Mastering LLM Fine-Tuning: Adapt and optimise with Python

Pymug Meetup March 2025

Presented by: Nirmal Rampersand, Python User Group Mauritius

March 2025

Where it all started 🤔

1
Initial Goal

2
The surprise

3
The realisation

4
Getting started

Agenda

Large Language models (LLMS)

Limitations of LLMs

LLM Fine Tuning

Technical deep dive

Live demo

Potential & Alternatives

Large Language Models

Large Language Models

Generative AI

Understand and generate “human-like” text using neural networks

Models

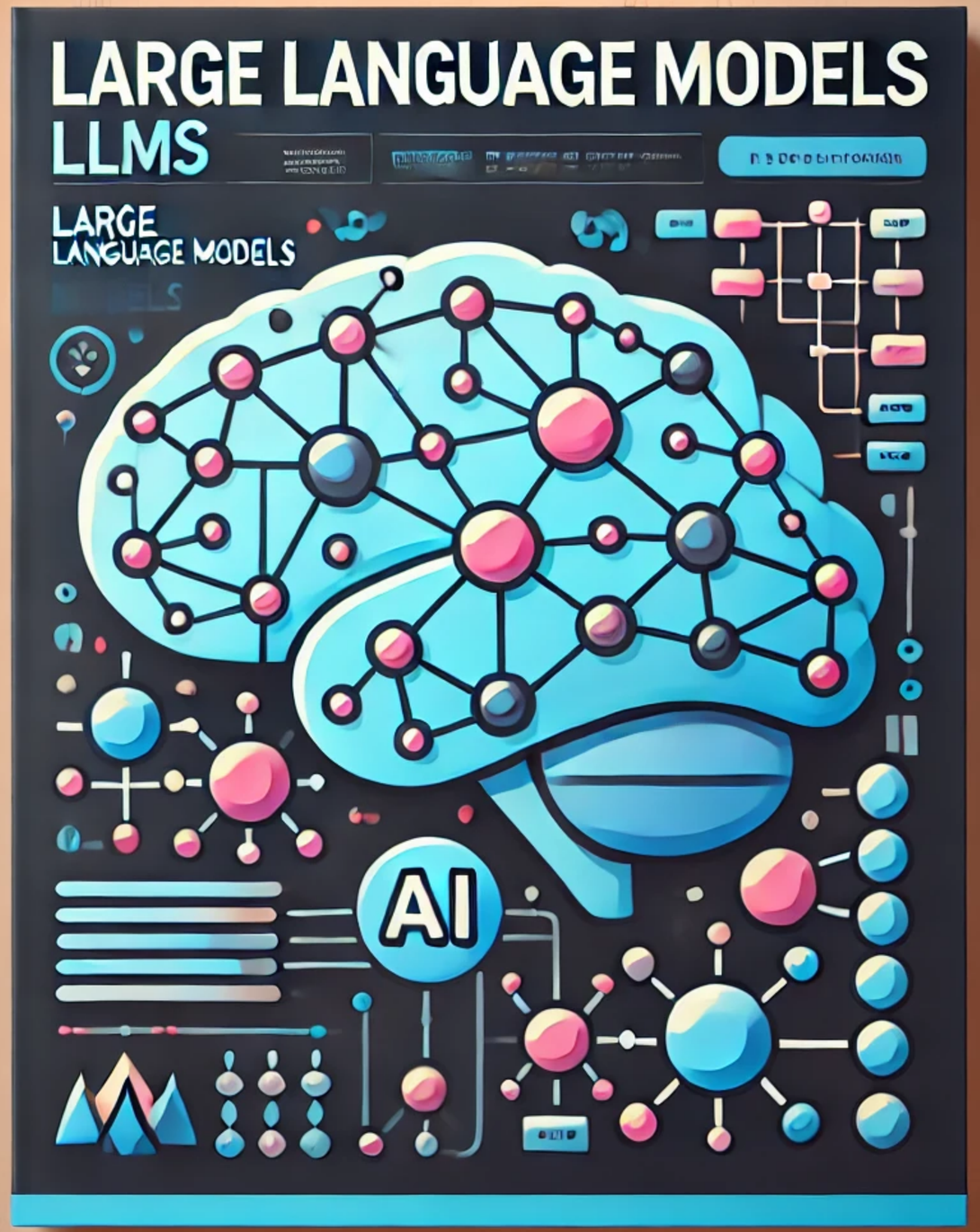
GPT O1, Llama 3.2, Mistral, Deepseek-R1, Qwen 2.5

Applications

Translation, summarisation, question answering, document analysis, coding assistance, research, customer service, etc

Architecture

Deep neural networks using transformer architectures with self-attention mechanisms



Transformer models

Neural network

Learns **context** of sequential data

Generates **new data**

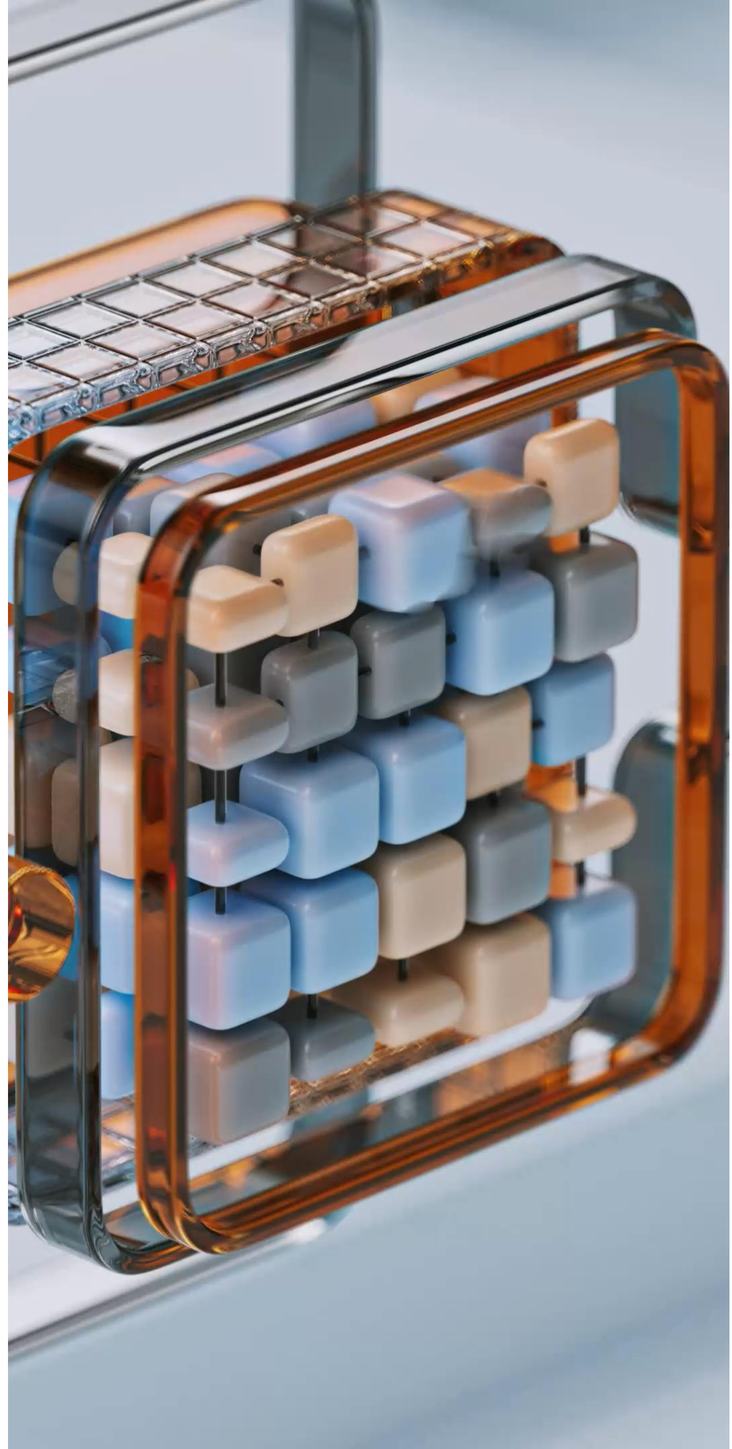
GPU execution !

