"Generative AI & LLM"

* Prompt → Model (LLM) → Completion.

* Text Generations-

↳ RNN's where used before as model to generate next word in sequence.

→ Problem of RNN was that it could see only one word or some previous word to predict next word, resulting wrong ans.

* Transformer Architecture:-

↳ Tokenization.

↳ Vector Creation.

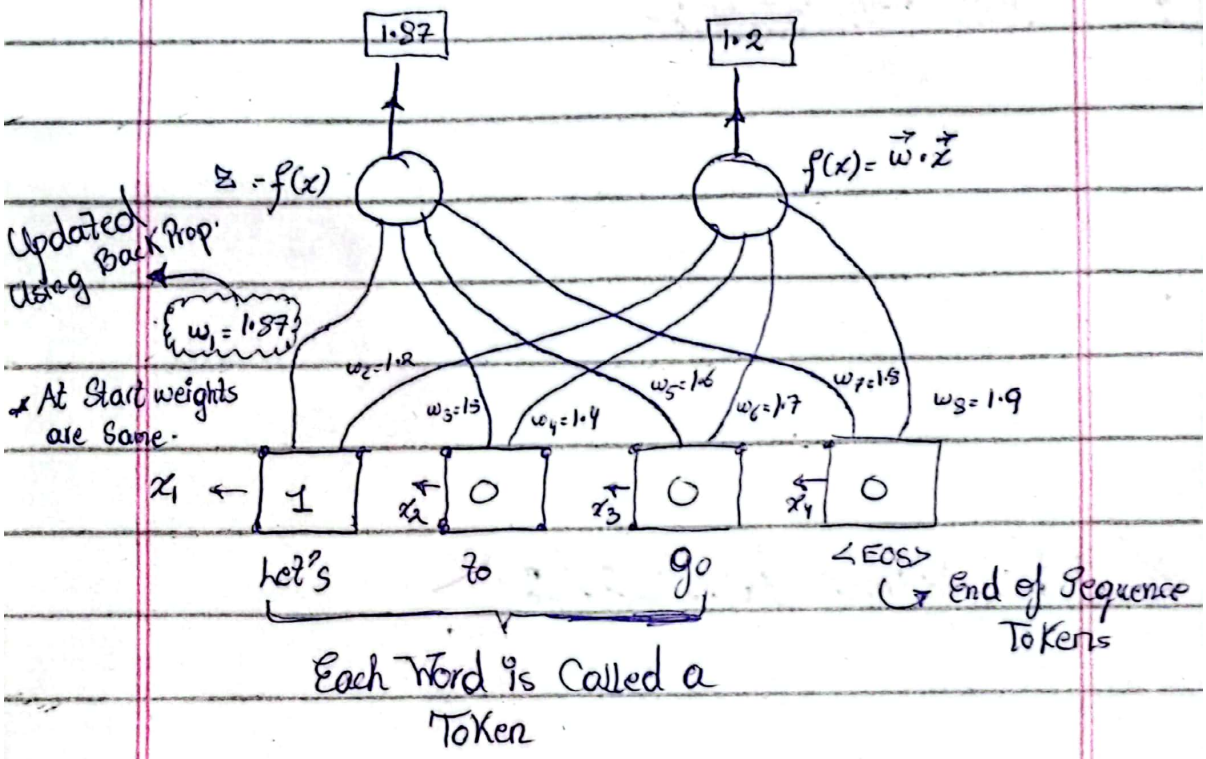
=> Embedding Space of GPT-3 = 12,288

↳ Dictionary Size: 50,K Tokens

