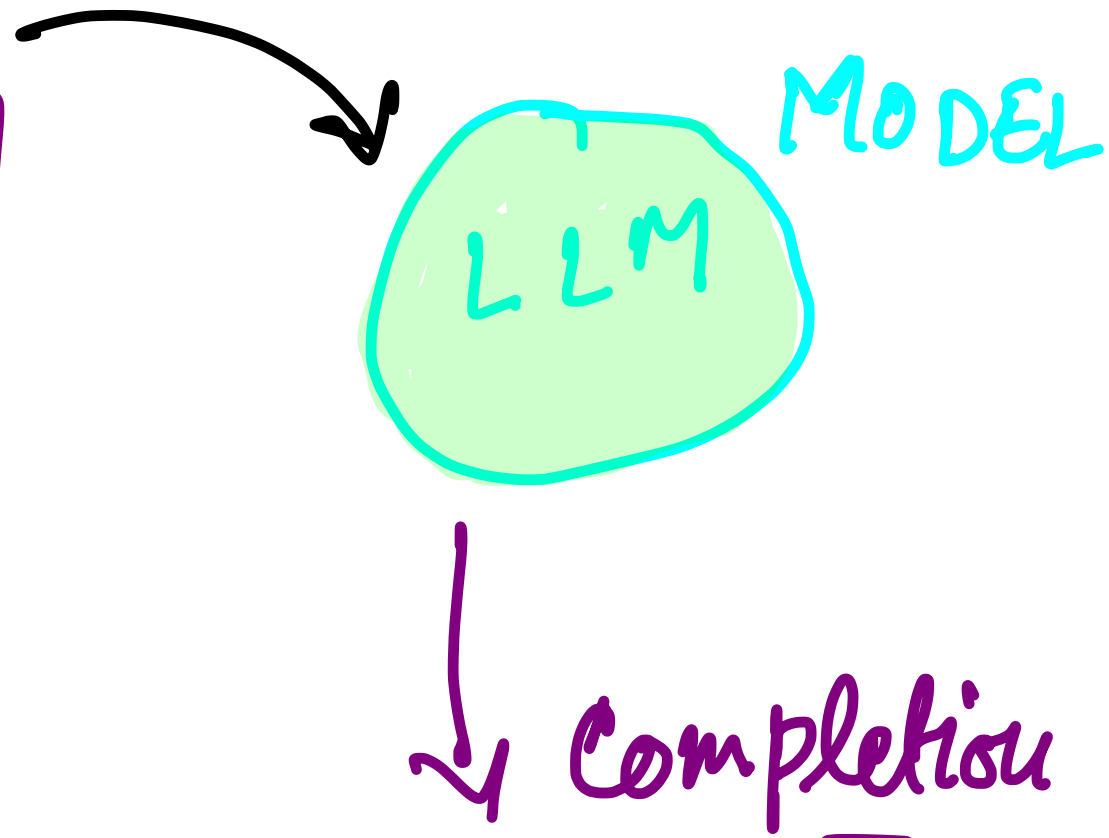
WEEK-01

Generative AI & LLM :-

PROMPT

Where is earth located in Solar System



Where is earth located in Solar System ?

Earth is a planet it is located within SUN orbit's.

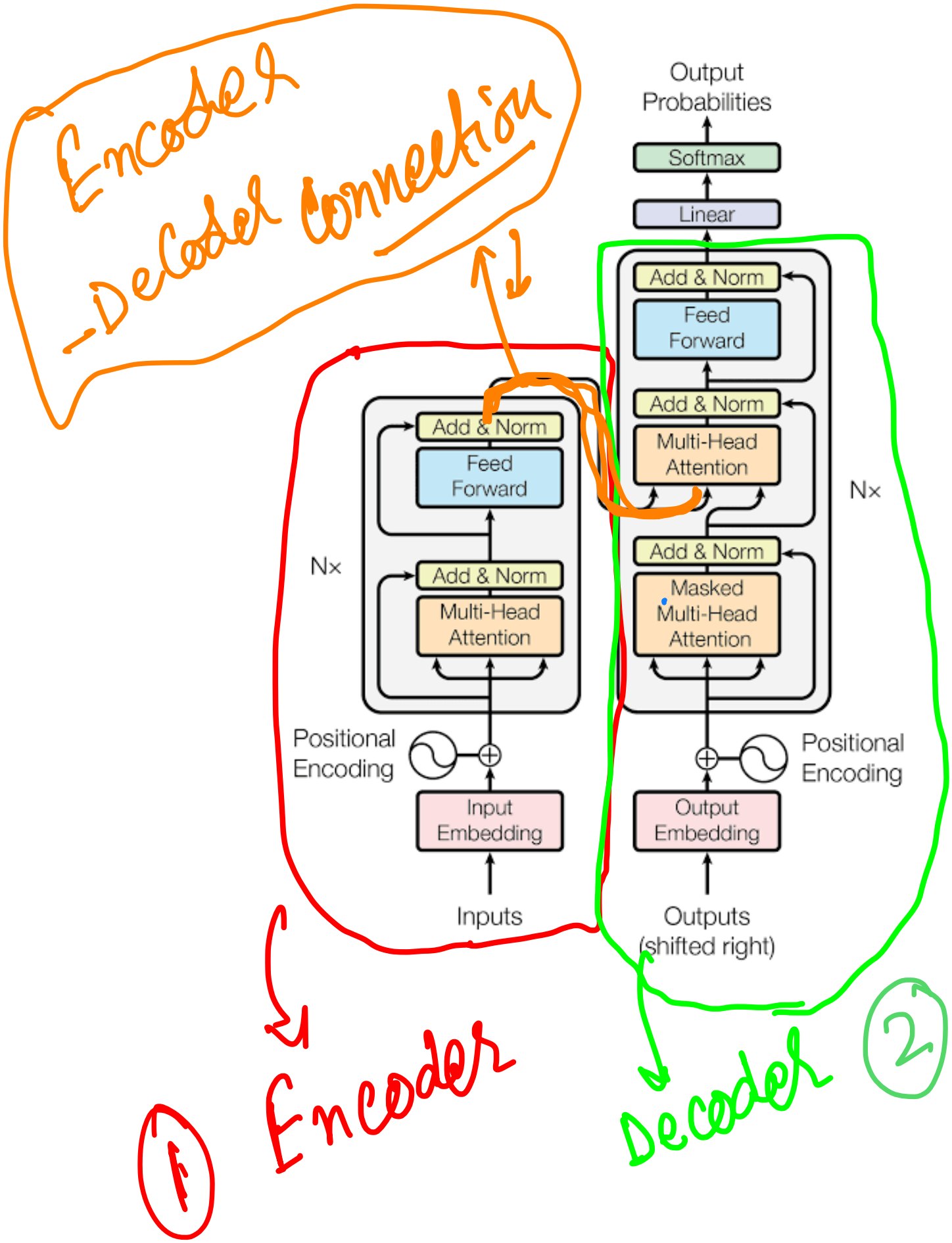
LLM USE CASES :-

1. Summarize 2. Translation

SCALE OF LLM :-

110 M

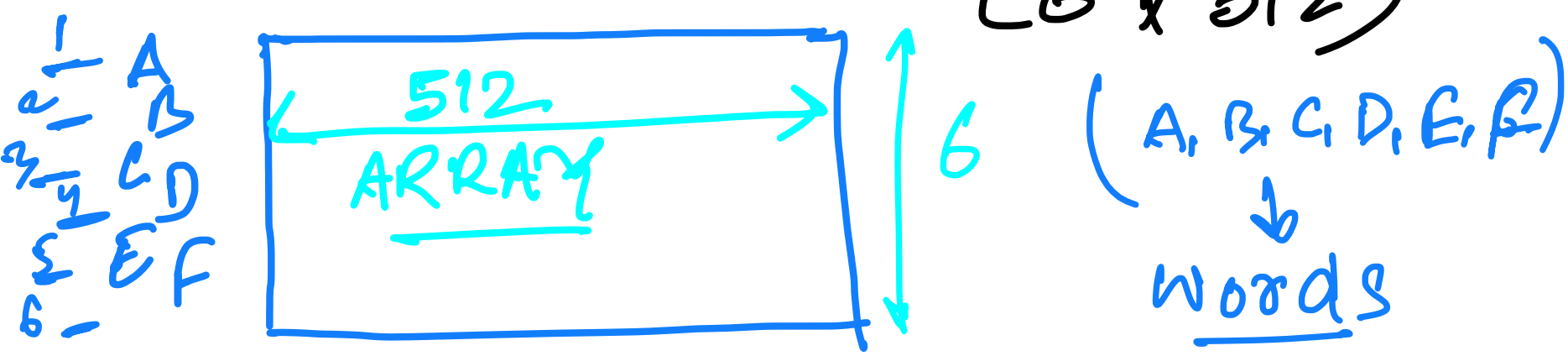
TRANSFORMER ARCHITECTURE



NOTATIONS :-

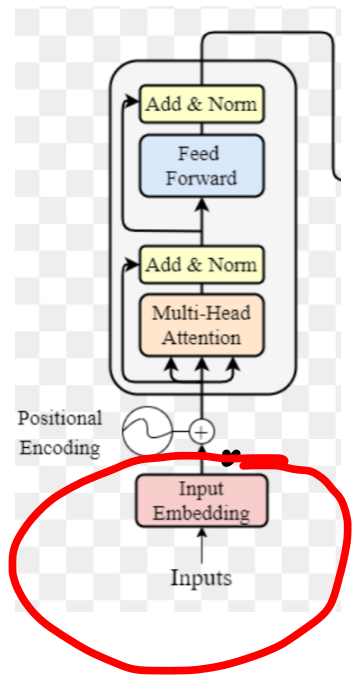
Input Matrix (sequence, d model)

(6 x 512)



Encoder:

INPUT EMBEDDINGS ?



Words are map to tokens.

Your → 105 ← map

Cat → 6587

is → 5475

a → 3578 ← map

Lovely → 65 ← map

Cat → 6587

↳ to position in Vocab

Input IDs

(position in Vocabulary)

Token

Your → 105

Cat → 6587

is → 5475

a → 3578

Input IDs

Lovely → 65

Cat → 6587

(Position in Vocabulary)

dmodel = 512

512 | 512 | 512 | 512

512 .... 'Embedding' of 512

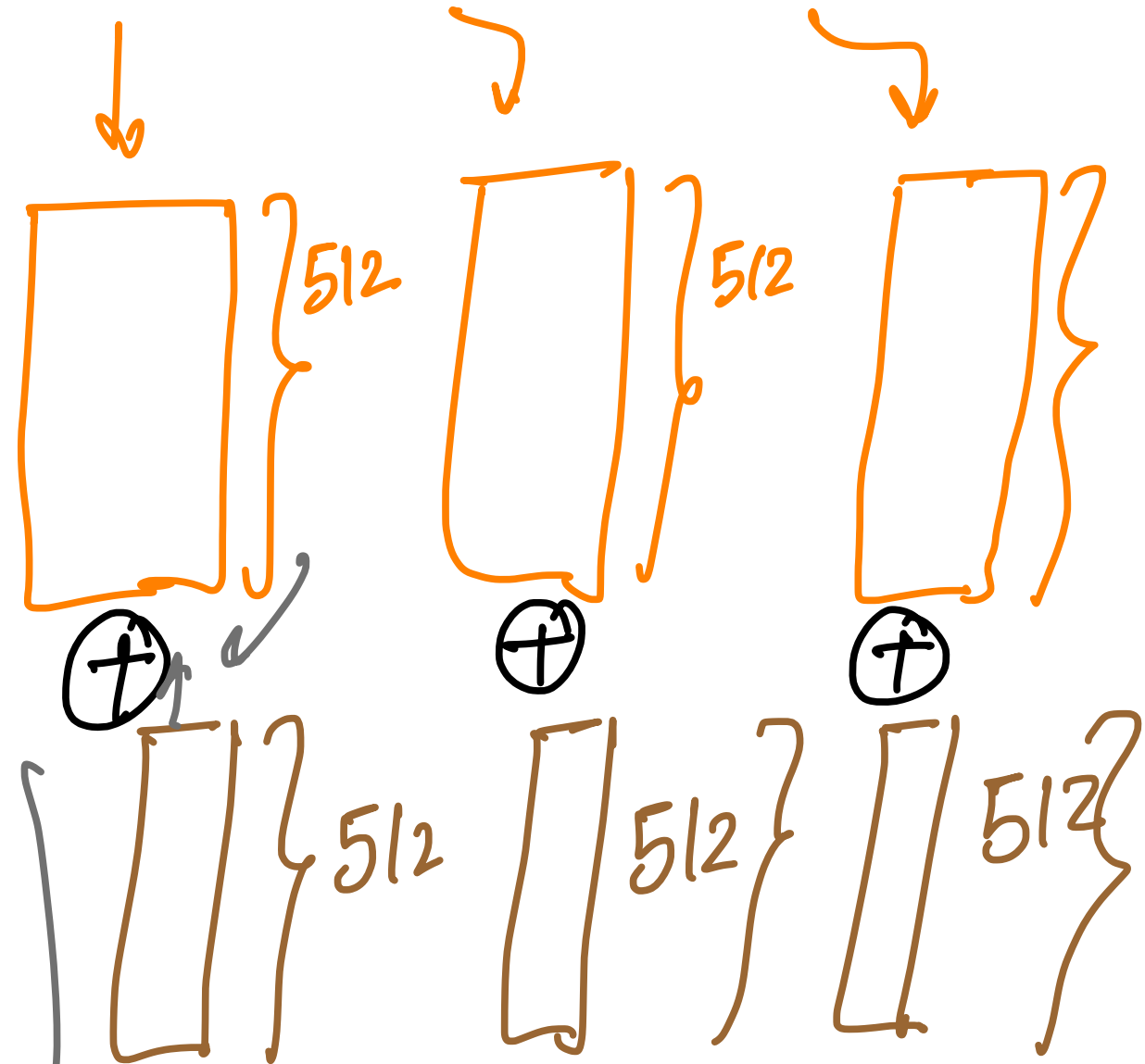
512 is a parameter. Not fix.

Positional Encoding :

- We want each word to carry some information about it's position in the sentence.
- We want model to treat words that appear close to each other as 'CLOSE'. & words that are distant as 'DISTANT'.

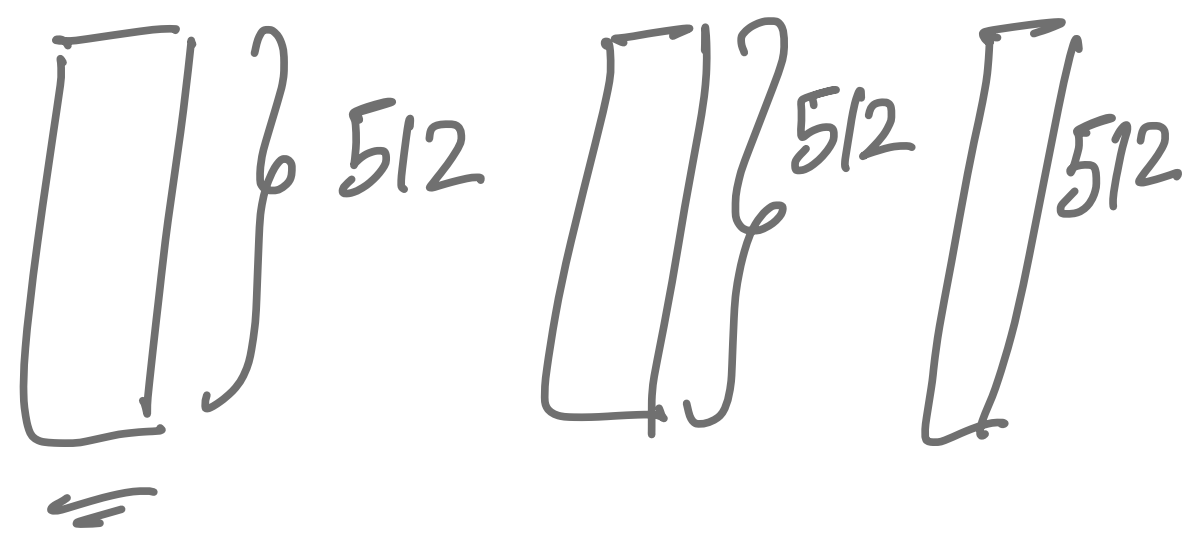
Original Sen = Your Cat is ...

Embeddings Vector of 512



Positional Embedding Vector 512

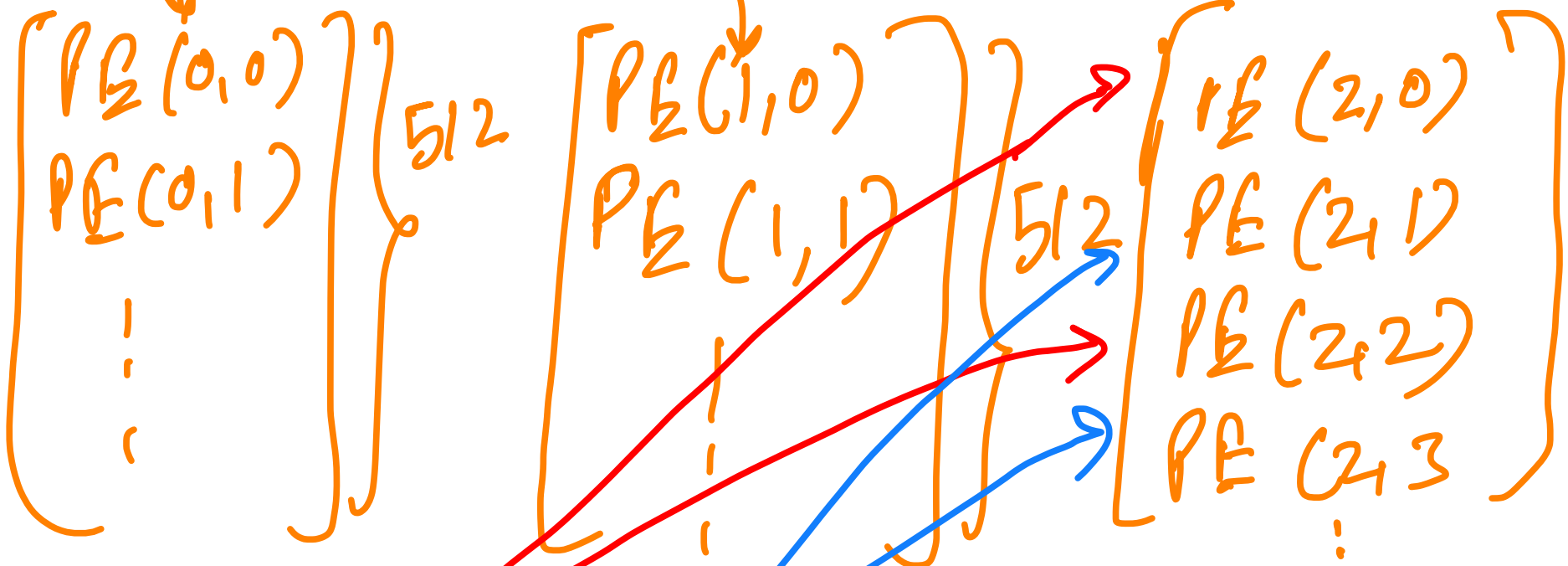
Encoder Input Vector 512



How positional Encoding (PE) calculated?

Your Cat is

pos=0 pos=1 pos=2



$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

even Rows

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

odd rows

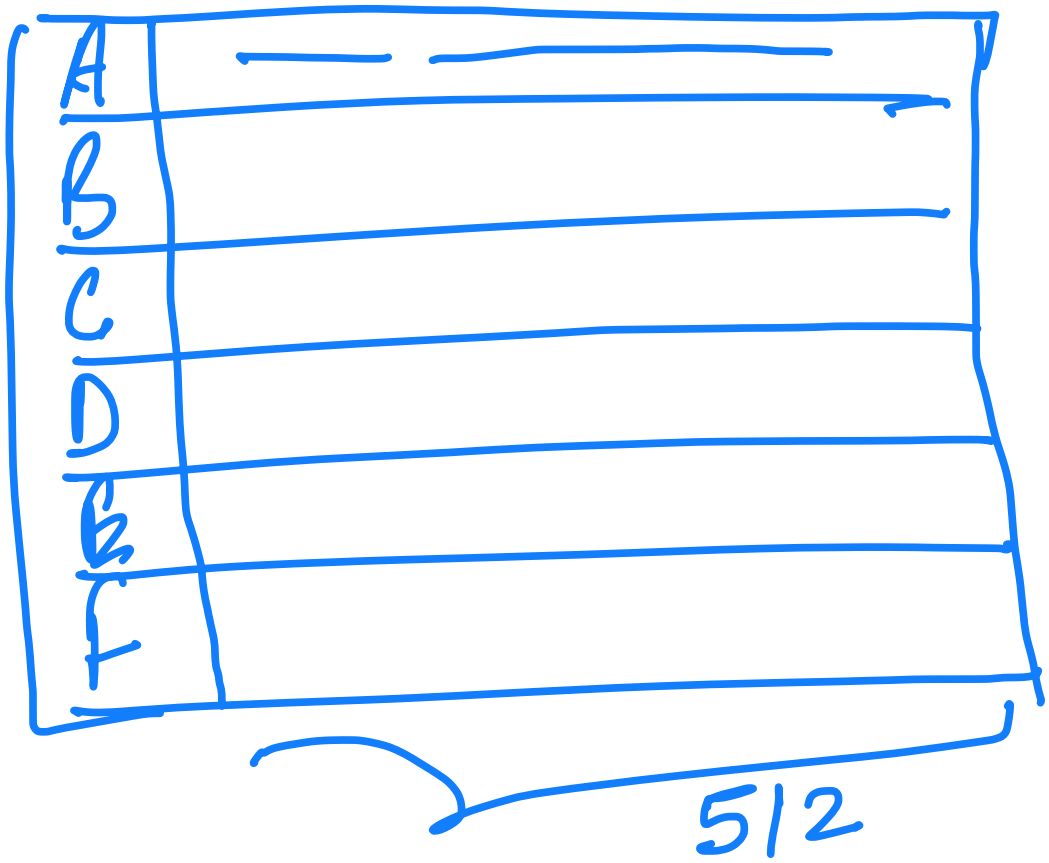
Encoder: Multihead Attention

Let's first understand 'Single Head Attention'.

Self-Attention:- Self. attention allows the model to relate words to each other.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Let's take simple example 6 word size = 512.



seq = 6

d model = 512

This can be re-presented by matrix Q, K, V, these are same matrix re-presenting input of 'SEX' words with dimension of '512' vector length. (dmodel = 512)

$Q = K = V =$ Matrix (5 X 512)

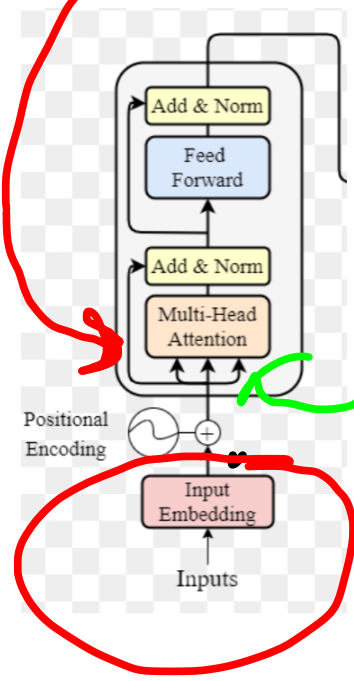
$$\text{Softmax}\left(\frac{Q_{(6,512)} \times K^{T}_{(512,6)}}{\sqrt{d_{model}} = \sqrt{512}}\right) = \underset{\text{matrix}}{6 \times 6} =$$

	Your	Cat	is	a	lovely	Cat	Σ
Your	.268	.119	1	1	1	1	1
Cat	1	1	1	1	1	1	1
is	1	1	1	1	1	1	1
a	1	1	1	1	1	1	1
Lovely	1	1	1	1	1	1	1
Cat	1	1	1	1	1	1	1

This is intermediate matrix of Attention which. Each cell is DOT product of Embeddings. Hence each cell represent a 'SCORE' How intense is a relationship between those words.

$$(6 \times 6) \times \underset{(6 \times 512)}{V} = \underset{(6 \times 512)}{\underline{\underline{\text{Attention}}}}$$

* Each row of Attention matrix captures not only the meaning (given by embedding) or the position in sentence (represented by the positional encodings) but also each words INTERACTIONS with other words.