Evolution

Replaces NLP encoder-decoder models

Shift from RNN LSTM to Transformers

Use **attention mechanism**

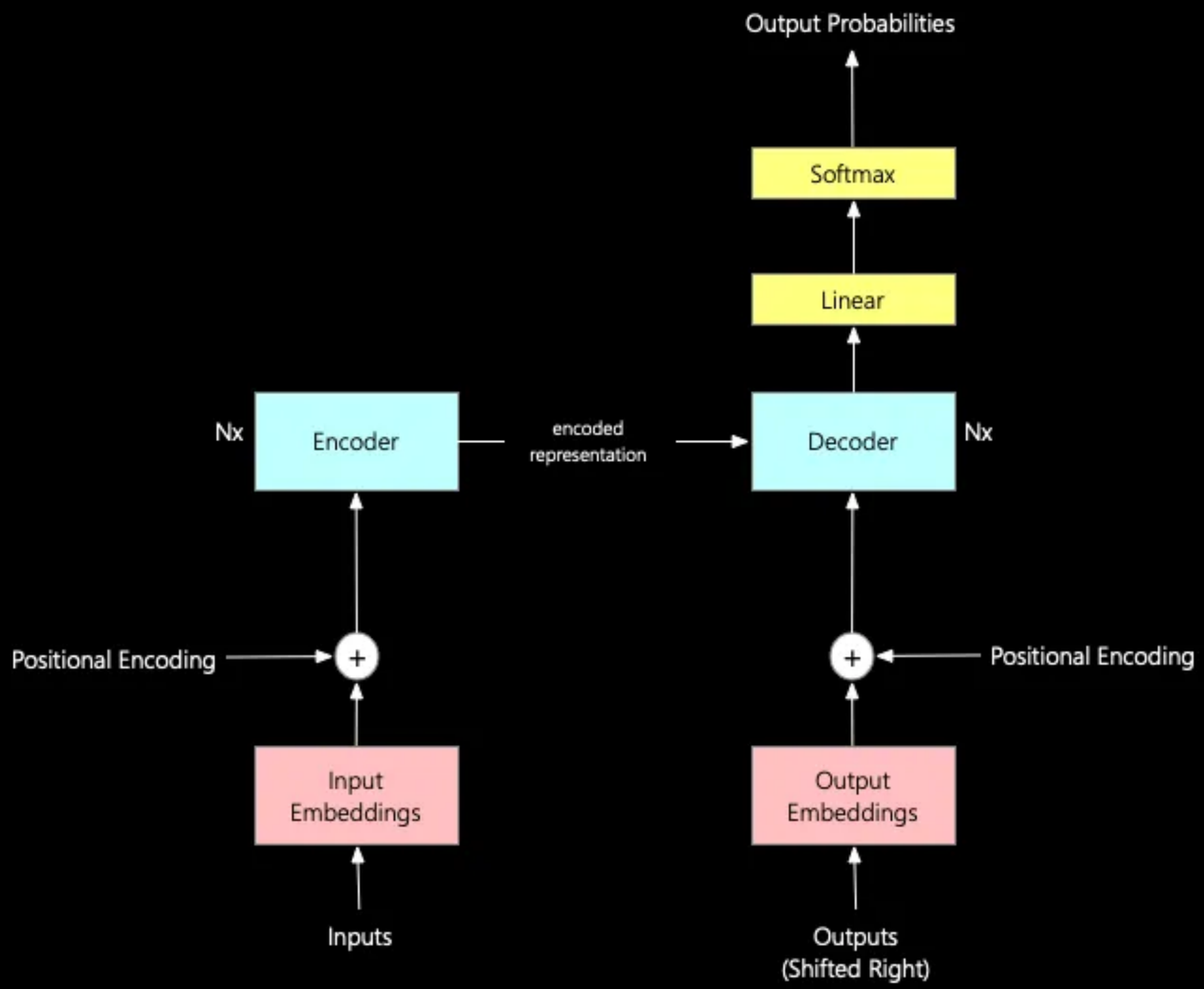


The Transformer Architecture

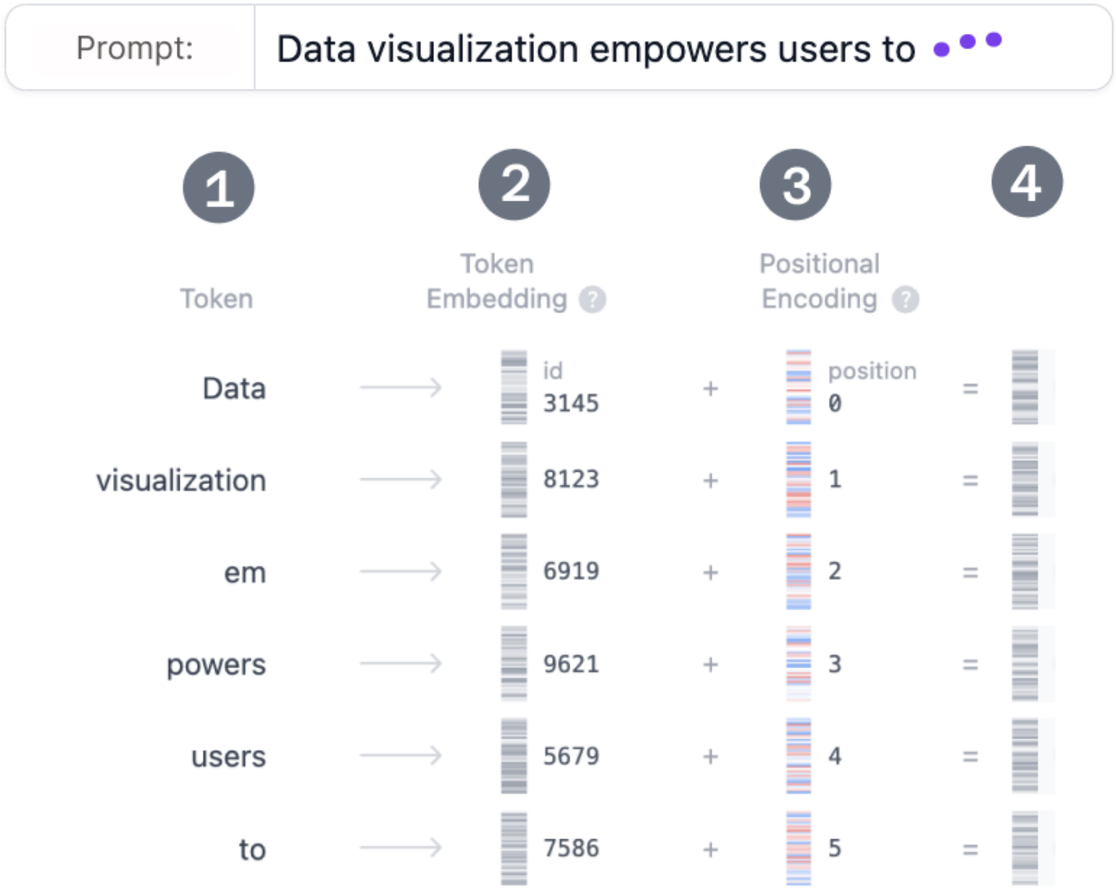
The power behind LLMs

An overview of transformers

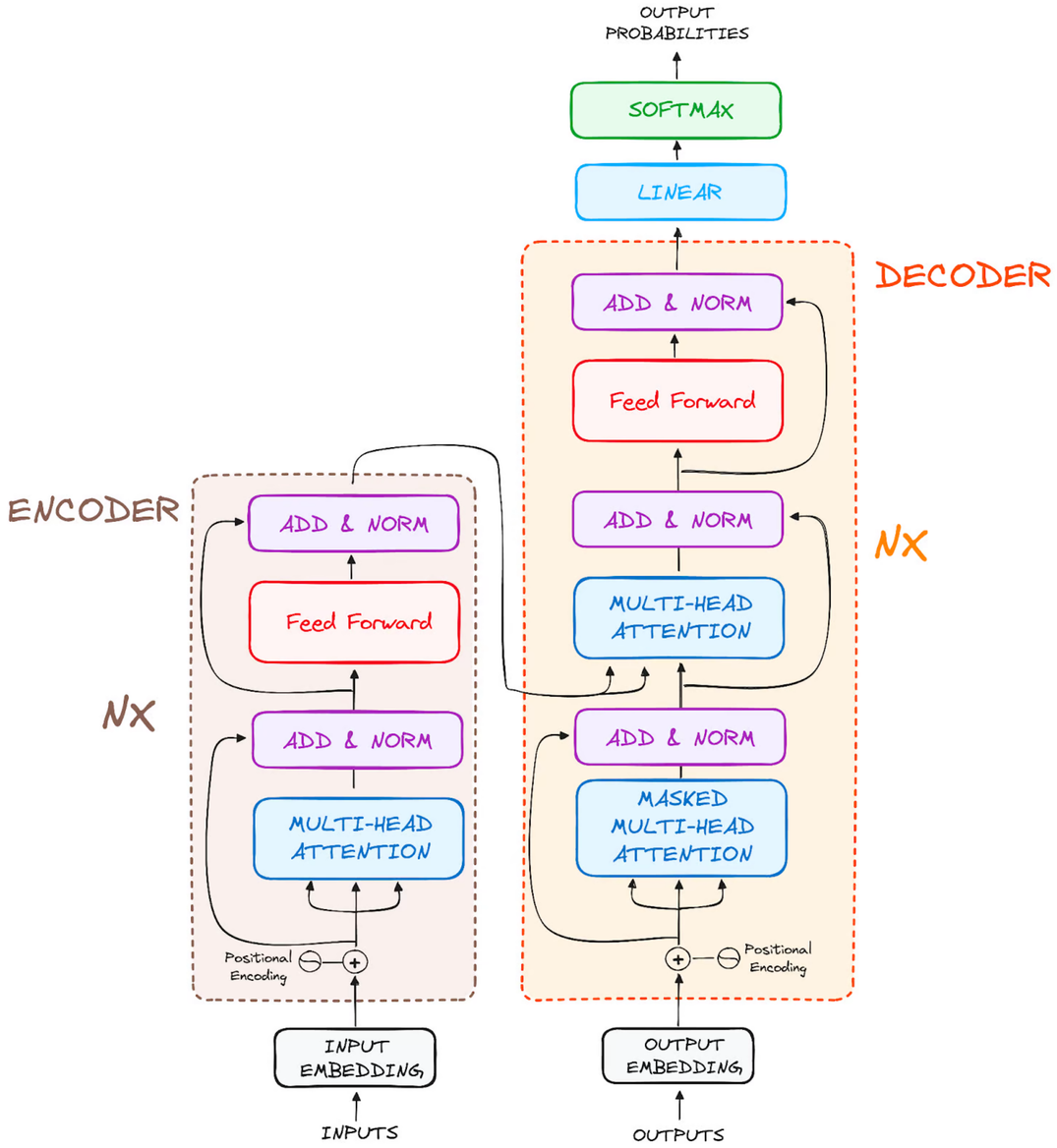
Under the Black box 🗃️



Embeddings 🔢


Prompt:
Data visualization empowers users to
1
2
3
4
Token
Token Embedding
Positional Encoding
Data
visualization
em
powers
users
to
id
3145
8123
6919
9621
5679
7586
position
0
1
2
3
4
5
+
=

Basic transformer architecture



Core problems

01

How to understand the position of words in sequences?

