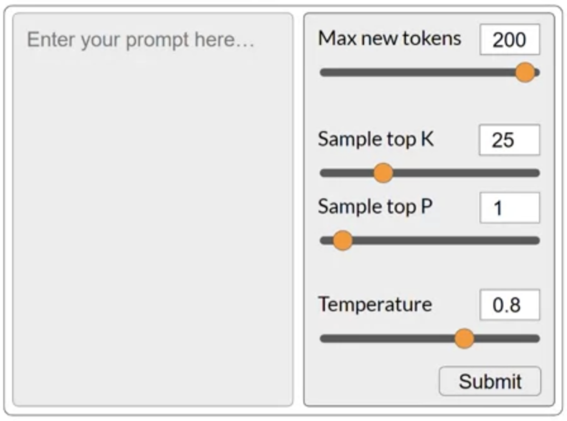
# Model Pretraining :

## what is Generative Configuration:

Generative configuration - inference parameters



Max new tokens 200

Sample top K 25

Sample top P 1

Temperature 0.8

Maximum number of words in response

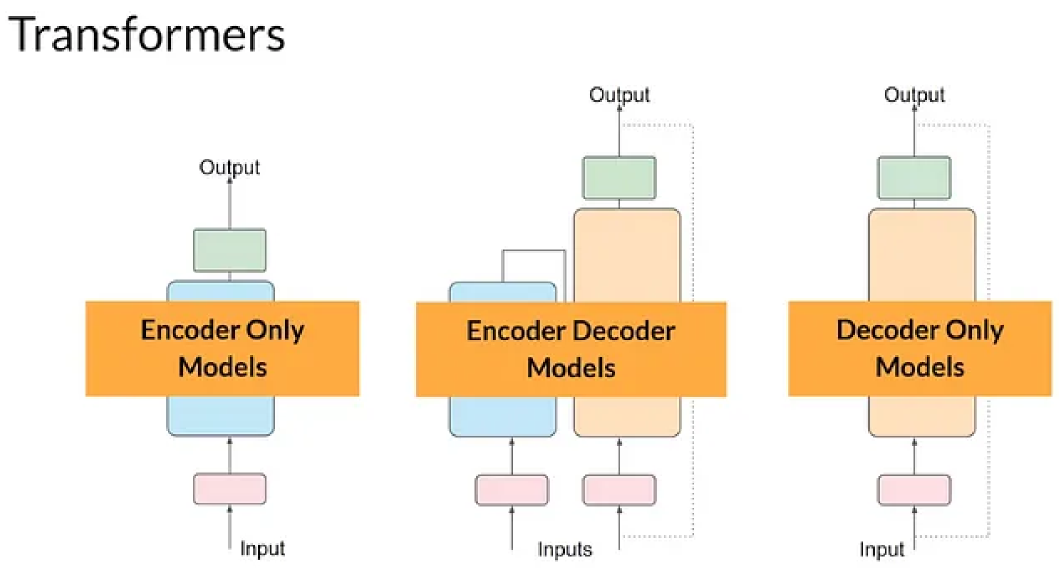
Limits the model to the K most probable tokens, controlling diversity and focus.

Limits token selection to a cumulative probability threshold, dynamically balancing diversity and focus.

Adjusts the randomness of token selection, influencing the creativity and variability of the output.

## How you decide which Model you have to choose for you work ?

## Differnece between Auto regressive Models and Auto encoding Models



## Autoencoding Models: Encoder-Only Models

Autoencoding models, also known as encoder-only models, are pre-trained using **masked language modeling.** In this approach, tokens in the input sequence are randomly masked, and the model's objective is to predict the masked tokens to reconstruct the original sentence.

## Autoregressive Models: Decoder-Only Models

Autoregressive models, or decoder-only models, are pre-trained using causal language modeling. The objective is to predict the next token based on the previous sequence of tokens. These models mask the input sequence and can only see the input tokens leading up to the token in question.

## Sequence-to-Sequence Models: Encoder-Decoder Models

Sequence-to-sequence models utilize both the encoder and decoder components of the original transformer architecture. The pre-training objective for these models varies depending on the specific model.