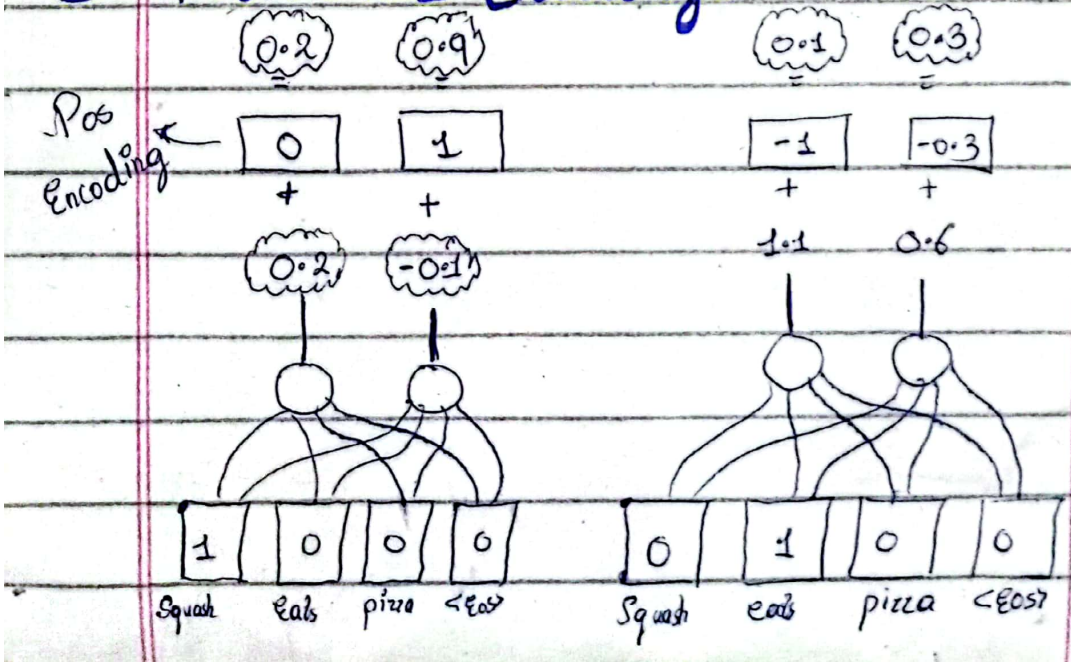
→ Context Size: 2,018 Tokens

(9) Word Embeddings-

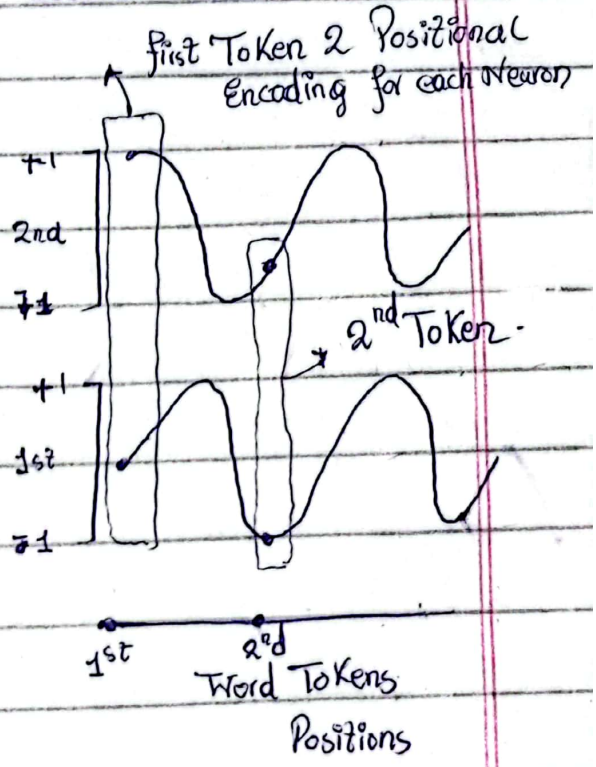
↳ Select words into numbers using its vocabulary created using word2vec:



(10) Positional Encodings-



↳ we get Positional Encoding using alter-nating Sine and Cosine Waves.



↳ When we Swap word's in Sentences, there word Embedding swap but the positional encoding values remains the same, so we end up with new pos encodings