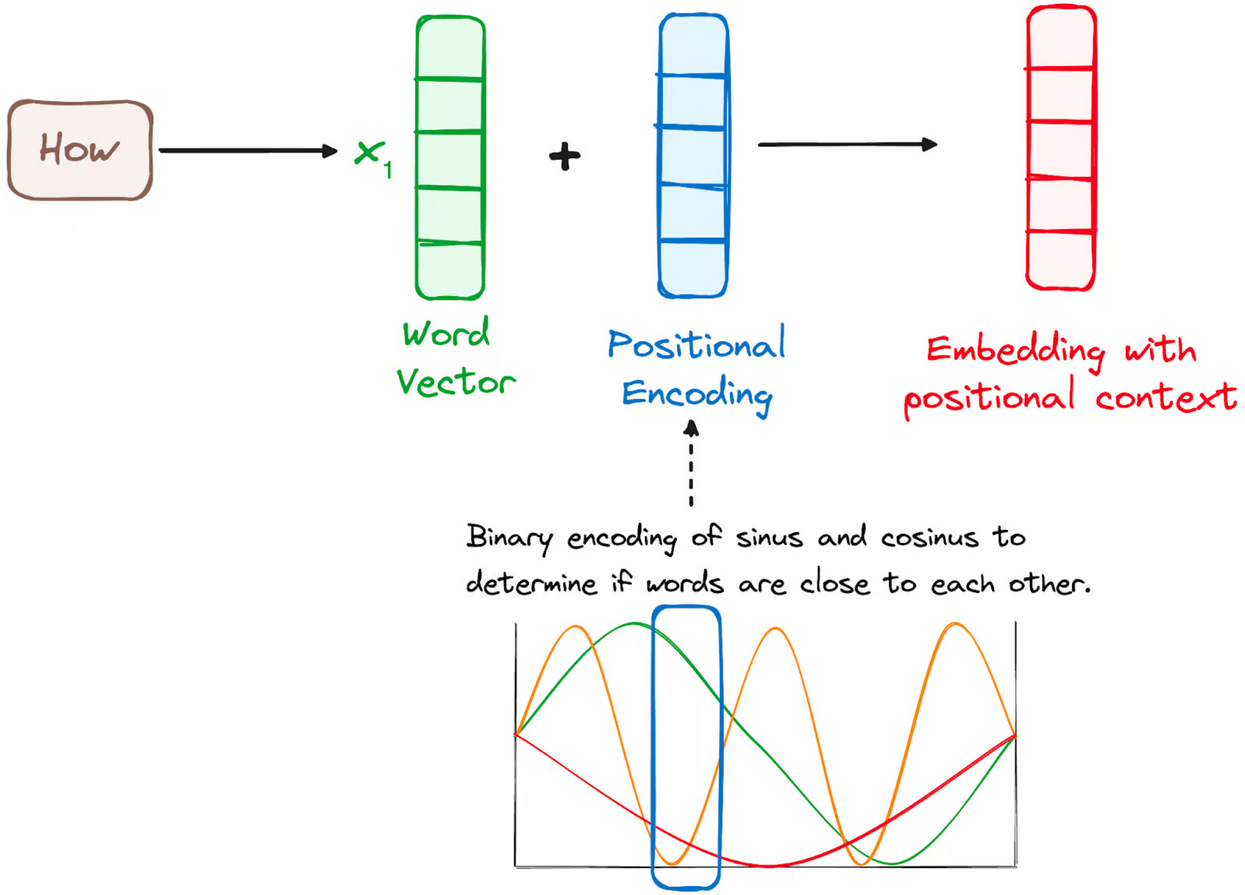
02

How do we handle the **context** of words?

02

How do we make the training process faster?

The way transformers address the position of each word



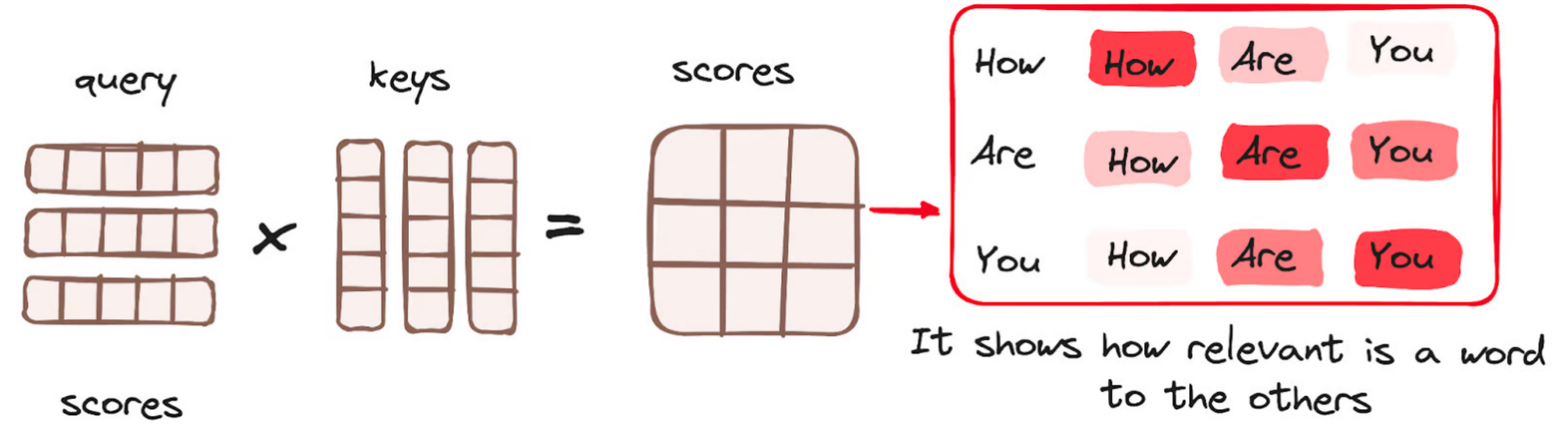
But how is context handled 🤯

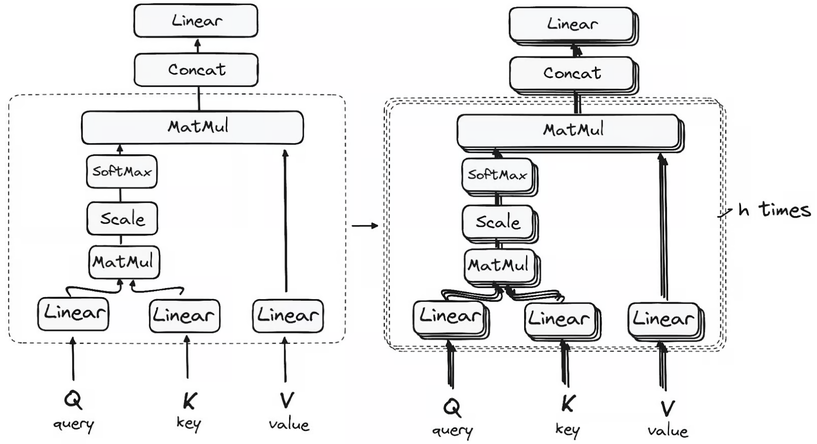
“Self-attention mechanism”

Enables each input word to relate to other words

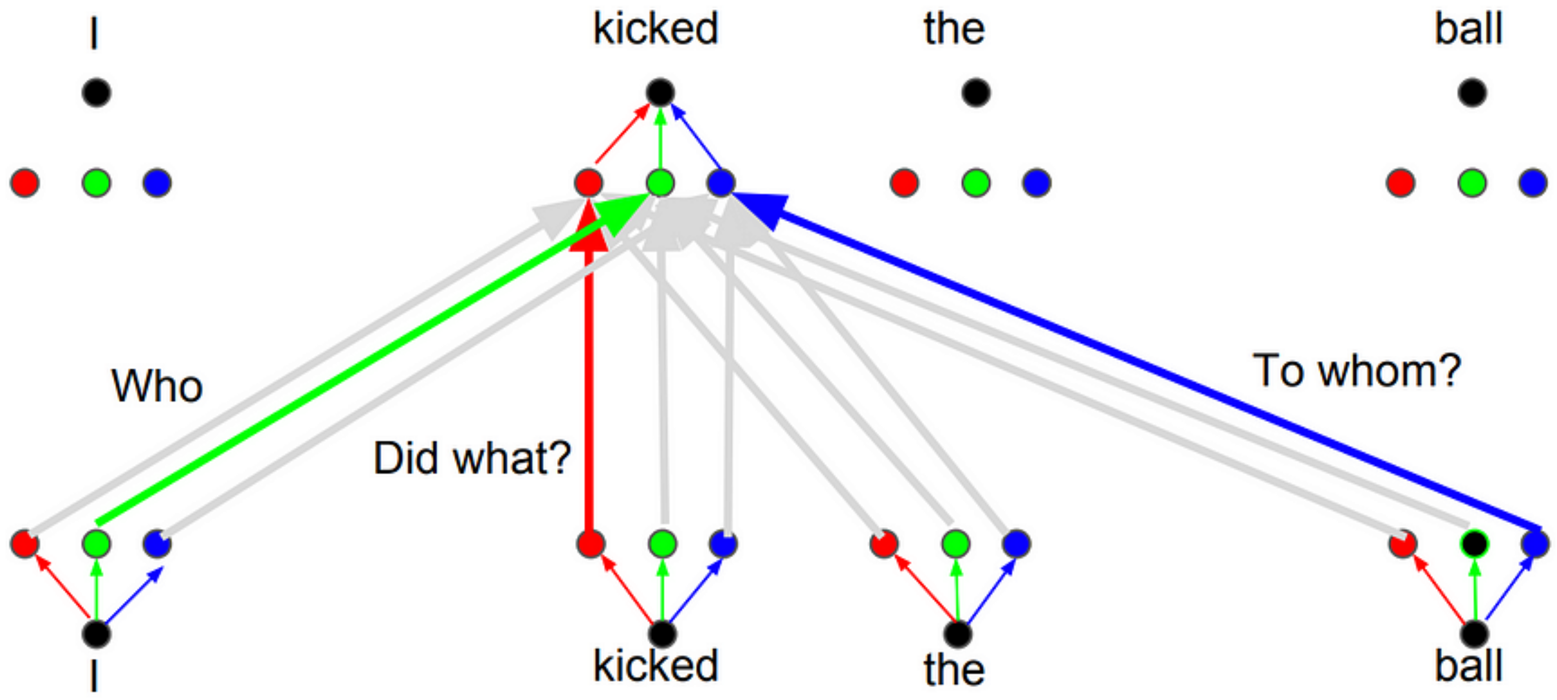
Only words with high scores are kept

In other words
=> CONTEXT 🔥





Attention process example



Can we make the training process any faster 💭?