Sequence-to-sequence models are commonly used for **translation, summarization, a**nd question-answering tasks**. BART** is another well-known **encoder-decoder mode**l.

## What Matter the Most :

**Model Size Matters**

**Training Dataset Size**

***increase the compute power and the time you are going to train the model.***

> A. Give your model more horse power — increase the compute power and the time you are going to train the model
>
> *B. Give your model more muscles — increase the model size or the model parameters*
>
> *C. Give your model more training material — increase the size of the dataset*

The paper **Scaling Laws for Neural Language Models** shows that increasing model size, dataset size, and training compute all independently enhance performance.

ChatGPT (175B parameters), Jurassic (178B parameters), or the massive Megatron-Turing NLG (530B parameters).

**B : *increase the model size or the model parameters***

DeepMind paper **Training Compute-Optimal Large Language Models** (also known as the **Chinchilla paper)**, the authors suggest that current big models might be over-parameterised and under-trained in terms of dataset size.

**cost issues:**

- *Inference cost — the cost of calling an LLM to generate a response*
- *Tuning cost — the cost of tuning an LLM to drive tailored pre-trained model responses*
- *Pre-training cost — the cost of training a new LLM from scratch*
- *Hosting cost — the cost of deploying and maintaining a model behind an API, supporting inference or tuning*

these choices are influenced by the available compute budget, encompassing factors such as hardware limitations, training time constraints, and financial considerations.

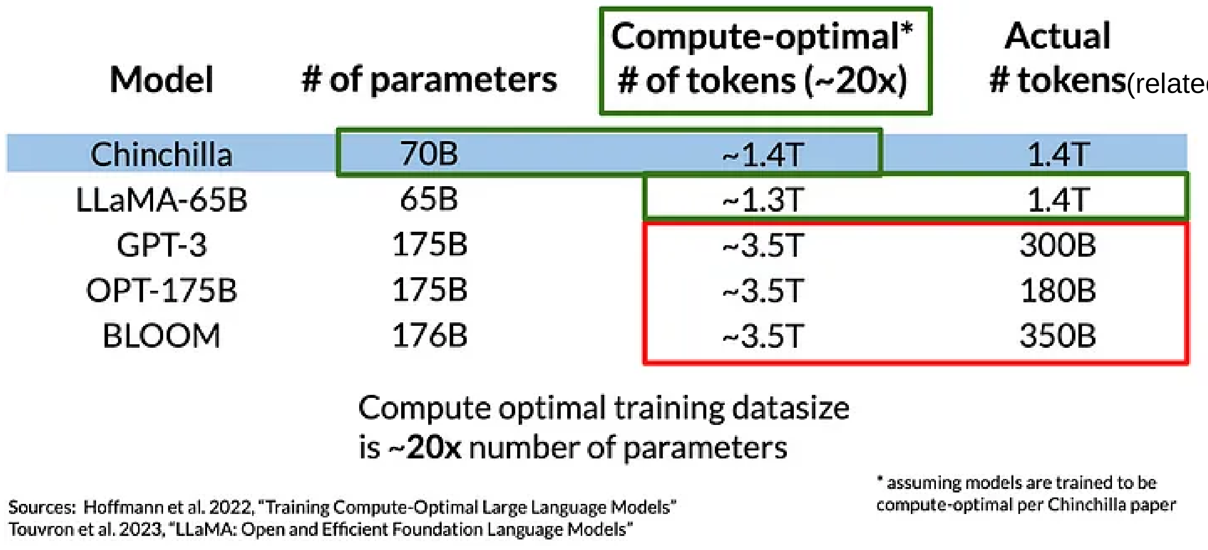
***C. Increase the size of the dataset:***

**The authors show that the optimal training dataset size for a given model is about**
**20 times larger**

**than the number of parameters of the model.**

The Chinchilla model trained by Deepmind was trained optimally and the size of the dataset was **1.4T** and the number of parameters is **70B**. Llama-65B also follows a similar pattern, in contrast to GPT-3 or BLOOM models highlighted in red in the picture below

