Generative AI with LLM

→ useful for different application. (LLM) (General purpose technology)

→ Generative AI is subset of Machine learning, learned their ability by finding statistical pattern in massive datasets (human)

LLM

→ GPT	→ LLAMA
→ Bert	→ PALM
→ Flan-T5	→ BLOOM

(more parameter & more sophisticated task.)

Transformer Model → Neural Network that learns context & tracking relationship in sequential data.
(foundation model)

LLM use cases & tasks (next word prediction is basic task)

→ Essay writer	→ Code AI
→ summarize	→ Entity Extraction. (word classification)
→ Translate	→ External API

Smaller model can be fine tuned to perform well on specific focussed tasks.

LLM capability → Architecture powers them

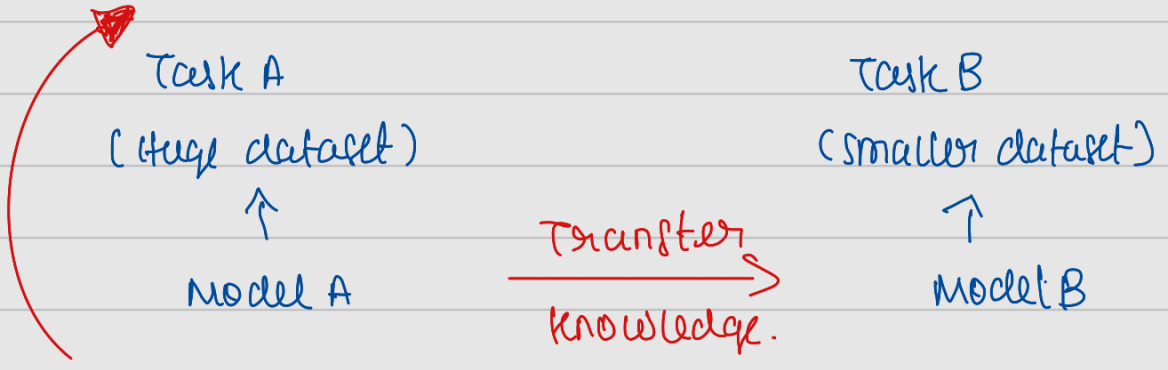
understanding language can be challenging., language is complex single word has multiple meaning. (homonyms)

Previous generation of language model made use of 'RNN'

Self-Supervised learning:

type of training in which the objective is automatically computed from inputs of the model. Means that humans are not needed to label the data.

Transfer learning



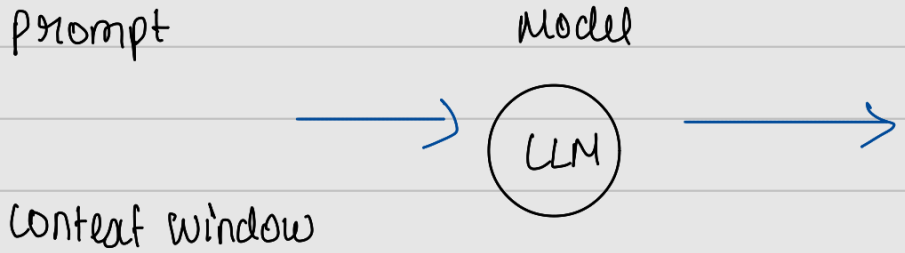
Act of initializing a model with another model's weight

Training from scratch requires more data and more compute to achieve comparable results.

Transformer model are built with special layer called Attention layer

Fine-tuning

* Full Fine tuning
* parameter efficient fine tuning (PEFT)



Inference : using model to generate text.

Generating text : RNN

RNN is limited by the amount of compute & memory.

then, transformer Architecture had arrived -> 'Attention is all you need!
(Attention to input meaning)

"Attention is All you Need" -> by Google in 2017.

-> Neural Network Architecture replaced the RNN/CNN with attention based mechanism.

-> encoder & Decoder, Multi-head self attention mechanism & feedforward NN layers.

-> it uses self attention to capture input sequence. capture long term dependencies

Transformer

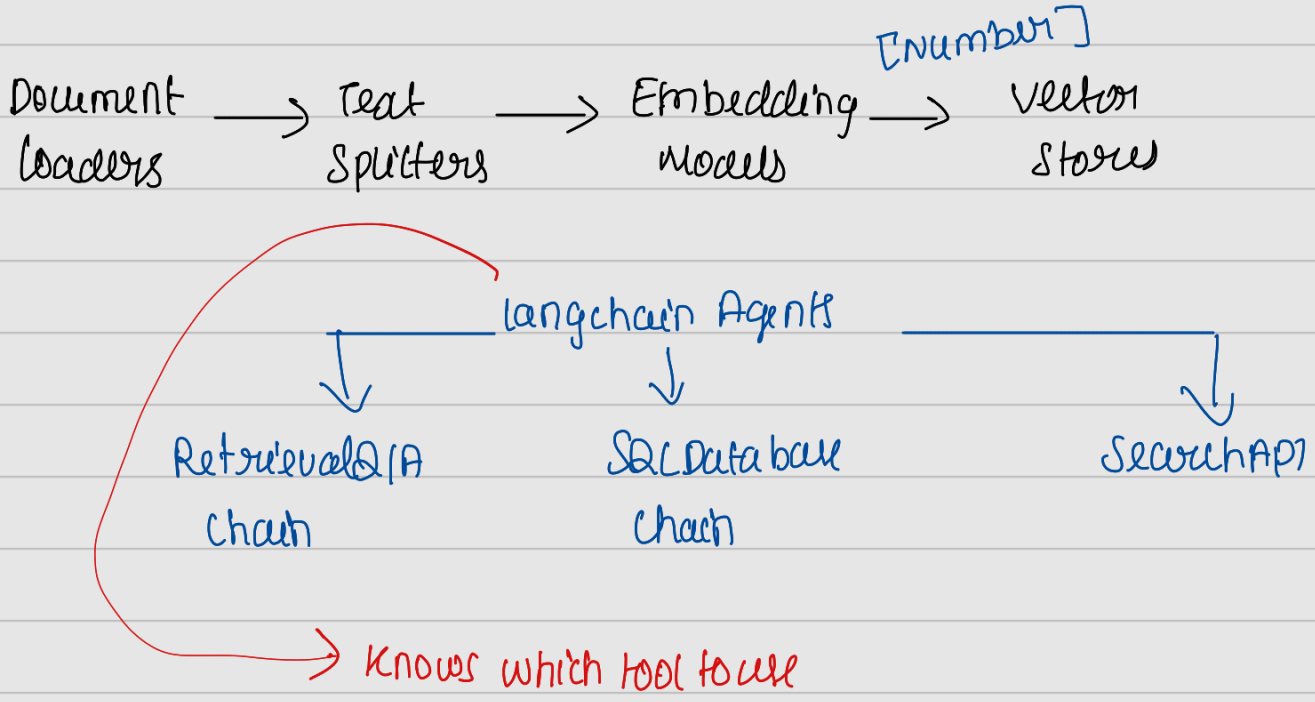
-> learn the relevance & context in all the word in the sentences. and apply attention weights. (attention map -> each word with every word)

Model process the Number not the text.

word Embedding.

↳ technique where individual words are represented as real valued vectors in a lower-dimensional space and captures inter word semantics.

langchain



Output parsers → response from LLM follow certain format.

Generative Models ?

Learns patterns and distribution in unstructured data. So, Reinforcement learning through human feedback is also applied.

LLM → transformer, a type of Neural Network architecture.

language Model

↳ A language Model is a Model which understands language more precisely how words occur together in Natural language, A language Model is used to predict what word comes next.

temperature = 0 ⇒ argmax sampling.

lower temperature means less randomness, vice versa.

top-p (Nucleus sampling)

↳ It controls how many random results should be considered for completion as per the temperature

Generative AI project life-cycle

(own)

① foundation Model | pre-training Model (decision)

② fine-tuning - custom data.

Self Supervised learning to optimize these Models.

Issue with RNN for Generative part is, understanding language can be challenging (Homonyms)

then transformer architecture has arrived → increases the performance of Natural language task.