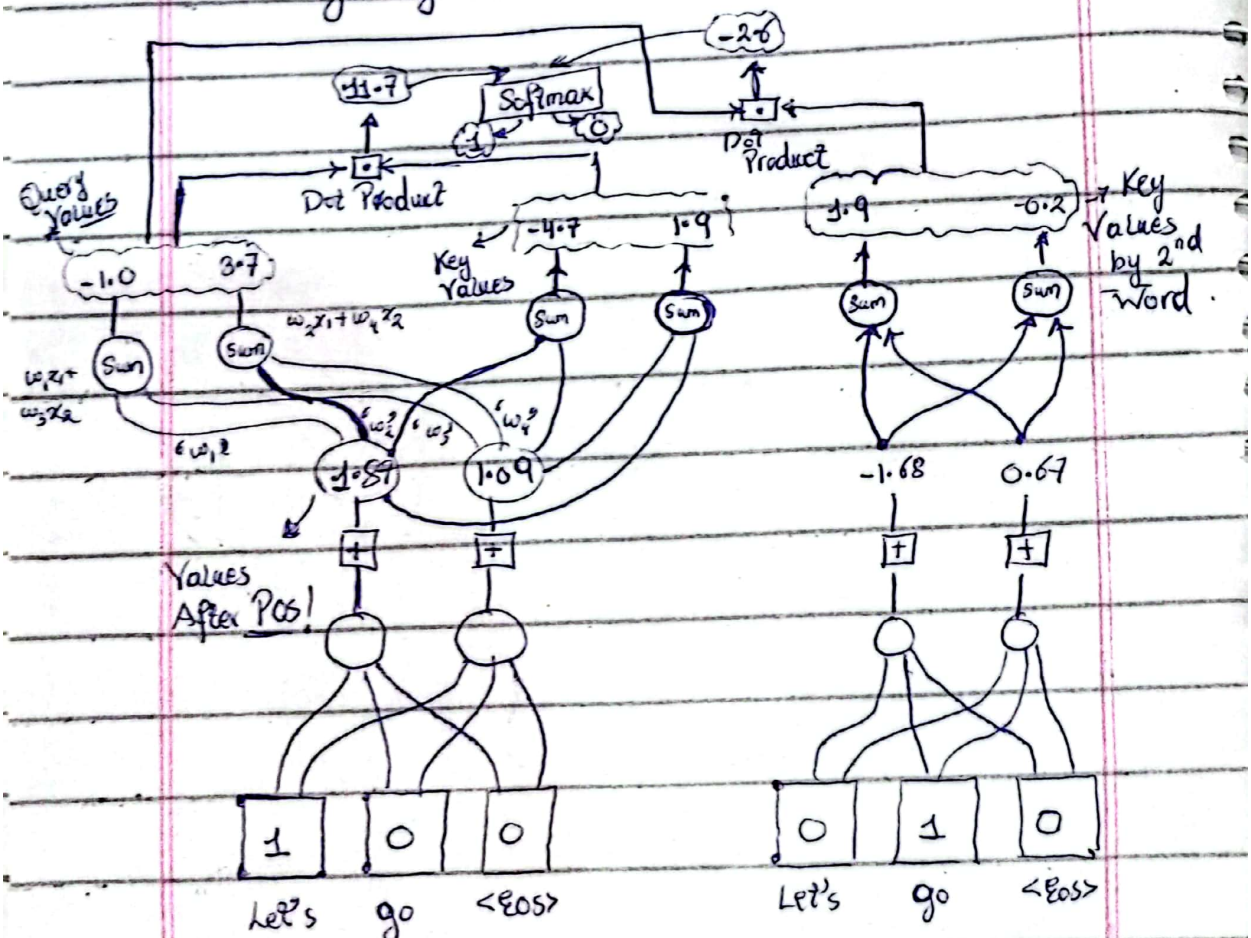
(iii) Self-Attention:-

↳ Method to Associate words with each other. It gives context that

$\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+2 \end{bmatrix} = \begin{bmatrix} 4 \end{bmatrix}$.

$1 \times 2 \quad 2 \times 1$

how much attention should a word be giving to other word in Sentences



→ You multiple Attention Scores with Every word Value vector.

* Multiply Softmax Values by Values matrix to generate Vector in Vector Space.

After Value Vectors Generated add them together. Like $\begin{bmatrix} 2.5 \\ -2.1 \end{bmatrix}$ → This Shows Self-Attention Values of Word 'Let's'.