Input (seq, d model)

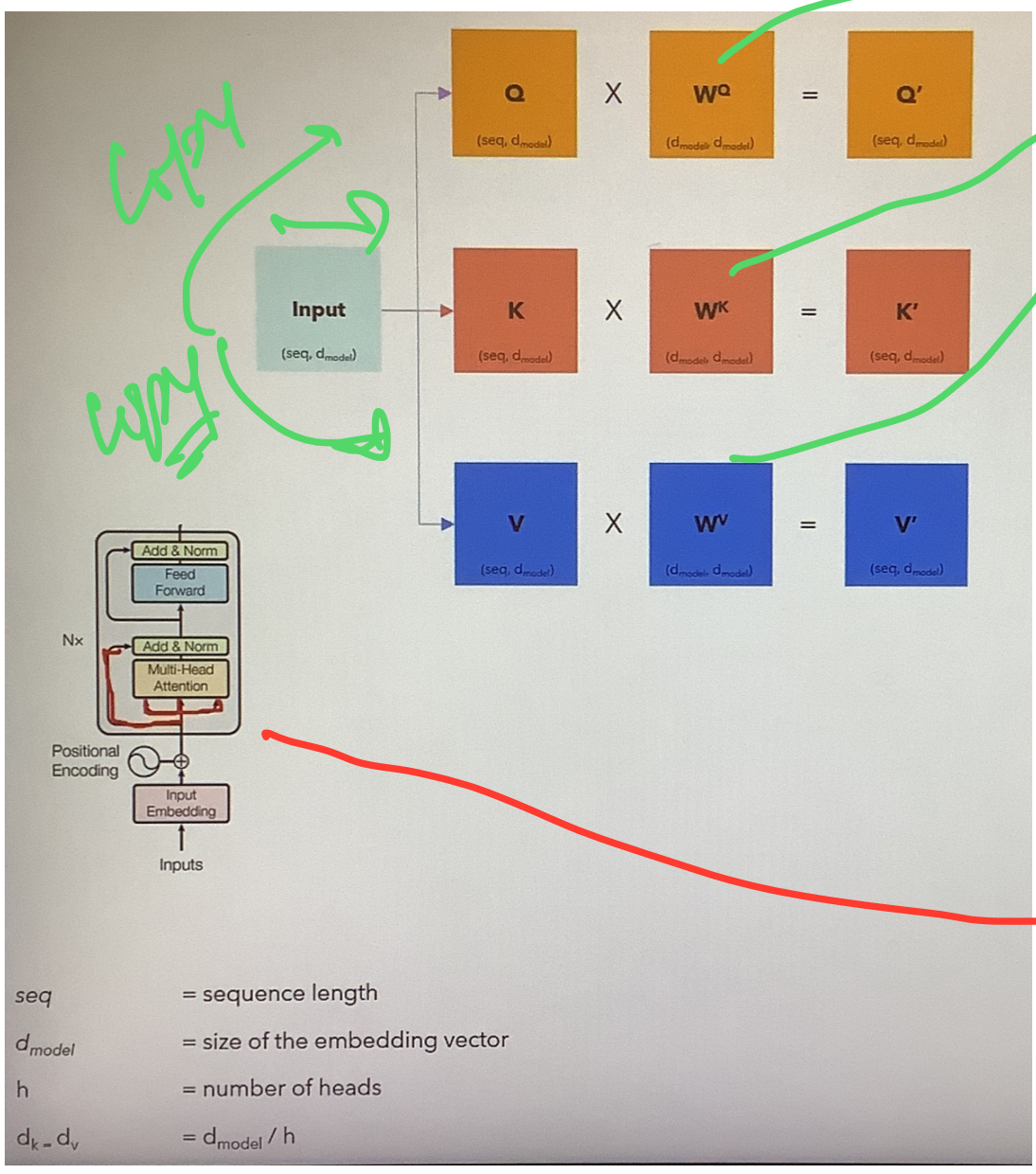
make 4 Copy

1

Sent [ADD, NORM] block

3 Sent

Multi head Attention Block



These all are Parameter Matrix

copy input matrix FOUR Times

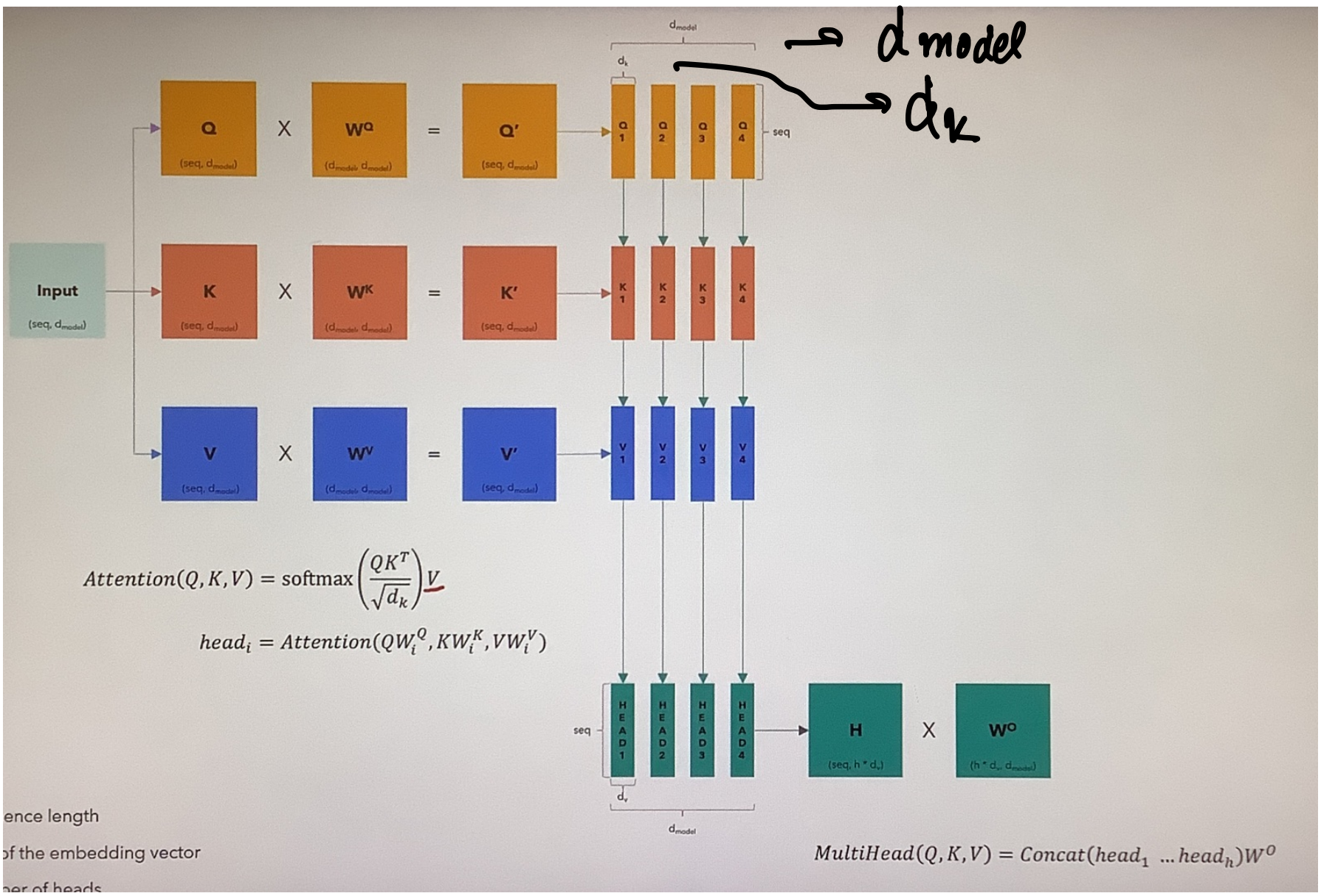
W^Q, W^K, W^V = Parameter Matrix.

h (h = no of heads)

* Each head watching full sentence but part of embedding.

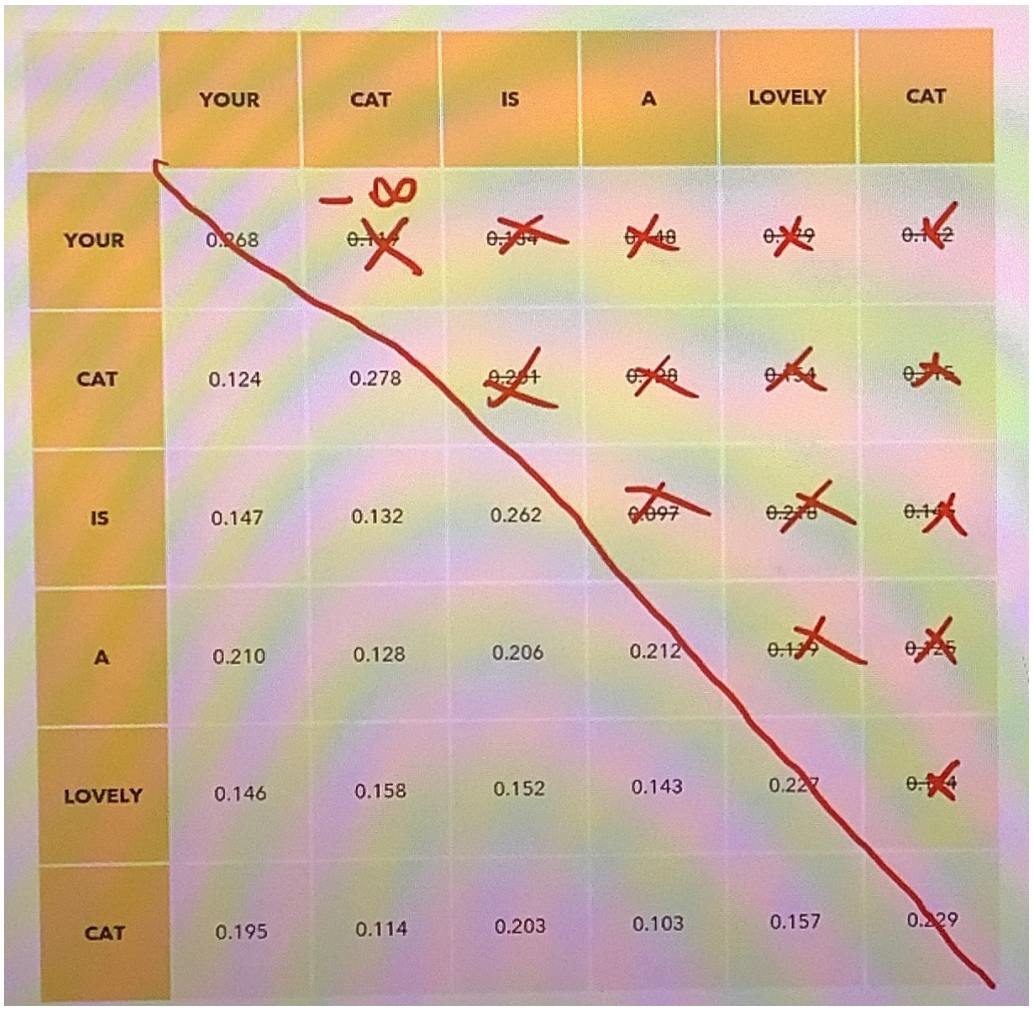
($d_k = 512/4$)

dmodel = embeding size.



Masked Multi-Head Attention :-

Our goal is to make model causal : It means output at certain position can only depend on the words on the previous positions. The model must not be able to see the Future Words.



We achieve it by making $= -\infty$ the next word & then softmax turn that into ZERO.

Training

Time Step = 1

It all happens in one time step!

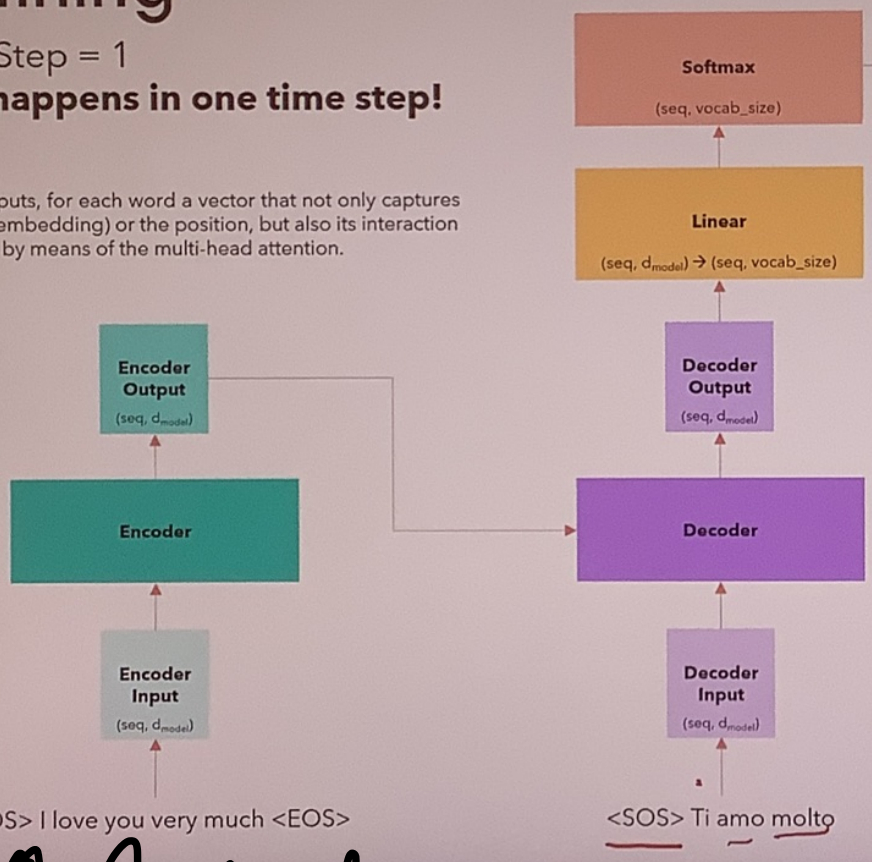
Label

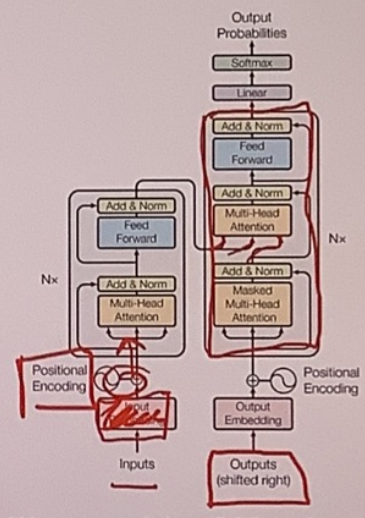
Ti amo molto <EOS>

* This is called the "label" or the "target"

Cross Entropy Loss

The encoder outputs, for each word a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.





We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

Umar Jamil - https://github.com/hkproj/transformer-from-scratch-notes

↑ Input

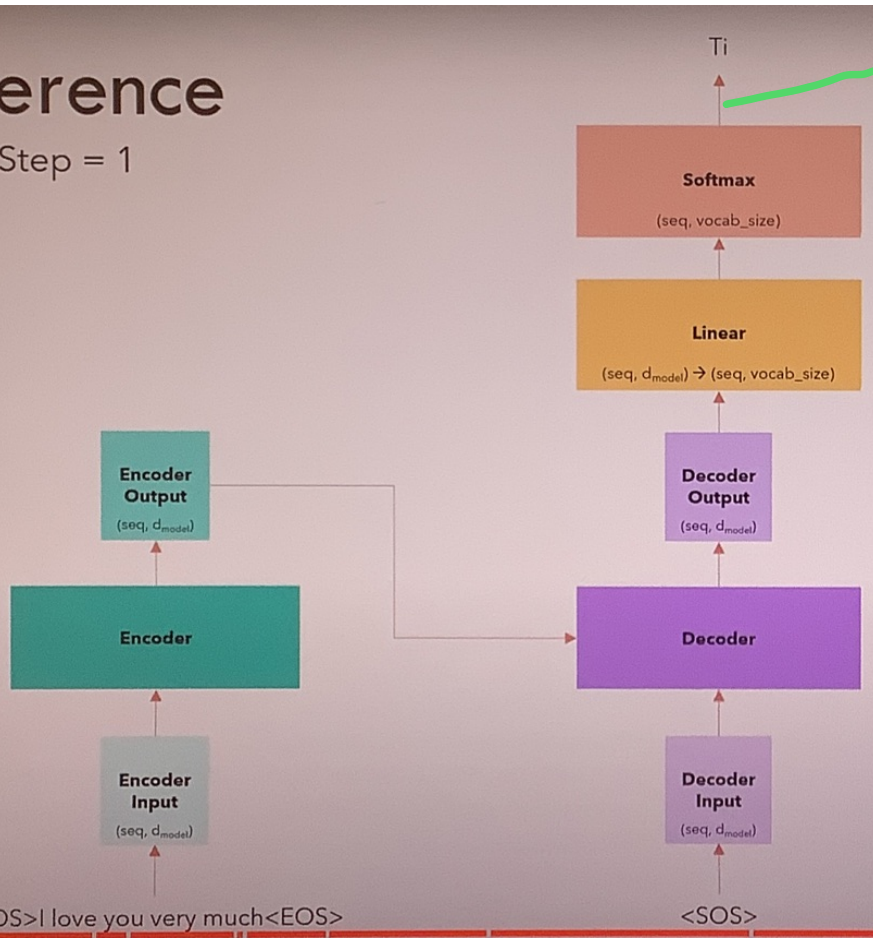
* One long sequence can be trained parallely faster.

Inference :- Inference is done Token-by-Token.

T1

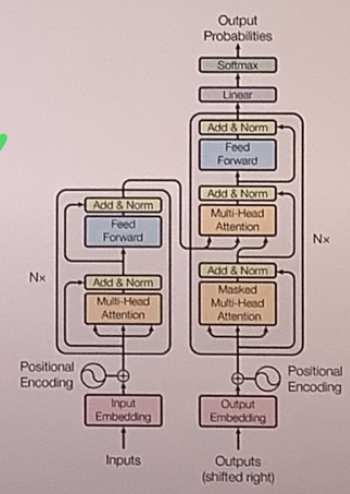
Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum value.

The output of the last layer is commonly known as **logits**



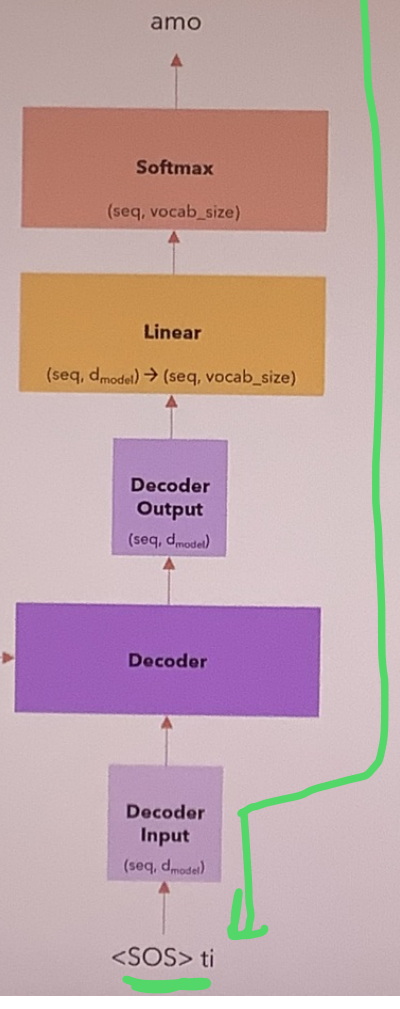
* Both sequences will have same length thanks to padding

T2

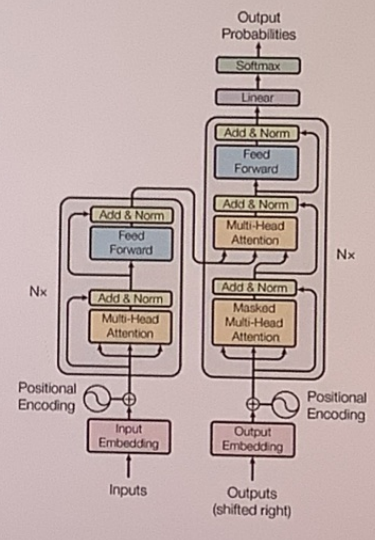
Inference

Time Step = 2

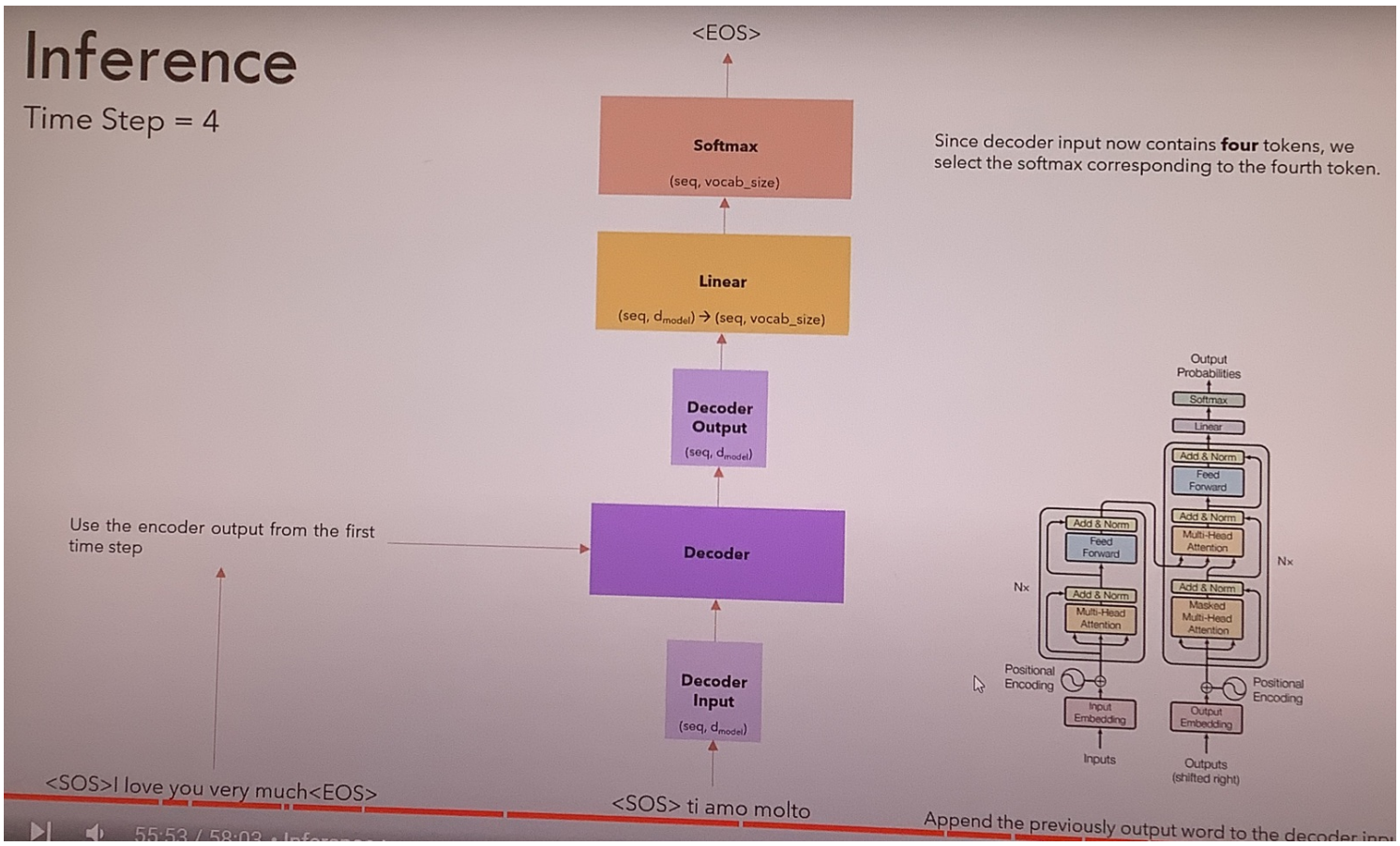
Encoder Output Re-used.



Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.



Append the previously output word to the decoder input

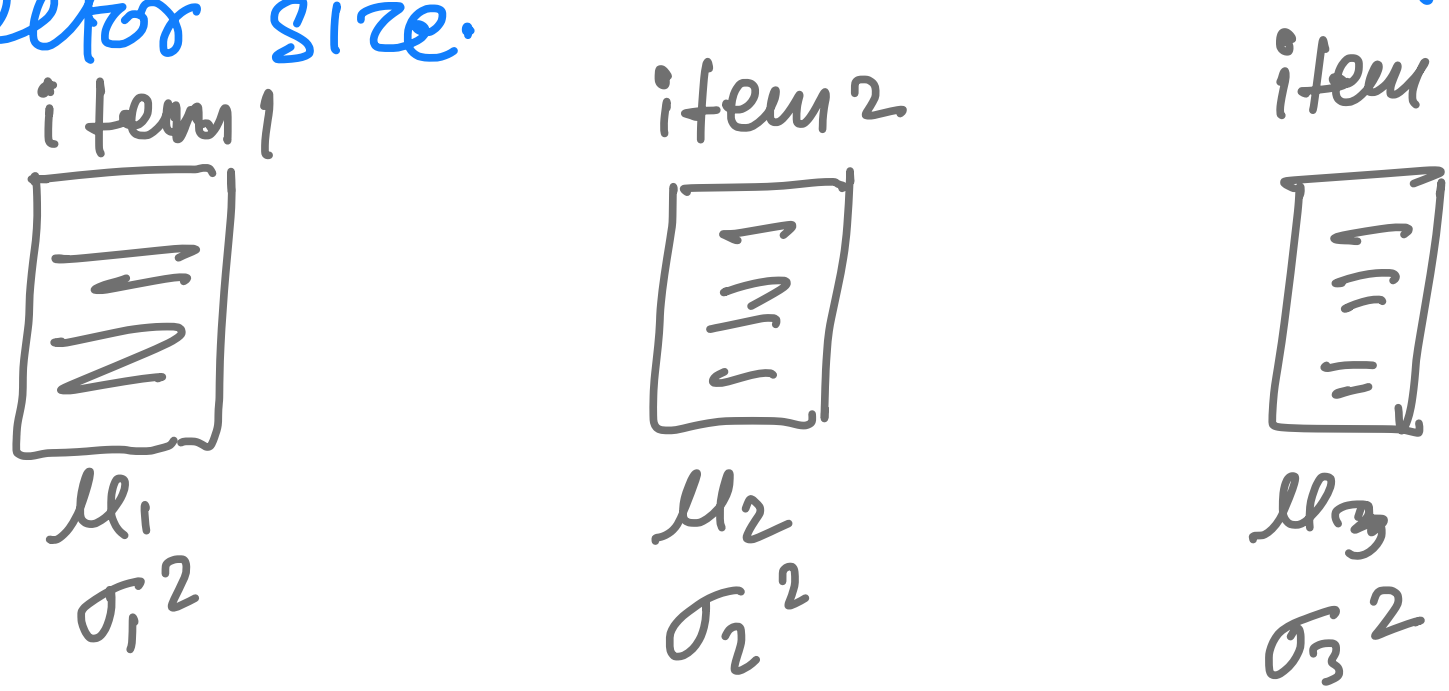


TB

We keep on iterating until decoder part produce <EOS> End of Sentence> token. Here we have used GREEDY Strategy. Beam search is better at each step we select TopB words & evaluate all the possible next words for each of them at each step keeping top B most probable sequences. This is the "BEAM SEARCH"

Add & Norm :- At this stage we perform 'Layer Nor-malization' not 'Batch Normalization'

Ex: A batch with 3 items having 512 vector size.



$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} * \gamma + \beta$$

γ, β → Learning Params.

Prompts & Completions :-

PROMPT

Where is Ganymede located in the solar system?

Model

* Context window
(typically a few 1000 words)

Completion

(Where is Ganyma---)
Ganymede is a moon of jupiter ...

Output

Encoder only

input

③ Sentiment Analysis.