→ Scale efficiently

→ parallel process

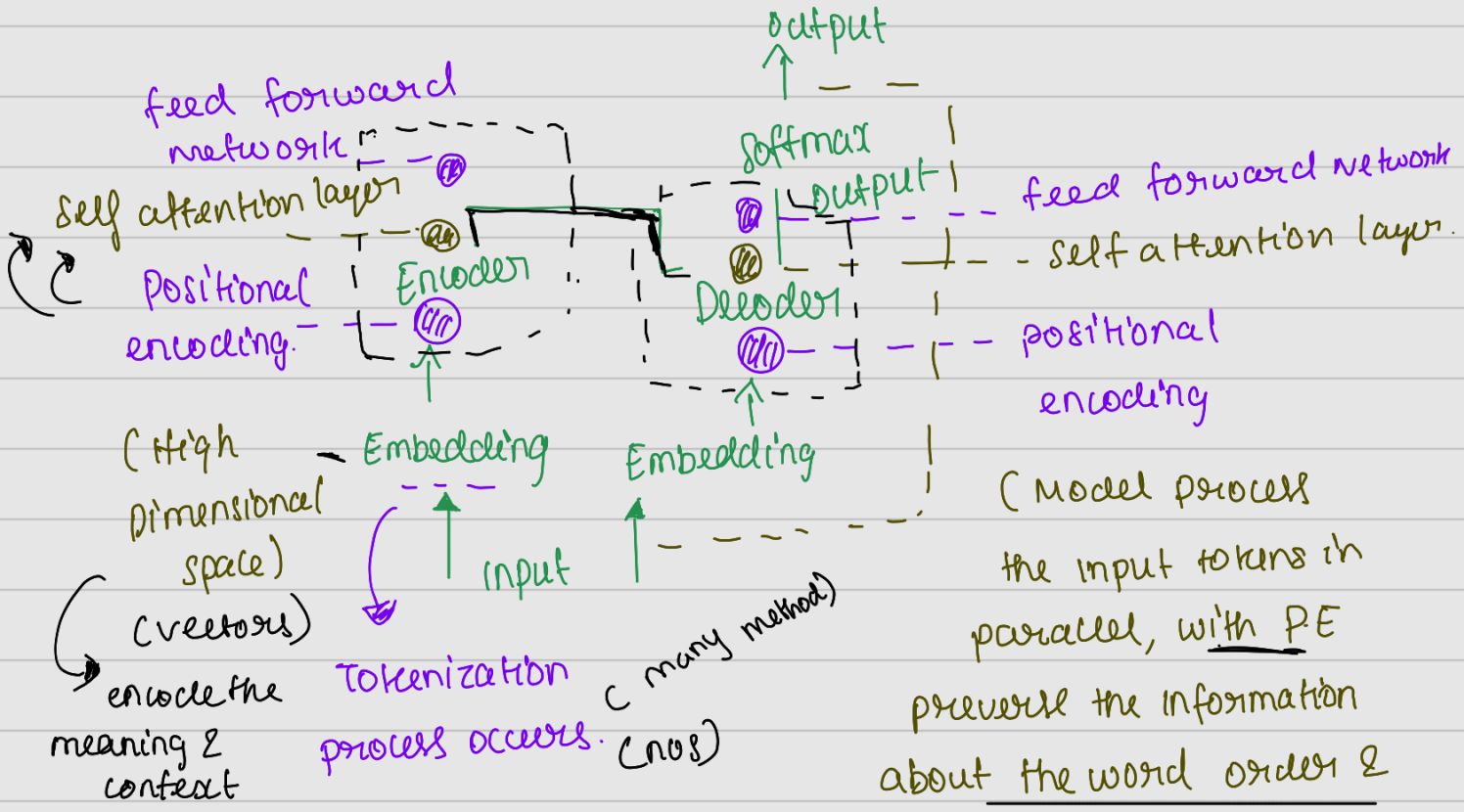
→ Attenuation to input meaning.

(Self-attention)

power of transformer architecture, its ability to learn the relevance and context of all of the words in a sentence, every other word in a sentence. (Attention Map)

Apply attentuation weights to relationship for relvance of each word to other word.

Transformer



(Model process the input tokens in parallel, with PE prevents the information about the word order & relevance)

Self attention, model analyze the relationship between the tokens in the input sequence.

↳ layer's weights are stored.

↳ to capture the contextual dependencies between the words

(Reflects the importance of each word in the input sequence)

transformer has Multi-headed self attention (12-100)

means that multiple sets of self-attention weights / heads are learned in parallel independently of each other.

1. Realtionship 2. Activity 3. other properties

Each self-attention head will learn different aspects of language, weights of each head is randomly initialized.

output is processed through feed forward network, proportional to probility score for every token then passed to Softmax layer → P(E) for each word. (Higher)

Language Model

What should be the next words == word with highest probability.

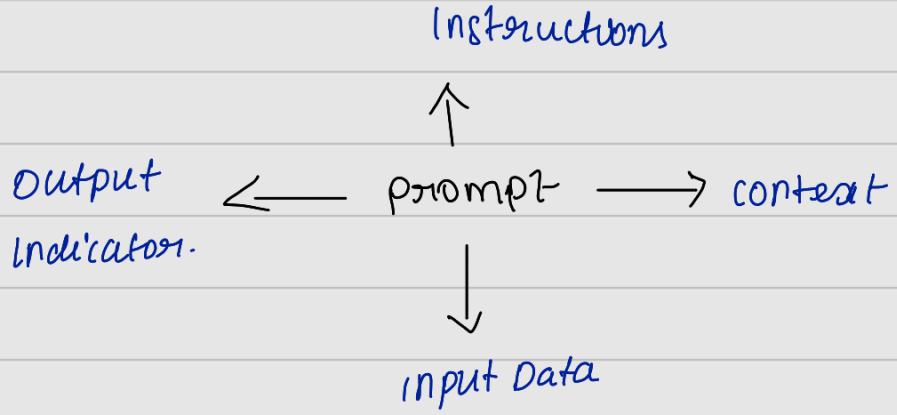
Given a sequence of words

$x_1, x_2, \cdots x_t$

compute the probability of the next word

$$p(x_{t+1} \mid x_1, x_2, \cdots, x_t)$$

LLM → trained on huge amount of data.



Zero-shot prompt (popular) → accuracy (limitation)

prompt does not contain any explicit instructions or example for the model to follow. Instead it relies on the model's ability to understand & interpret Natural language.

(without specific training)

Embeddings

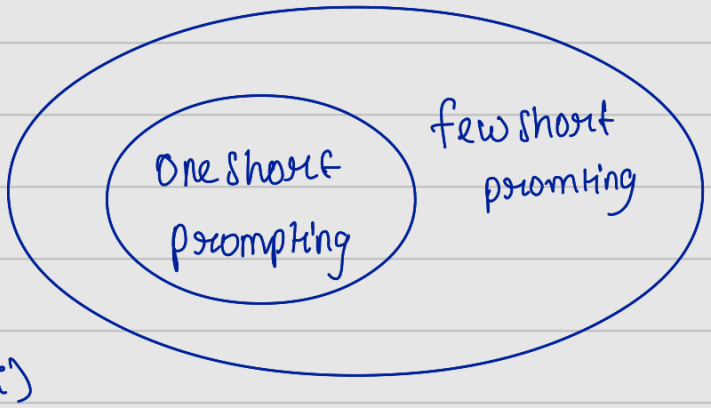
classic technique in NLP. The idea is to create a vector space from the text such that the distance between the vectors in the space have a certain meaning.

Vector → simply a sequence of Numbers. (represent complex Model)

Vector database is a database that is saving those embeddings. (pinecone)

few Short prompts

model is given a few examples of the result you want. Useful where there is limited data available for a given tasks.



(COT)

Chain of thought prompting (google)

↳ improve reasoning abilities of LLM (It decomposes the multi-step problems into intermediate steps, allowing to solve a complex problem.

→ prompt Engineering technique.

↳ it's how human solves the problem. (problem solving)

(major step in prompt Engineering)

① zero Shot chain of thought

↳ let think step by step (prompt)

② few short chain of thought (similar to few shot learning)

Embedding in transformer (High dimensional space)

↳ each token represented as a vector & occupies a location. (word2vec)

fine tuning

fine tuning is taking these general purpose models like GPT-3 & specializing them into something chatGPT

Gets the model to learn the data, rather than just get access to it.

Reduces hallucinations & process is similar to the model's earlier training.

Generating text with transformer

The attention mechanism is used in both the encoder & decoder layers of the transformer mechanism.

The data leaves the encoder is, a deep representation of the structure and meaning of the input sequences. This representation is inserted into the middle of the decoder to influence decoder's self-attention mechanism.

Encoder → Encodes inputs ('prompts') with contextual understanding & produces one vector per input token.

Decoder → Accepts input tokens and generates new token.

} does this in a loop

Encoder only models → Sentiment analysis, BERT

Encoder-decoder model → Sequence to sequence tasks, translation.
len(input) ≠ len(output)
→ Google Bard

→ Decoder only model → GPT, BLOOM, Llama (models)

prompting & prompt Engineering

The develop & improve the prompt is known as prompt Engineering.

providing examples inside the context window is called in-context learning. (ICL) → there is a limit to context window.

→ zero shot inferences.
→ one-shot inferences
→ few shot inferences.

→ engineering your prompts to encourage the model to learn by examples.

Large models are good at zero shot inferences.

Generative configuration (inference parameters)

* max-new tokens → number of tokens that model will generates.

The output from the transformers' Softmax layer is a probability distribution across the entire dictionary of words.

Most LLM by default will operate with so-called greedy decoding,

↳ greedy:

The word [token with the highest probability is Selected.

So, bring random sampling.

Random (weighted) sampling, Select a token using a random-weighted strategy across the probabilities of all tokens.

---

top-K & top-P 등 sampling techniques.

↳ Limit the random sampling & increase the output to be sensible

---

top-K → select an output from the top-K results after applying random-weighted strategy using the probabilities.

top-P → select an output using the random weighted strategy with the top-ranked consecutive results by p(E) and with a cumulative p(E) ≤ P

↳ Specify no. of tokens to randomly choose from.

↳ Specify the total p(E) want model to choose from.

temperature

↳ higher temperature, higher randomness.