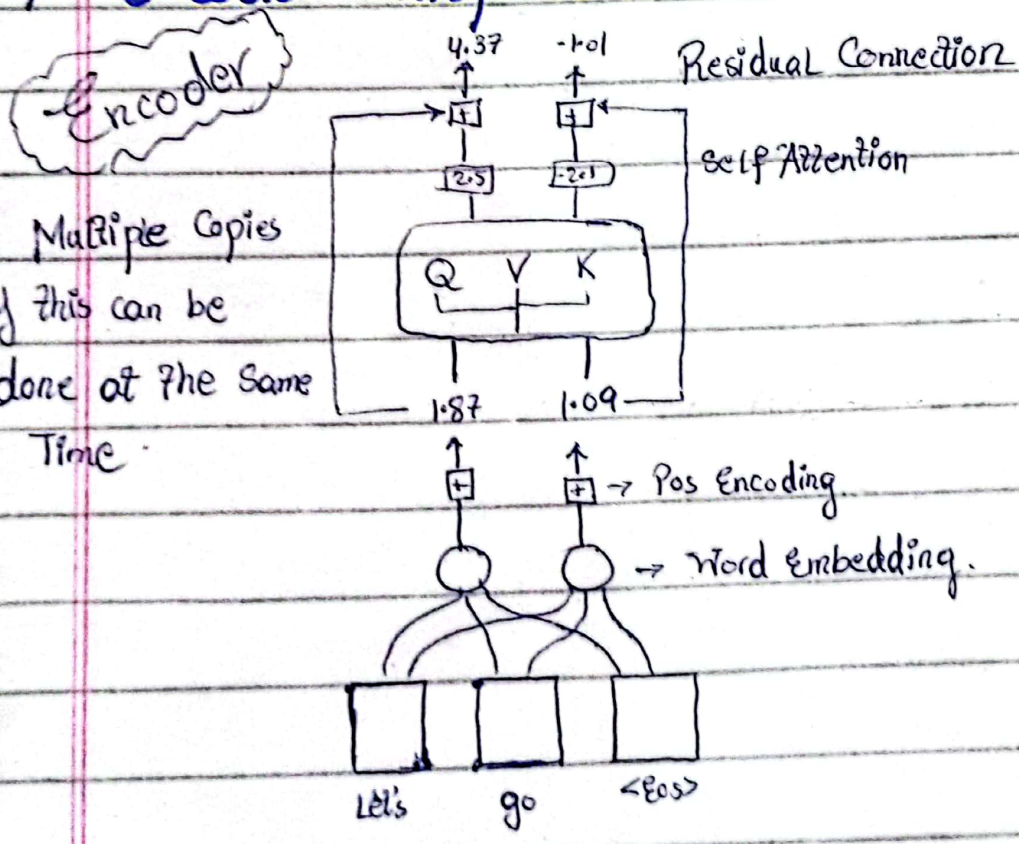
This can Run Parallel

↑

* Note the Weights Values used for Queries, Keys and Values remains the same for all the words in the input of transformer.

↳ For Multi-head, each uses its own set of weights for Query, Key and Values

iv) Encoder Transformer :-



Multiple Copies of this can be done at the Same Time.

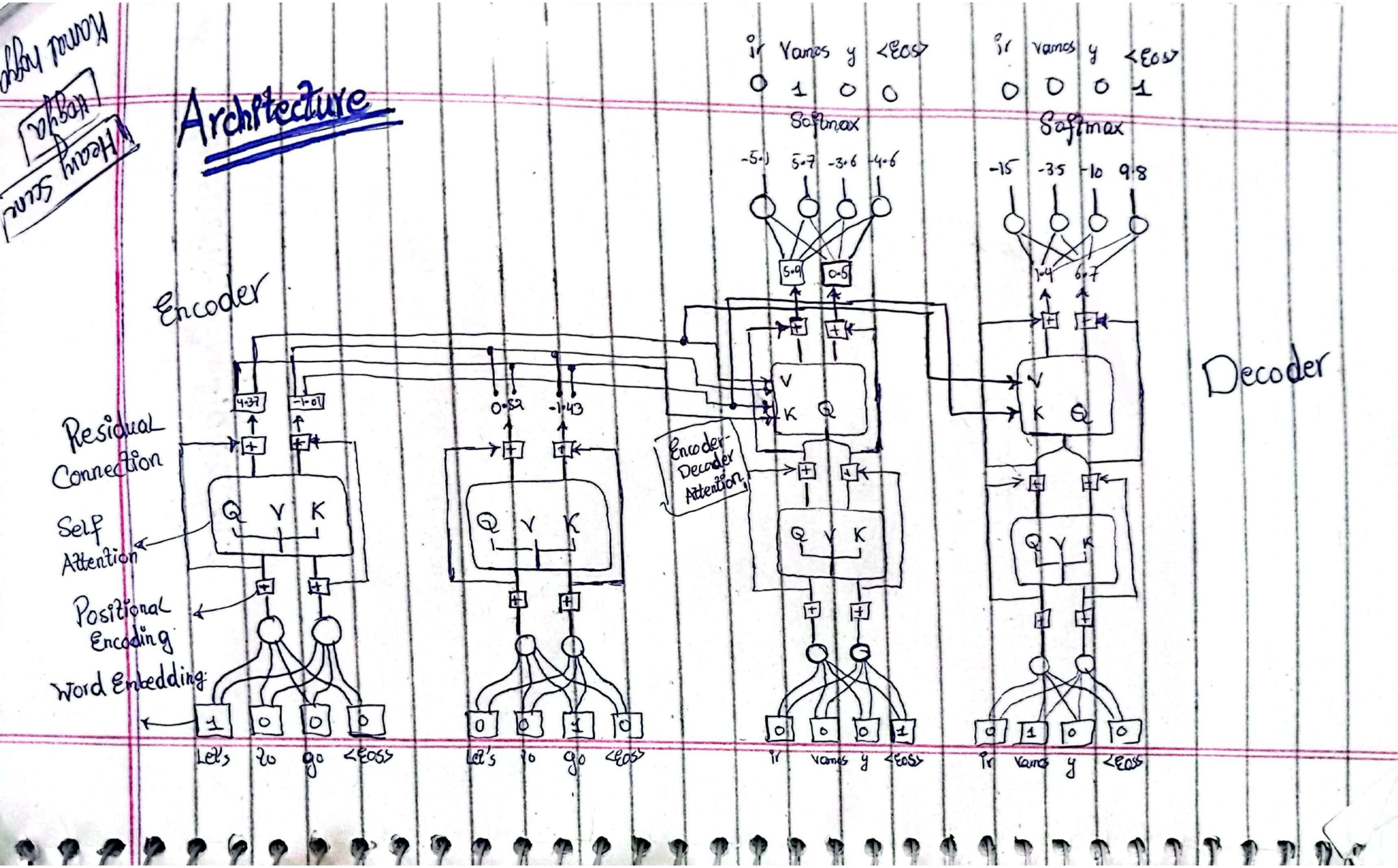
Transformer.

