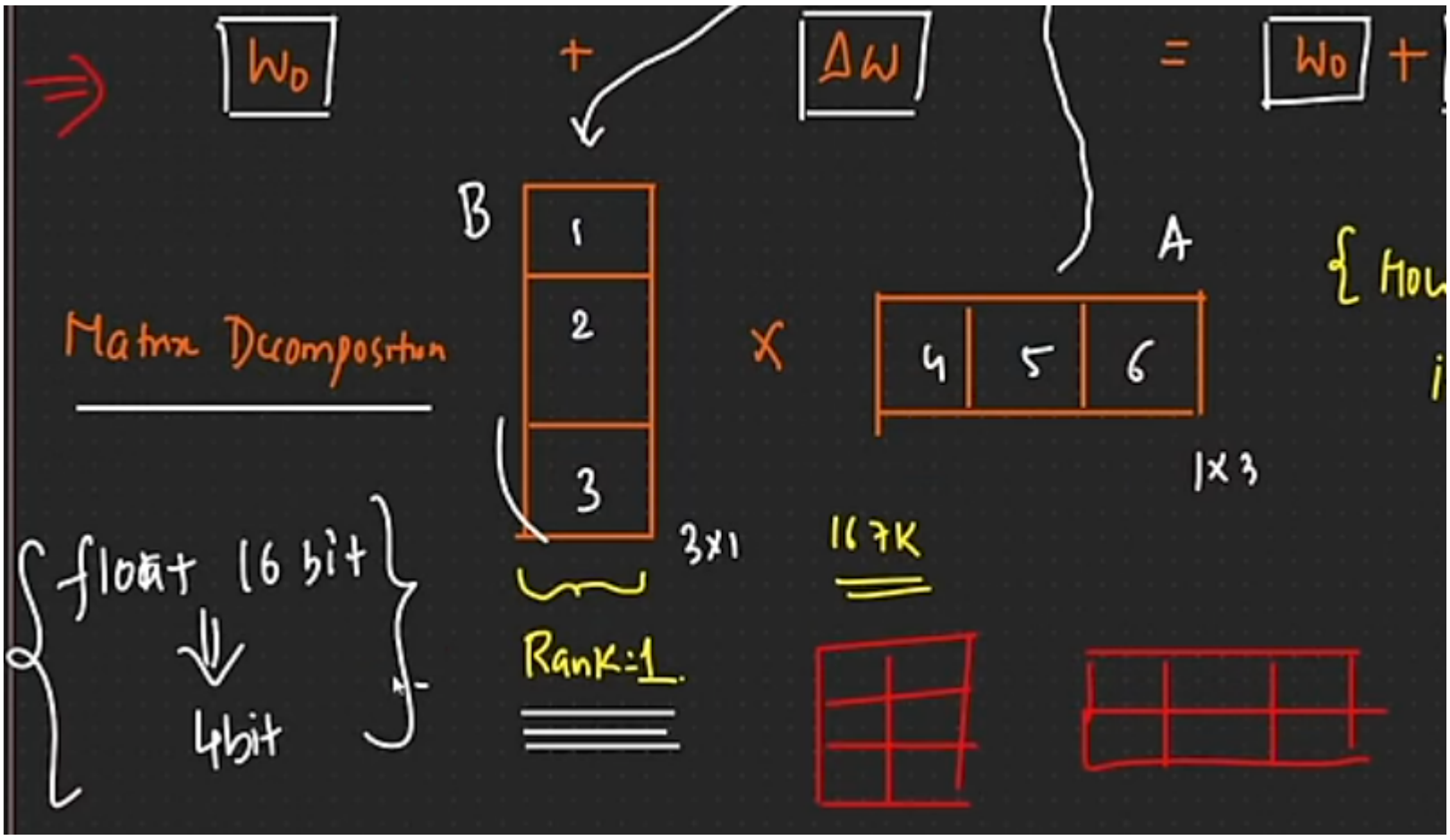
When Fine Tuning the first thing is you really need to set Rank

Because of this technique the Fine Tuning, parameters is done less and this is how the main Resource Constraint is done w.r.t all the DownStream task it becomes very much easy.

"Ok this a LoRA, what about 2.0 ⟹ QLoRA ?"

Quantized LoRA (QLoRA): Further Optimization

QLoRA builds upon LoRA by introducing a quantization step, further reducing the memory footprint of the fine-tuned model.



Concept : The decomposed matrices (B and A) within LoRA are typically stored in FP16 (half-precision). QLoRA converts these matrices to INT4 (4-bit integers), further reducing the memory required to store them.

Precision Trade-off : While QLoRA achieves significant memory savings, it involves a slight reduction in precision. This reduction in precision might lead to a small decrease in accuracy, but this is often offset by the significant memory savings and the ability to deploy the model on devices with limited memory.

Benefits : QLoRA allows for the deployment of fine-tuned LLMs on devices with extremely limited memory, making it suitable for mobile applications and edge computing.

Detail Examination :

Quantized LoRA what happens here is in Matrix Decomposition B&A that are stored in Float 16 bit we will try to convert this into 4 bit, by this we will reduce the precision and reduce the values also by this we won't require more memory that is the reason we say QLoRA technique.

Summary : LoRA and QLoRA are valuable techniques for fine-tuning LLMs while overcoming memory limitations. By efficiently tracking parameter changes using matrix decomposition and leveraging quantization, these methods empower developers to deploy fine-tuned LLMs on diverse platforms and for various real-world applications.

Key Takeaways :

- *Memory Efficiency* : LoRA and QLoRA significantly reduce memory requirements for fine-tuned LLMs, making them more practical for deployment on various devices.
- *Computational Efficiency* : The reduced number of parameters in LoRA and the compressed representation in QLoRA contribute to improved computational efficiency during inference.
- *Accuracy Considerations* : While LoRA and QLoRA offer memory advantages, they might lead to a slight reduction in accuracy. It's crucial to carefully evaluate the trade-off between memory savings and potential accuracy loss.