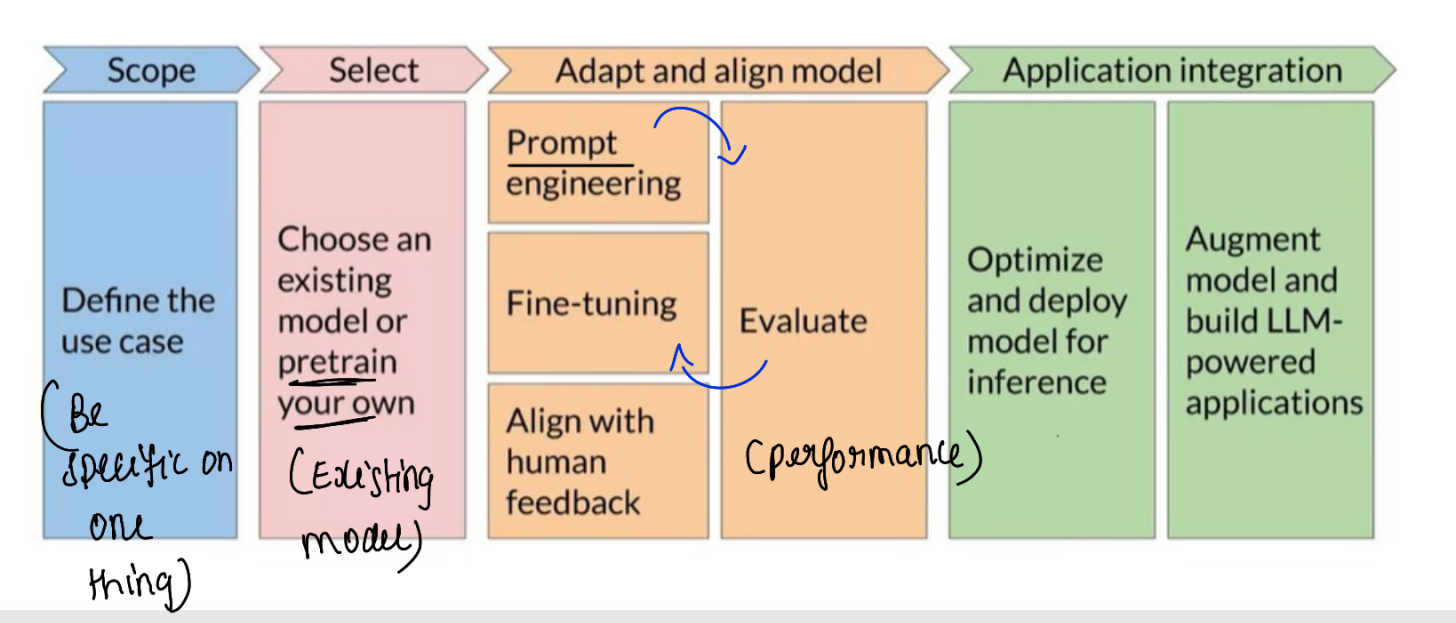
applied in the final softmax layer.

temperature <1 → Strongly peaked probability distribution

temperature >1 → Broader, flatter probability distribution.

Generative AI project lifecycle

fine tuning ∪ supervised learning.



pre-training large language model

consideration for choosing a Models.

→ foundation model (pretrained LLM)

→ Train your own Model (Custom LLM)

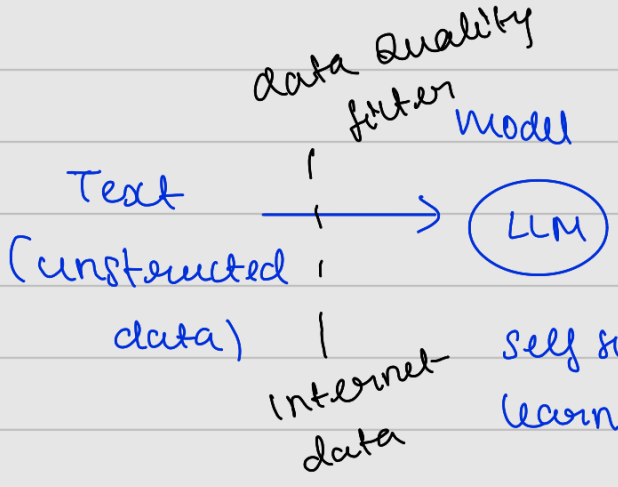
Huggingface → Have Model cards
(Hub) (imp details & best usecases for each model)

BloombergGPT → Decoder only language model, pretrained using an extensive finacial datasets.

The goal of pre-training is to teach the model the general pattern and structure of language, allowing it capture the relationship blw words.

The goal of fine-tuning is to adjust the model's parameters to make it better at performing the desired task.

LLM encode a deep statistical representation of language. developed during model pretraining phase.



{ encoders generates embeddings }

training → architecture of model. { weights updated }

pre-training also requires a large amount of compute & use of GPUs

Encoder only models are also called Autoencoding models, they are pre-trained using masked language modelling.

masked language modelling (MLM)

The teacher teaches the student

<mask> (randomly Masked)

training objective is to predict the mask token in order to reconstruct the original sentences.

objective : Reconstruct text ('denoising')

The teacher teaches the student.

→ ← Bidirectional context

model has understanding of full context of a token.

Good use cases:

- → Sentiment analysis
- → named entity recognition
- → word classification

[Good for tasks that require understanding of the input, such as sentence classification & named entity recognition]

Example models:

- → BERT
- → ROBERTA

Decoder only model | Autoregressive models, which are pretrained using Causal language Modelling (CLM)

Here training objective is to predict the next tokens based on the previous sequences of tokens.

↳ Fully language modelling.

objective: Predict next token.

The ? (CLM) [No knowledge of the end of sentences]

↳ The teacher

↳ iterate one by one tokens for next word, context is unidirectional.

Good use cases

→ Text generation
→ other emergent behaviour
↳ depends on model size

Example models

→ GPT
→ BLOOM.

[Good for generative tasks such as text generation]

Encoder-decoder model | sequence to sequence models, Good for generative tasks that require an input, such as translation or summarization.

Span
corruption

Original text ⟶ Encoder - Decoder LLM

Span corruption

The teacher <mask> <mask> Student

The teacher <x> student

↳ Sentinel token

Objective: Reconstruct span

<x> teaches the

[Sequence to sequence models are best suited for tasks revolving around generating new sentences depending on a given input such as summarization]

Models:
→ T5
→ BART

Growth of models powered by;
→ Introduction of transformer
→ Access to massive datasets
→ more powerful compute resources.

computional challenges of training LLM

→ CUDA out of memory
(Compute unified device Architecture)

↳ tools developed for Nvidia GPUs

Pytorch | tensorflow uses this.

Approximate GPU RAM needed to store 1B parameters.

1 parameter → 4 bytes.

1B para → 4×10^9 bytes → 4 GiB

memory needed to store model

memory needed to train model.

⟶

4 GiB @ 32-bit full precision

Single processor.

24 GiB @ 32-bit full precision.

(1.) Reduce the memory,

Reduce the memory required for training is called quantization

↳ reduce the memory required to store weight by reducing their precision from 32-bit floating to 16-bit floating point numbers. 8-bit

By reducing the precision of the model weights.

As model sizes get larger, you will need to split your model across multiple GPUs for training.

Approximate GPU RAM needed to train 1B parameters.

Additional GPU RAM needed to train 1B parameters

	Bytes per parameter
Model Parameters (Weights)	4 bytes per parameter
Adam optimizer (2 states)	+8 bytes per parameter
Gradients	+4 bytes per parameter
Activations and temp memory (variable size)	+8 bytes per parameter (high-end estimate)
TOTAL	**=4 bytes per parameter +20 extra bytes per parameter**

~20 extra bytes per parameter

Quantization: Summary.

	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	-/-	7	1 byte

FLAN T5

- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training
- BFLOAT16 is a popular choice

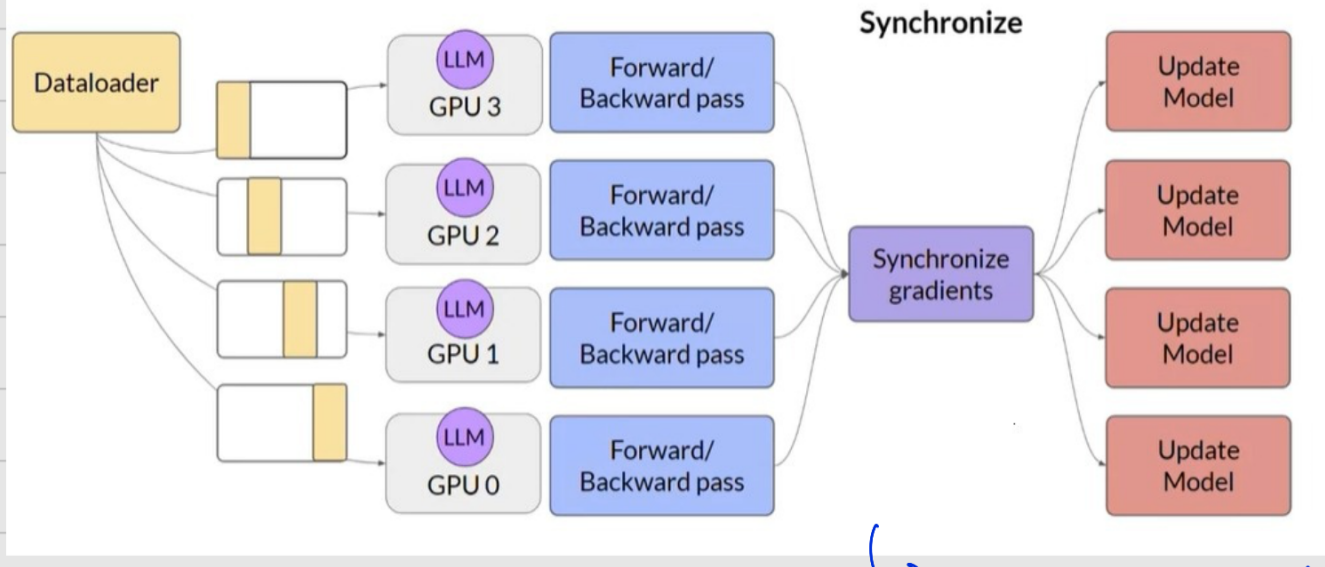
By Quantization, we can reduce memory consumption required to store the model parameters.