(Y) Decoder Transformer :-

→ After encoding values are generated they are passed to Encoder-Decoder Attention Part.

* Note it has its own pair of Query, Key and value weights. After Attention Apply Residual Connection with Output enbeddings, to generate a vector which is passed to fully Connected layer having no. of Outputs equal to decoded vocabulary.

Architecture



★ Generative Configurations:-

(i) → Max New Tokens

→ Greedy Sampling take highest Probability word [For short text Generation].

↳ Random [-weighted] Sampling :- select a token using random-weighted strategy.

(ii) Top K :-

↳ Select Output from Top-K results and apply random-weighted strategy.

(iii) Top-P :-

Top-K ↑

↳ Combined words propability should be Less than or equal to 'p'.

(iv) Temperature :-

Broader, flatter Probability Distribution ↑

↳ high value → Increase Randomness

↳ low value → Decrease Randomness

★ Prompt-Engineering :-

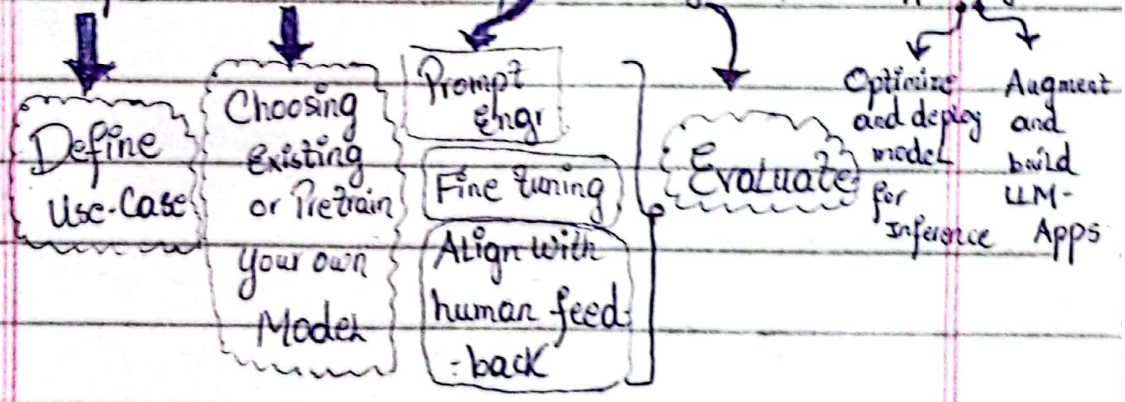
In-Context Learning
- (i) Zero-Shot Prompting → No Context.
- (ii) One-Shot Prompting → Single Example
- (iii) few-Shot Prompting → two or More Examples in Prompt

↳ In-Context learning has limitation of Context window size and for too small model low performance

★ Generative Configurations-

↳ Generative Life Cycle

Scope → Select → Adapt and Align Model → App Integration



↳ LLM-Scope and Use-Case :-

(i) Essay Writing
(ii) Summarization
(iii) Translation
(iv) Information Retrival.
(v) Invoke API and Actions

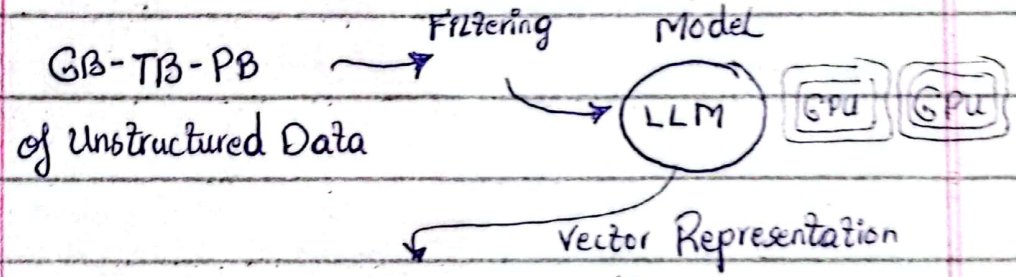
★ Pre-training Large Language Models-

↳ Foundational Models

↳ Custom Models (Train your own Model)

★ Model hubs { Model usecases & Limitation }

↳ Pre-training At high levels-



Token	TokenID	Embedding
'The'	37	[-0.05, 2.37, ...]
'_teacher'	3145	[2.5, 3.2, ...]
'_teaches'	11749	[3.5, 4.7, ...]
'_the'	8	[2.2, 5.2, 2.8, ...]