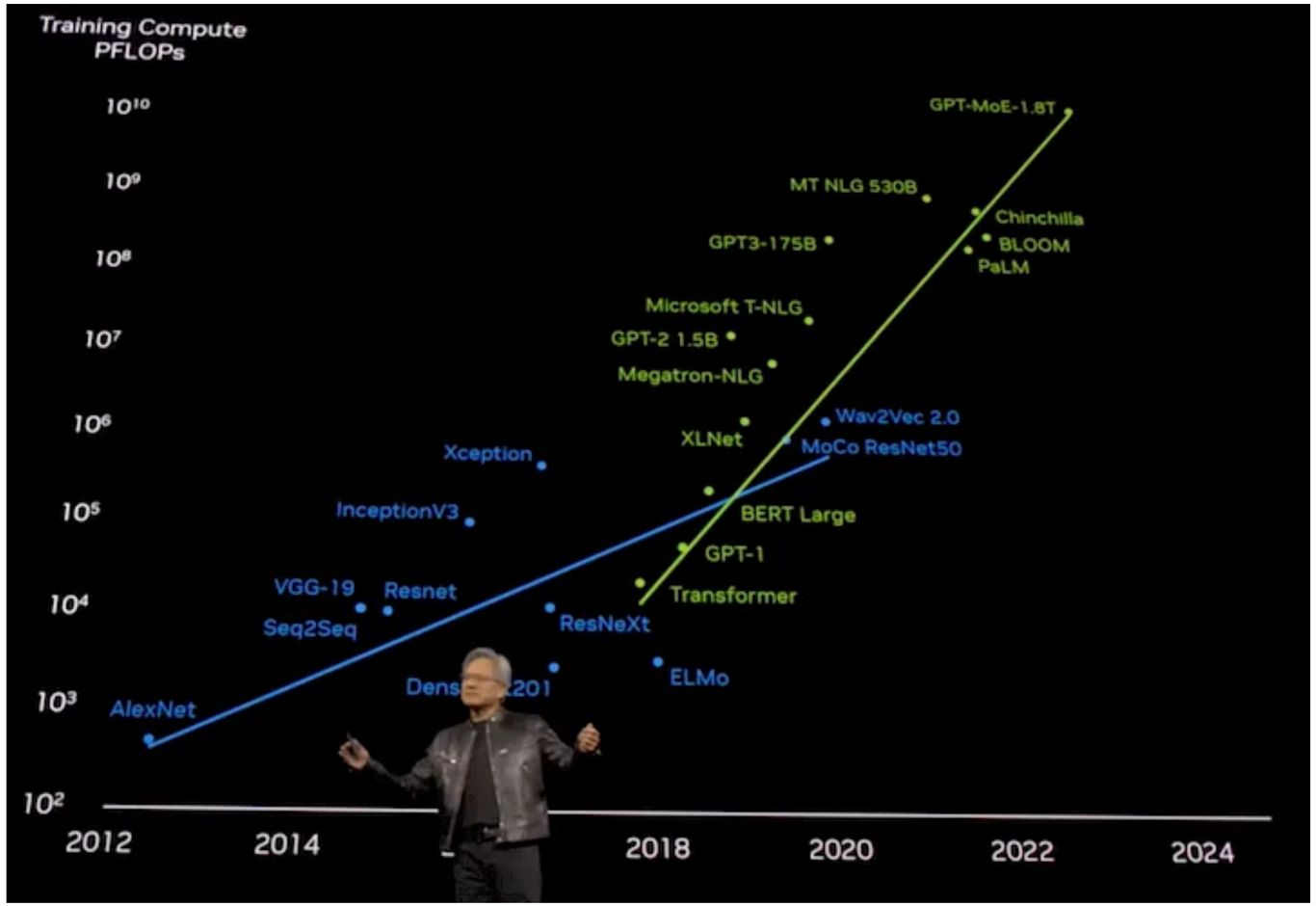


Table 2. H100 speedup over A100 (H100 Performance, TC=Tensor Core)

	A100	A100 Sparse	H100 SXM5	H100 SXM5 Sparse	H100 SXM5 Speedup vs A100
FP8 Tensor Core	NA	NA	1978.9 TFLOPS	3957.8 TFLOPS	6.3x vs A100 FP16 TC
FP16	78 TFLOPS	NA	133.8 TFLOPS	NA	1.7x
FP16 Tensor Core	312 TFLOPS	624 TFLOPS	989.4 TFLOPS	1978.9 TFLOPS	3.2x
BF16 Tensor Core	312 TFLOPS	624 TFLOPS	989.4 TFLOPS	1978.9 TFLOPS	3.2x
FP32	19.5 TFLOPS	NA	66.9 TFLOPS	NA	3.4x
TF32 Tensor Core	156 TFLOPS	312 TFLOPS	494.7 TFLOPS	989.4 TFLOPS	3.2x
FP64	9.7 TFLOPS	NA	33.5 TFLOPS	NA	3.5x
FP64 Tensor Core	19.5 TFLOPS	NA	66.9 TFLOPS	NA	3.4x
INT8 Tensor Core	624 TOPS	1248 TOPS	1978.9 TFLOPS	3957.8 TFLOPS	3.2x

SO,

What’s the key challenges in LLMs today 🤔

—

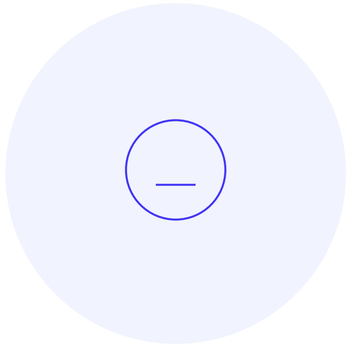
Limitations of Large Language Models

1/3

Hallucinations

2/3

Knowledge cut off



3/3

Resource intensive !

As developer, how can we address these problems?

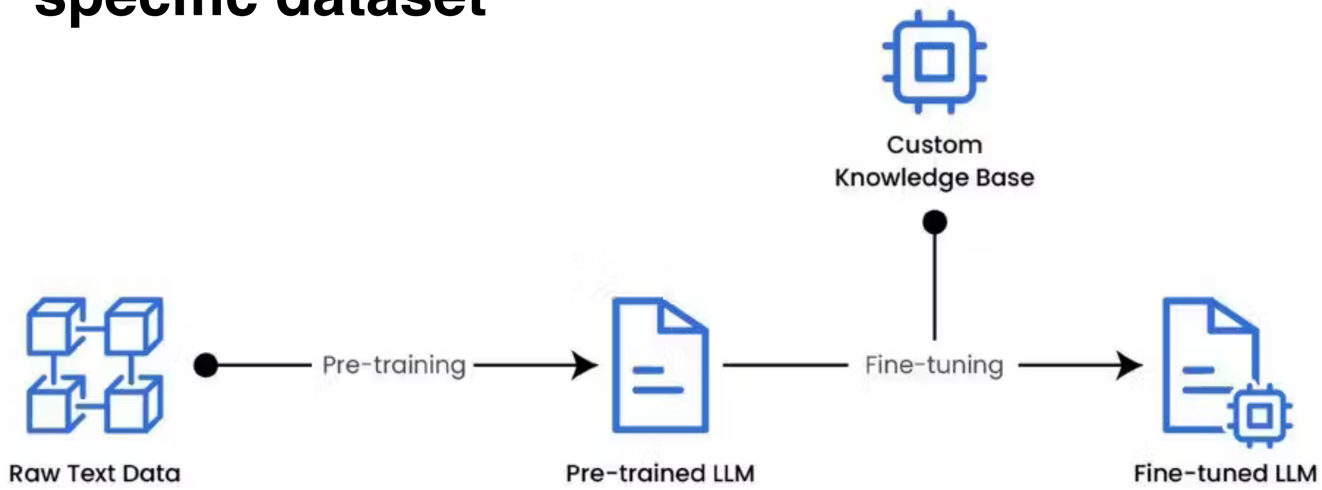
“Can I really train my own LLM ... 😔”

LLM
Fine tuning

What is LLM Fine-tuning?

- Use the existing knowledge embedded in the pre-trained model

- Specialise the model towards a specific task by further training it with a task-specific dataset



Why fine tune?