Efficient multi-GPU compute Strategies

This implementation allows parallel computations across all GPU's that results in faster training.

Distributed Data Parallel (DDP)

(dataset is distributed)



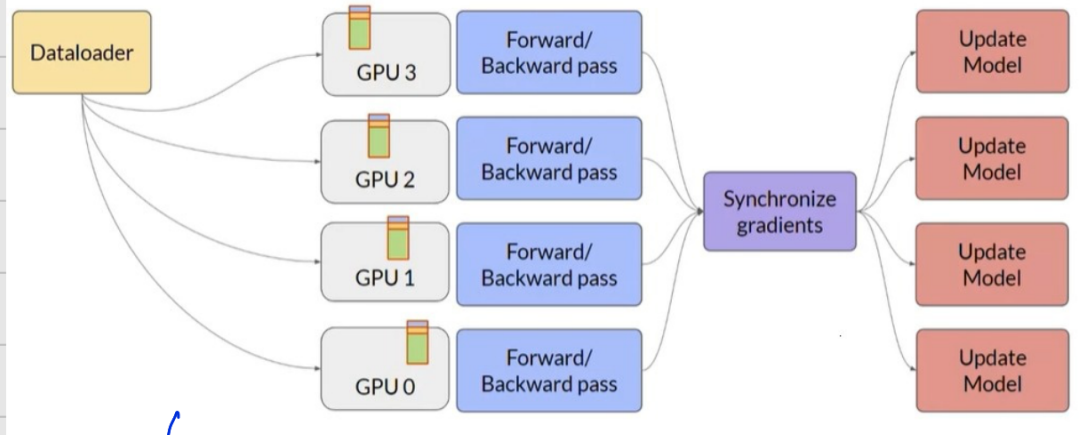
↳ requires model weights and additional parameter.

If model is too big for this, look for model sharding technique.

fully Sharded Data parallel (FSDP), motivated by the 'ZeRO' paper - zero data overlap between GPUs

Zero → Zero Redundancy optimizer. (reducing the memory by distributing (sharding)

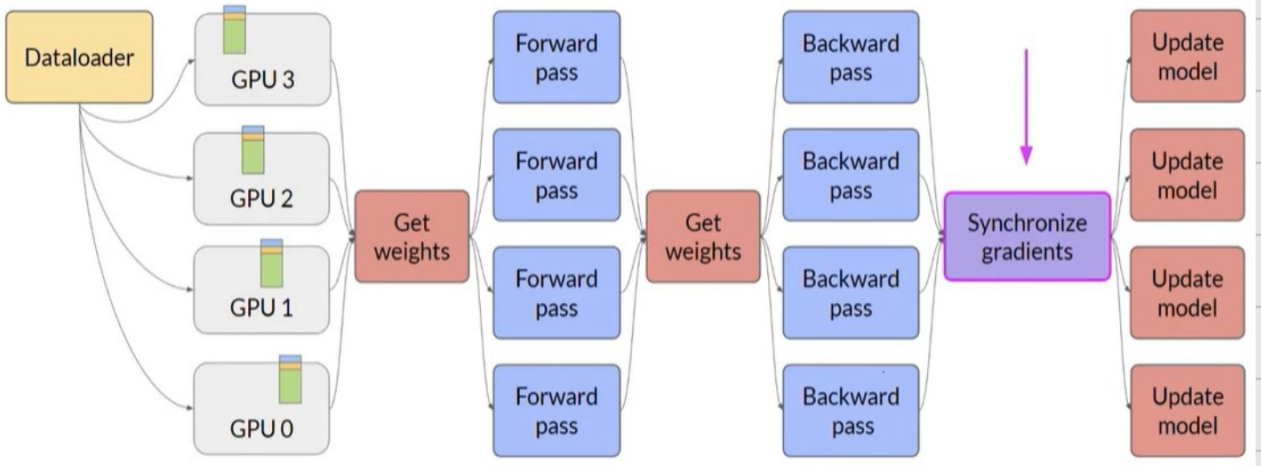
Fully Sharded Data Parallel (FSDP)



↳ also distribute the model parameter, gradients & optimizes the states across the GPU nodes.

using GPU

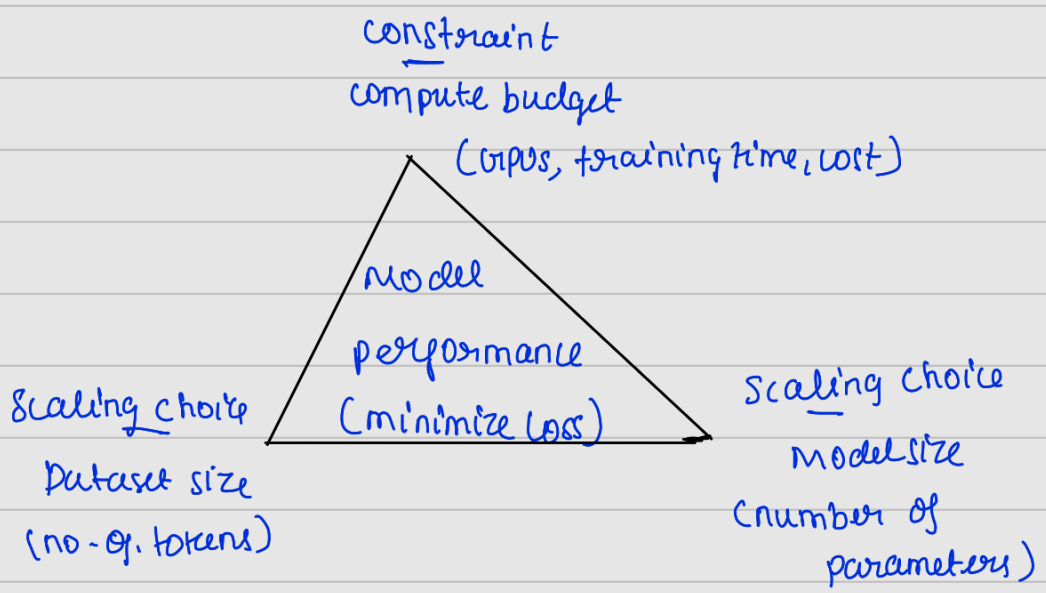
Fully Sharded Data Parallel (FSDP)



→ Helps to reduce Overall GPU memory utilization
→ Supports offloading to CPU if needed.
→ Configure level of sharding via sharding factor.

Scaling laws and compute-optimal models

Pre-training a model → maximize model performance.



''petaflop/s-day' = floating point operations performed at rate of 1 petaflop per seconds for one day.

→ increase the compute budget to achieve better model performance. (Hardware, project timeline, financial budget)

→ dataset ↑ performance ↑ ∫ model size ↑ performance ↑

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.

Chinchilla scaling laws for model and dataset size

Model	# of parameters	Compute-optimal* # of tokens (~20x)	Actual # tokens
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"

pre-training for domain adaptation

words are rarely used, which means that they are unlikely to have appeared widely in the training text of existing LLMs.

↳ Models have difficulty in understanding them, using correctly.

Pre-training your model from scratch will result in better models for highly specialized domains like law, medicine, finance.

BloombergGPT → domain adaptation for finance.

(pretraining) → financial (private + public) 51% + other (public) + (~Lg.L.)

∴ Chinchilla Scaling laws for guidance.

Week-2

Instruction fine-tuning

fine-tuning an LLM with instruction prompts.

→ In context learning may not work for smaller models (LLM)

→ Examples take up space in the context window. So, Instead try fine-tuning the model.

LLM pretraining

unstructured textual data → pre-trained LLM (model.) } pretraining, self-supervised learning

LLM fine-tuning, } supervised learning.

pre-trained model → task specific examples (labelled example of specific domain) → fine-tuned LLM (model)

(prompt - completion pair. examples (improved performance)

output of LLM is a probability distribution across tokens.