(i) Encoder-Only Models:- [Auto-encoding Models]

↳ Masking Language Models

↳ Each word has full context of the words before and after it.

Use Cases :-

(i) Sentiment Analysis

(ii) Named Entity Recognition

(iii) Word Classification

Example Models :- (i) BERT (ii) ROBERTA

(ii) Decorder-Only Models :- [Auto-Regressive Models]

↳ For Text-Generation

↳ Each Token sees Token Before it

(iii) Encoder-Decoder Models :- { Seq-to-Seq }

↳ Translation

↳ Text Summarization

↳ Question Answering.

★ Computational Challenges :-

↳ GPU RAM for 1B Parameters

∴ 1 Param = 4 bytes (32-bit float)

1B Param = 4 x 1B = 4 GB { GPU Memory }

Not Only This

Adam Optimizer :- +8 bytes Per Param

Gradients : +4 bytes per Param

Activations : +8 bytes per Param

∴ Total for a Single Parameter = 4 bytes + 20 extra bytes

This makes it 24 GB RAM Required

∴ To Reduce This Use Quantization Process.

↳ Reduction of 32-bit floating Point to 16-bit FP or 8-bit Integer, reduces precision but are memory efficient

↳ Another ALternative is to Use BFLOAT-16. Which Optimizes the 16 bit Precision.

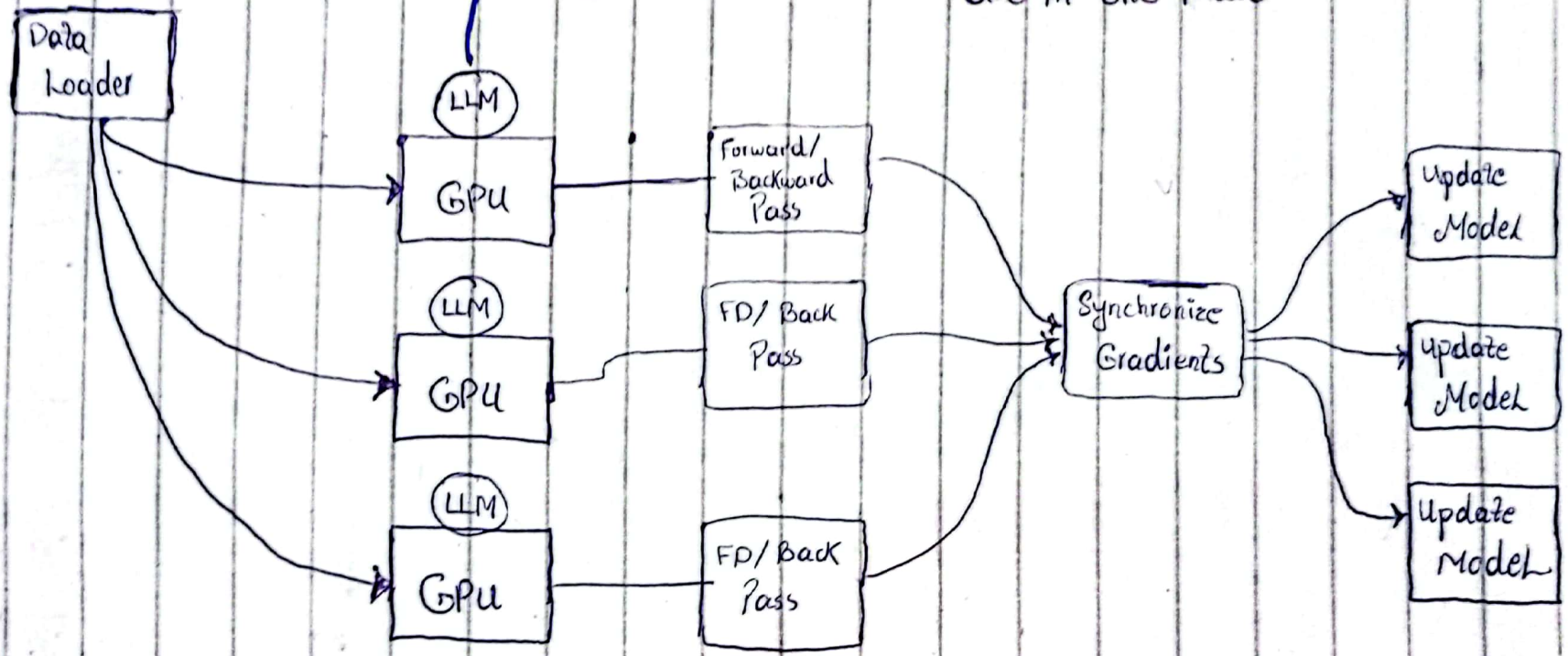
↳ Quantization Aware Training Leaks the scaling factors during training.