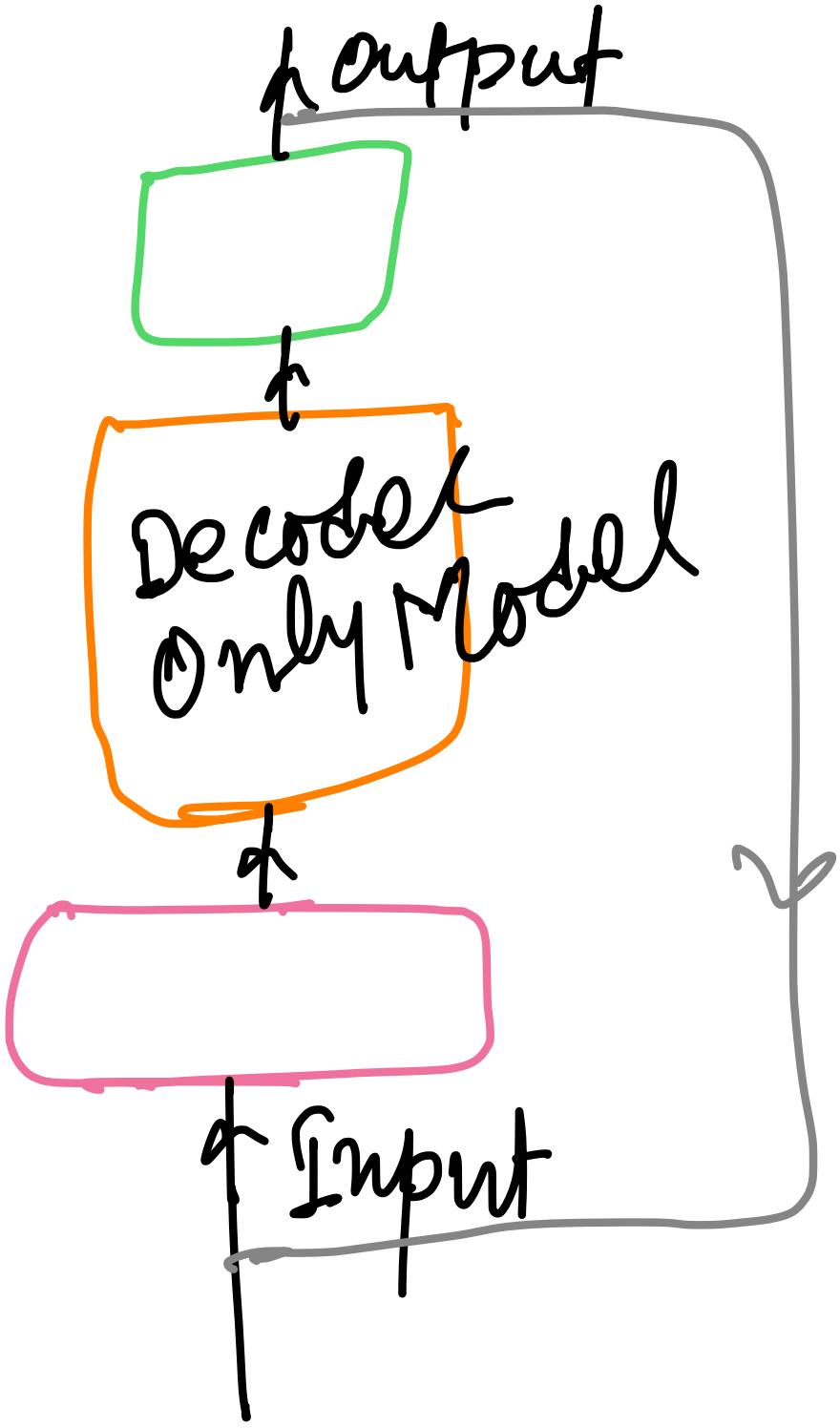
output

Encoder

Deco der model

input

④ Sequence to Seq uence task



® GPT family of models like Bloom, LAMA, etc...

Generative AI Project Life Cycle

Scope : Define the use case.

Select :- Choose existing model or pre-train yours.

Adapt and Aligned model :-

1. Prompt engineering
2. Fine Tuning
3. Align with Human Feedback

4. Evaluate.

Application Integration :-

1. Optimized Deploy model for inference
2. Augment model and build LLM powered applications.

Model Architectures and pre-training Objectives :-

GB-TB-PP

@ Consider this for how much data need to be collected to pre-train our own model.

1. Encoder Only Models:
2. Encoder-Decoder Models:
3. Decoder only Models:

Computational Challenges of Training LLM.

Approx GPU RAM needed to store 1B params.

1 parameters = 4 bytes (32-bit float)

1B Params = 4×10^9 bytes = 4 GB

® This is just to load model in memory. If we want to train

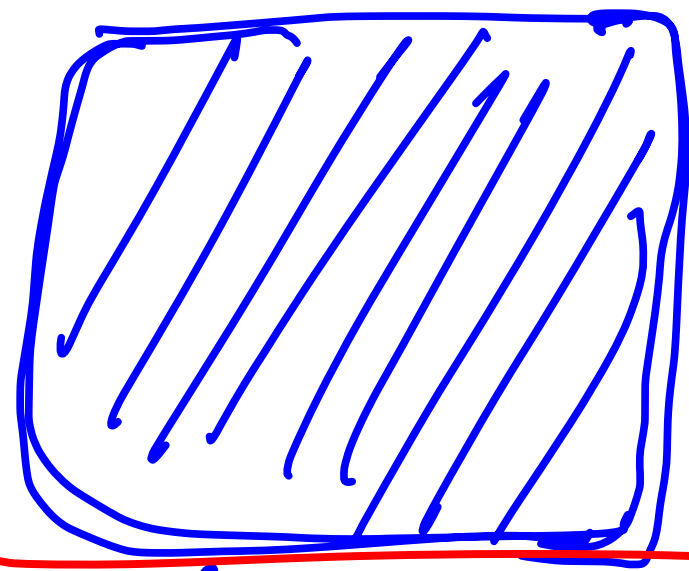
1. Model Params (wts) = 4 bytes / param
2. Adam Optimizer (2 states) = 8 bytes / param
3. Gradients = 4 bytes / param

4. Activations & tmp memory

≃ 8 bytes / param

Summing All = 24 bytes / param

⊗ It needs 6 times more GPU RAM to train model.



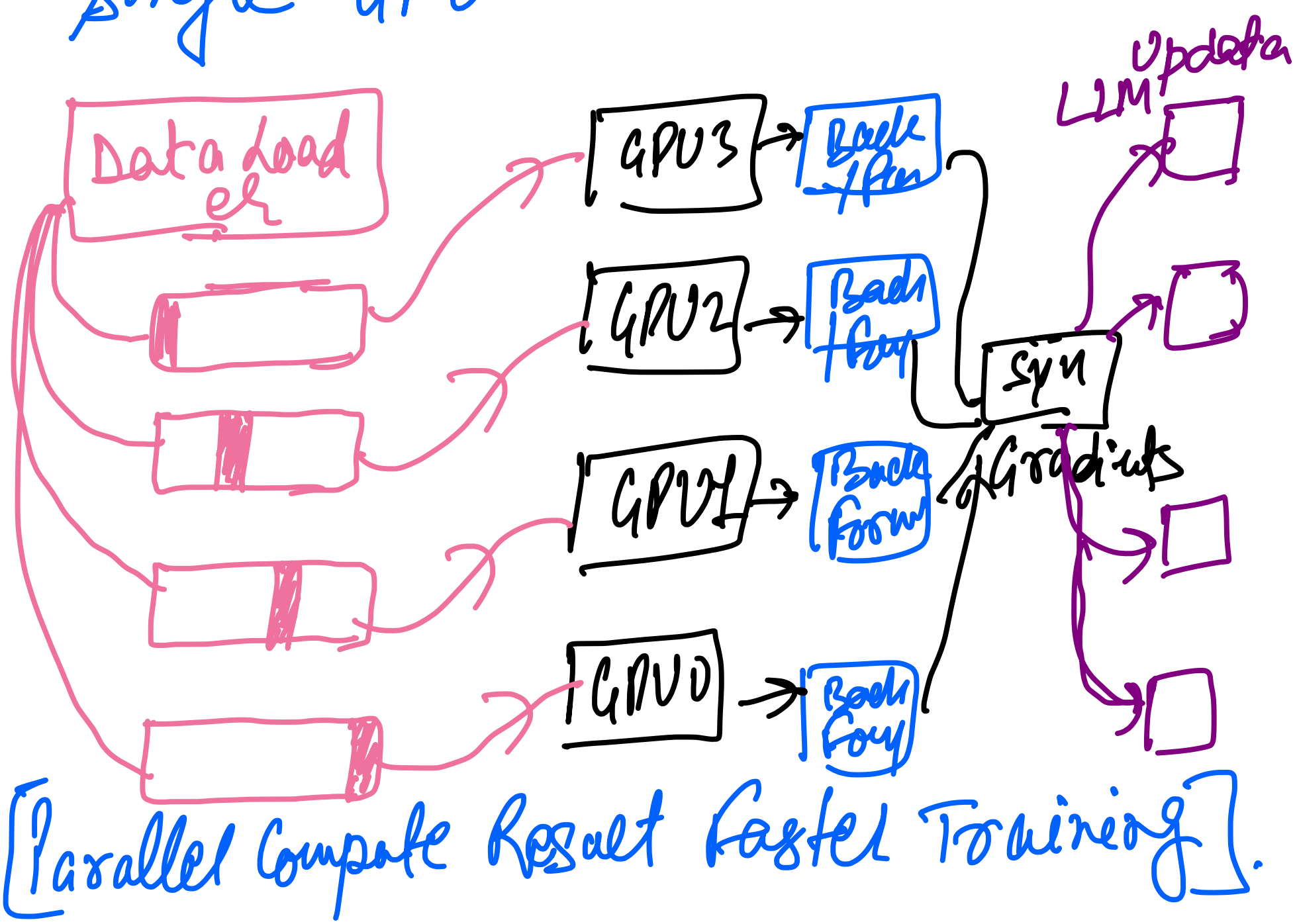
4GB Store Model

24GB Train Model.

Efficient Multi-GPU Strategy

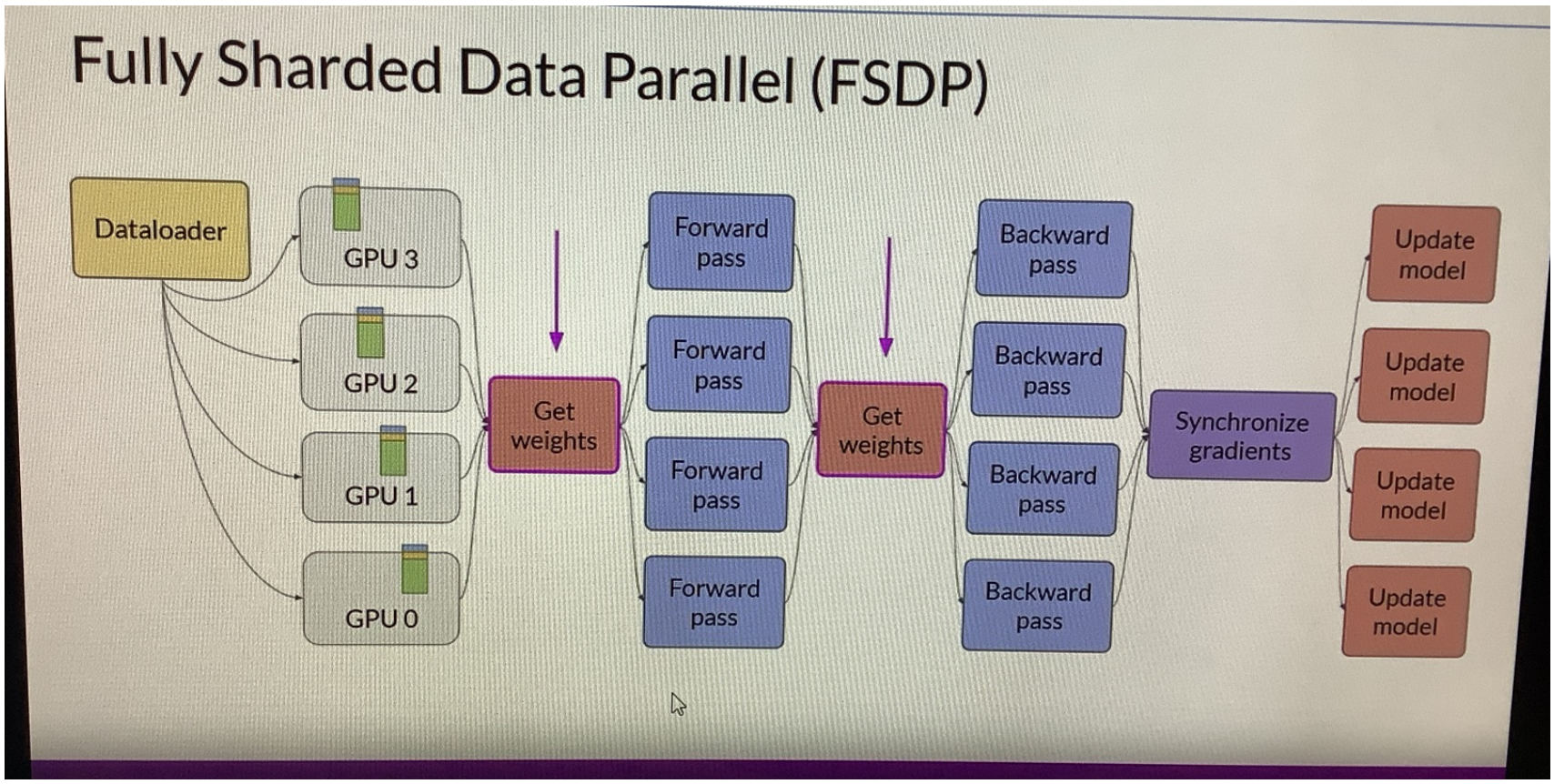
1. Distributed Data Parallel (DDP)

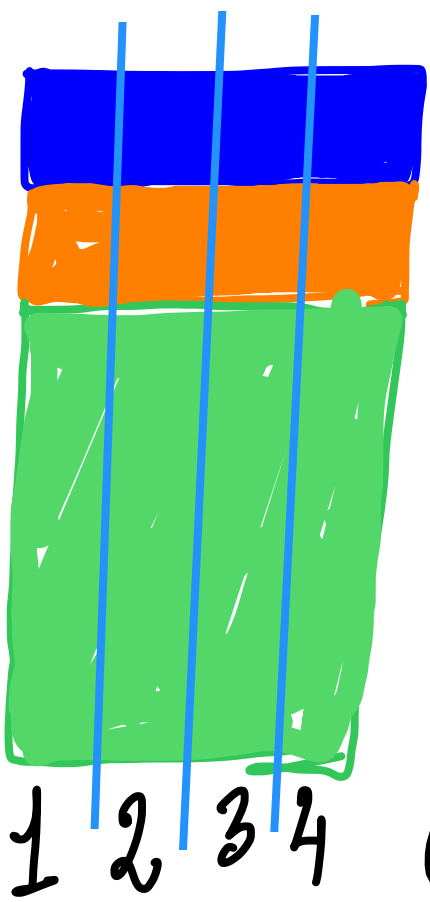
Assuming Model can be fit on single GPU.



[Parallel Compute Result Faster Training].

Fully Sharded Data Parallel : (FSDP)





→ Parameters
→ Gradients
→ Optimizer States.

1 2 3 4 (GPU 1, 2, 3, 4)

In case of FSDP we distribute Data & Model parameters both across the GPUs. FSDP

Require to collect the data before forward / Backward pass.

Scaling laws & Compute Optimal

Chinchilla Paper :- Compute

Optimal training data size is ~20x number of parameters.

Model	# of Params	# Compute Optimal Token
Chinchila	70B	~1.4T

Optimal Token ≈ 20x times of no of Parameters.

Domain Adaption Pre-Training

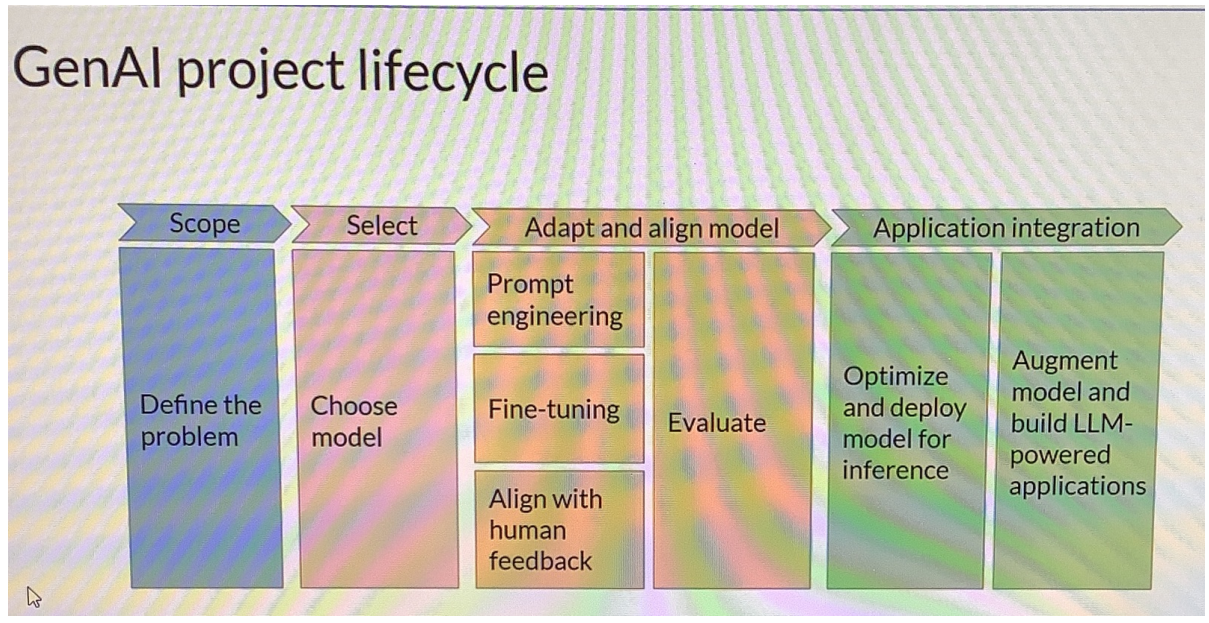
Model need specific domain training (Finance, Medical) etc.

Bloomberg GPT: domain Adaption for Finance: Bloomberg GPT was train on ~51% (financial public & private) & 49% (other) public.

{: Read ⇒ Bloomberg GPT: A Large Language Model for Finance.}

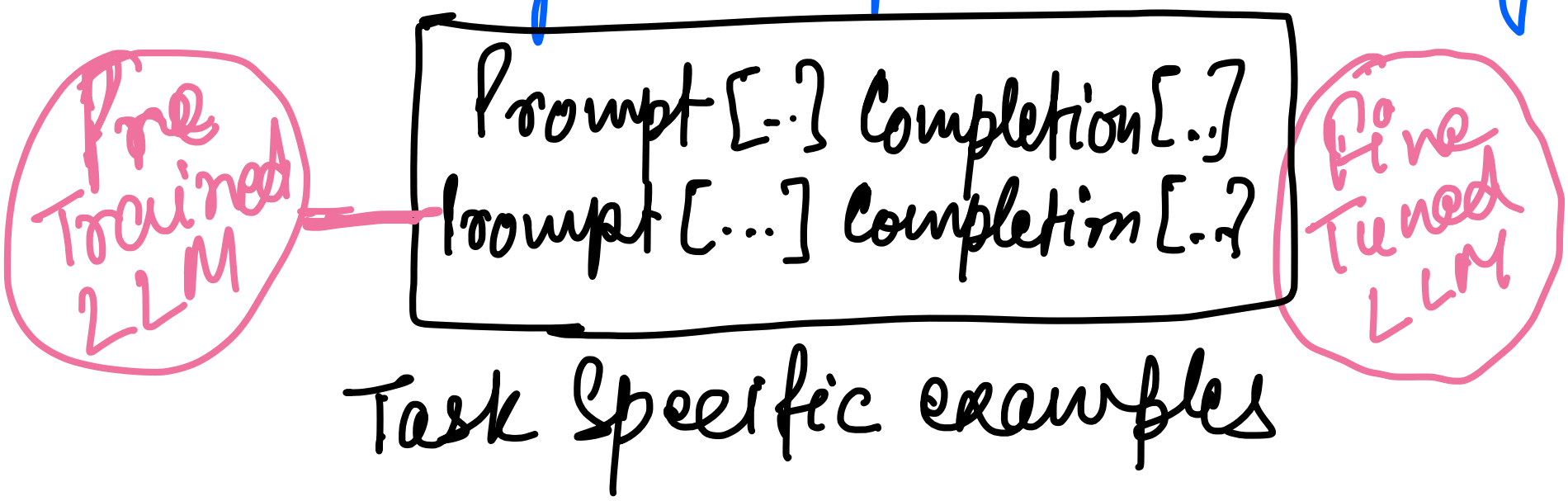
WEEK-02

Gen AI life cycle



Fine Tune LLM with Instruction Prompt:

Fine tuning is Supervised learning



Using Prompts Fine Tune LLM

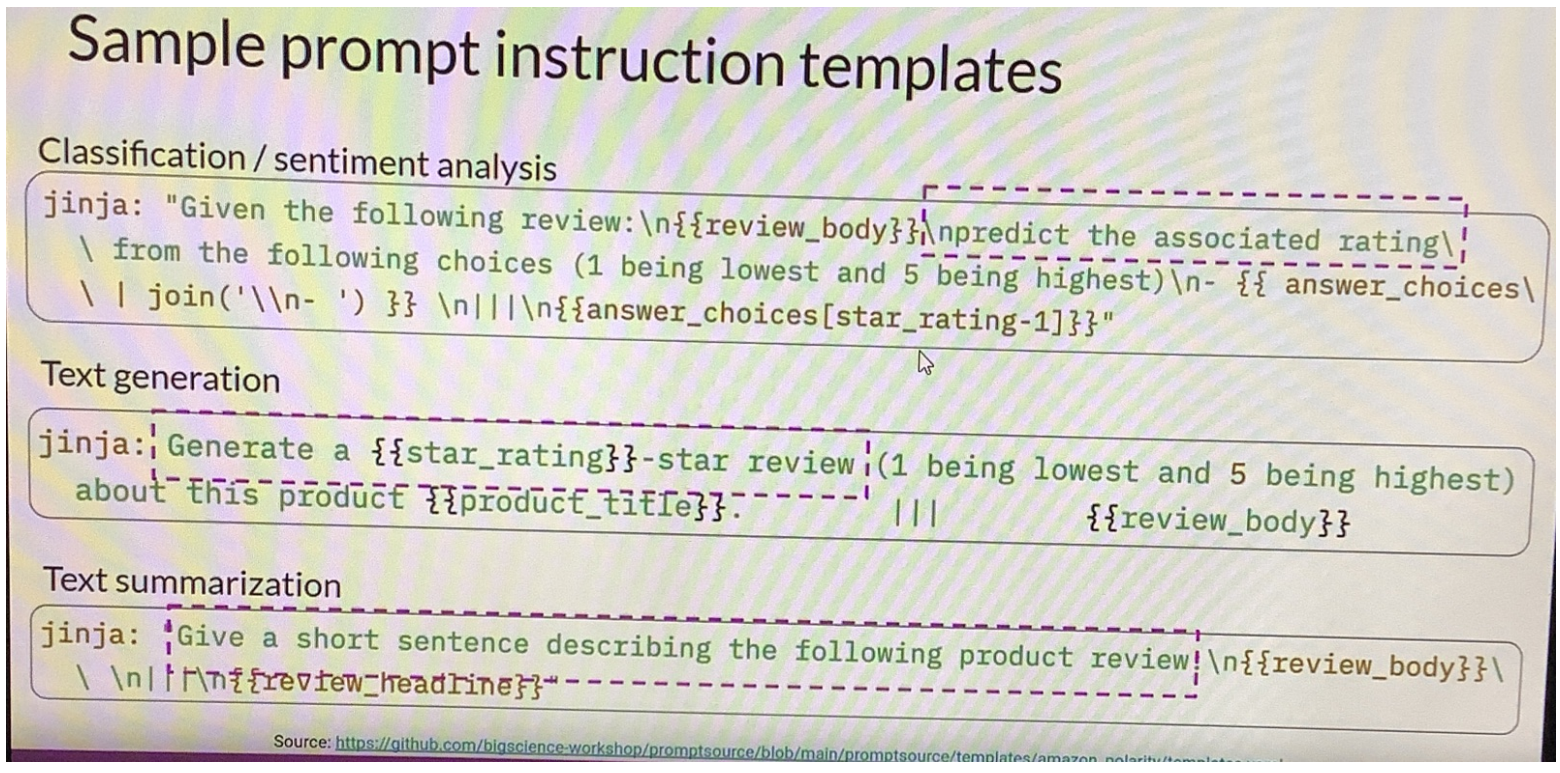
Prompt
Example Text
Example Completion

{ ® Summarize the following text
® Example Text
® Example Completion. }

⇒ In this process all of the model weights are updated. Full fine tuning require memory like Pre-Training processes.

Data set Preparation

Below is the sample of 'Prompt instruction templates'



Most of the time Fine Tuning means instruction Fine Tuning.

Fine Tuning on a Single Task

We can fine tune a Pre-Trained Model need to perform 'A Single Task' with [500 - 1000] examples with a good performance. In contrast Billions of Token model saw during pre-training.

Down side ⇒ Catastrophic Forgett

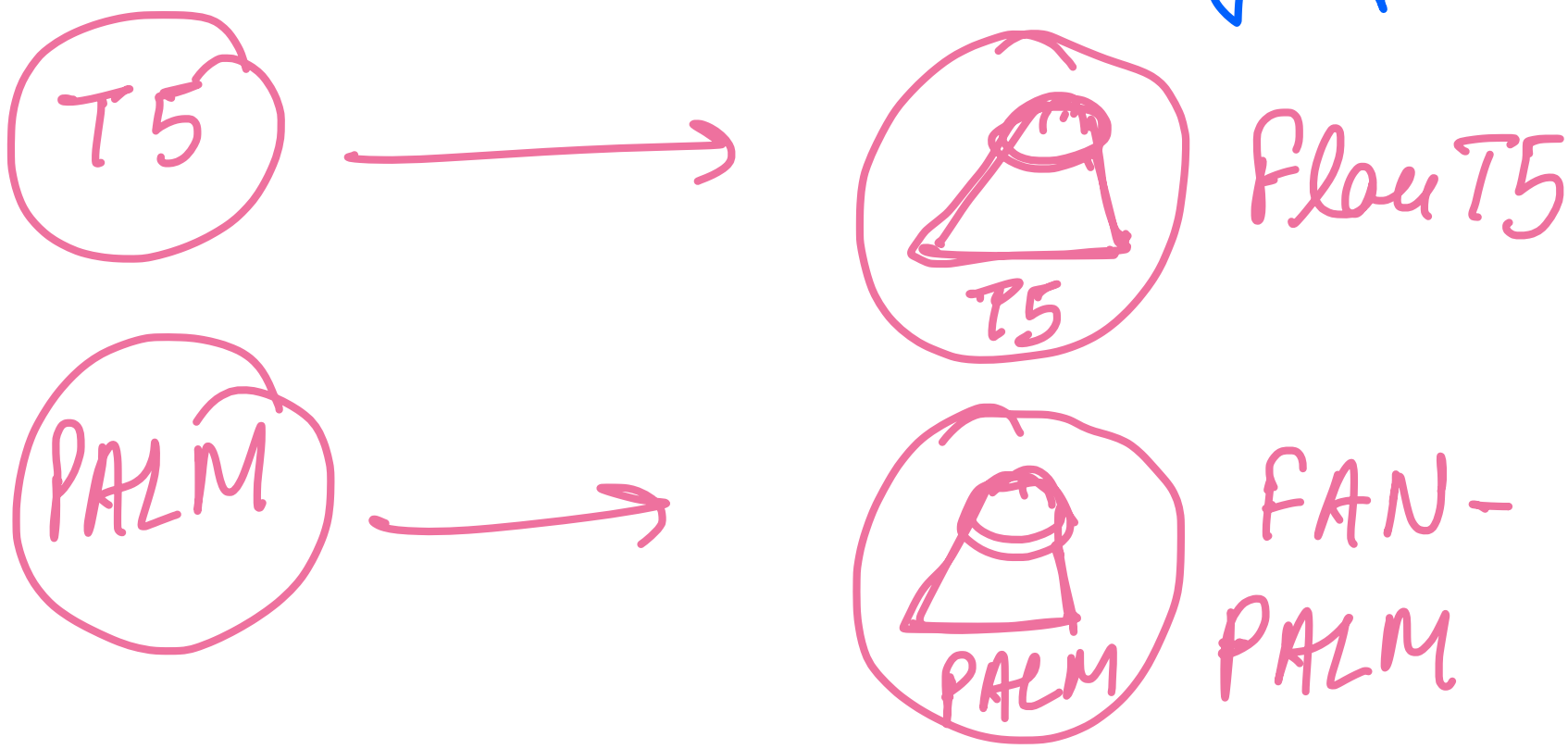
* Fine tuning can significantly increase model performance on specific task. It can de-grade performance on other task.