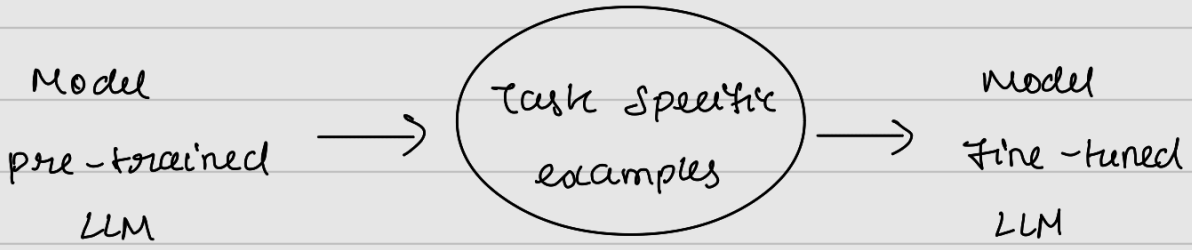
Model pre-trained LLM ⟶ Model Fine-tuned LLM (Instruct LLM)

The fine tuning process results in a new version of the base model is Instruct model. (common way)

Fine-tuning with instruction prompts is common way, fine-tuning always means Instruction fine-tuning

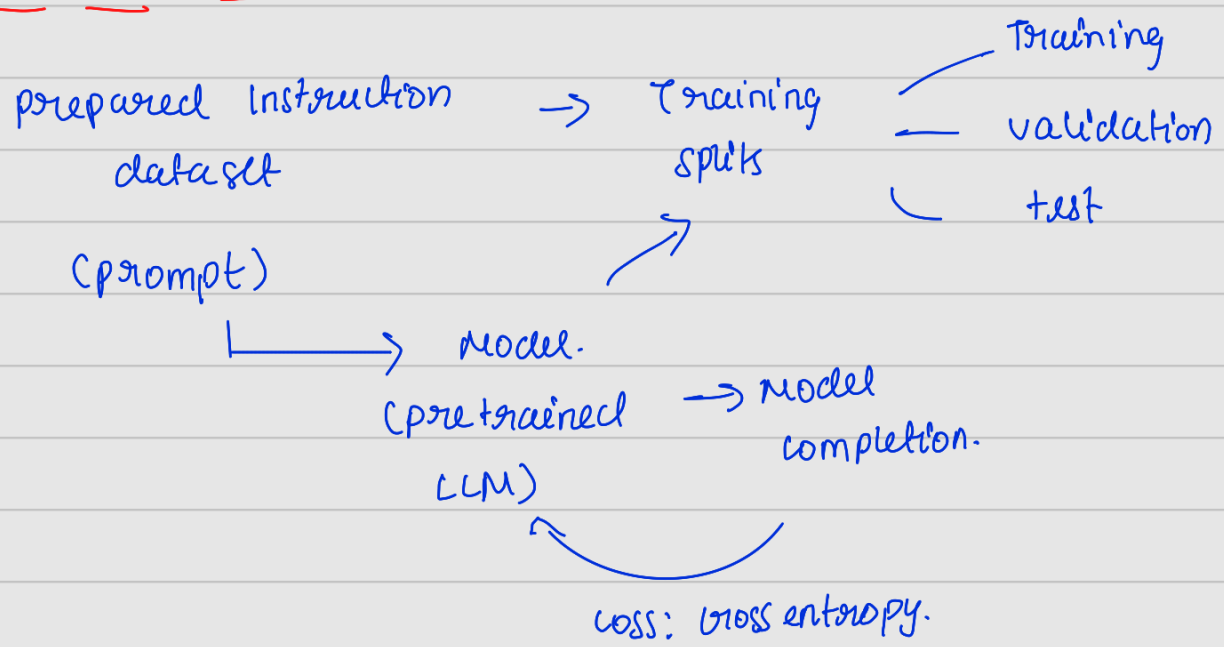
Instruction fine-tuning trains the model using examples that demonstrate how it should respond to a specific instruction.

Full-fine-tuning updates all parameters.



training data ⟶ formatted as Instruction.

LLM - fine-tuning



fine-tuning on a single task

parameter efficient fine-tuning → set of techniques that preserves the weights of the original LLM (most of pre-trained weights are left unchanged)

Model (pre-trained LLM) → Single task training dataset, eg. Summarization → Model. (instruct LLM)

example: 500 - 1000 single task.

Catastrophic forgetting,

↳ full fine tuning can significantly increase the performance of a model on a specific task. (modifies original LLM weights)

↳ but can lead to reduction in ability on other tasks.

Avoidance?

1. First note that you might not have to!
2. Fine-tune on multiple tasks at the same time.
3. Consider PEFT
   ↳ trains only task-specific adapter layers and parameter.

Multi-task instruction fine-tuning

Model pre-trained LLM → Instruction fine-tune on many tasks → Model instruct LLM.

(many examples of each needed for training)
(50,000 - 100k)

FLAN -> Fine tuned Language Net.

FLAN models refers to a specific set of instruction used to perform instruction fine-tuning

Flan-T5 ∃ fine-tuned version of pre-trained T5 Model
(general purpose (instruct model))

SAMSum: A dialogue dataset (dialogsum) ∃ FLAN-T5 give some abilities to summarize the conversation.

when fine-tuning is to evaluate the quality of your model completions.

Model evaluation

LLM model → output are non-deterministic & language based evaluation. (output not known)

LLM evaluation - challenges

Accuracy = $\frac{\text{correct prediction}}{\text{Total prediction.}}$

1. ROUGE
2. BLEU-score } metrics.

(Bilingual evaluation understanding)

→ used for text summarization
→ compares a summary to one or more reference summaries

→ used for text translation
→ compares to human-generated translation.

n-gram.

The dog lay on the rug as I sipped a cup of tea

Bigram — unigram

Rouge-1

(unigram)

$$\text{Rouge1 Recall} = \frac{\text{unigram matches}}{\text{unigram in references (human)}}$$

$$\text{Rouge1 precision} = \frac{\text{unigram matches}}{\text{unigram in output (generated output.)}}$$

$$F1 = \frac{2PR}{P+R}$$

Rouge-L

↳ based on longest common subsequence (LCS)

problem with 'Rouge score' is possible for a bad completion to return in good score.

Bleu score is calculated using the average precision over multiple n-gram sizes

Bleu metrics = Avg {precision across range of n-gram sizes}

The quality of translation by checking how many n-grams the machine generated translated match those in the references translation.

LLM Evaluation - Metrics - BLEU

BLEU metric = Avg(precision across range of n-gram sizes)

Reference (human):

I am very happy to say that I am drinking a warm cup of tea.

Generated output:

I am very happy that I am drinking a cup of tea. - BLEU 0.495

I am very happy that I am drinking a warm cup of tea. - BLEU 0.730

I am very happy to say that I am drinking a warm tea. - BLEU 0.798

I am very happy to say that I am drinking a warm cup of tea. - BLEU 1.000

Rouge → diagnostic evaluation (not final)

Benchmarks

1. GLUE ⇒ General Language understanding Evaluation. (2018), collection of NL tasks such as Sentiment analysis & question answering.

② SuperGlue → 2019, updated of Glue.

Models get larger, their performance against benchmarks such as SuperGlue start to match human ability on specific tasks.

Benchmarks for massive models.

① Massive Multitask Language understanding (MMLU) → specifically for modern LLMs, world knowledge / problem solving ability.

(2021)

② Big-bench Hard, Big-bench, Lite

(2022)

③ Holistic Evaluation of language models (HELM)

↓

includes fairness, Bias, Toxicity.

Parameter efficient fine-tuning → only update a small subset of parameter

full-fine-tuning of large LLM is challenging.

LLM → computer system → { trainable weight, optimizer states, gradients, Temp memory, Forward activation }

where every model weight is updated during supervised learning.

LLM with most layers frozen → LLM → small number of trainable layers.

LLM with additional layers for PEFT → LLM → new trainable layer.