★ Efficient GPU Strategies:-

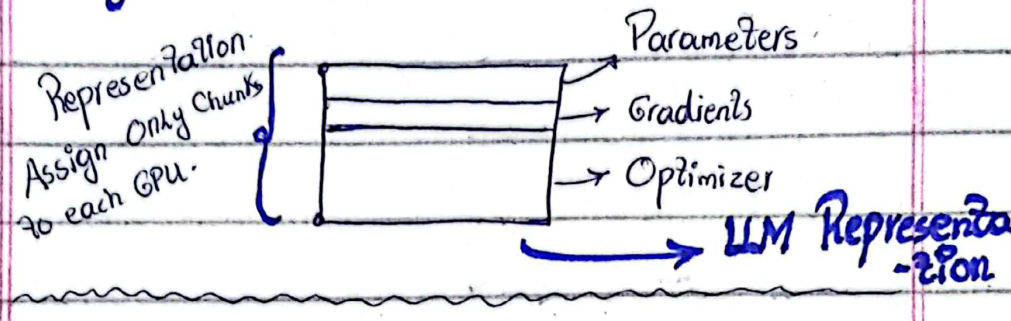
(1) Distributed Data Parallelism -

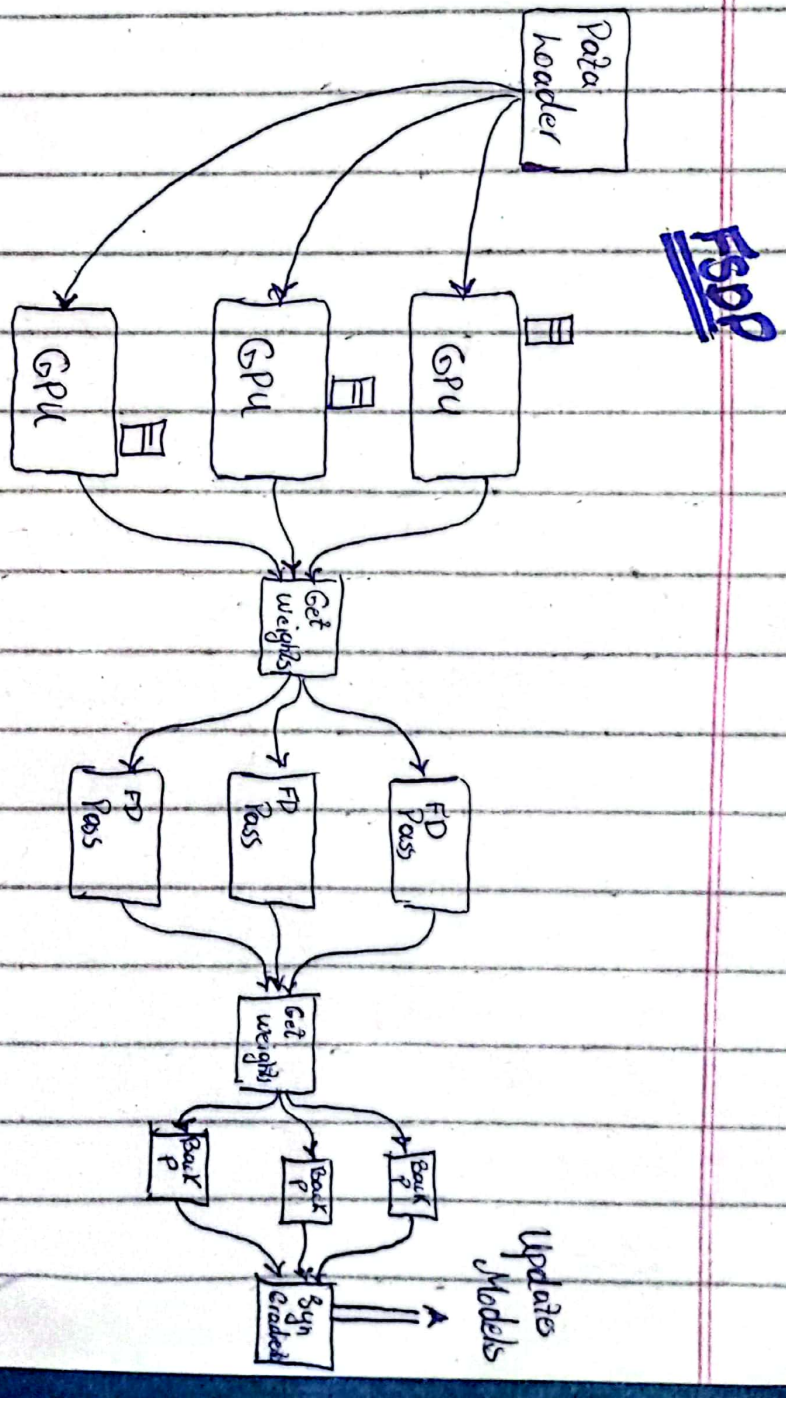
DDP :-

Full Copy of LLM on Each GPU → Not feasible as All Parameters, there gradients and Optimizer States are in one Place.