How to avoid Catastrophic forgetting

- We might don't care about this
- Fine-tune on multiple tasks at the same time.
- Consider PEFT (Parameter Efficient Fine-Tuning)
  - ⇒ Set of Techniques that preserves the weight of original LLM and train specific Adapter Layer which are task specific.

Instruction fine-tuning with FLAN !

FLAN (fine-tuned LAnguage Net)



FLAN Paper :⇒ Scaling Instruction-finetuned Language Models.

LLM Evaluation Metrics :-

1. ROUGE
2. BLEU SCORE

ROUGE :- 1. Use for text summarization.

2. Compares a summary to one or more reference summaries

BLEU :- 1. Text Generation

2. Compares to human-generated translations.

Rouge-1 = F1 score of Rouge-1

1 ⇒ Unigram Matches

Rouge-2 = F1 score of Rouge-2

2 ⇒ Biagram Matches.

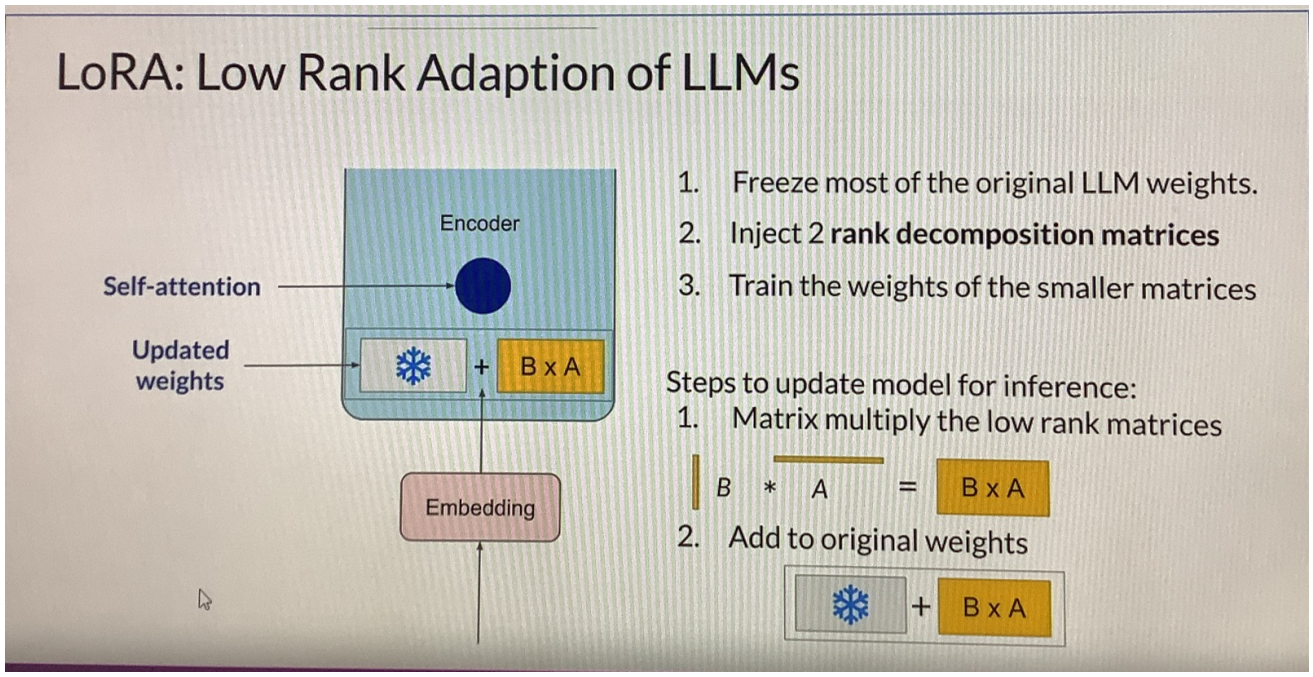
BLEU Metric: Avg. (precision across range of n-gram sizes)

* Rouge & BLEU are basic metric these should not be our final metric to bench mark a LLM Model.

More Complex Metrics :-

1. GLUE 2. Super GLUE
3. HELM 4. MMLU (Massive Multi task Language Understanding)
5. BIG- BENCH.

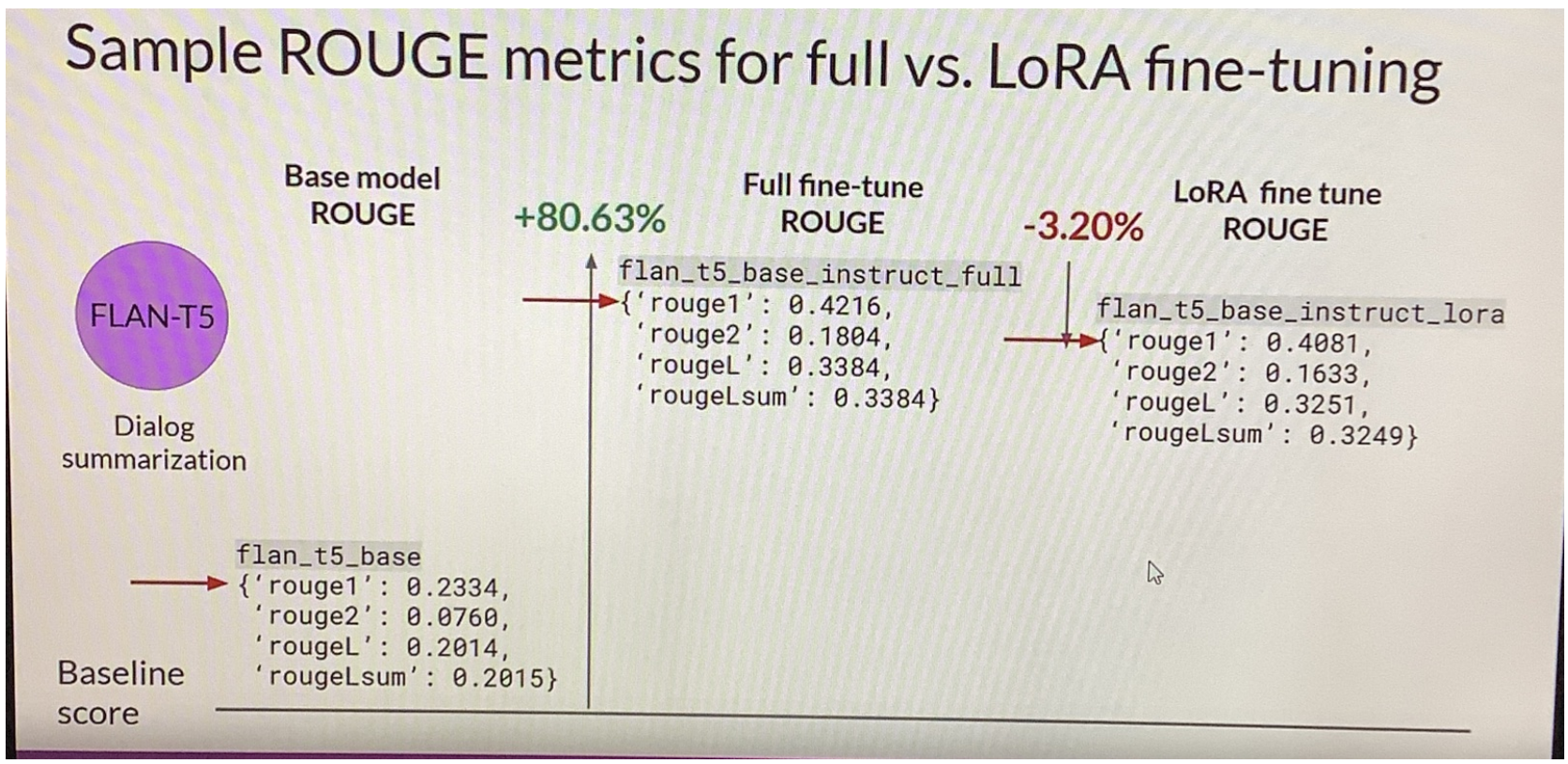
PEFT (LORA)



LORA introduces TWO LOW Rank

(r) typically 4, 8, ... 64. We train smaller matrices only. This requires very small matrices two store and train

Comparison of Base VS Full Tuning



Vs LoRA Finetuning Dialog Summarization.

Choosing LoRA Rank :=

Choosing the LoRA rank

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

Bold are the best score in each category.

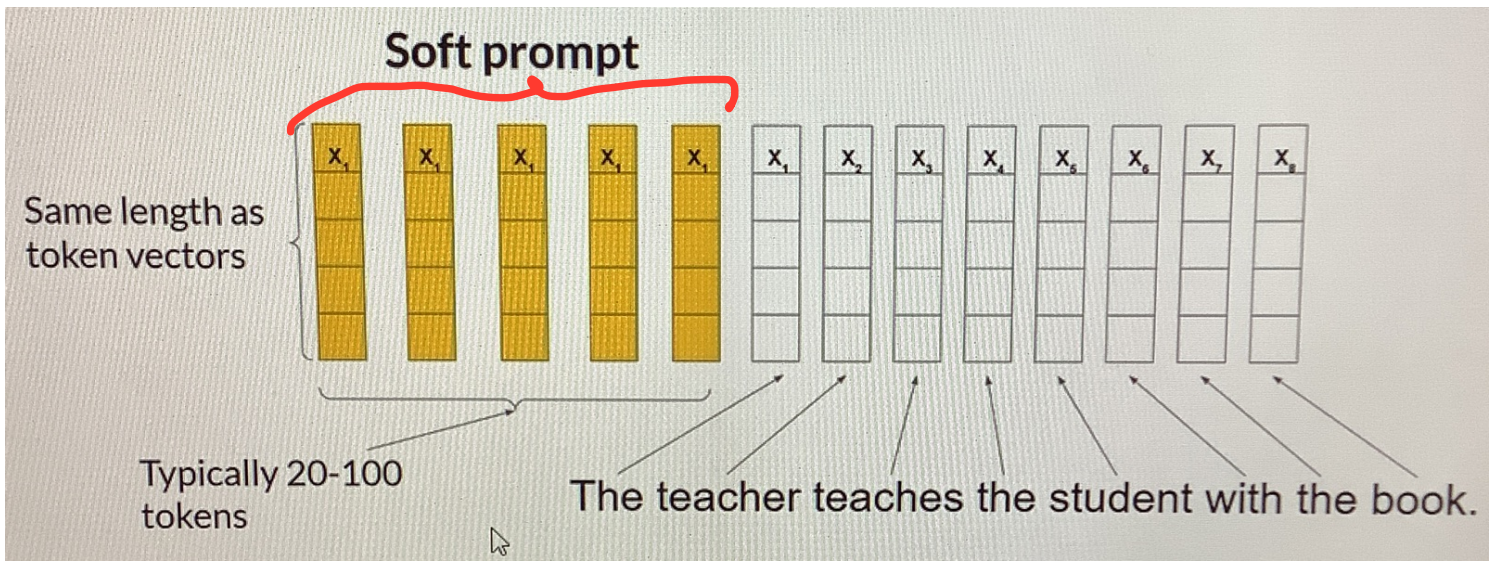
We can see 8 or 16 Rank nothing much improvement on those metrices.

Hence Rank (4 → 32) good tradeoff.

PEFT (Prompt Tuning)

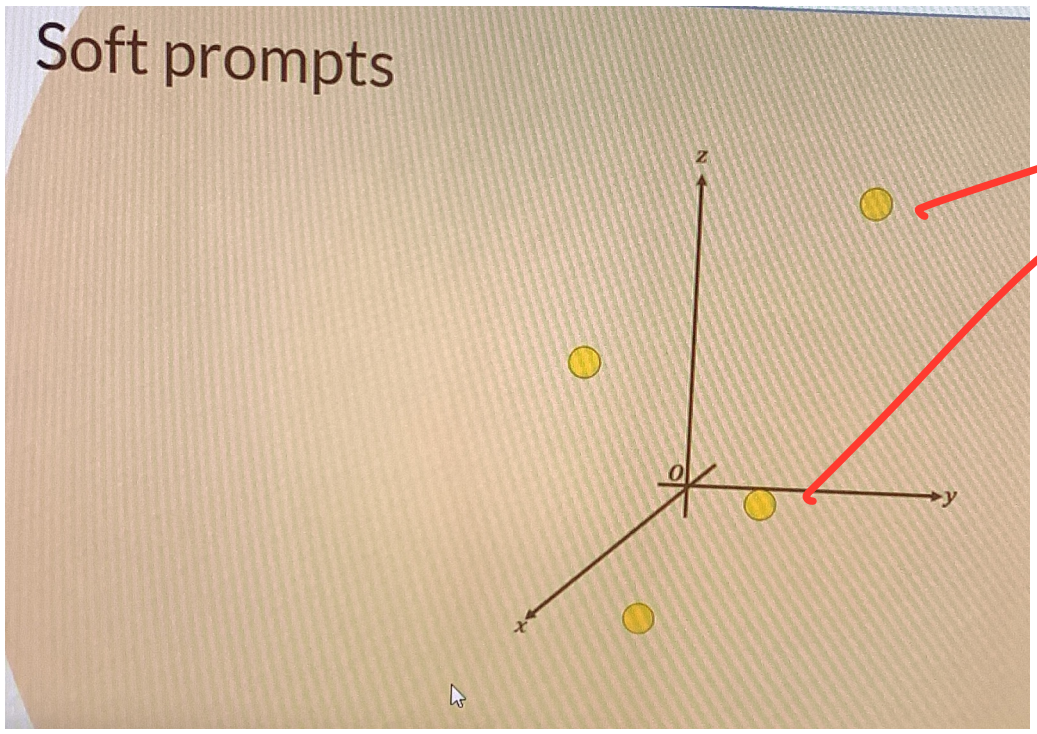
Prompt Tuning is NOT Prompt Engineeg.

like Zero Shot of Few Shot inference. Prompt tuning adds additional trainable token 'Soft Prompts".



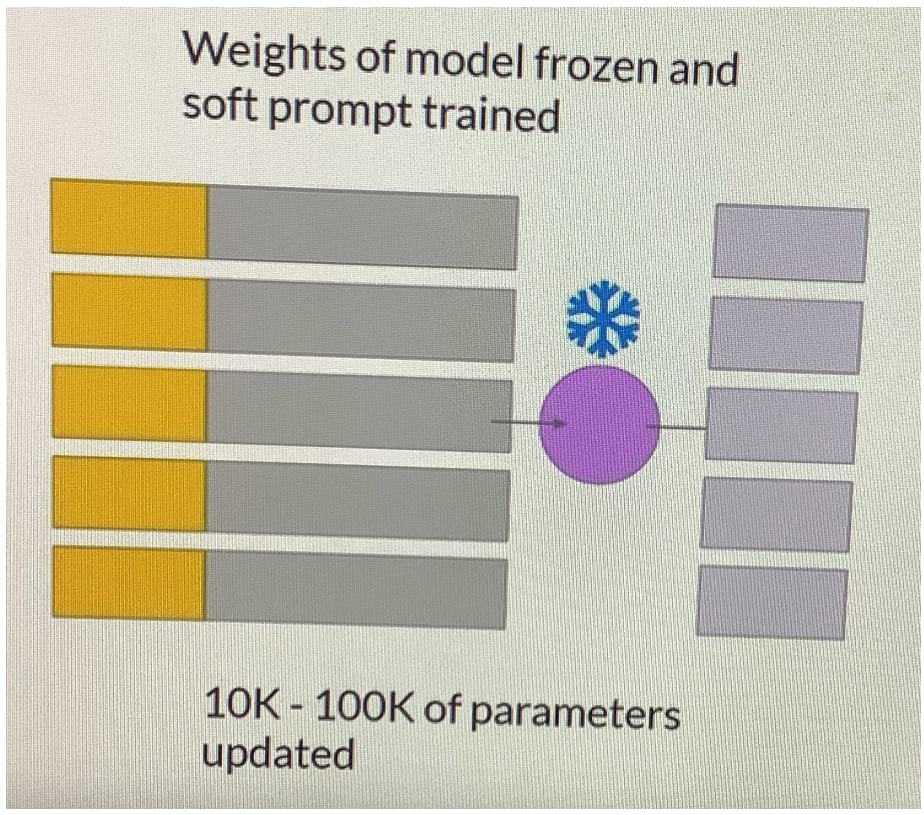
Typically 20-100 tokens are enough

Soft prompts can be think of as Virtual Tokens that can take any value in Multidimension Embeding Space. Via Supervised Learning model learns about these toffees.

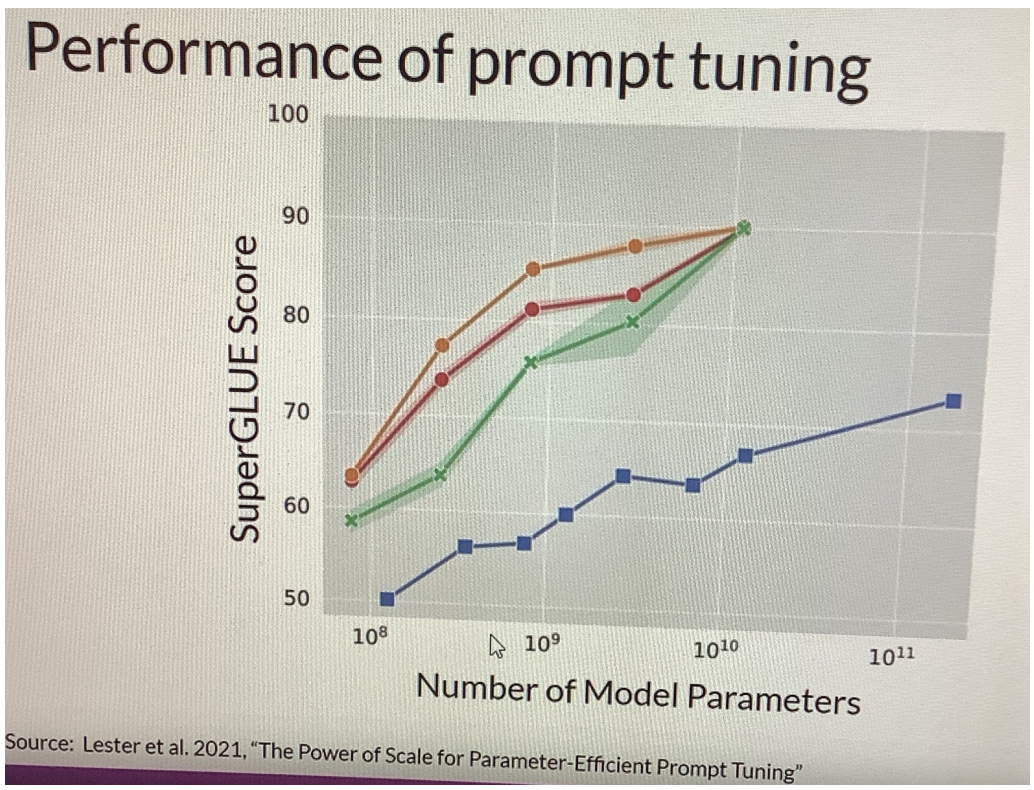


Virtual Tokens

Soft Prompt Training

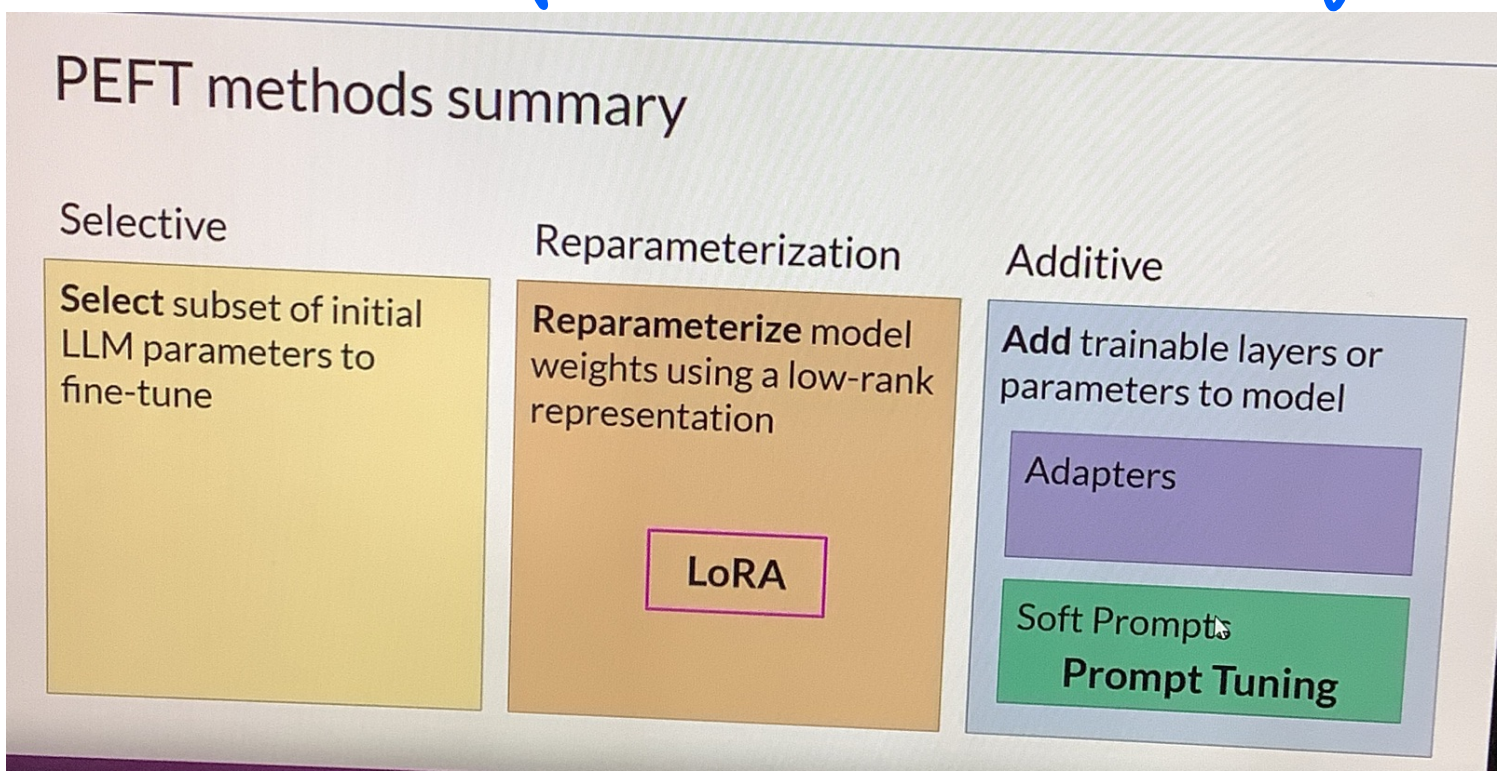


Performance Prompt Tuning :



- Multi Task Fine Tuning
- Full Fine Tuning
- x Prompt Tuning
- Prompt Engg.

As the model size increase so the performance of Prompt Tuning.



Week 2 Resources :-

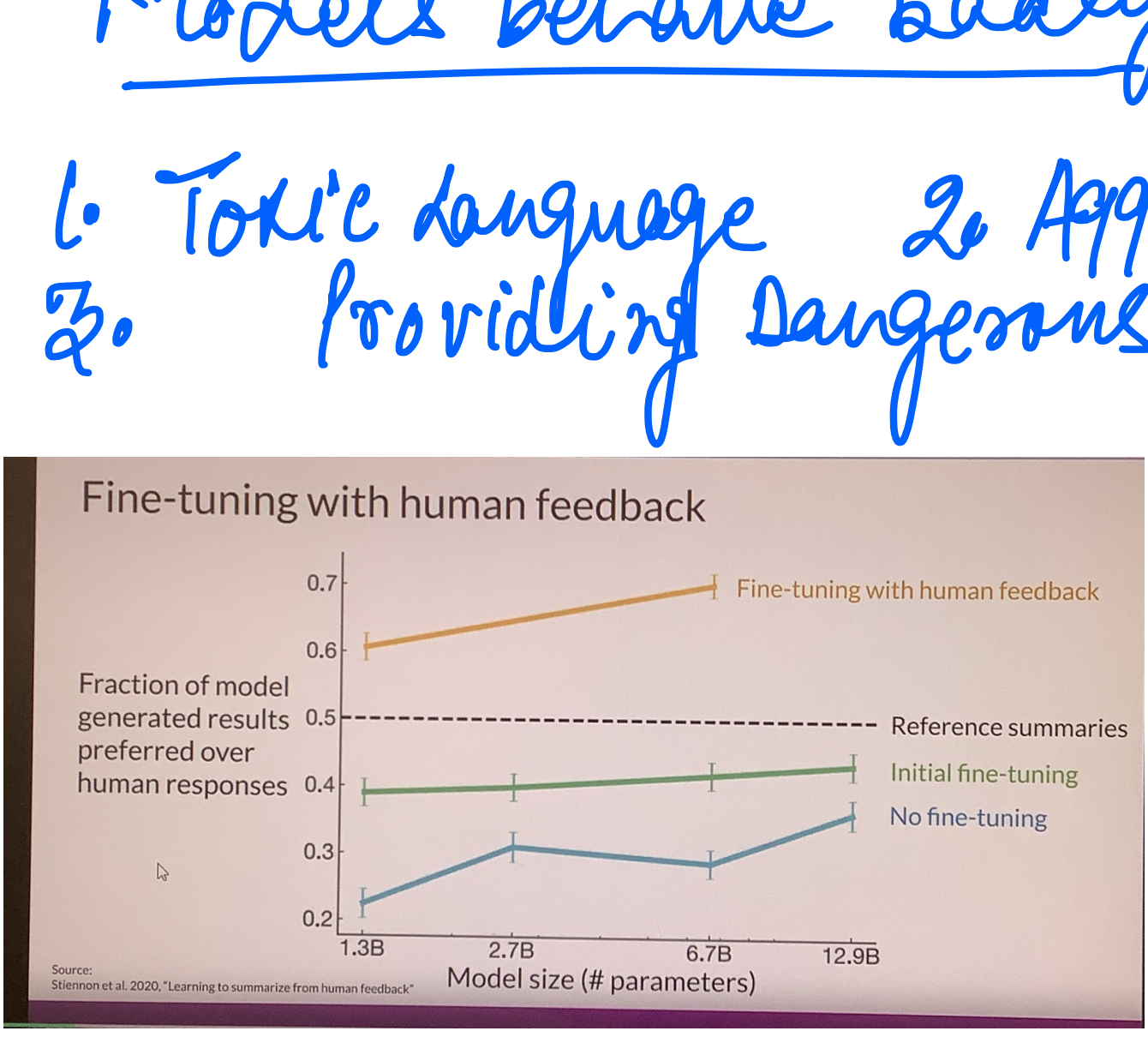
1. Scaling Instruction - finetuned LM
2. Introducing FLAN: More generalizable Language Models with Instruction Fine Tuning.
3. Scaling down to scale up: A Guide to PEFT.
4. On the Effectiveness of Parameter Efficient Fine Tuning.
5. LoRA & QLoRA
6. The power of Scale for Parameter-Eff Prompt Tuning.