This column indicates the
optimal number of tokens
model should be trained on
to make the most efficient
use of computational resources



| Model | # of parameters | Compute-optimal* # of tokens (~20x) | Actual # tokens(related to NLP) |
|---|---|---|---|
| Chinchilla | 70B | ~1.4T | 1.4T |
| LLaMA-65B | 65B | ~1.3T | 1.4T |
| GPT-3 | 175B | ~3.5T | 300B |
| OPT-175B | 175B | ~3.5T | 180B |
| BLOOM | 176B | ~3.5T | 350B |

Compute optimal training datasize
is ~**20x** number of parameters

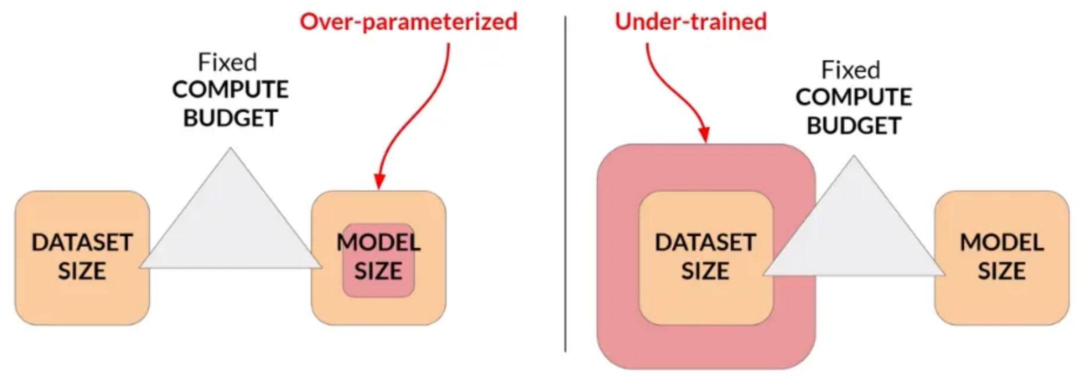
* assuming models are trained to be
compute-optimal per Chinchilla paper

Sources: Hoffmann et al. 2022, "Training Compute-Optimal Large Language Models"
Touvron et al. 2023, "LLaMA: Open and Efficient Foundation Language Models"

## Optimal Models:

### Compute optimal models

- Very large models may be **over-parameterized** and **under-trained**
- Smaller models trained on more data could perform as well as large models