General models

Available pretrained on diverse data

Lack specialisation

Hallucinate with unknown queries

Fine tuned models

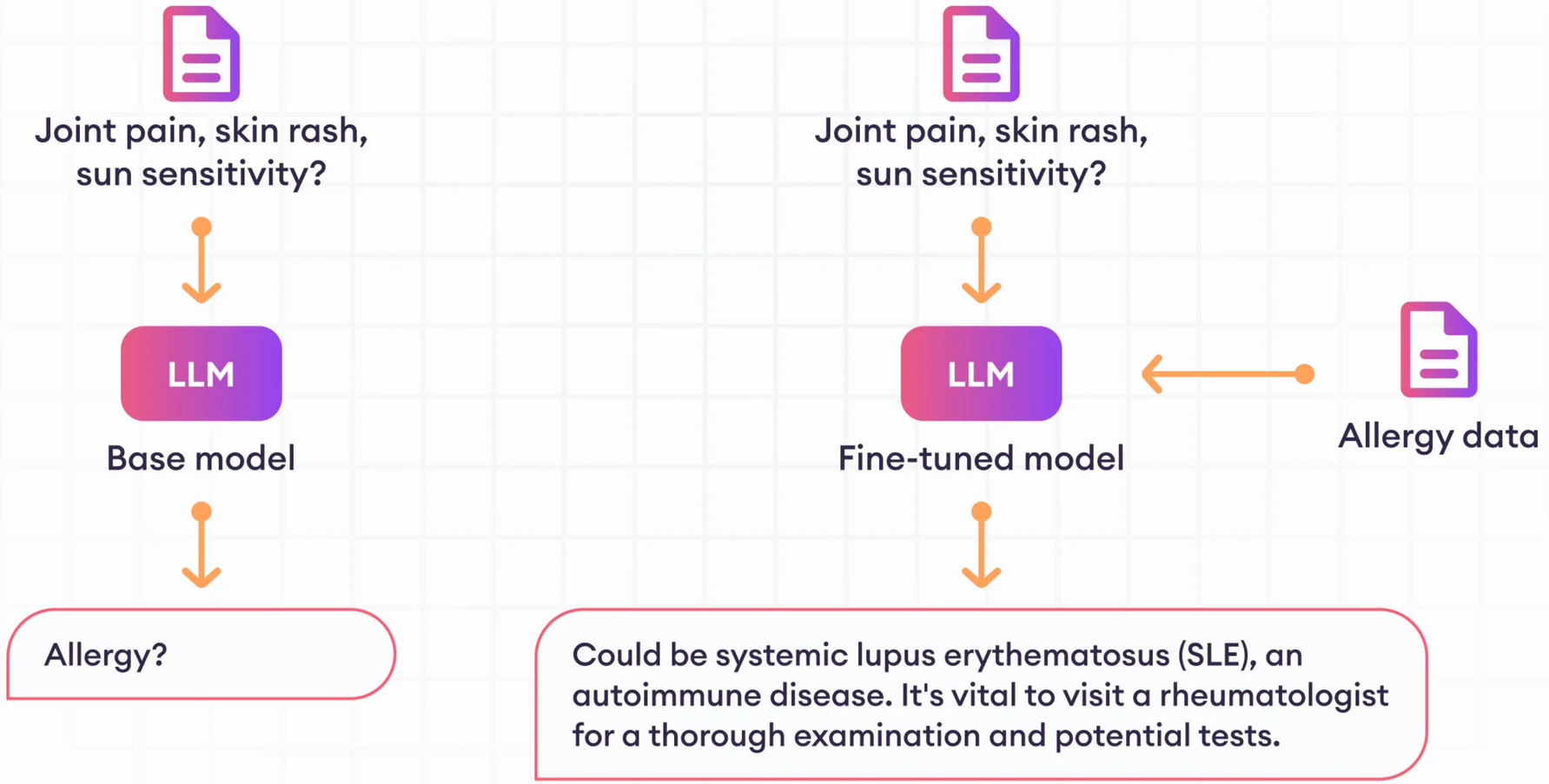
Improved **accuracy and relevance**

Domain trained

Mitigates hallucinations and knowledge cutoff

Builds **trust**

Example of LLM fine-tuning



Fine tuning approaches