Week - 03

Reinforcement learning from Human feedback

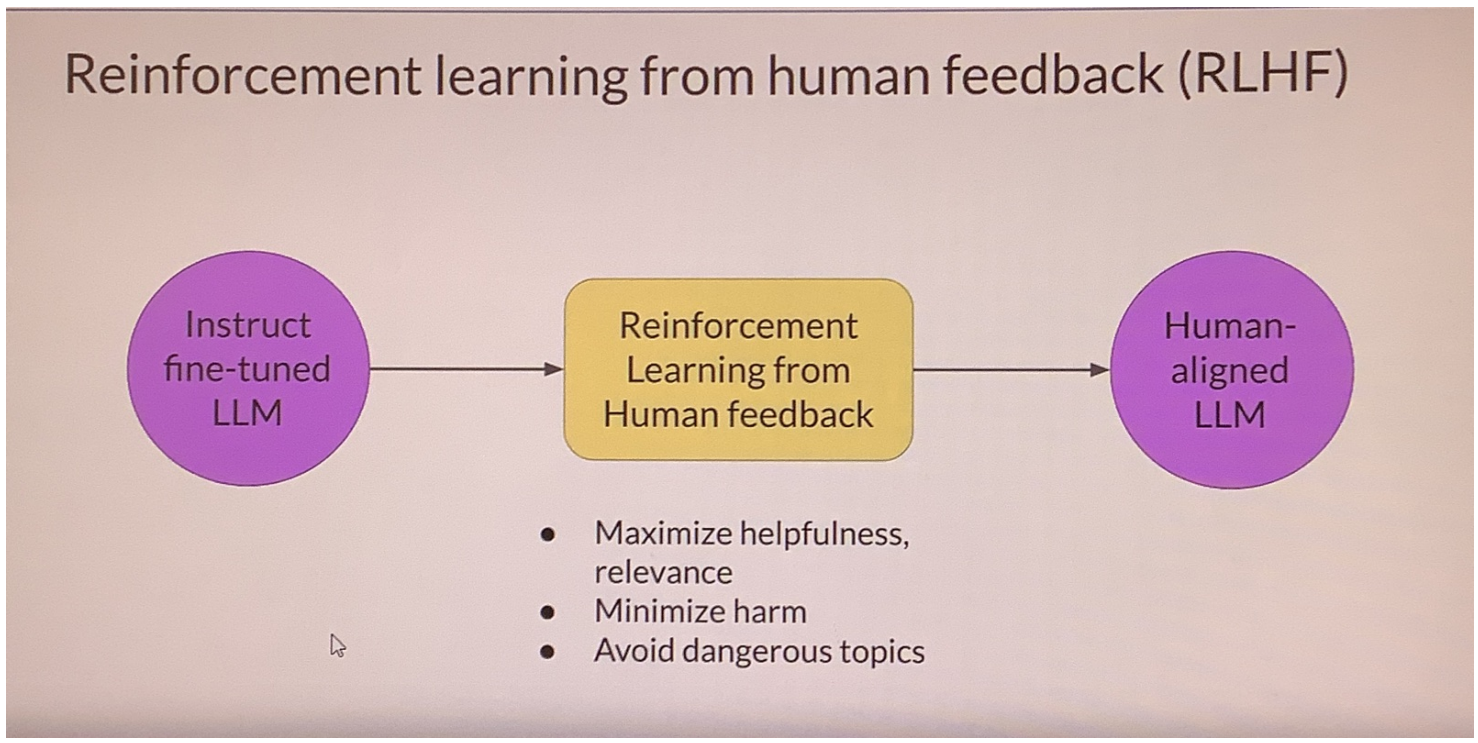
Models behave badly :-

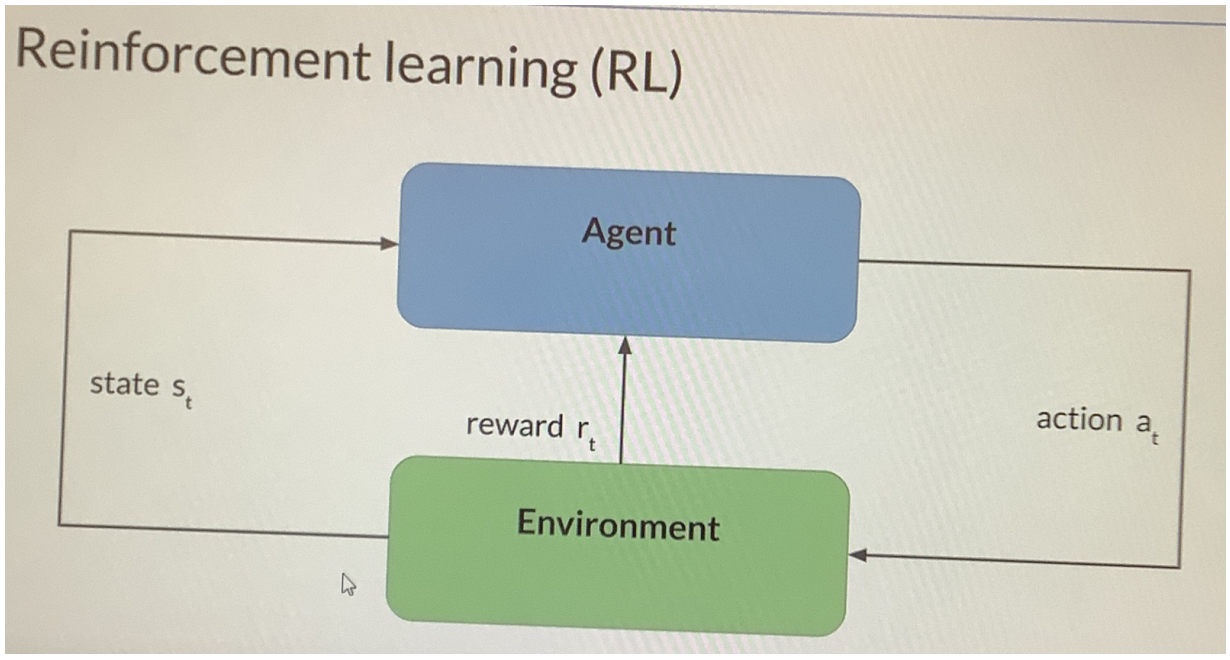
1. Toxic language 2. Aggressive Response
3. Providing Dangerous Info.



⊗ As we can see fine tuning with Human feedback Exceed Reference Summaries.

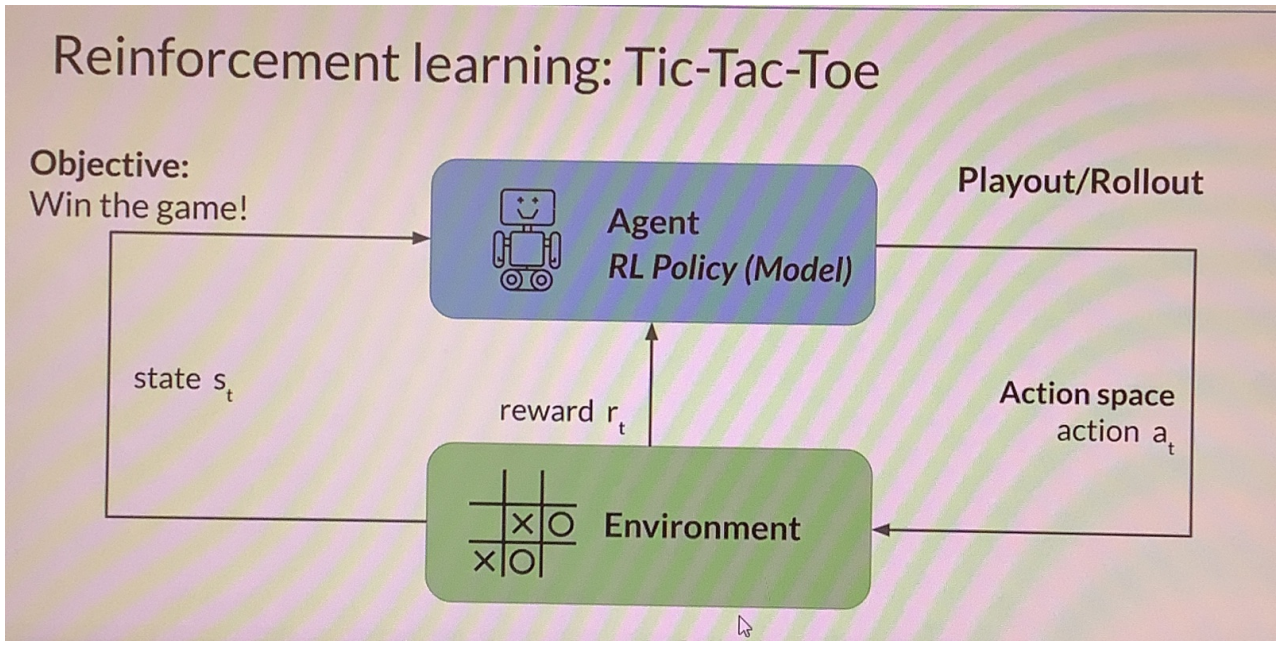
RLHF



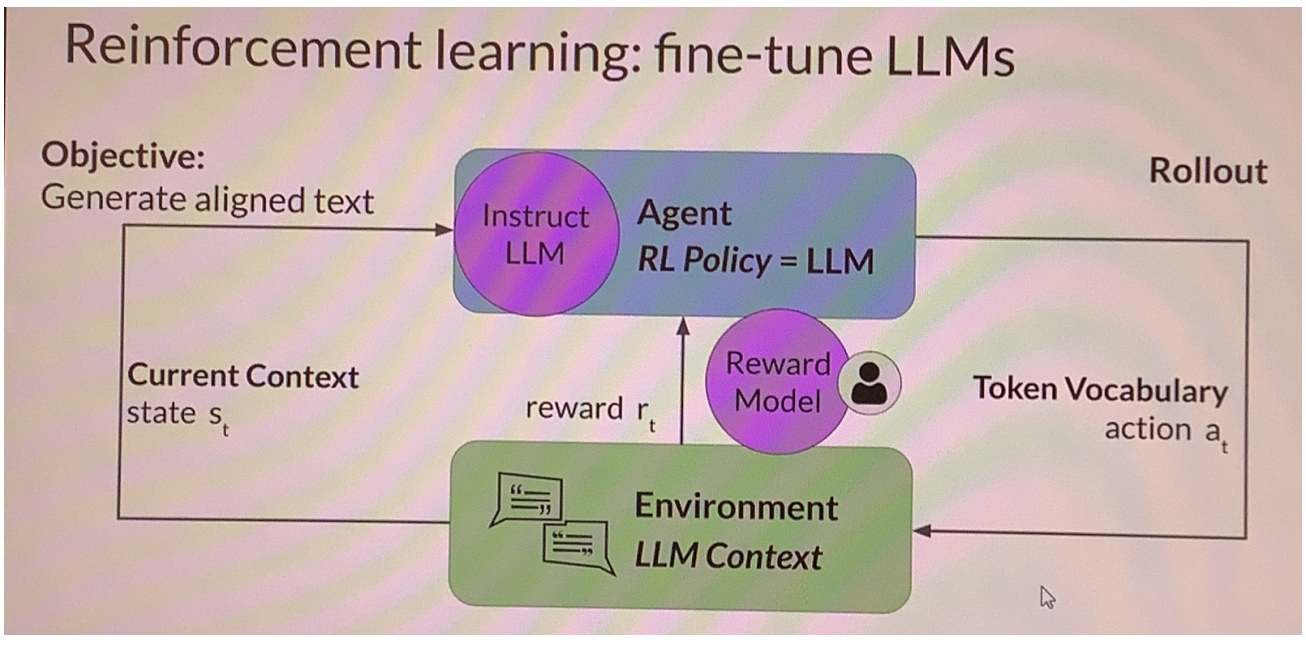


Iterating through the loop 'AGENT' define a policy for best performance over a task.

Reinforcement learning: Tic Tac Toe

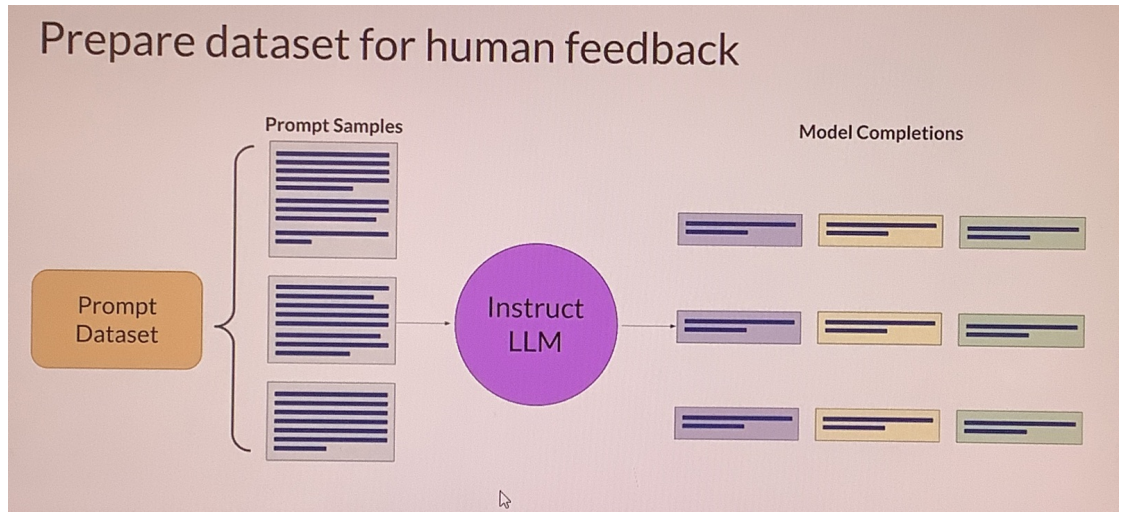


This model can be extended to generate text aligned with human (Montoric ...)

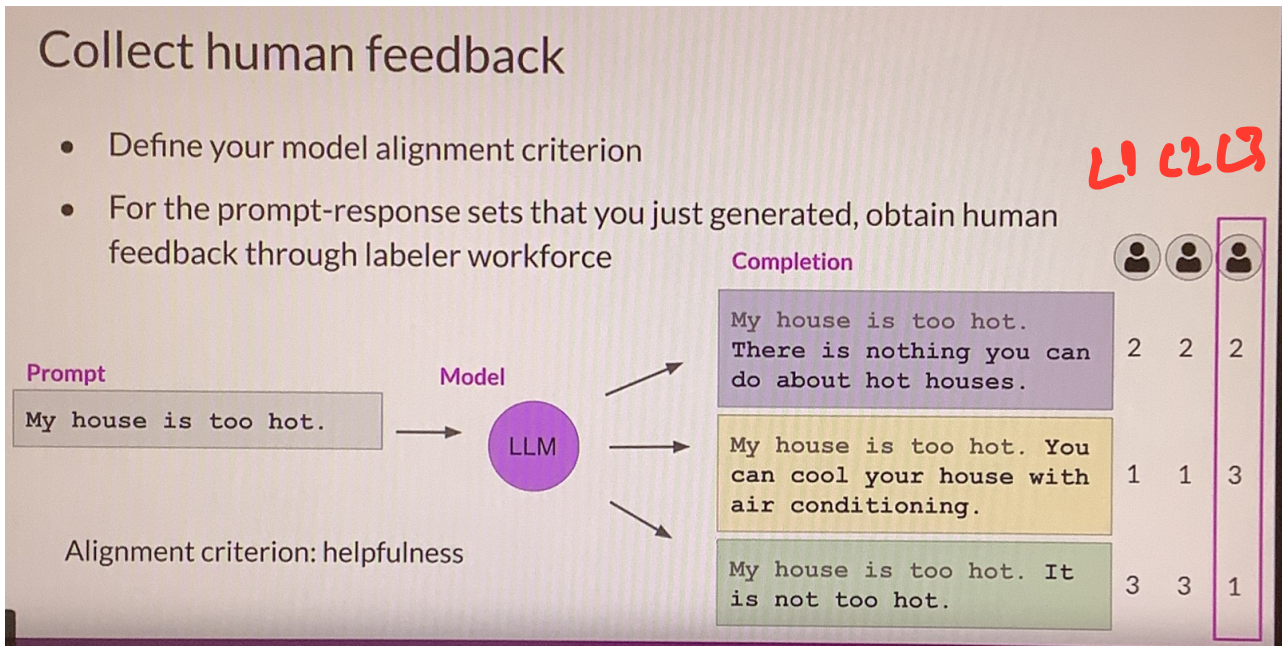


Prepare Dataset for RLHF based Training :-

1. Choose LLM model with some capability for the task.
2. Prepare Dataset with Human feedback



3. Collect human feedback on Model completion.



Human(labe-ler) feeback on the model from multiple sources. To minimise 'subjectivity'. Ex: L3 feels YELLOW Response is worst. Hence we have should have very clear & objective set of instructions for labelling.

Sample Instructions for Humans Labelers.

Sample instructions for human labelers

* Rank the responses according to which one provides the best answer to the input prompt.

* What is the best answer? Make a decision based on (a) the correctness of the answer, and (b) the informativeness of the response. For (a) you are allowed to search the web. Overall, use your best judgment to rank answers based on being the most useful response, which we define as one which is at least somewhat correct, and minimally informative about what the prompt is asking for.

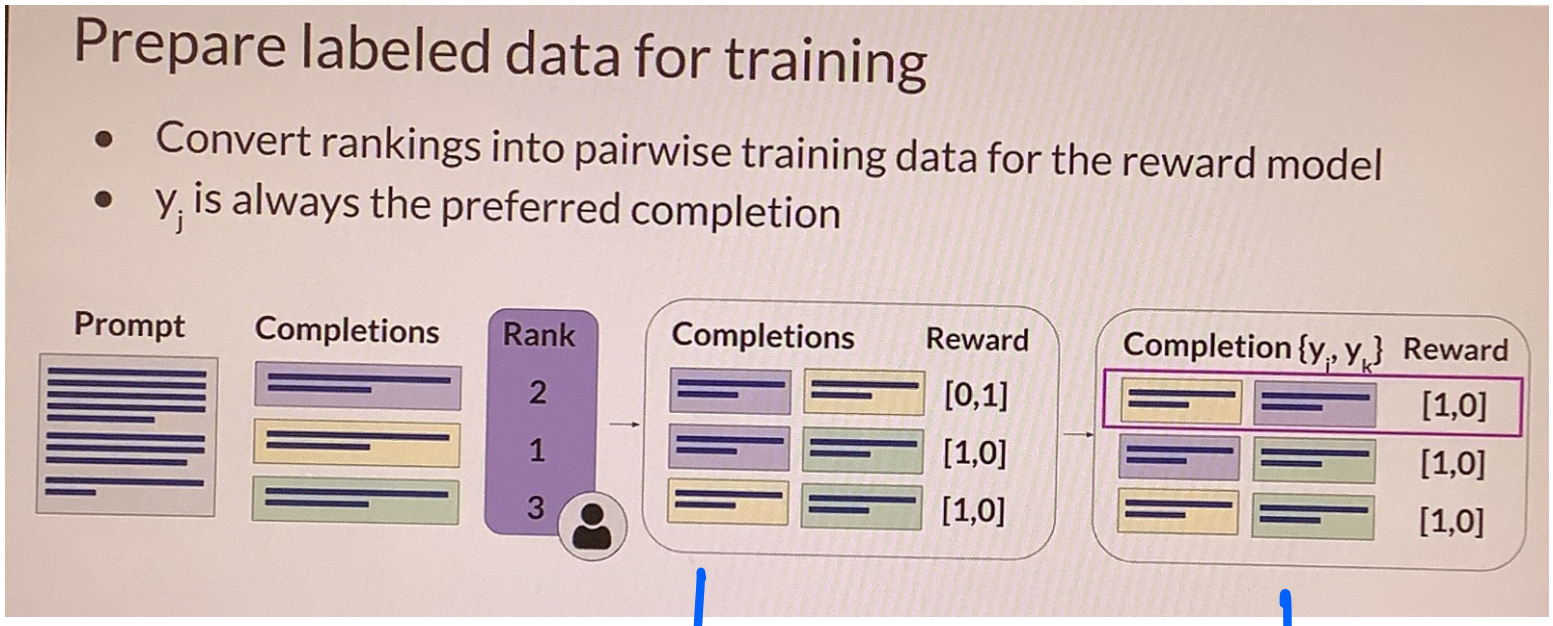
* If two responses provide the same correctness and informativeness by your judgment, and there is no clear winner, you may rank them the same, but please only use this sparingly.

* If the answer for a given response is nonsensical, irrelevant, highly ungrammatical/confusing, or does not clearly respond to the given prompt, label it with "F" (for fail) rather than its rank.

* Long answers are not always the best. Answers which provide succinct, coherent responses may be better than longer ones, if they are at least as correct and informative.

Source: Chung et al. 2022, "Scaling Instruction-Finetuned Language Models"

Prepared Labelled Data for Training



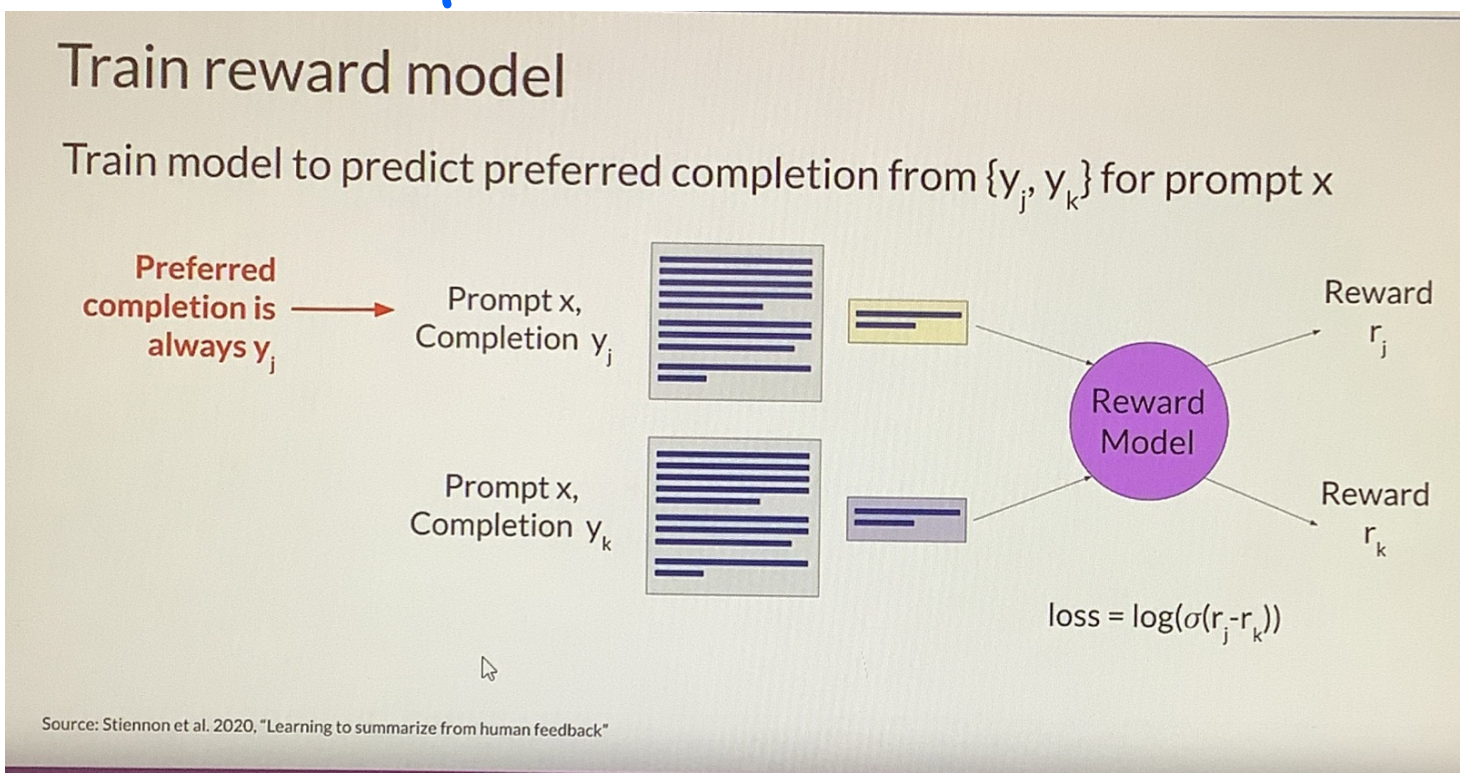
Combination of responses

Re-arrange such that Preferred Response comes first.