## Why choose a Small Language Model?

### Benefits and shortcomings of Small Language Models

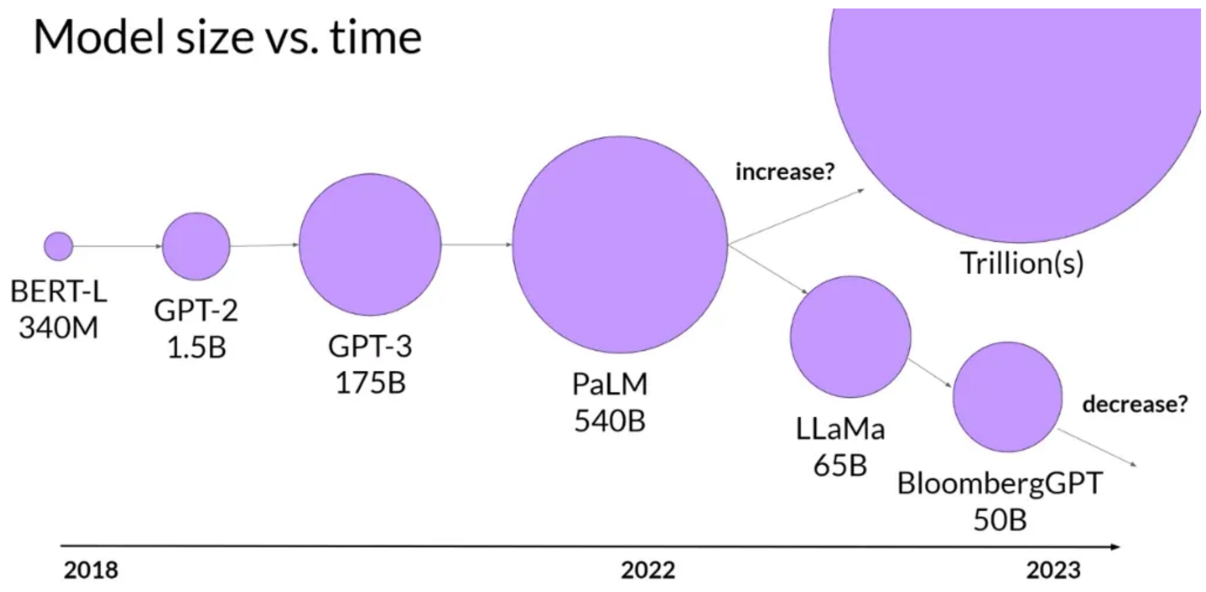
### Benefits

- **Efficient —** SLMs are more nimble and require **less computational power** which makes them more **efficient to deploy** in production
- **Cost-effective —** Less parameters means that you need **less resources to train**, maintain and run an SLM compared to an LLM
- **Specialised —** SLMs can be trained o**n high quality datasets for specific domain tasks**. This often leads to better performance within that niche
- **Explainable —** Because these are **less complex** and use more targeted data can offer more transparency into their outputs. Explainability is valued in most Enterprises especially in sensitive applications

### Shortcomings

- **Task-limited —** Due to their specialised nature SLMs might struggle to perform as well on tasks outside their **training domain.** They lack the breadth of knowledge that LLMs possess
- **Performance-limited —** SLMs have a lower capacity for learning and **understanding complex language patterns compared to larger models.** This can lead to limitations in the types of tasks they can handle effectively
- **Dataset-dependent —** Smaller but less curated datasets can lead to less robust models as the performance of SLMs relies heavily on the quality and relevance of the data they are trained on

## Why Customizing Language Models for Specialized Domains:



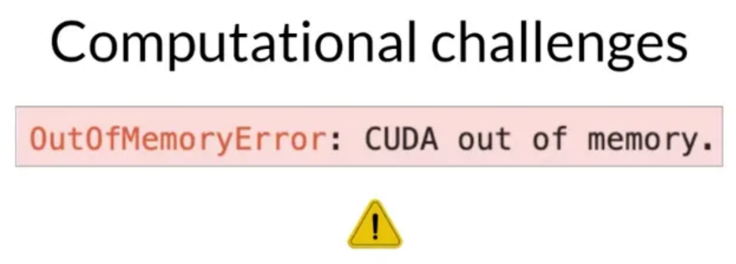
## Understanding Domain Adaptation

Certain domains, such as law, medicine, finance, and science, possess their own vocabulary and unique language structures. Common terms and phrases in these domains may be unfamiliar outside their respective fields.