Task specific
Full fine tuning

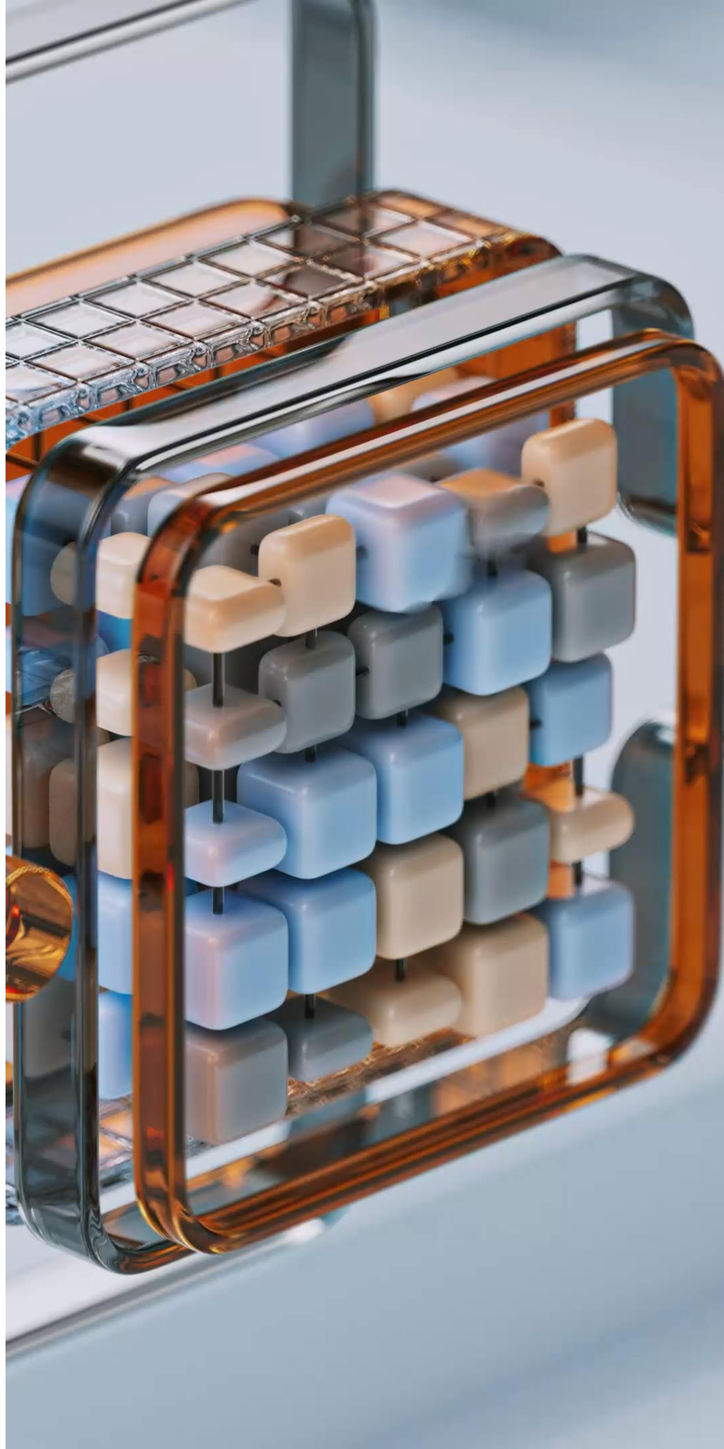
Parameter efficient fine tuning

Instruction fine tuning

Transfer learning

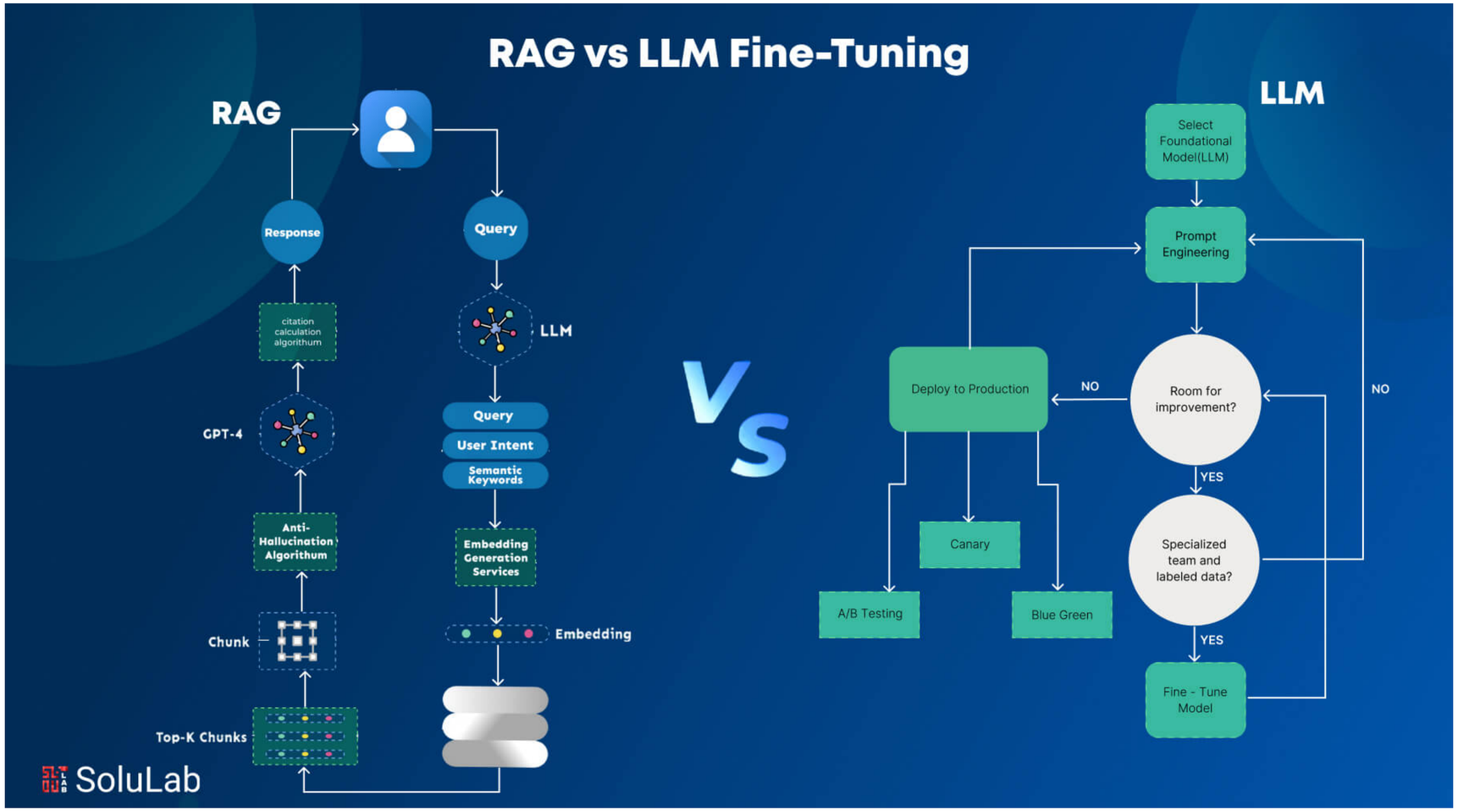
Multi-task learning

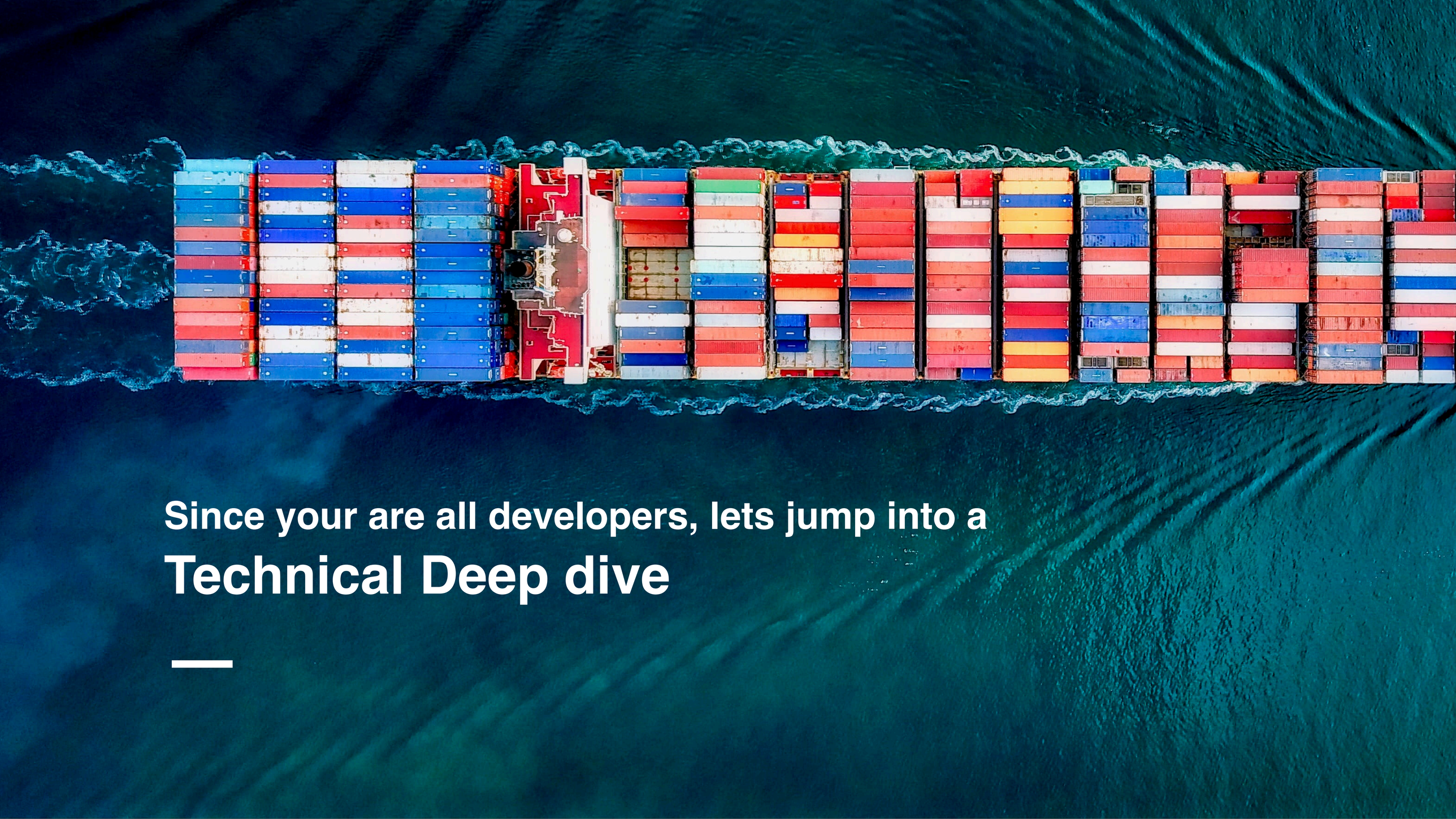
Sequential fine tuning



RAG vs Fine tuning

“training process”