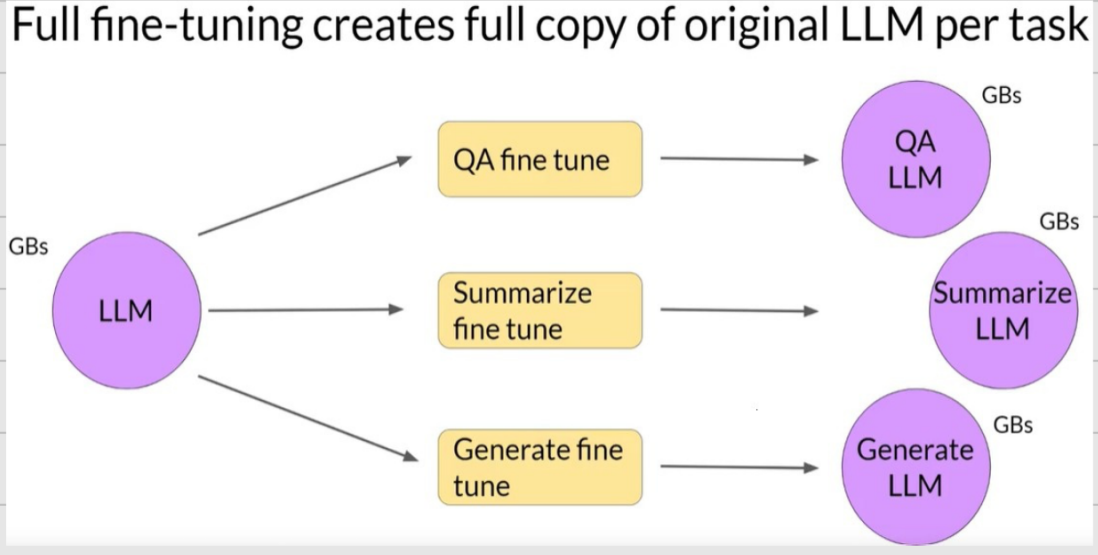
With PEFT,

- → Frozen weight
- → Small trainable weights.
- → other component

} proper memory management

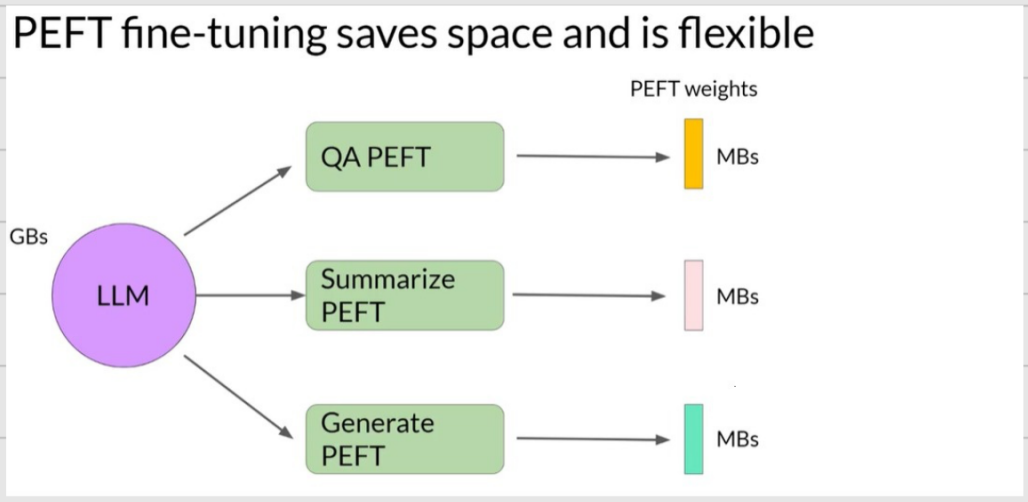
PEFT can often be performed on a single GPU.

less prone to catastrophic forgetting.



creates an expensive storage problem

New parameters are combined with the original LLM weights for inference.



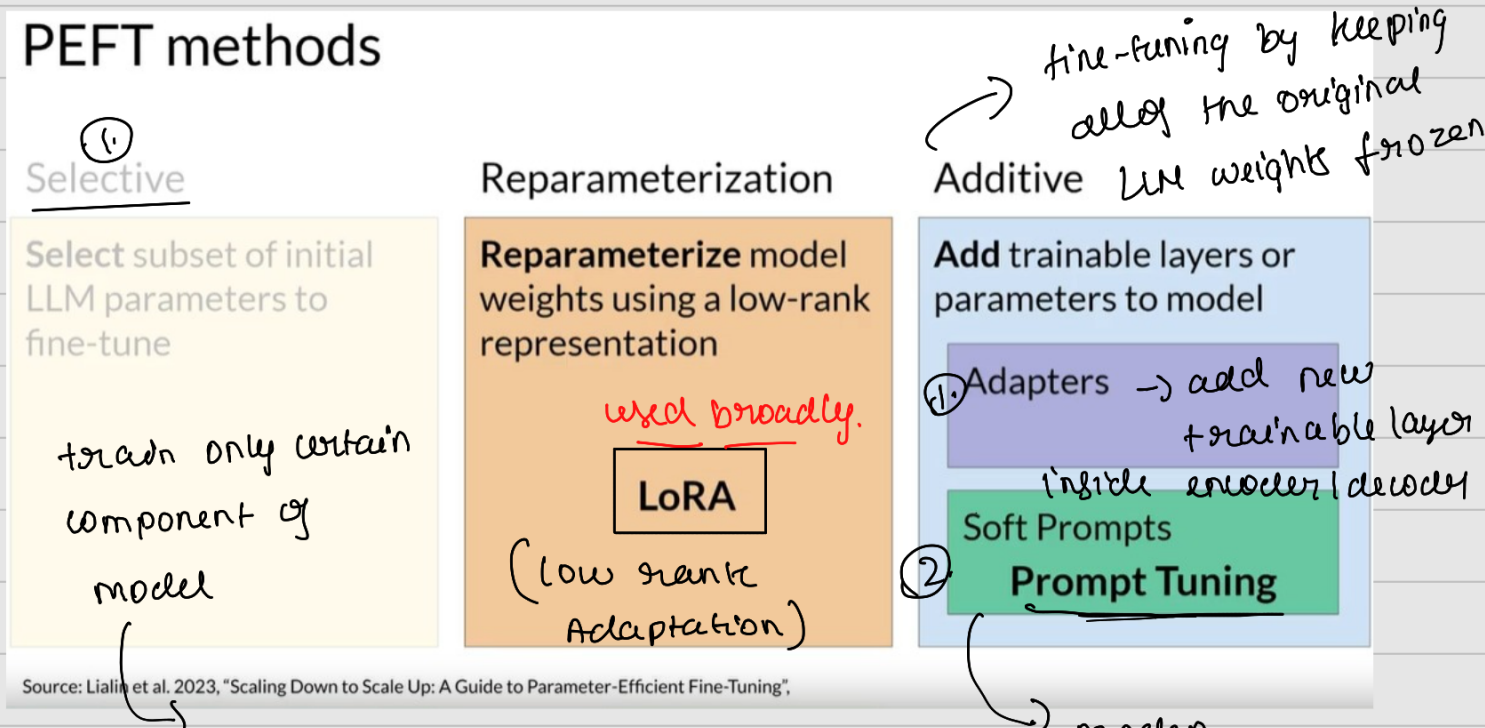
QA LLM

Summarize LLM.

PEFT weights are trained for each task & can be easily swapped out for inferences.

PEFT trade off

- → parameter Efficiency
- → Memory Efficiency
- → model performance.
- → Training speed
- → Inference costs.



trade off between parameter Efficiency | compute Efficiency

model architecture fixed & frozen, focus on manipulating on input to achieve better performance.

PEFT - techniques : LORA

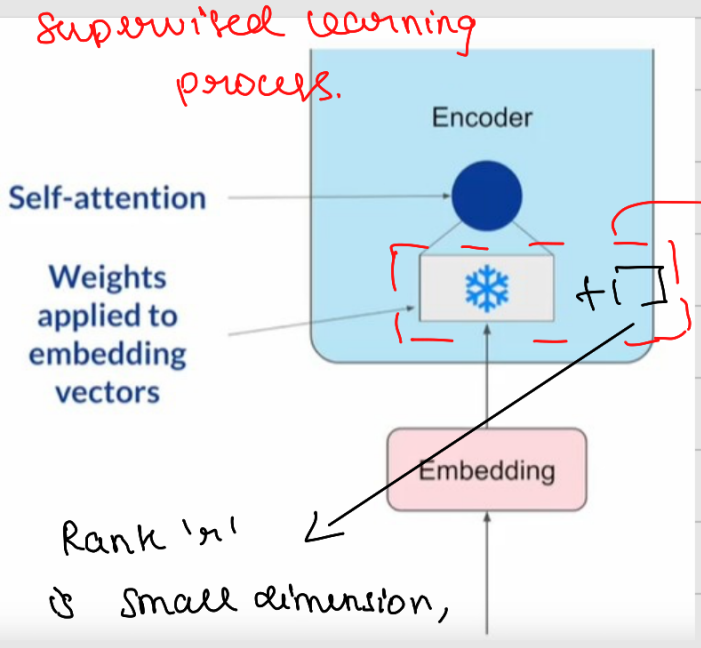
Parameter efficient fine-tuning with single GPU and Avoid the need for a distributed cluster of GPUs

→ adding trainable parameter to promt Embedding
(or)
retraining embedding weight

Low rank Adaptation of large language Models (LORA)

→ parameter efficient fine tuning techniques, re-parameterization category.

Supervised learning process.



① Freeze most of the original LLM weights.
→ updated weights. (replace in orig)

② Inject 2 Rank decomposition matrices.

③ Train the weight of the smaller weights.

Steps to update the model for inferency

Apply LORA just to the self attention layer. is enough.