RLHF: Reward Model :-

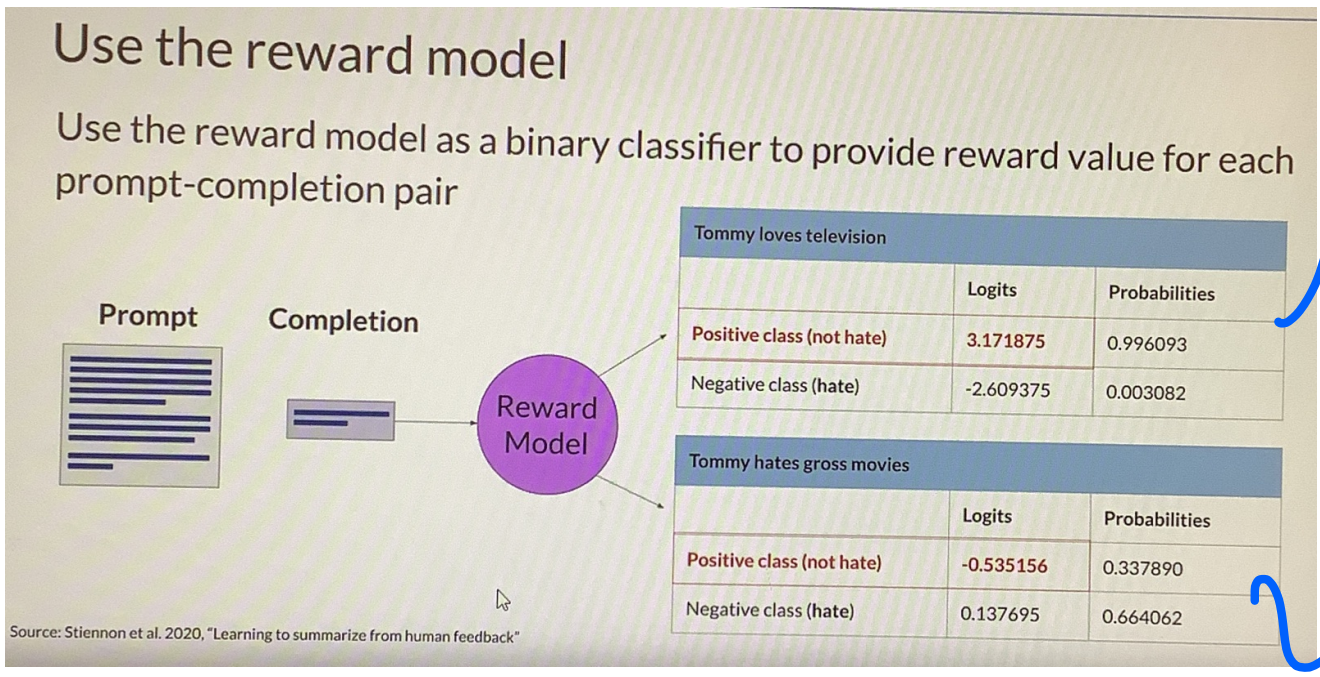
Train reward model to predict completion from $\{y_j, y_k\}$ for prompt x.

Reward model learn to favor the Human labeled completion.



Using Reward Model :-

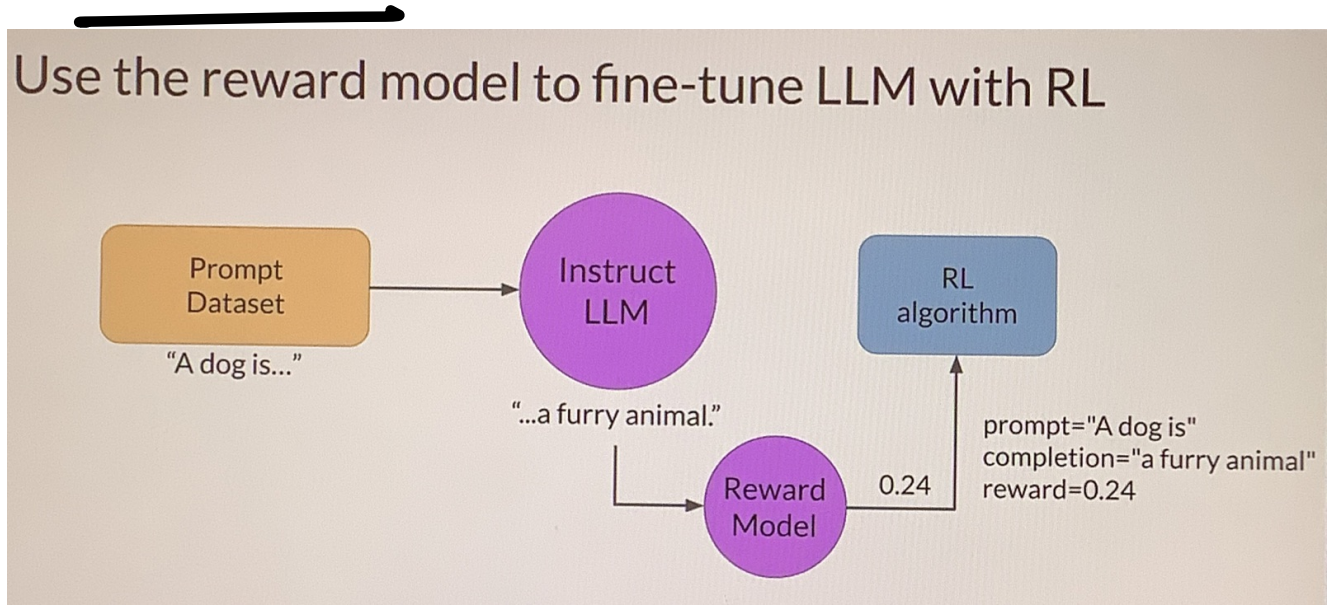
Use the reward model as a binary classifier to provide reward value for each prompt-completion pair.



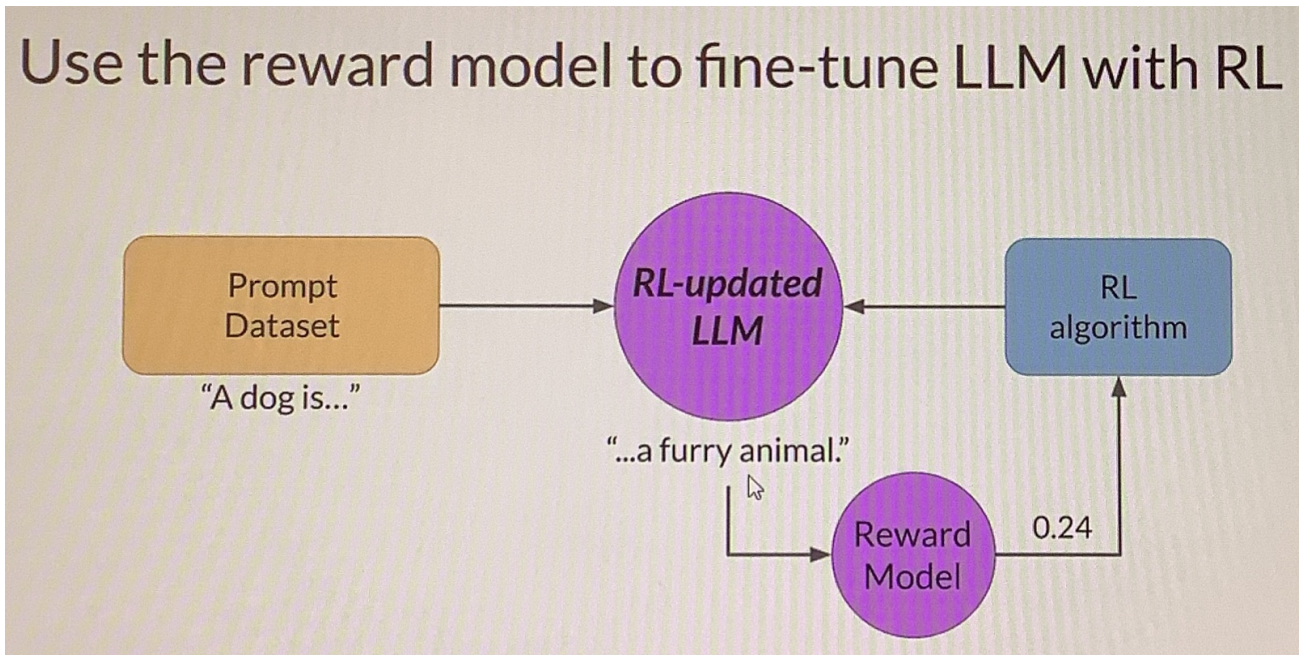
Good Reward

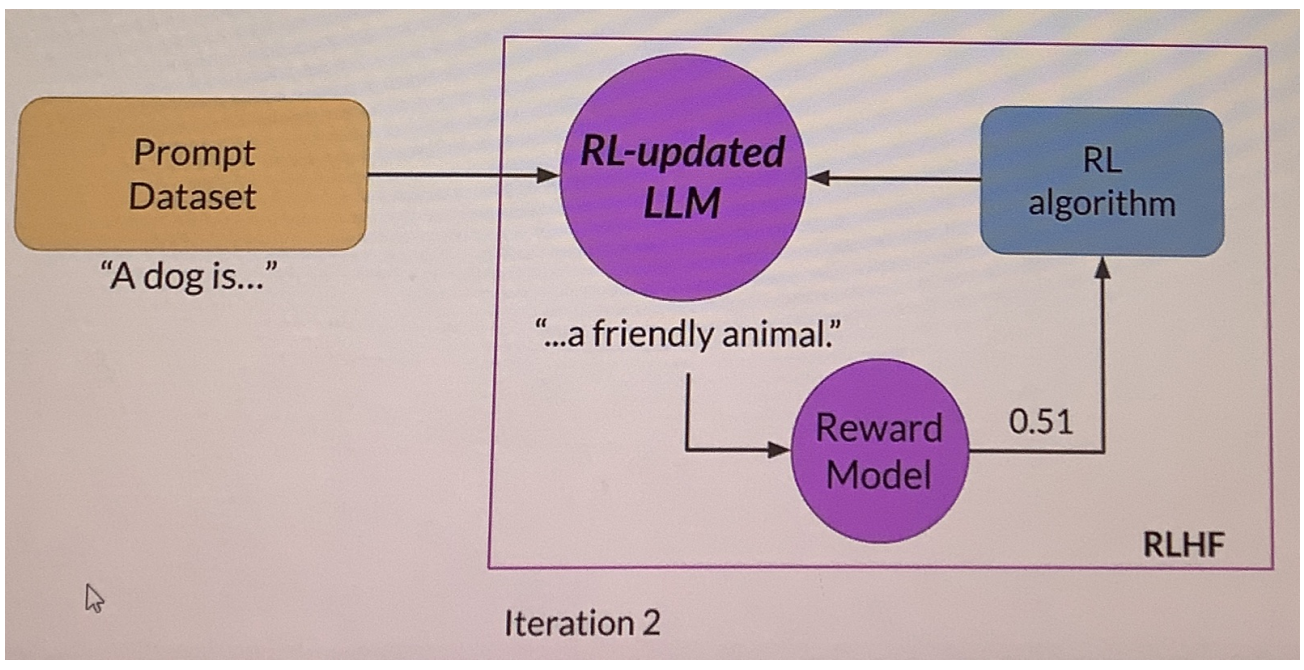
Bad Reward

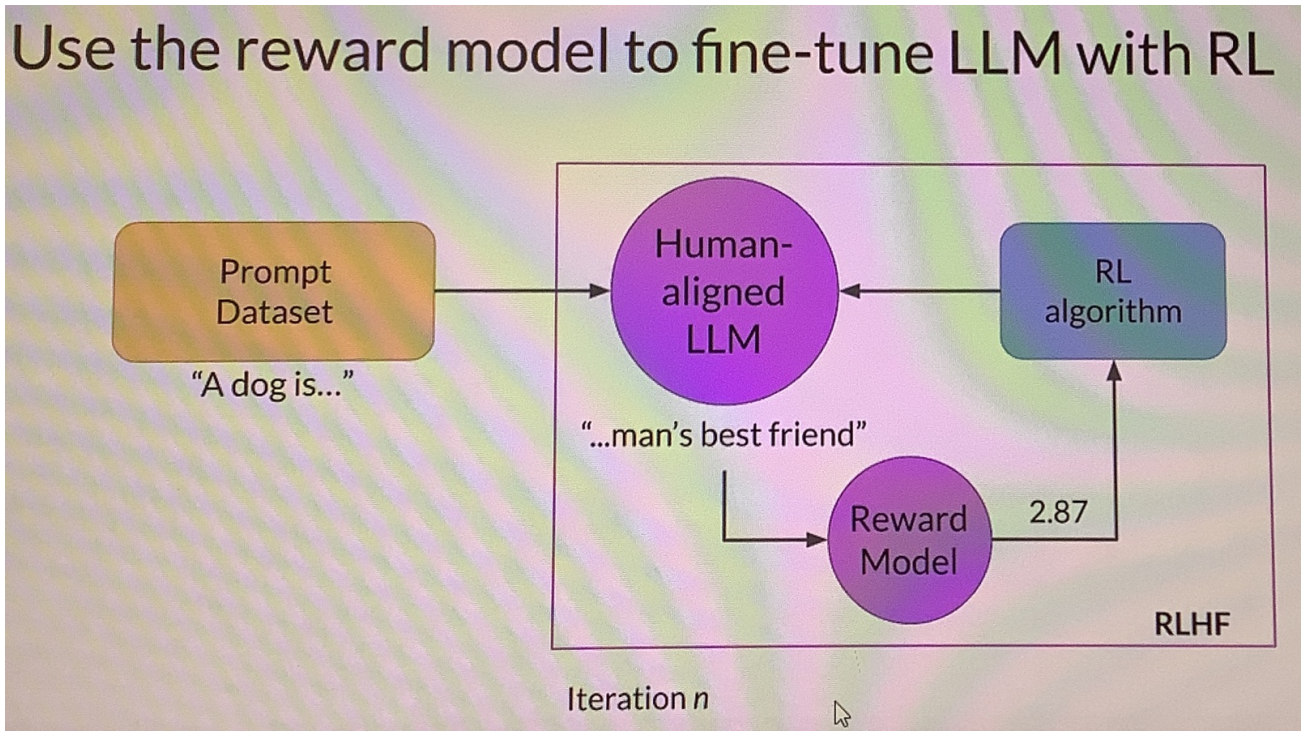
Using Reward Model to fine-tune LLM with RL :-



RL then update your LLM weights.



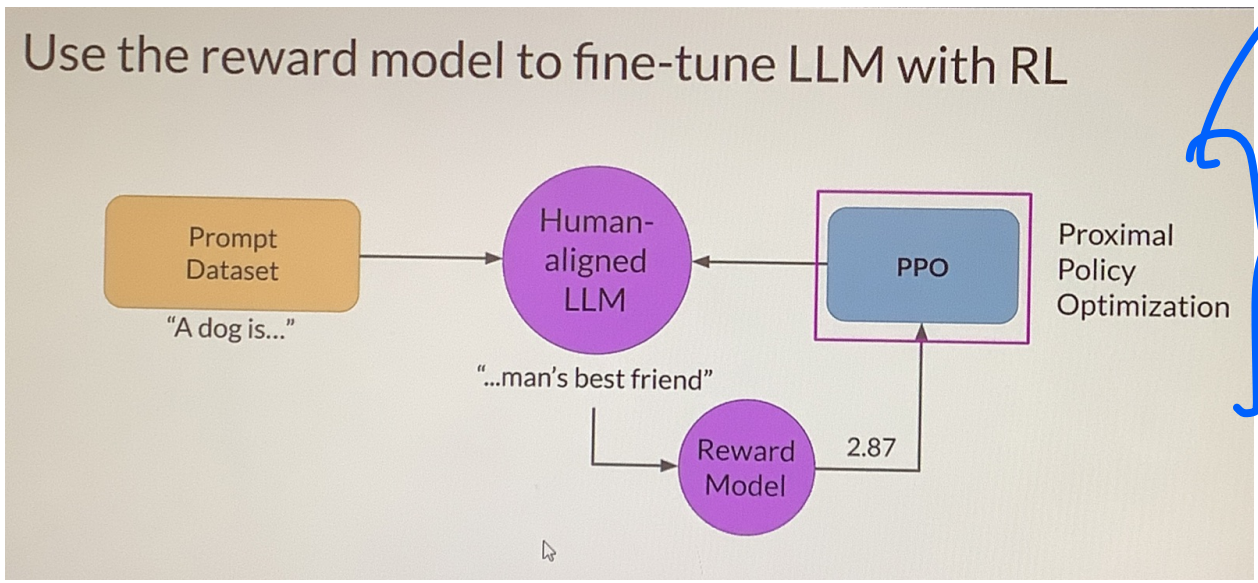




Then over multiple iterations. LLM slest receiving higher reward score.

Will continue this process until model reaches some criteria ex: Helpfullness score.

RL Algorithm Type:

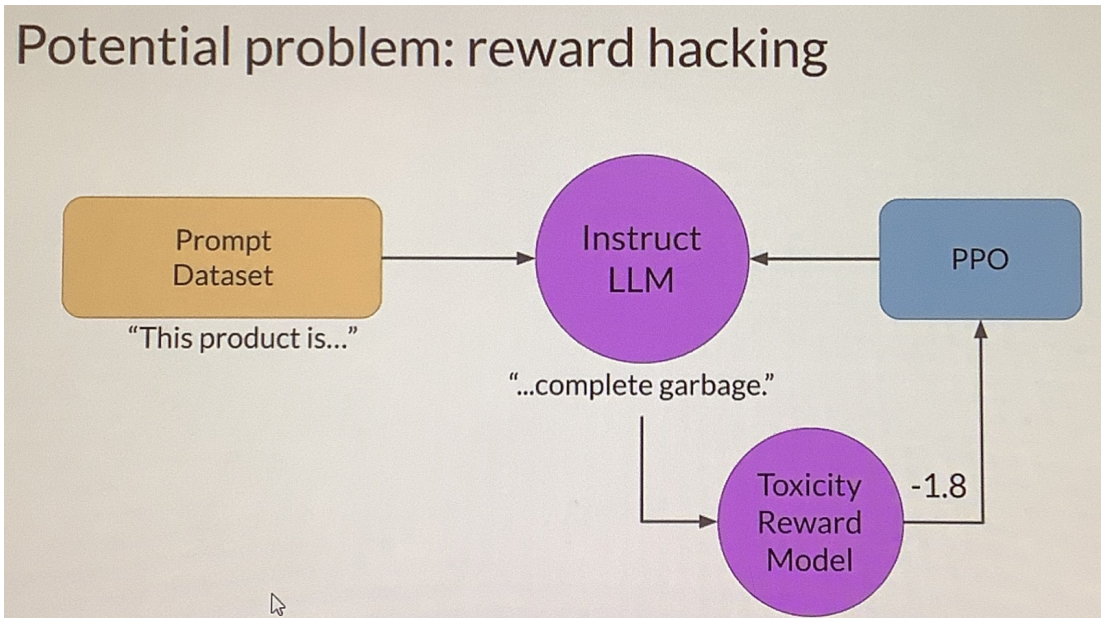


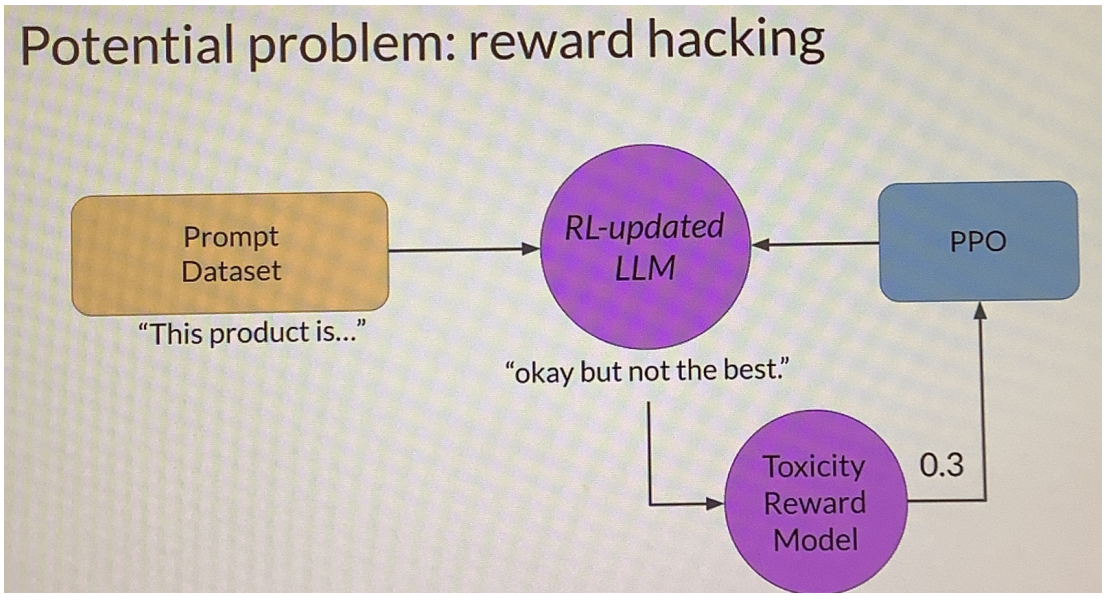
RL Algorithm

(PPO)

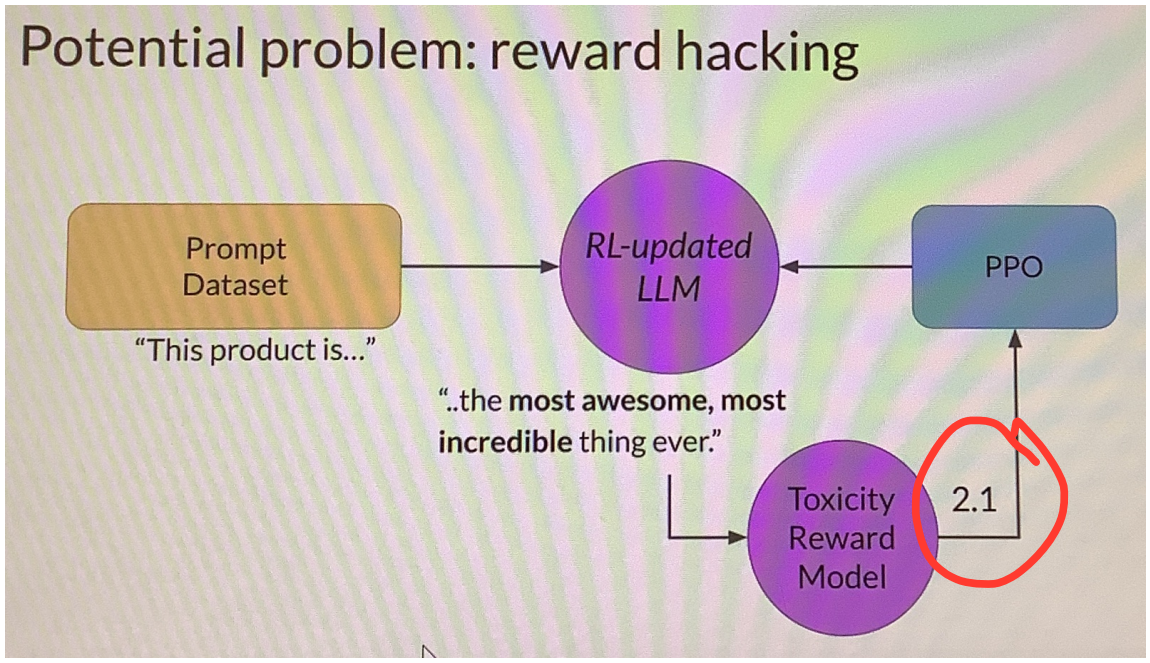
Potential problem : Reward Hacking

Agent learn to cheat the system by maximising reward. but the end Output is not that usefull.



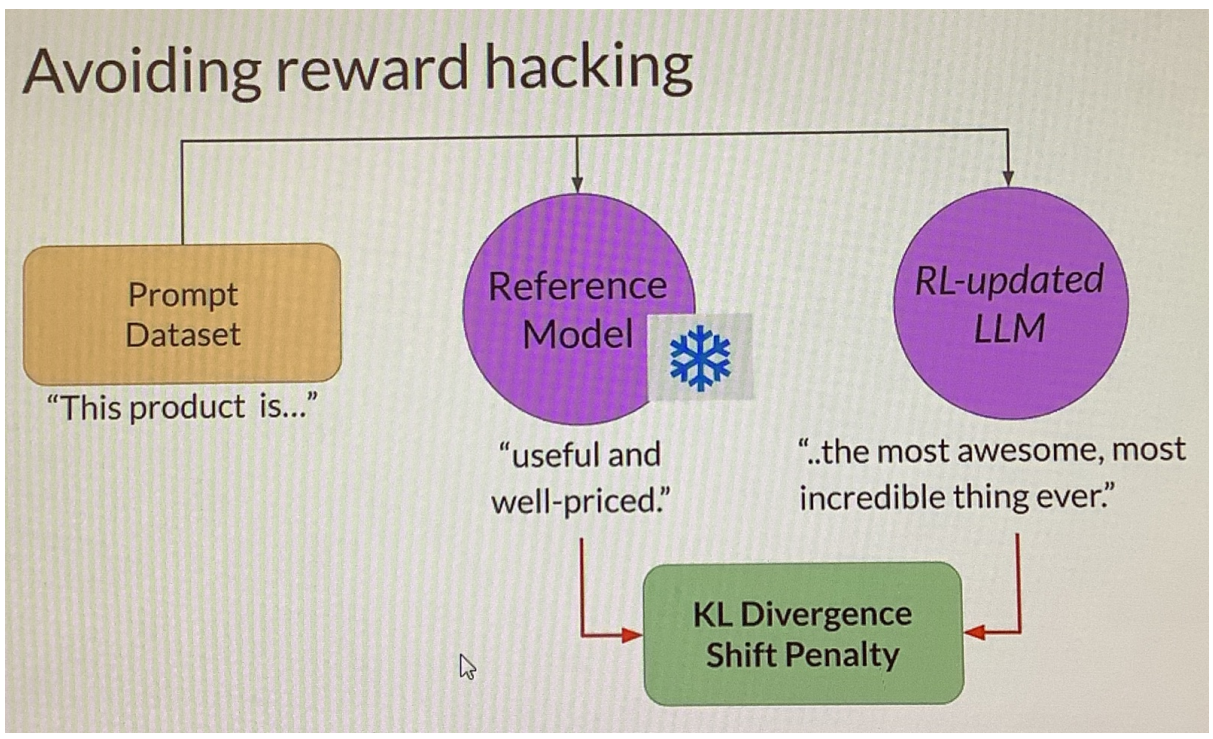


1. Toxic Response by LLM.
2. PPO algo update the weights to less toxic response.

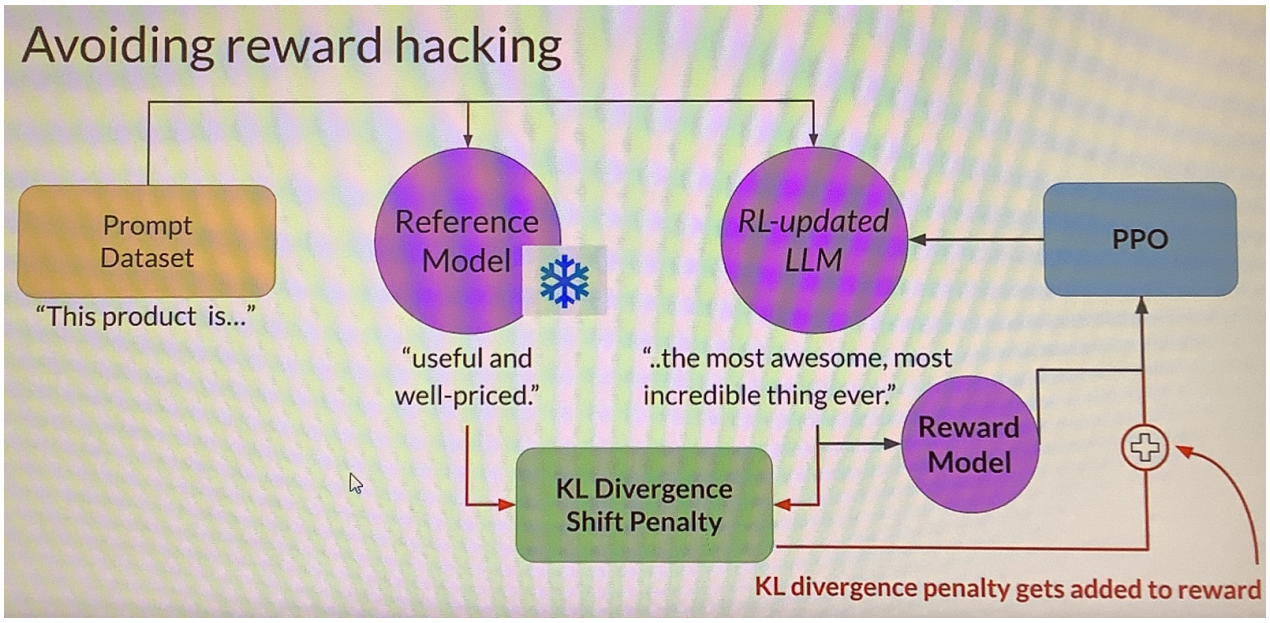


3. However as the policy try to reward more it can divert to much from initial model. Hence output is not that USEFULL any more. (Very low toxic score)

Avoid Reward Hacking :-



With KL Divergence we can find How much Updated Model has a divergence.