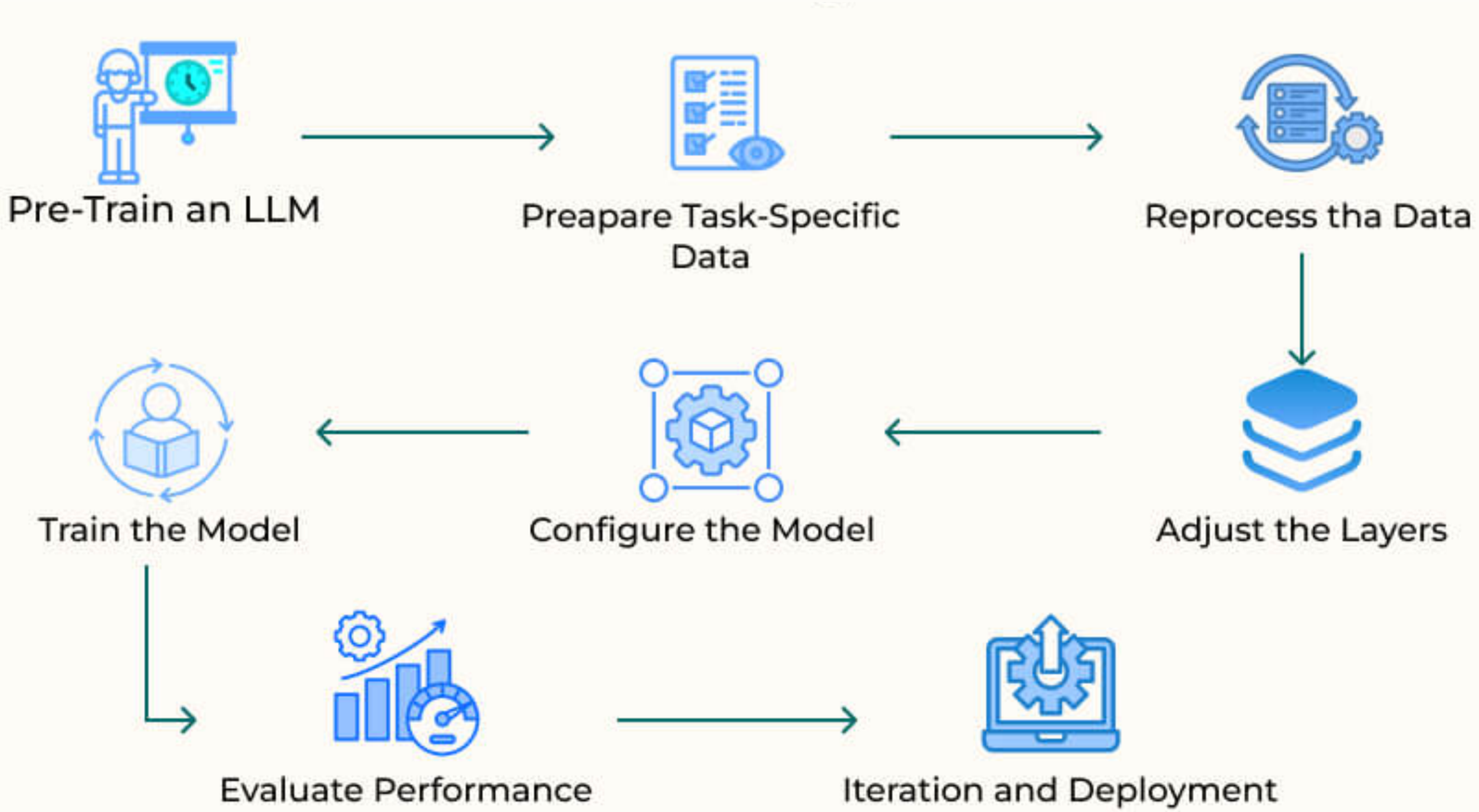




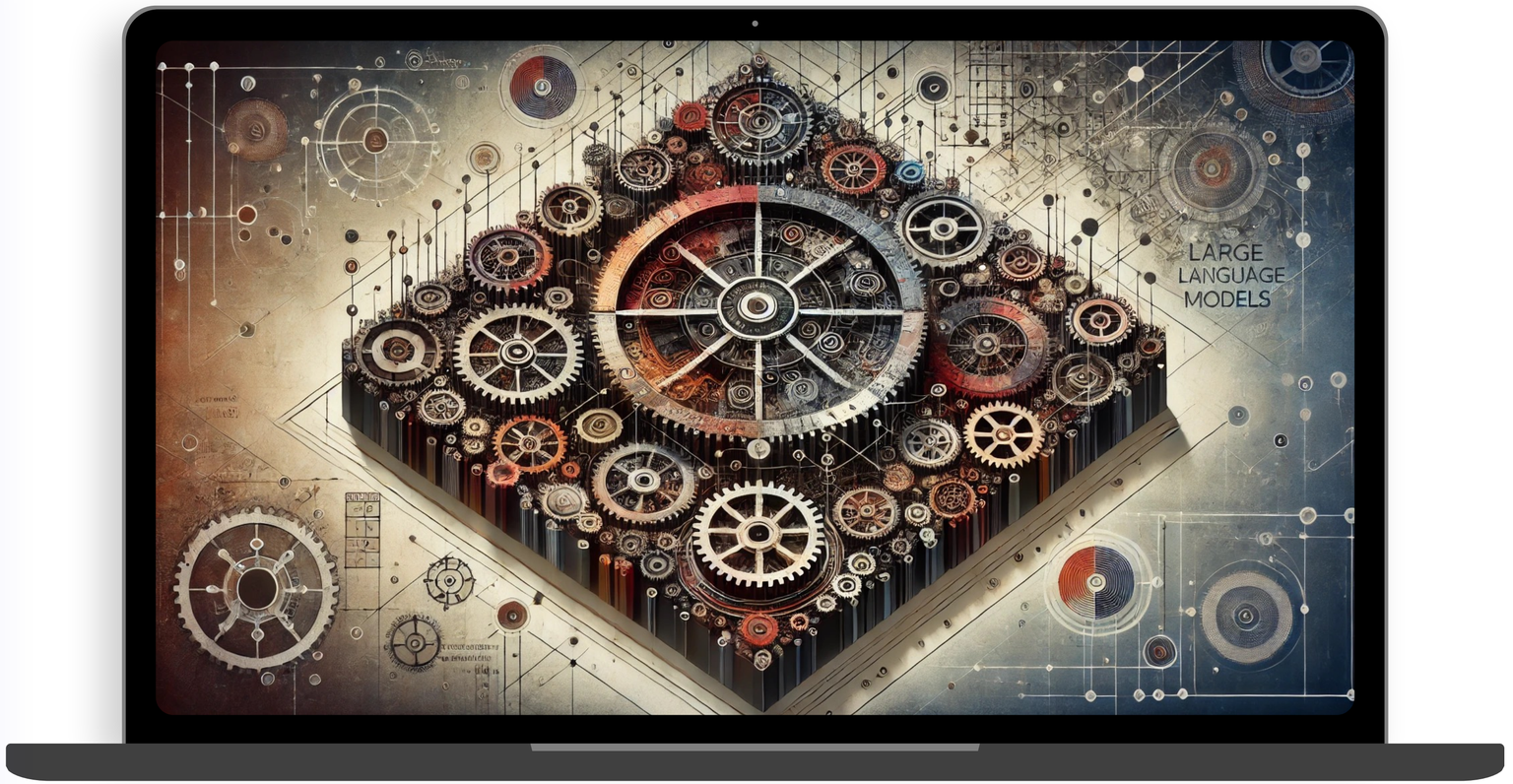
Since your are all developers, lets jump into a

Technical Deep dive

An overview of LLM fine tuning



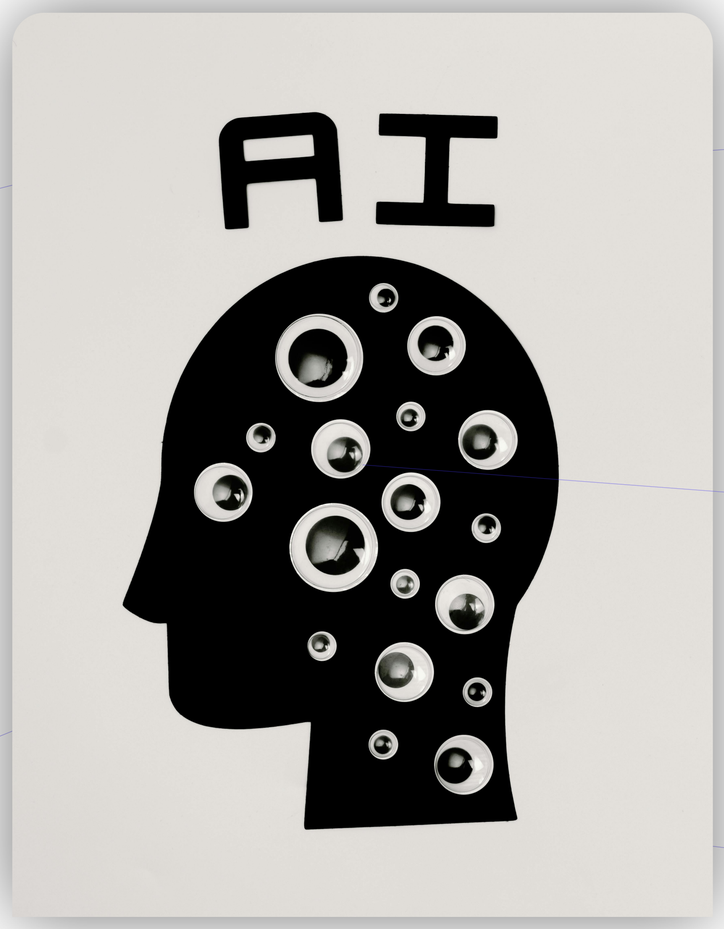
Let’s fine tune a pretrained model



What do we need?

01
LLM model

02
Custom dataset



03
GPU 👾

04
Fune tuning pipeline

05
Some coffee ☕

Tools and technologies

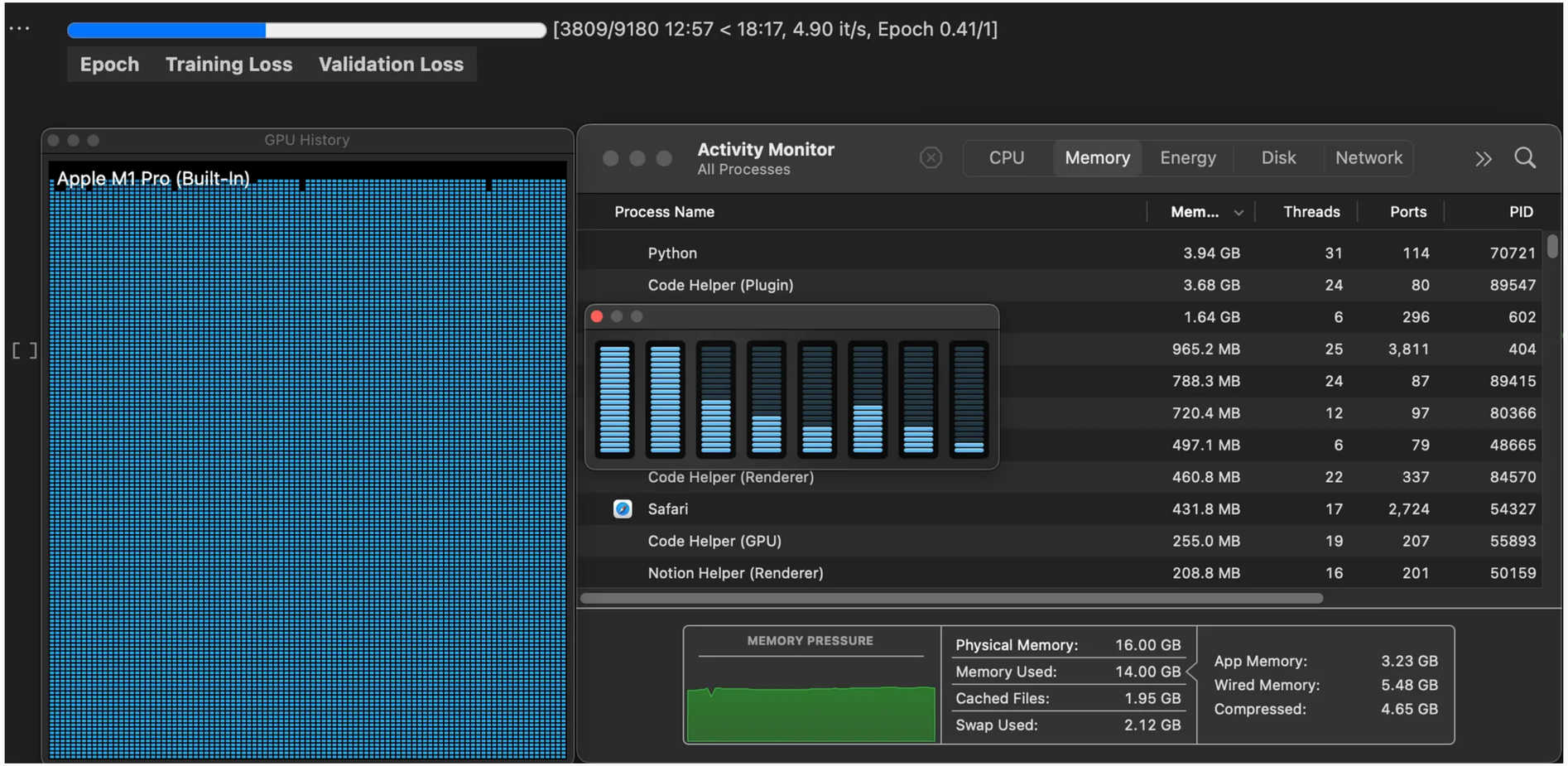
So, what did I use?