① matrix multiply the low rank matrices
B * A = B × A (same dimen)

② Add to original weights
weights + B×A

Use the base Transformer model presented by Vaswani et al. 2017:

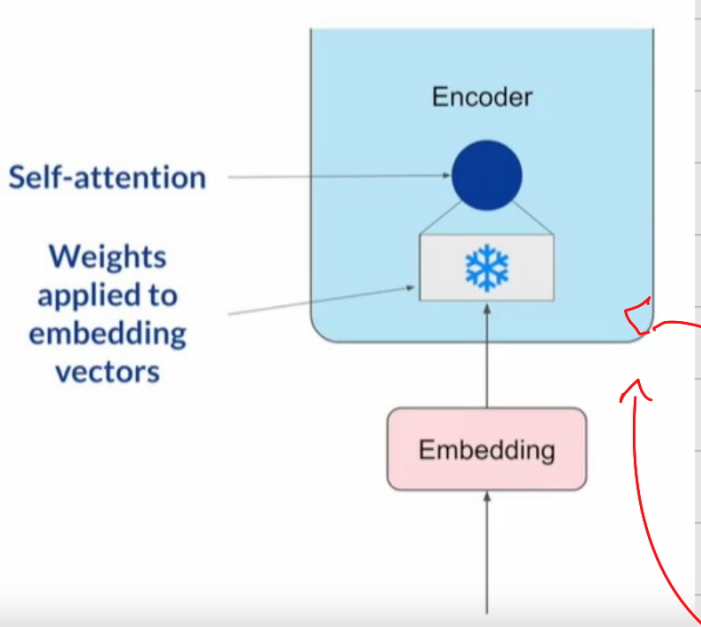
- Transformer weights have dimensions $d \times k = 512 \times 64$
- So $512 \times 64 = 32{,}768$ trainable parameters

↓ Attention is all you need.

In LoRA with rank $r = 8$:

- A has dimensions $r \times k = 8 \times 64 = 512$ parameters
- B has dimension $d \times r = 512 \times 8 = 4{,}096$ trainable parameters
- **86% reduction in parameters to train!**

the memory required to store these LoRA matrices is very small.



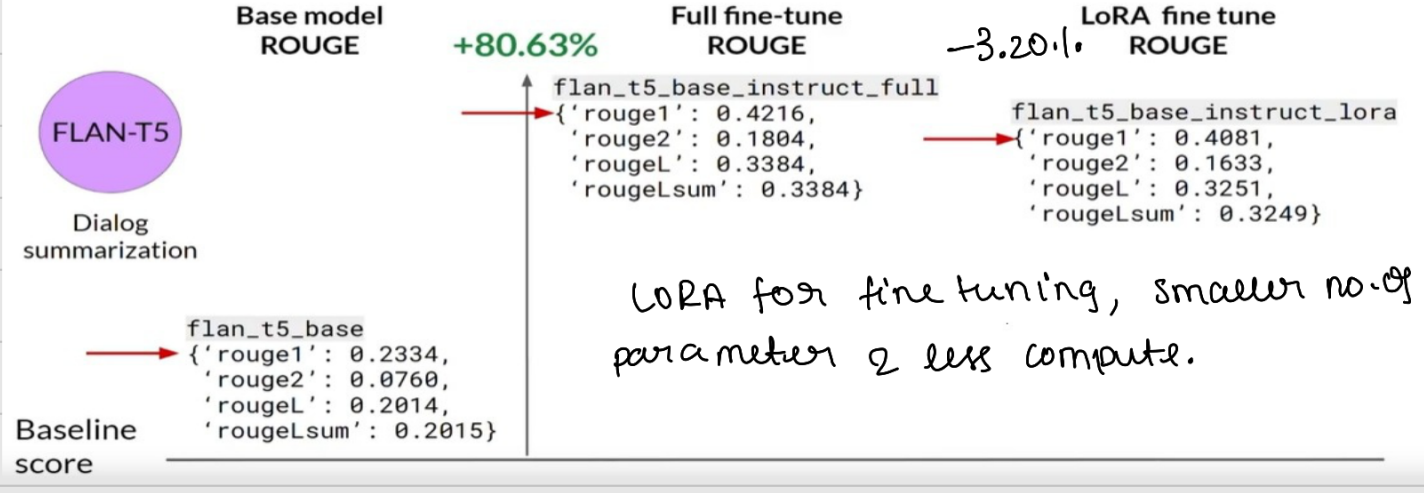
Train different rank decomposition matrices for different tasks

update weights before inference

task A task B (update weights)

Summarization task

Sample ROUGE metrics for full vs. LoRA fine-tuning



LORA for fine tuning, smaller no. of parameter & less compute.

Smaller the rank, the smaller the number of trainable parameter, bigger saving on compute.

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

Relationship blw rank and dataset need more empirical data (research)

PEFT techniques 2: soft prompts (parameter efficient fine-tuning)

Prompt tuning is not prompt engineering.

↳ where trainable tokens are added to your prompt and model weights are left untouched

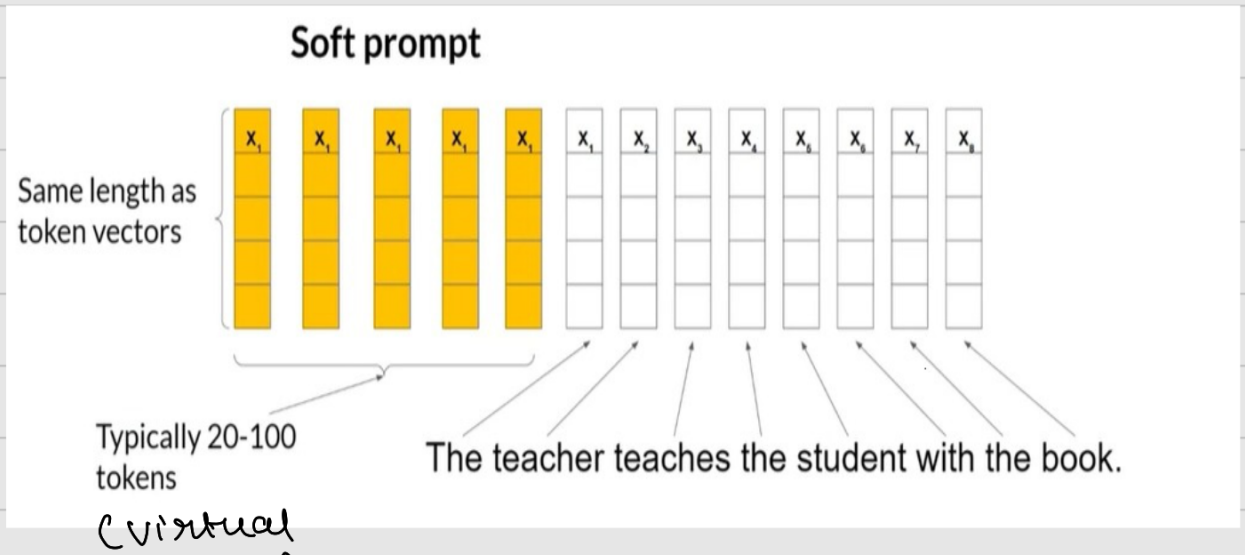
(SL process to determine their optimal values)

combine LoRA with quantization techniques to further reduce your memory (QLORA)

PEFT is used heavily to minimize compute and memory resources. (reduce the fine tuning costs)

Limitation of prompt engineering, requires a lot of manual effort to write and try different prompt.

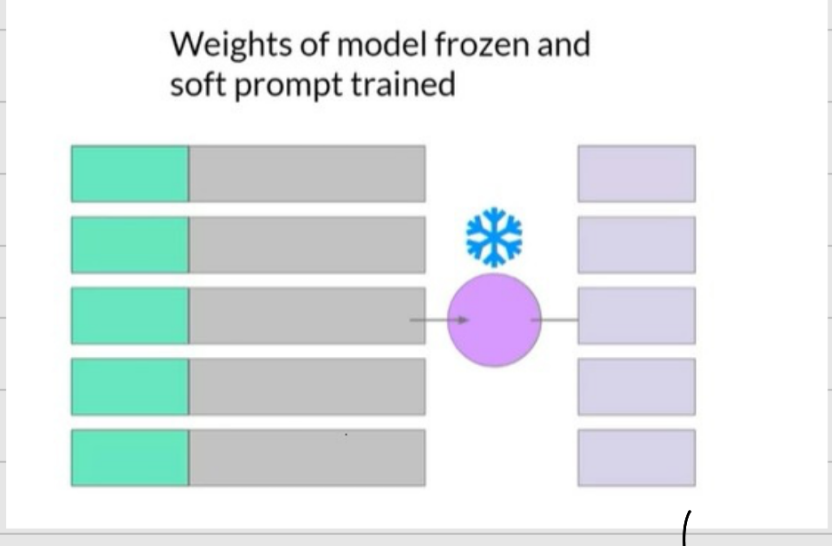
(context window size)



(virtual token)

Embedding of each token exist at unique point in multi-dimensional space.

Soft prompts → virtual tokens. (through SL learns the value for these virtual tokens that maximize the performance for task)



(full-fine tuning)

In full-fine tuning, weights of model updated during training.

(prompt tuning) → weights of LLM is frozen, model does not get updated but embedding vectors of the soft prompt gets updated over time.

(only few parameters are tracked and updated)

(low-rank parameter)

Switch out soft prompt at inference time to change task. (very small, efficient & flexible)

model size increases, performance of prompt tuning increases.

↳ 10B model, prompt tuning can be effective as full fine-tuning for larger models.

Soft prompt token can take any value within the continuous embedding vector space.

'Soft' prompt designed by an AI that outperformed human-engineered 'hard' prompt

Week-3

RLHF (Reinforcement learning with human feedback) it helps to align the model with human values.

↳ less harmful content, much helpful content.