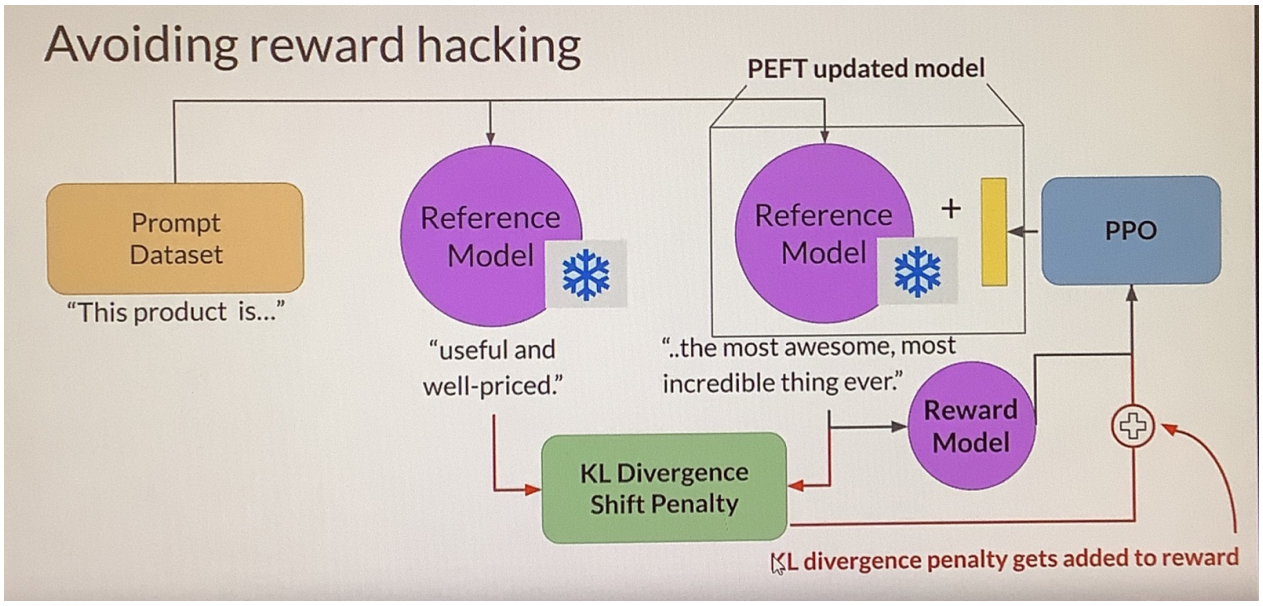
KL divergence

Score is added to Reward Model.

This will penalise if model is shifting too much from Original Model.

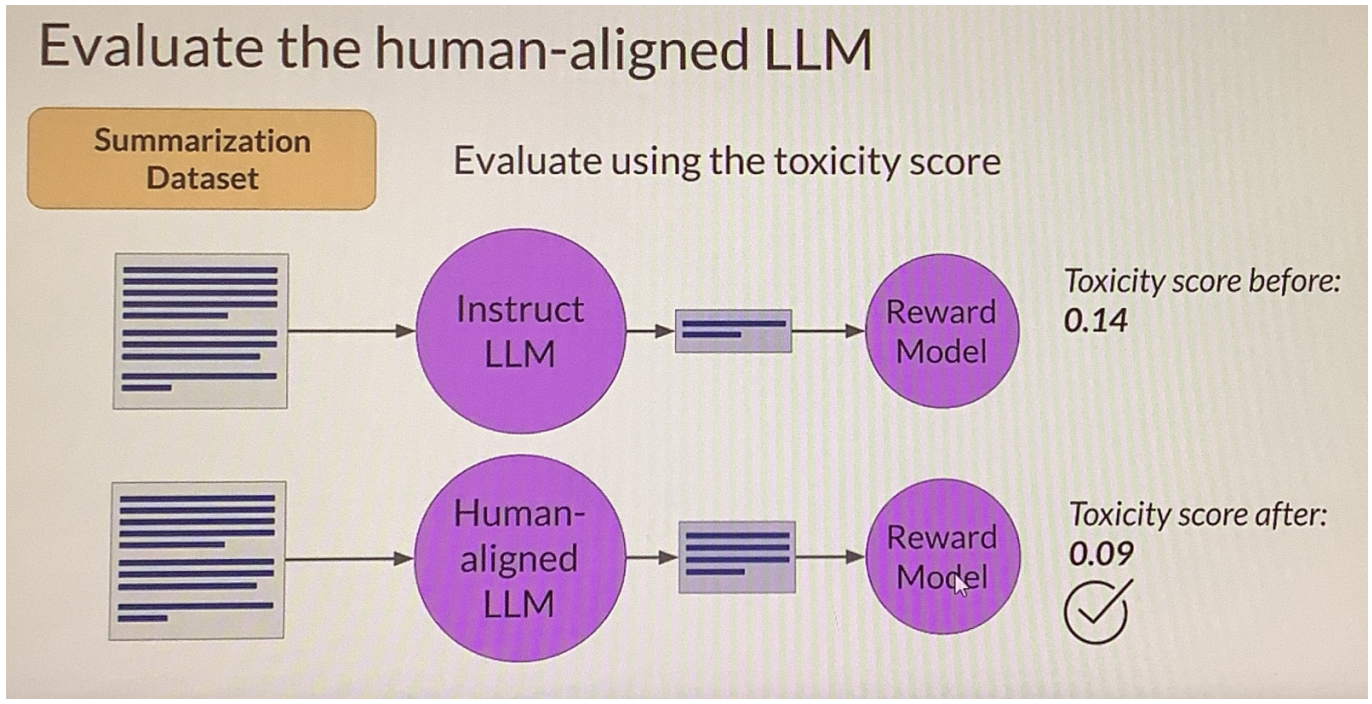
* We now need TWO full copies of LLM.
To avoid these we can combine RLHF + PEFT. In this case

We only update the weights of PEFT adapter.

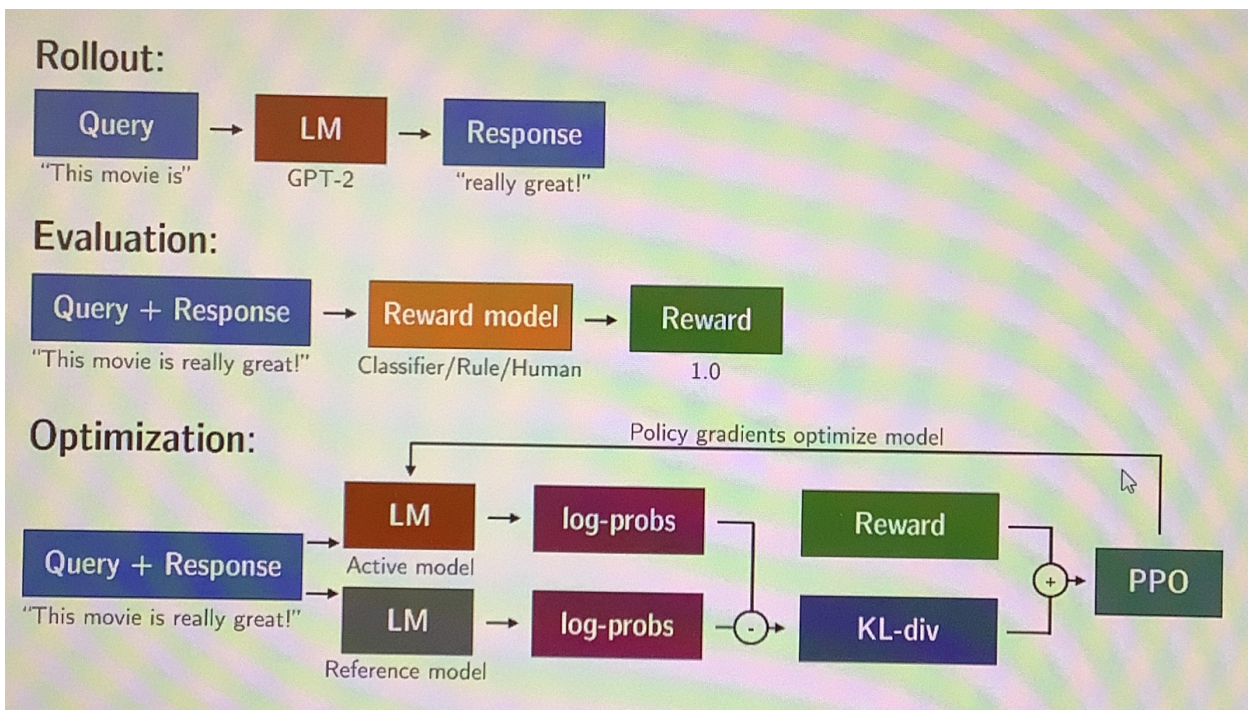


Combine PEFT with Reference model.

Evaluating Human Aligned Model :-



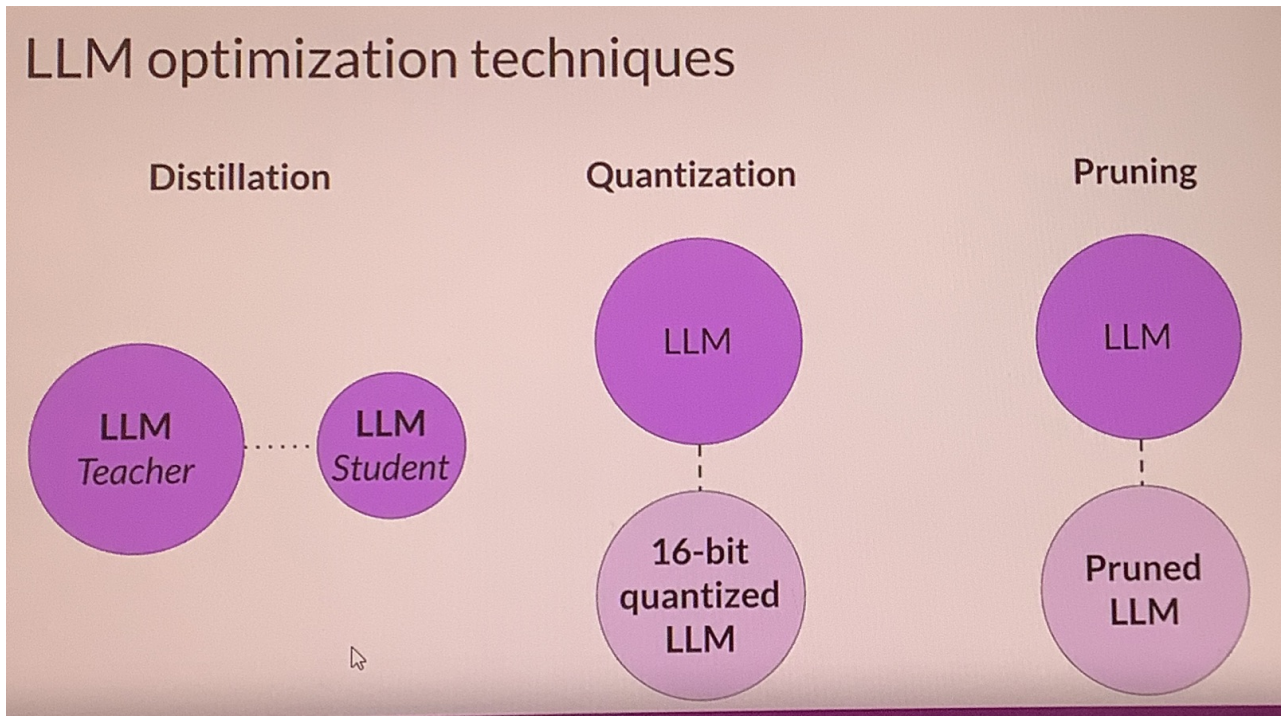
KL Divergence :-



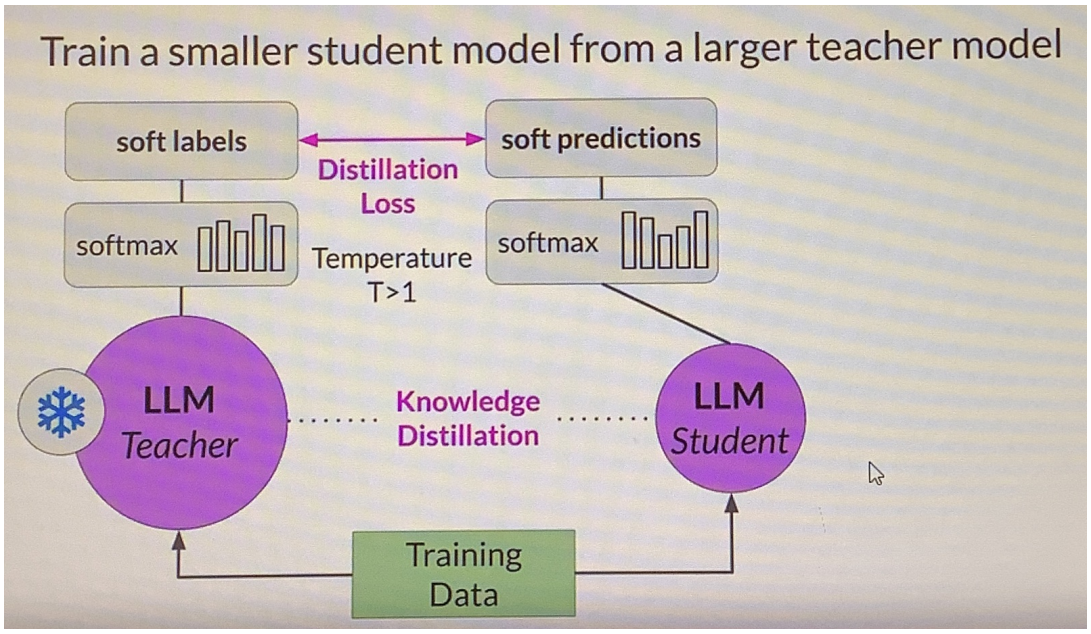
Model Optimizatim for Deployment

1. Distillation 2. Quantizatim

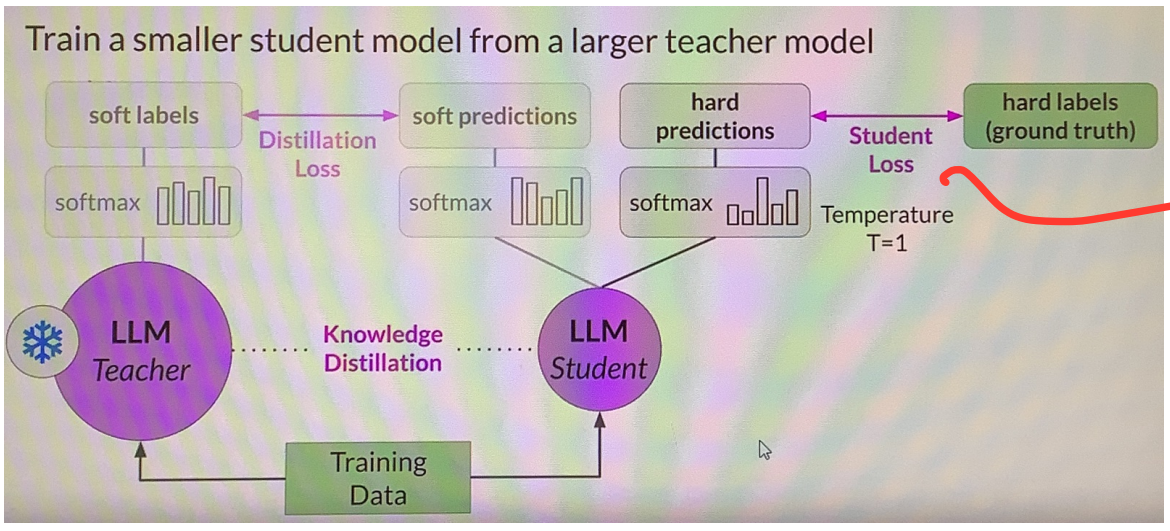
3. Pruning (Remove redeundent)



Distillation :



In parallel we also use dataset to train 'LLM Student' model.

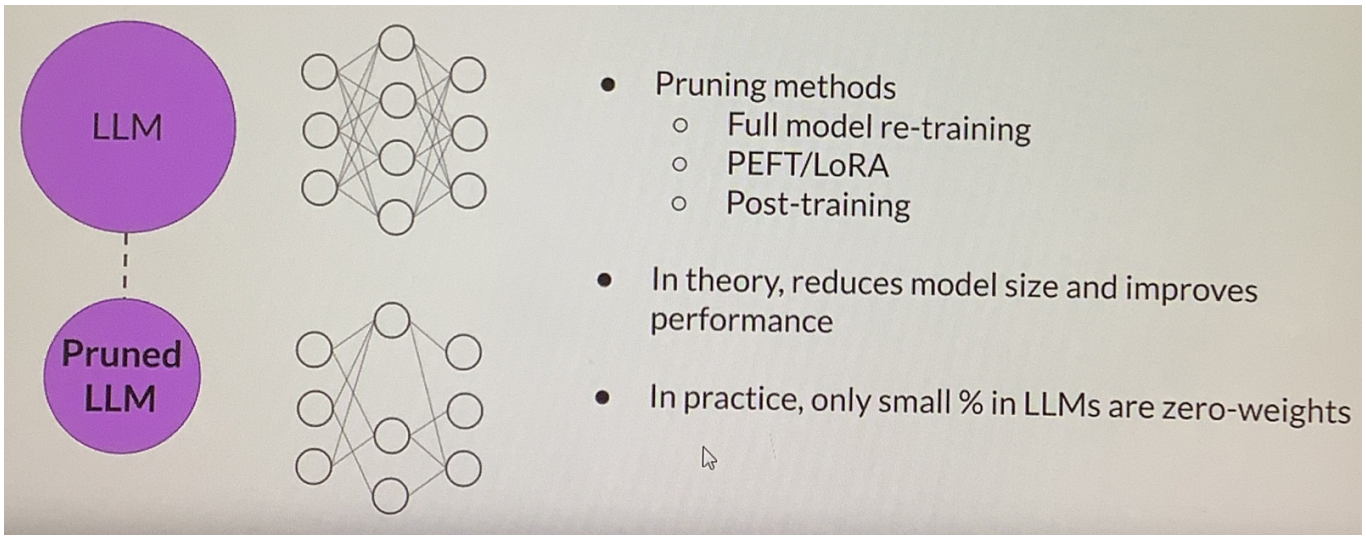


To calculate 'Student loss'

Combine of Distillation Loss + Student loss is use to update. Student weight via 'Back Propagation'

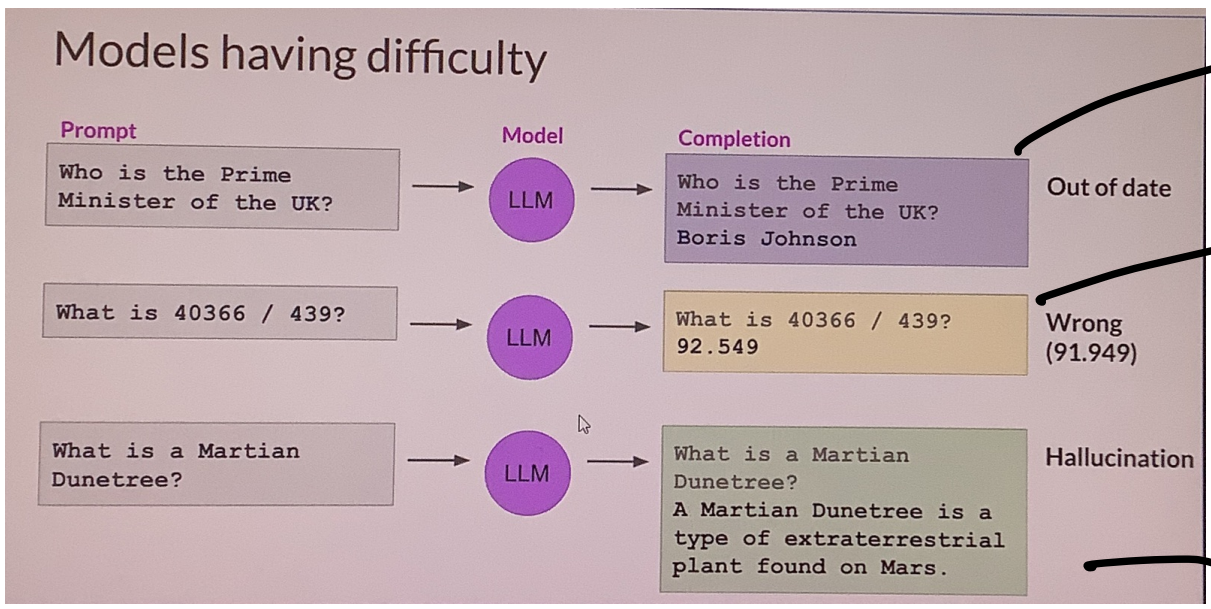
⊛ In practice 'Distillation' is not as 'effective for' Generative Decoder' model. It is more effective 'Generative Encoder' model. Such as 'BURT' that have lot of representation redeen-dancy.

Pruning :- Remove model weights with values closer to ZERO.



Time & Effort in the Life Cycle :-

	Pre-training	Prompt engineering	Prompt tuning and fine-tuning	Reinforcement learning/human feedback	Compression/ optimization/ deployment
Training duration	Days to weeks to months	Not required	Minutes to hours	Minutes to hours similar to fine-tuning	Minutes to hours
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and # of tokens for input/context Large amount of domain training data	No model weights Only prompt customization	Tune for specific tasks Add domain-specific data Update LLM model or adapter weights	Need separate reward model to align with human goals (helpful, honest, harmless) Update LLM model or adapter weights	Reduce model size through model pruning, weight quantization, distillation Smaller size, faster inference
Objective	Next-token prediction	Increase task performance	Increase task performance	Increase alignment with human preferences	Increase inference performance
Expertise	High	Low	Medium	Medium-High	Medium



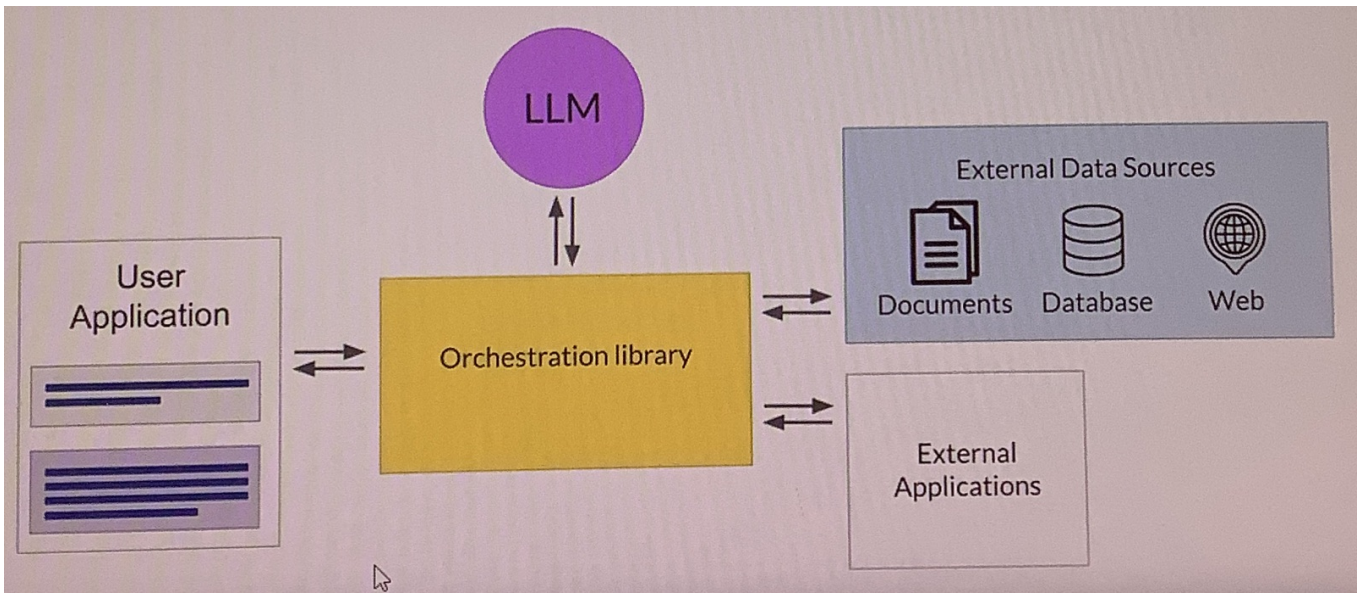
→ old Data

→ Models predict next no calculation

→ Generating

wrong information.