- Language: Python 💥
- Model: GPT2
- GPU:
  - Apple MPS (16 Gb Unified memory)
  - RTX 4060 Ti (8 Gb Ram)
- Training lib: Pytorch
- Dataset: Wikitext

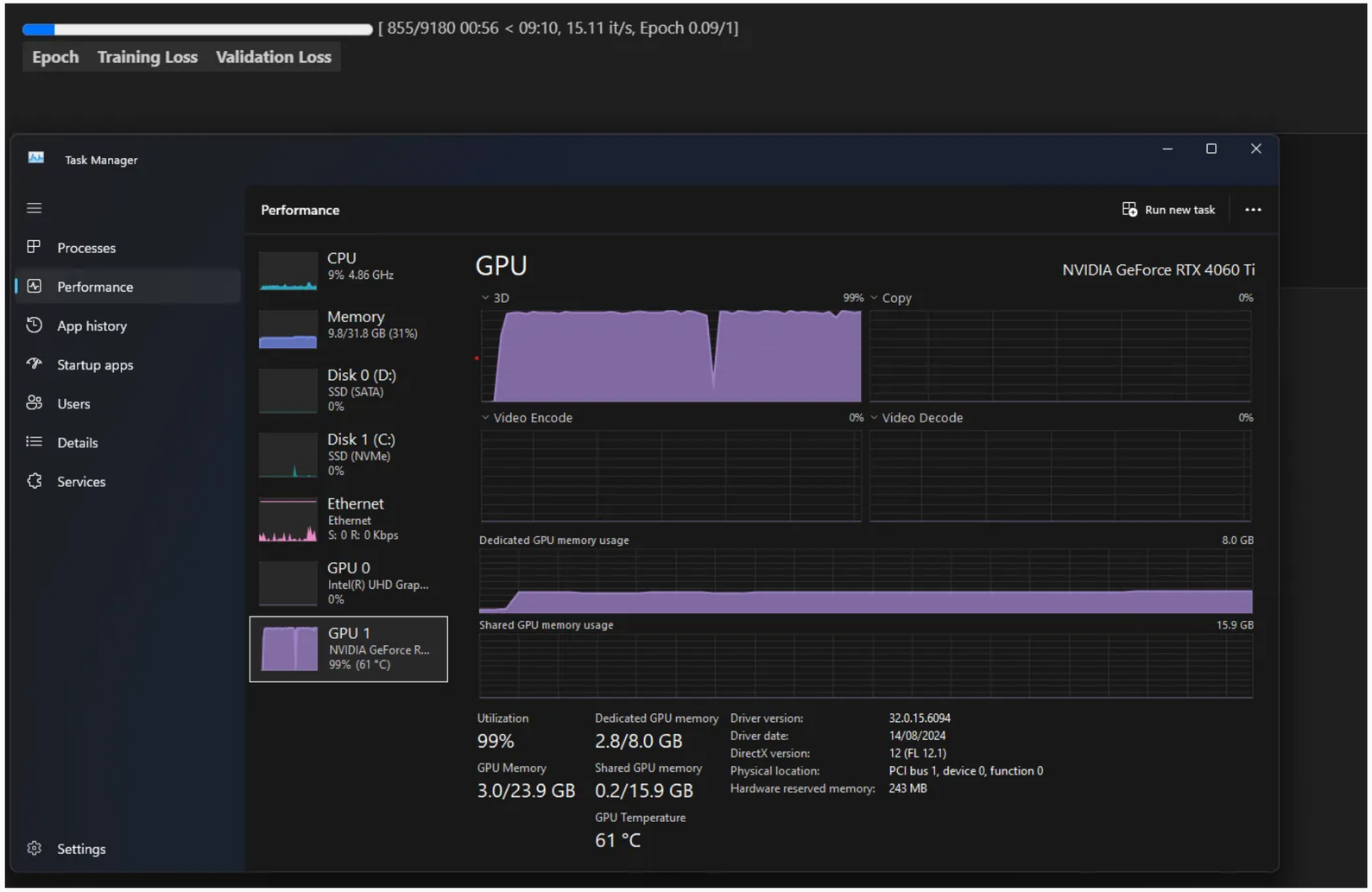
📝 GPT2 model was chosen due to small number of parameters (124 million params)

Lets look at some code code 👾

Use Case 1: Full LLM fine-tuning (MPS)



Use Case 1: Full LLM fine-tuning (CUDA)



Bad Results 😭

[9180/9180 31:08, Epoch 1/1]

Epoch	Training Loss	Validation Loss
1	1.379100	1.382495

```
TrainOutput(global_step=9180, training_loss=1.4348974506319998, metrics={'train_runtime': 1876.0005, 'train_samples_per_second':
```

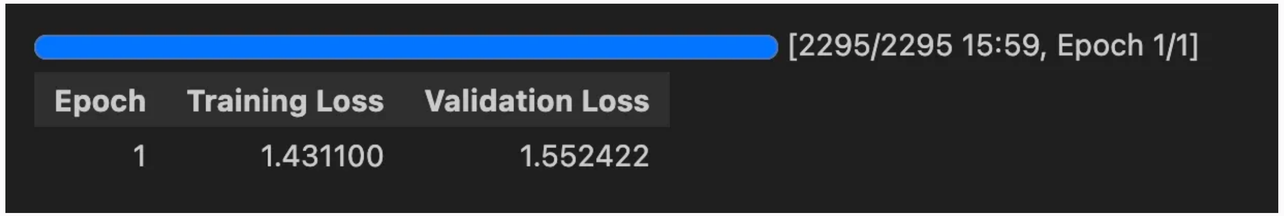
```
generate_text("Once upon a time")
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Total number of paramerers: 81912576
Once upon a time
```

Use Case 2: Parameter Efficient fine-tuning

Introduced handicaps 💀

- **Smaller distilgpt2 model**
- **1 epoch only**



Trained parameters: 147,456

```
 Generated:
Once upon a time of the moon, an image of a meteor was taken, and the comet was seen as being in its final resting place. However
```

Did it really work 😮

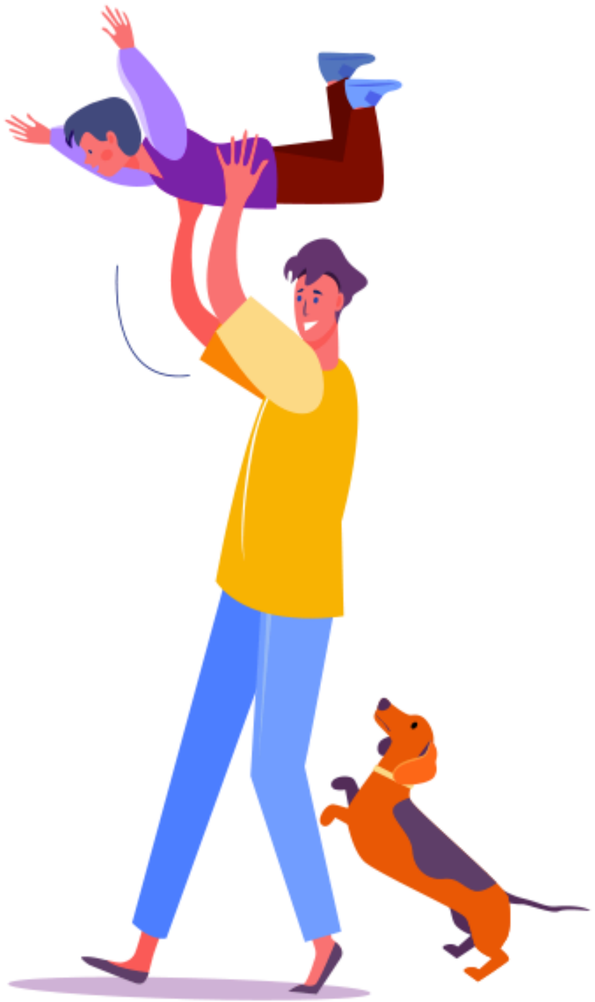
```
📦 Base model output:
== Early Life ==
John Keats was born in Chicago, Illinois, in 1842. He moved to Chicago from St. Petersburg, Florida, to Chicago in 1844. Upon arr
Keats was born in Richmond, Virginia
🔧 LoRA fine-tuned model output:
== Early Life ==
John Keats was born in London in 1794 in Yorkshire and was raised in Edinburgh, and attended the school of James. He was educated
```

The base model is completely wrong here 😂

However the fined-tuned model is correct!

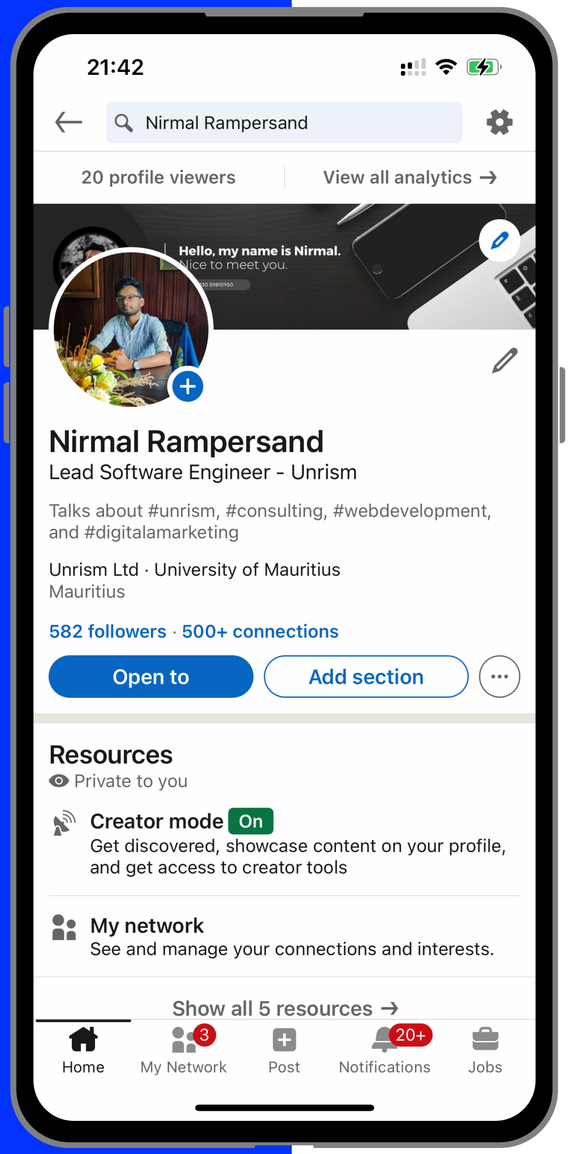
SUCCESS :)

Not only can we now run local LLM’s, but we can also **fine tune them to make them accurate in task-specific environments**!



Some real world applications of fine tuning

- Customer support
- Academic research
- Healthcare and diagnostics
- Documentation and development
- Legal document analysis
- Financial analysis



Thank you for your attention!

Nirmal Rampersand

+230 59810950

nirkramp@gmail.com