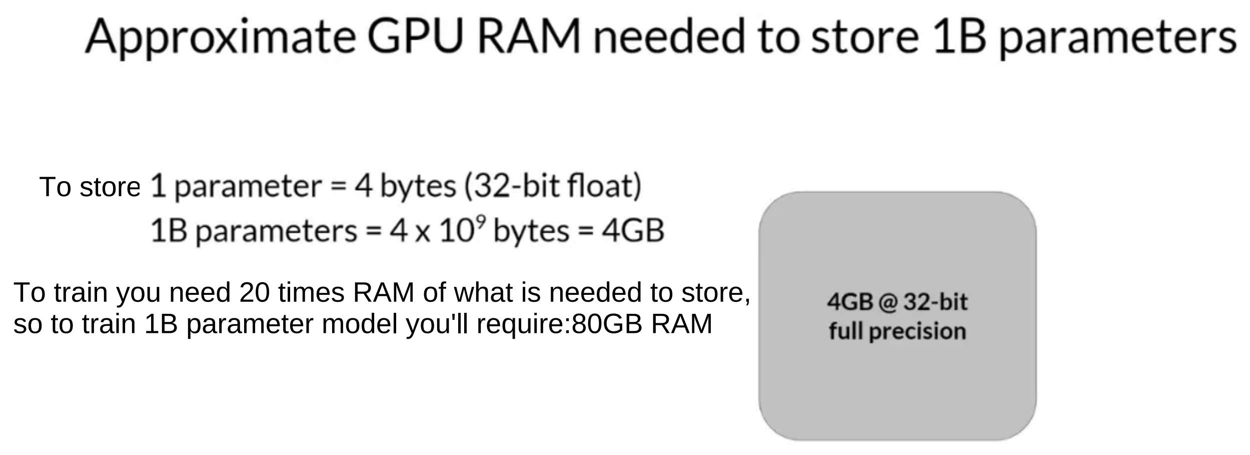
## BloombergGPT

A Finance-Focused LLM: BloombergGPT serves as a prime example of a specialized LLM in the financial domain. Developed by Bloomberg researchers, this model combines finance-specific data with general-purpose text during pretraining. By maintaining a balance between finance and public data (51% financial data and 49% public data), BloombergGPT achieves superior results on financial benchmarks while still demonstrating competitive performance on general-purpose LLM benchmarks.

## What is Quantization?


Computational challenges

```
OutOfMemoryError: CUDA out of memory.
```


Approximate GPU RAM needed to store 1B parameters

To store 1 parameter = 4 bytes (32-bit float)

1B parameters = 4 x $10^9$ bytes = 4GB

To train you need 20 times RAM of what is needed to store, so to train 1B parameter model you'll require:80GB RAM

4GB @ 32-bit
full precision

**Quantization** involves reducing the memory required to store model weights by decreasing their precision. Instead of the default 32-bit floating-point numbers (FP32) used to represent parameters, quantization employs 16-bit floating-point numbers (FP16) or even 8-bit integers (INT8). This reduction in precision helps optimize the memory footprint of the models.

## Quantization Process and Memory Savings

Quantization statistically projects the original 32-bit floating-point numbers into lower-precision spaces, utilizing scaling factors derived from the range of the original numbers. For instance, if a model with one billion parameters requires approximately 80 gigabytes of GPU RAM at full 32-bit precision, quantization can yield significant memory savings.

By employing 16-bit half precision (FP16), the memory requirement can be reduced by 50%, resulting in only 40 gigabytes of GPU RAM. Furthermore, representing the model parameters as 8-bit integers (INT8) can reduce the memory footprint even further to just one gigabyte, representing a total 98.75% reduction compared to full 32-bit precision.

**BFLOAT16** (BF16) has emerged as a widely adopted precision format in deep learning. Developed by Google Brain, BF16 serves as a hybrid between FP16 and FP32, capturing the full dynamic range of FP32 with just 16 bits.

**1.reduced  require memory for to store and train  models**

2. l**ower precision spaces**
3. **QAT qunatization aware training**
4. **BFIOAT16**

**THINK:**

500B param : 32 bit ?

## How Much Gpu RAM Needed to train 500B parameter model ?

## what are  the Scaling techniques for Model Training with Multiple GPUs

**Improving Efficiency and Performance**

**Two popular techniques**

**1.** Distributed Data-Parallel (DDP)

2. Fully Sharded Data Parallel (FSDP)

**DDP** is a widely used model replication technique that distributes large datasets across multiple GPUs, enabling parallel processing of batches of data. With DDP, each GPU receives a copy of the model and processes data independently. Afterward, a synchronization step combines the results, updating the identical model on each GPU. **DDP is suitable when the model and its additional parameters fit onto a single GPU, resulting in faster training.**

### large to fit in the memory of a single GPU.

**Sharding** involves **splitting and distributing one logical data set across multiple databases that share nothing and can be deployed across multiple servers**.

**Fully Sharded Data Parallel (FSDP):**

**FSDP,** inspired by th**e ZeRO technique,** provides a solution when the model is too large to fit in the memory of a single GPU. ZeRO (Zero Redundancy Optimizer) aims to optimize memory usage by distributing or sharding model parameters, gradients, and optimizer states across GPUs. FSDP applies sharding strategies specified in ZeRO to distribute these components across GPU nodes. This enables working with models that would otherwise exceed the capacity of a single chip.

## Memory Optimization with ZeRO:

ZeRO offers three optimization stages:

- Stage 1 shards only optimizer states, reducing memory usage by up to a factor of four.
- Stage 2 shards gradients, further reducing memory usage by up to eight times when combined with Stage 1.
- Stage 3 shards all components, including model parameters, with memory reduction scaling linearly with the number of GPUs.