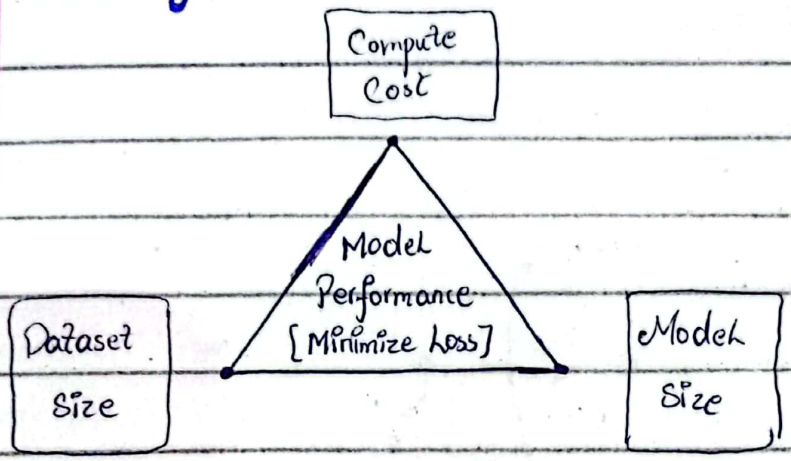
★ Fully Sharded Data Parallelism [FSDP]:-





1 PetaFlop/s-day → 1 PetaFlop Operations for whole day. (per second)
↓
1 Quadrillion floating Point Operations Per Second.

* Scaling Laws 8-



* Increasing Dataset Size on a fixed Smaller model can Also Increase performance and Reduce Loss.

↳ Finding the Optimal No. of Parameters and Dataset size for max performance.

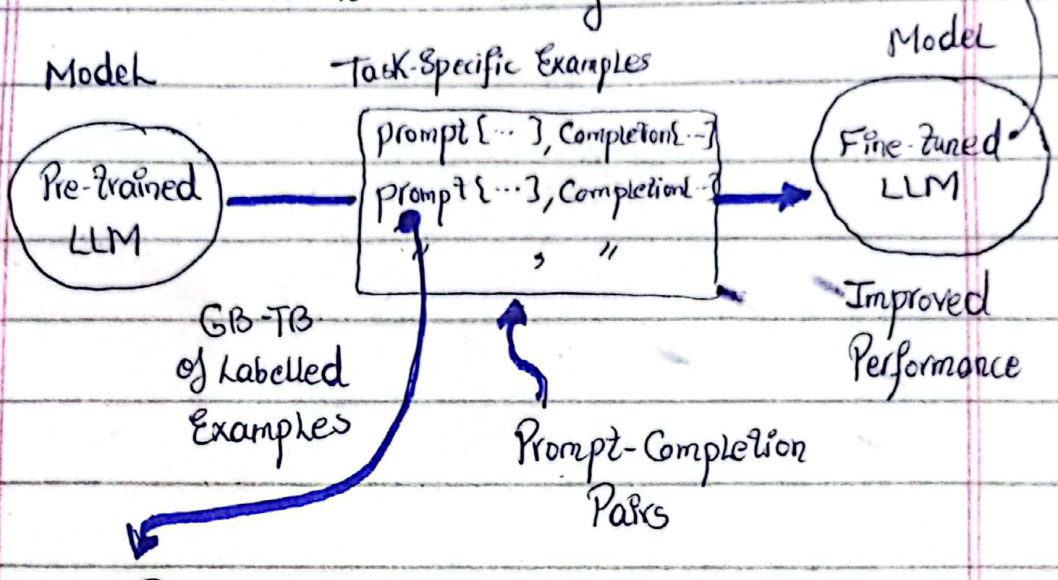
↳ GPT-3 Under Acheive during Training because it has 175 B Parameters yet dataset size was only 500 B Tokens

↳ LLAMA uses 65B Parameters but dataset size is 20x times, so 1.4 Trillon Tokens. Utilizing All Parameters during Training.

★ Fine-tuning:-

Also known as Instruct-Model is case of Instruction fine-tune

↳ Instruction- Fine tuning



Each Prompt Consist of instruction associated with the text:-

Like:- (i) Do Sentiment Analysis / Classify the Review

(ii) Predict the Rating

(iii) Generate a text

(iv) Provide small short summary of sentence.

↳ Split dataset in three Portions:- train, validation and test set.

↳ Full Fine-tuning deals with using all parameters, gradients and Optimizer while training.