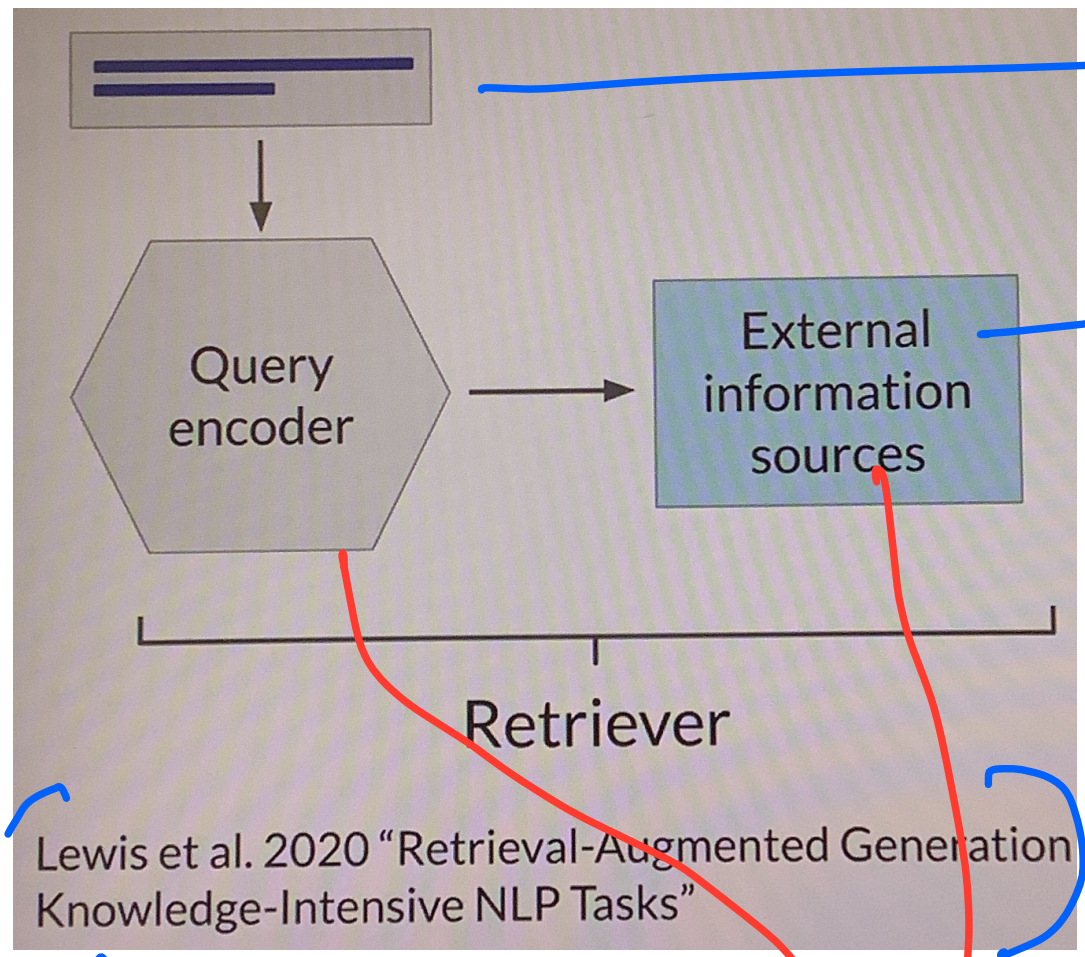
LLM powered Applications :=



Our application must manage the passing of user input to the LLM & return the completions. This often done by using 'Orchestration library'.

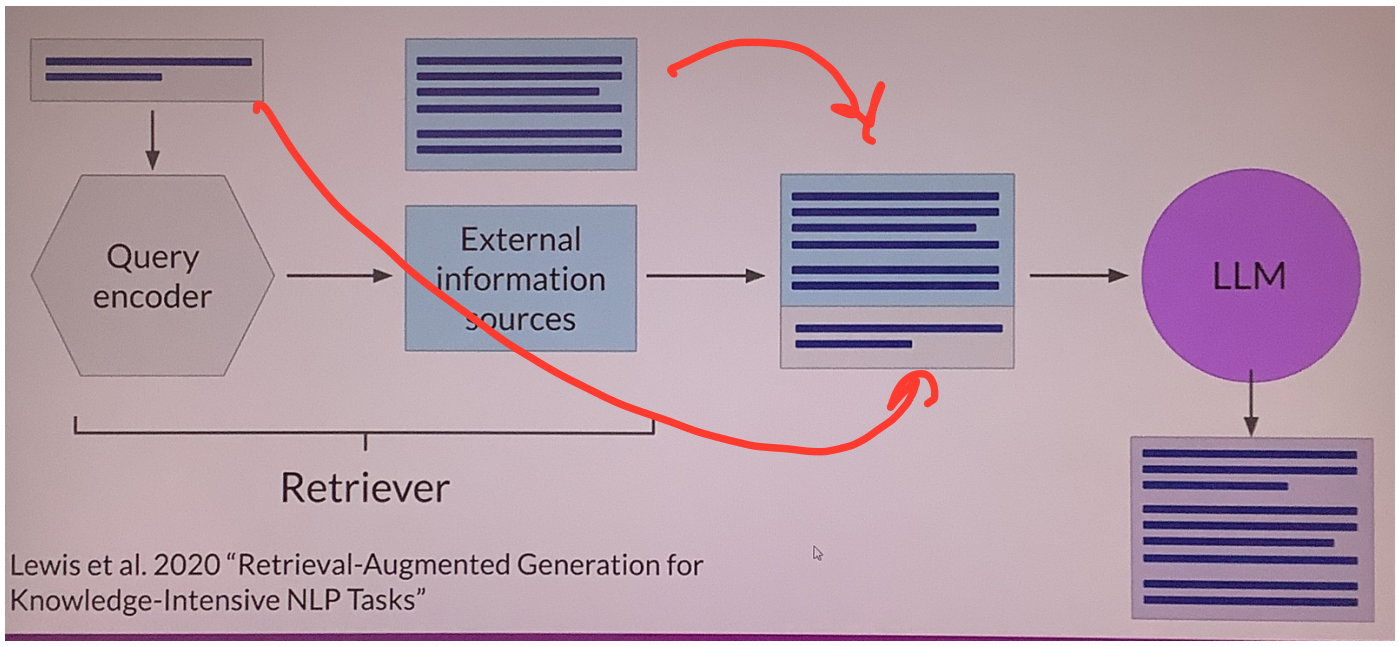
The layer can enable some powerful technologies that augment & enhance the performance. By providing External data sources or connecting with existing API's of other applications!

Retrieval Augmented Generation (RAG)



Query

→ DB (It can be in the form of CSV, SQL etc. but in the facebook paper it was stored as Vector DB. These two component are train together find DOCUMENT most RELEVANT for QUERY.

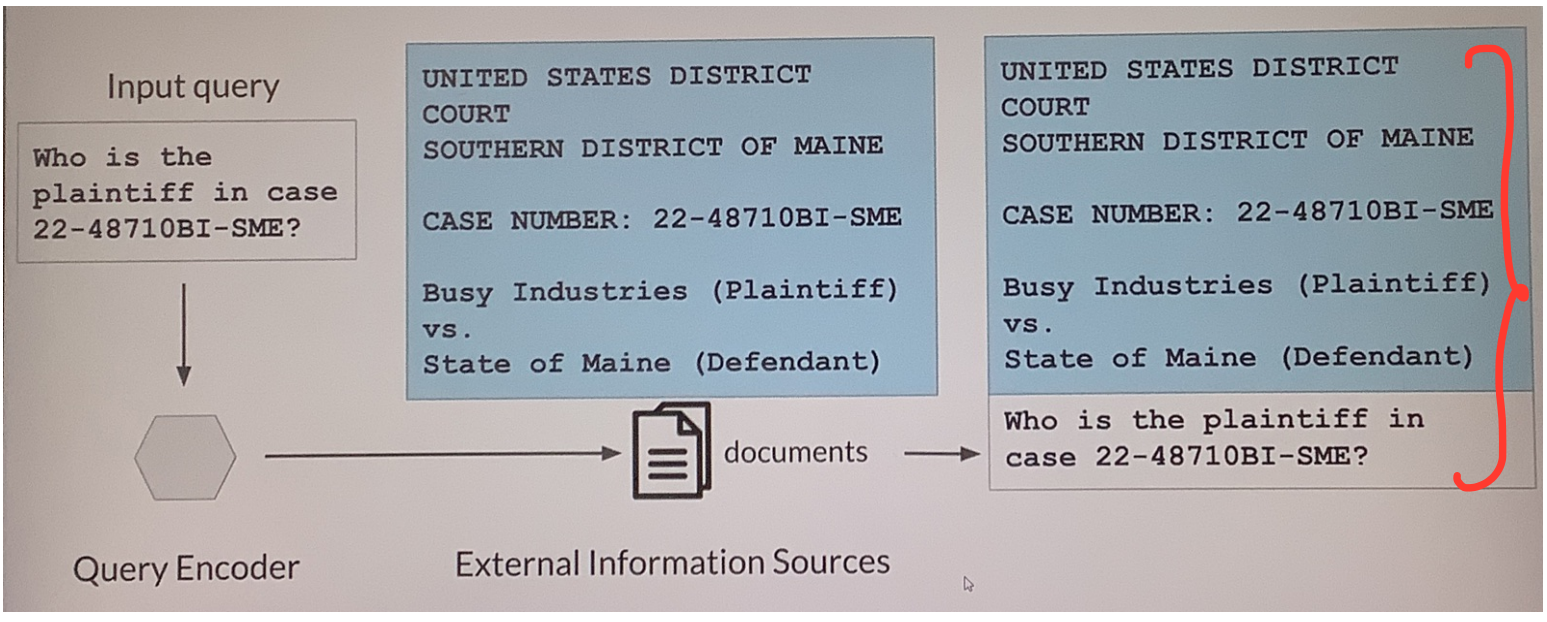


Query + output

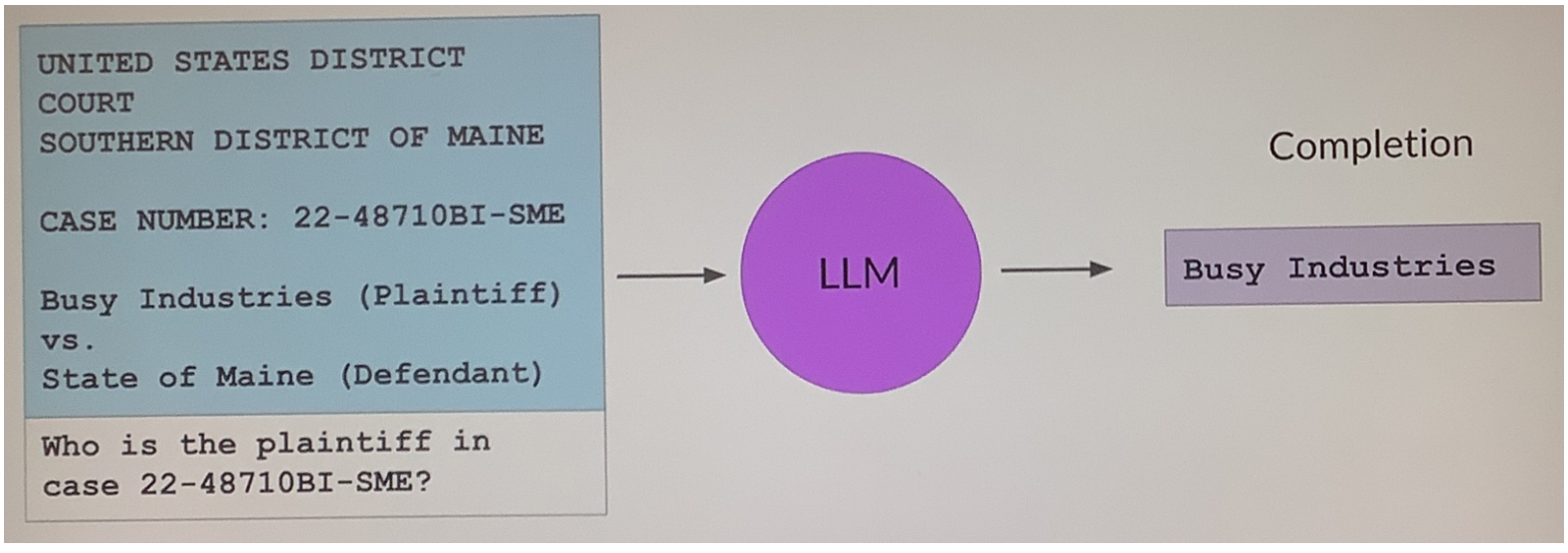
Query both are concatenated & passed as prompt into LLM.

Let's Take Specific Example

Searching legal Documents:



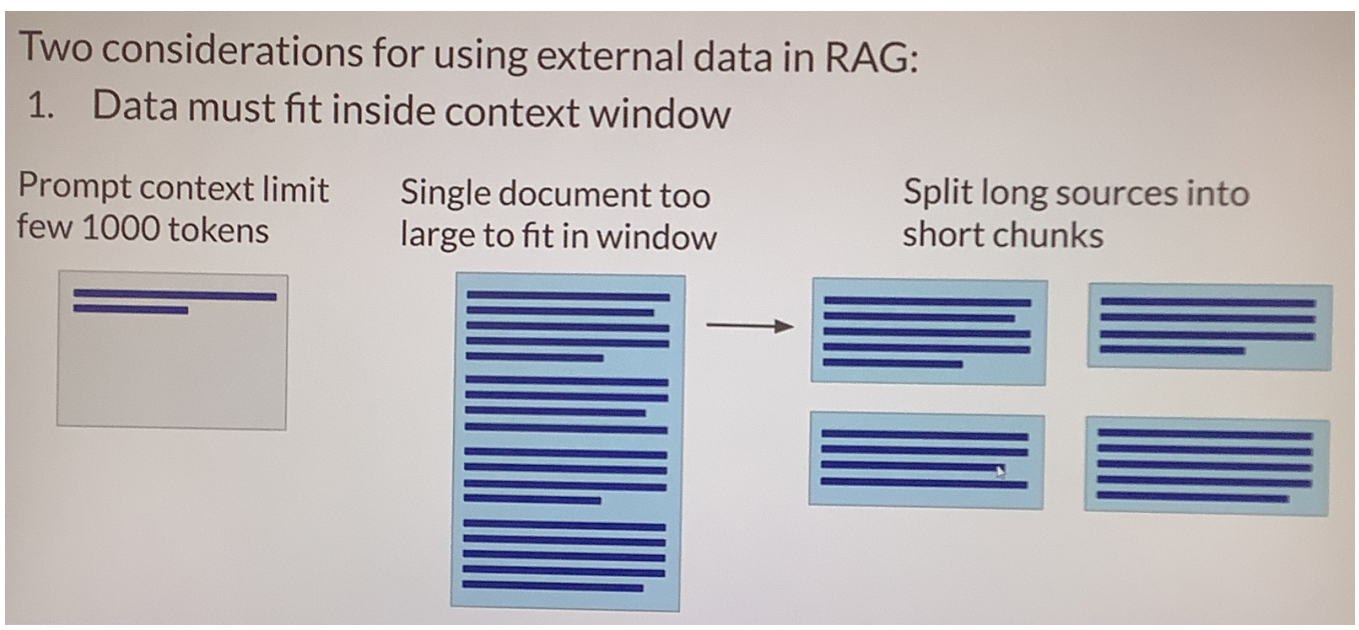
Concat Retrie- -ved Document + Query.



Model then uses the Context & Query as Prompt & generate desired output.

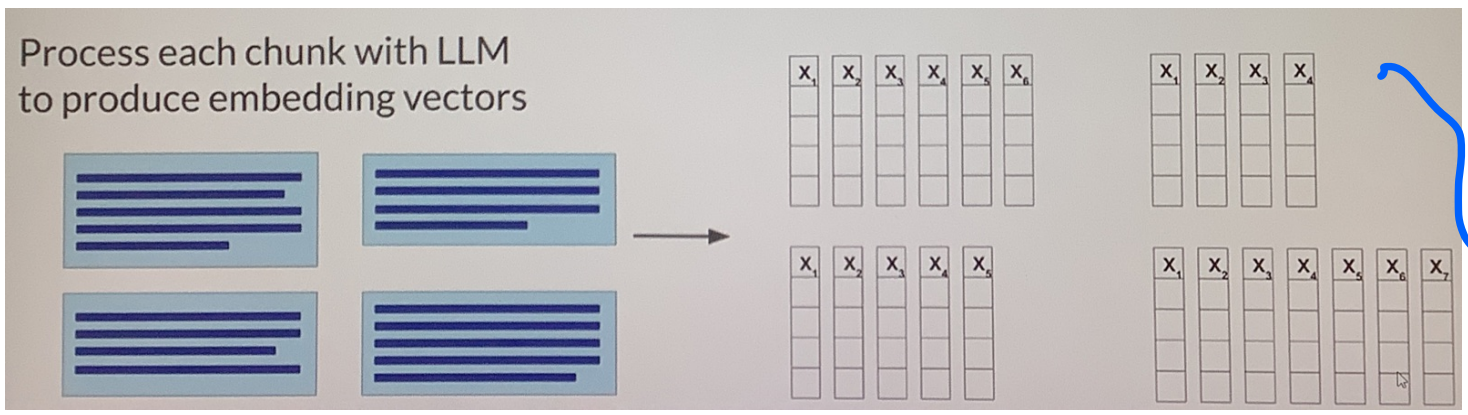
* RAG also helps to avoid the problem of MODEL Hallucination.

Data Preparation for Vector Store RAG



LANGChain can handle this kind of work.

2. Data must be in format that allows it's relevance to be assessed at inference time : Embedding Vectors.



Store Vectors in Vector Database.

to allow fast search.

Programming-aided Language (PAL) models :-

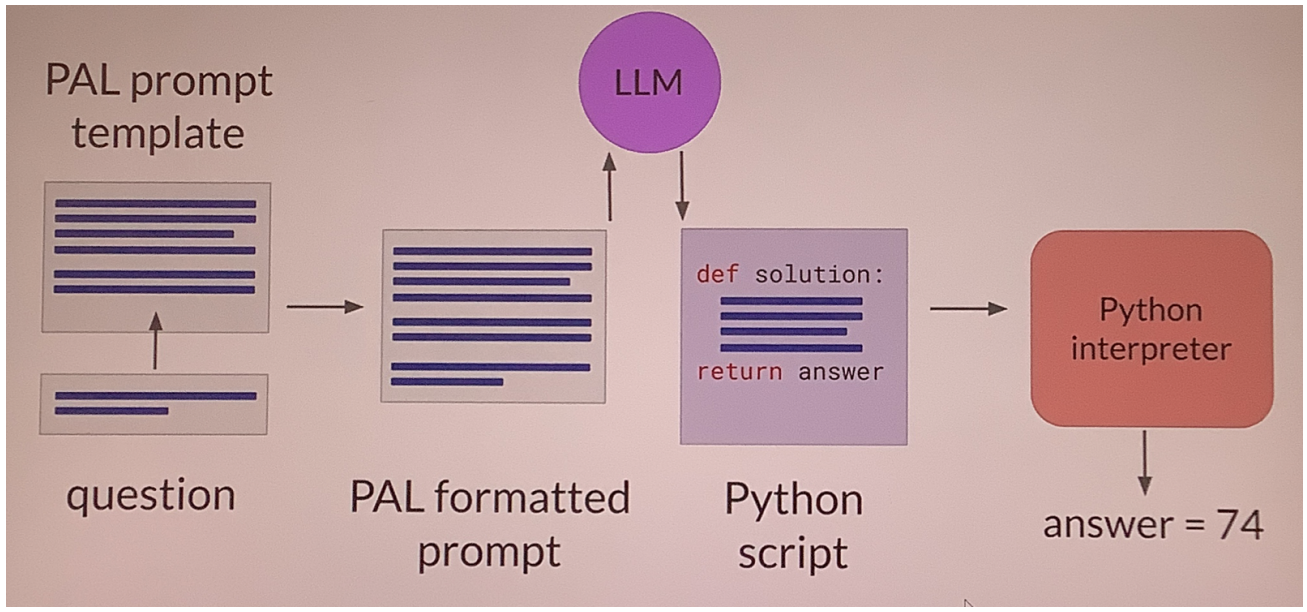
Prompt with one-shot example

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Answer:

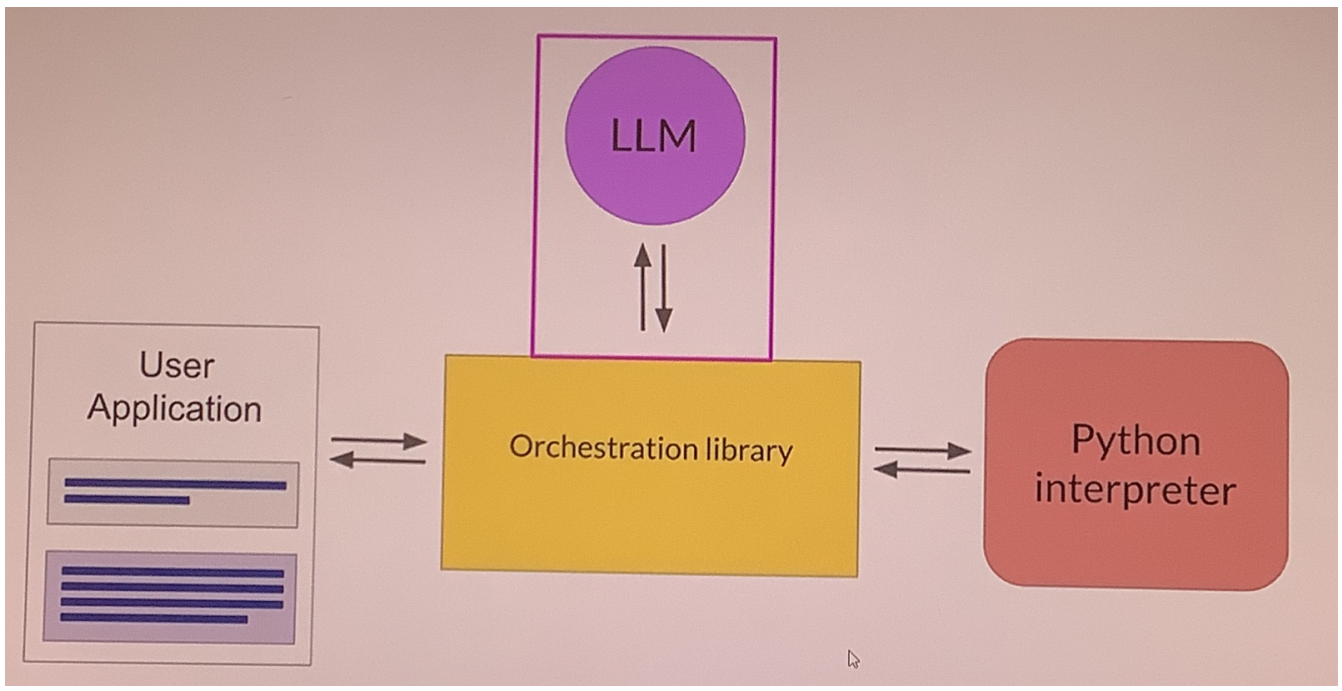
```
# Roger started with 5 tennis balls
tennis_balls = 5
# 2 cans of tennis balls each is
bought_balls = 2 * 3
# tennis balls. The answer is
answer = tennis_balls + bought_balls
```

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves did they have left?



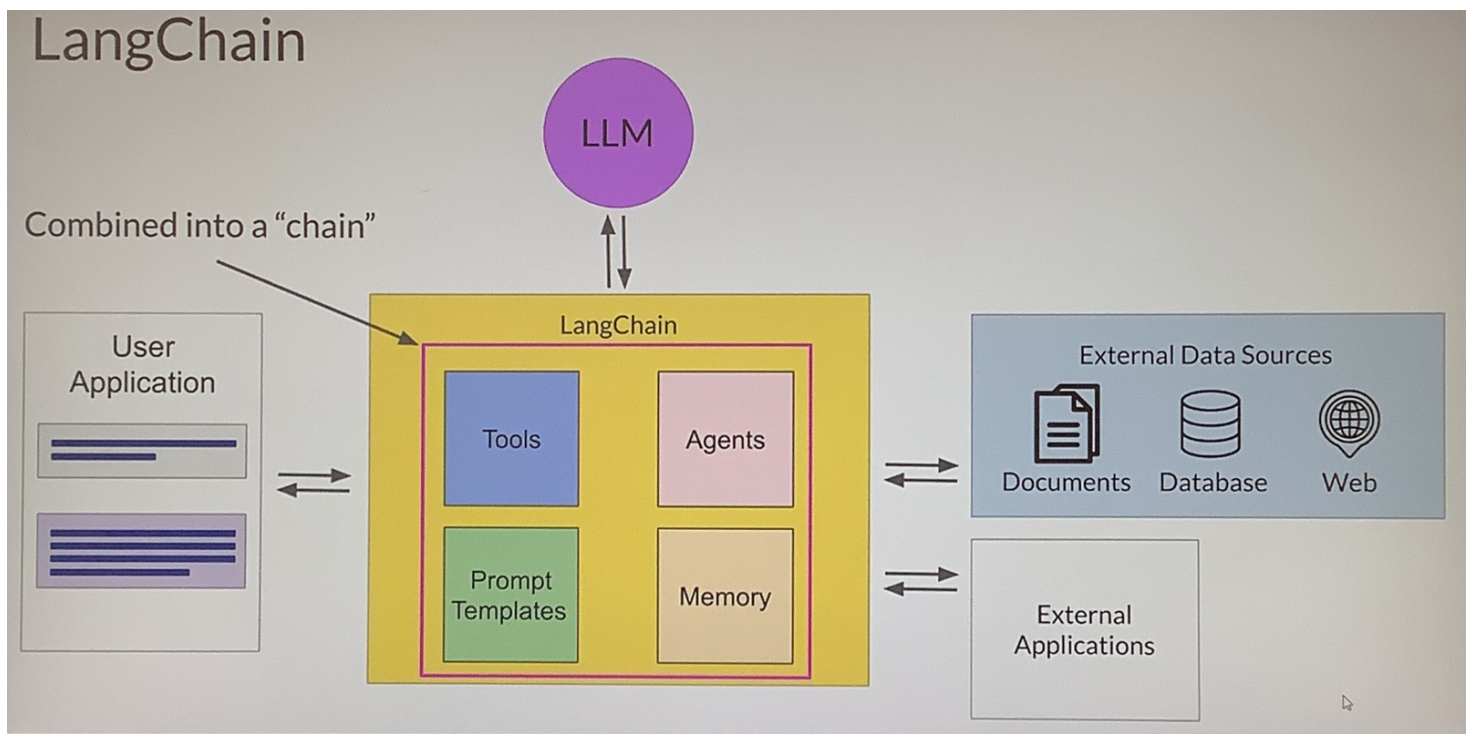
PAL Flow

PAL Architecture :-



ReACT: Combining Reasoning and Action in LLMs :-

Lang Chain :-



LLM Powered Application Architectures :-