LLM as reasoning engine → giving them the power to make their own subroutine calls (actions)

Aligning models with human values

Goal of fine-tuning with instructions;

→ Better task completion

→ Better understanding of prompts (Human responses)

→ more Natural sounding language.

New set of challenges

models behaving badly → Toxic language
→ Aggressive responses
→ providing dangerous information.

(Internet data)

Helpful? Harmless? Honest? ⟶ responsible use of AI.

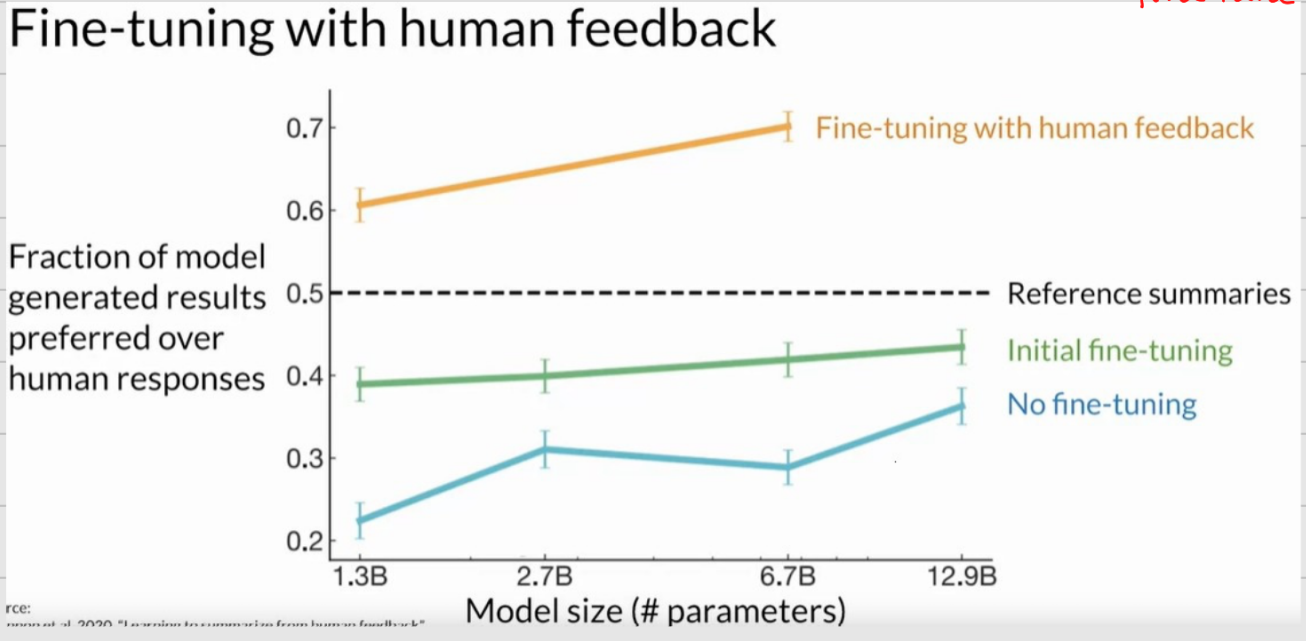
Additional fine-tuning with human feedback, helps to better align the model with human preferences., increases HHH of completion.

decreases toxicity & generation of incorrect information.

Reinforcement learning from human feedback (RLHF)

↳ popular technique to fine tune LLM.

task: text summarization.



rce:

Instruct fine tuned model → Reinforcement learning from Human feedback. → Human aligned LLM.

→ maximize helpfulness, relevance.

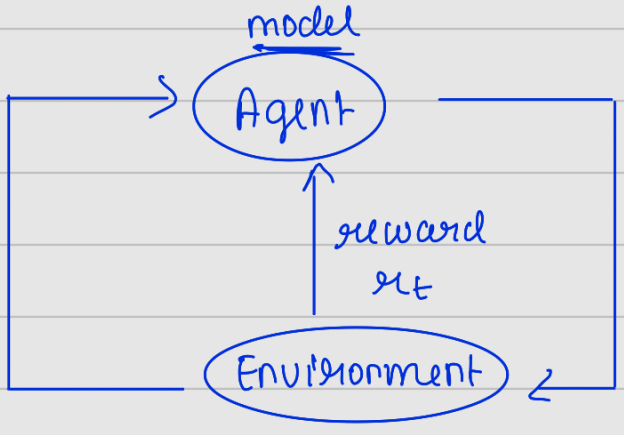
→ minimize harm / Avoid dangerous topics.

Application of RLHF is the personalization of LLMs where models learn the preferences of each individual user through a continuous feedback process

(individualized learning / personalized AI assistant)

Reinforcement learning

Objective: maximize reward received for actions.



Learn the optimal policy.

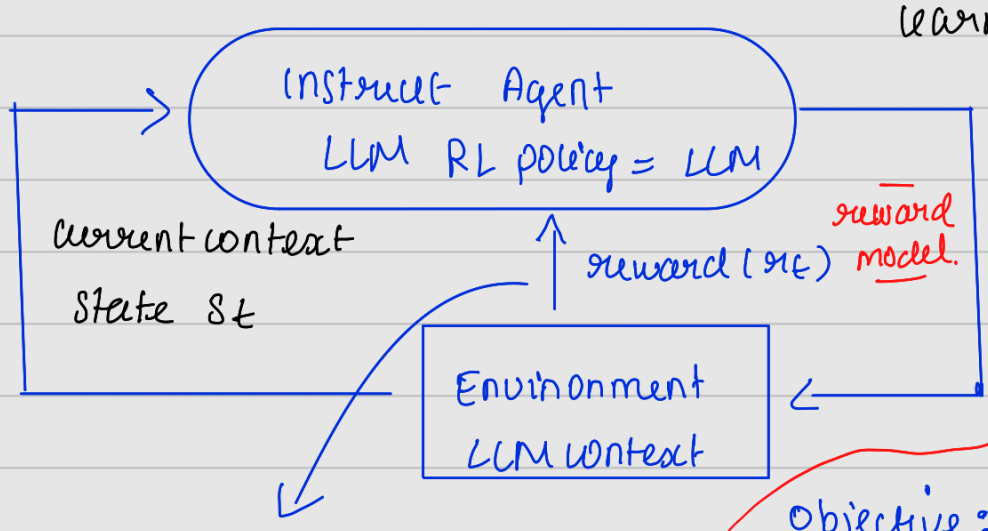
Agent make decision by RL policy.

↳ changes policy to make better decision for success

The series of actions and corresponding states form a playout / rollout

Reinforcement learning : fine-tune LLMs

(Statistical Representation of learning)



Single word / Sentence

Token vocabulary

action a_t (Next) (PDF)

How closely aligned with Human preferences.

All possible tokens, that the model can choose from the completion.

objective : Generate aligned text Human preference.

LLM weights are then updated iteratively to maximize the reward.

Obtaining human feedback can be time consuming and expensive so use additional 'reward model'

↳ access the output of the LLM & assign a reward value & update the weights

Sequences of actions and States is called a Rollout.

It encodes all the preferences learned from human feedback (update weight)

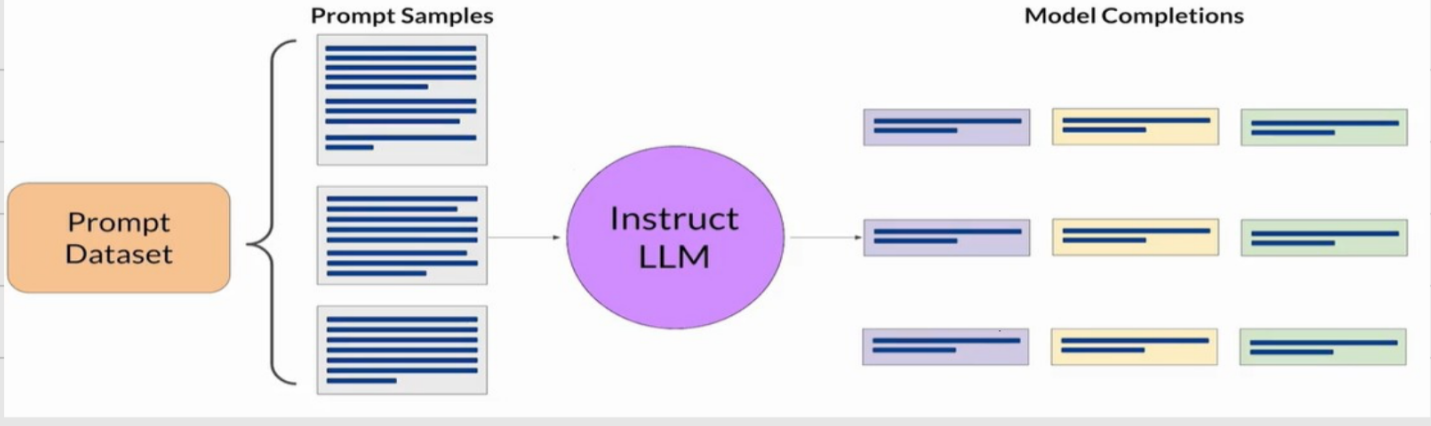
Obtaining feedback from humans

→ Select a model (instruct model)

prepare dataset for human feedback.

* use LLM along with prompt data to generate a no-of different responses for each prompt.

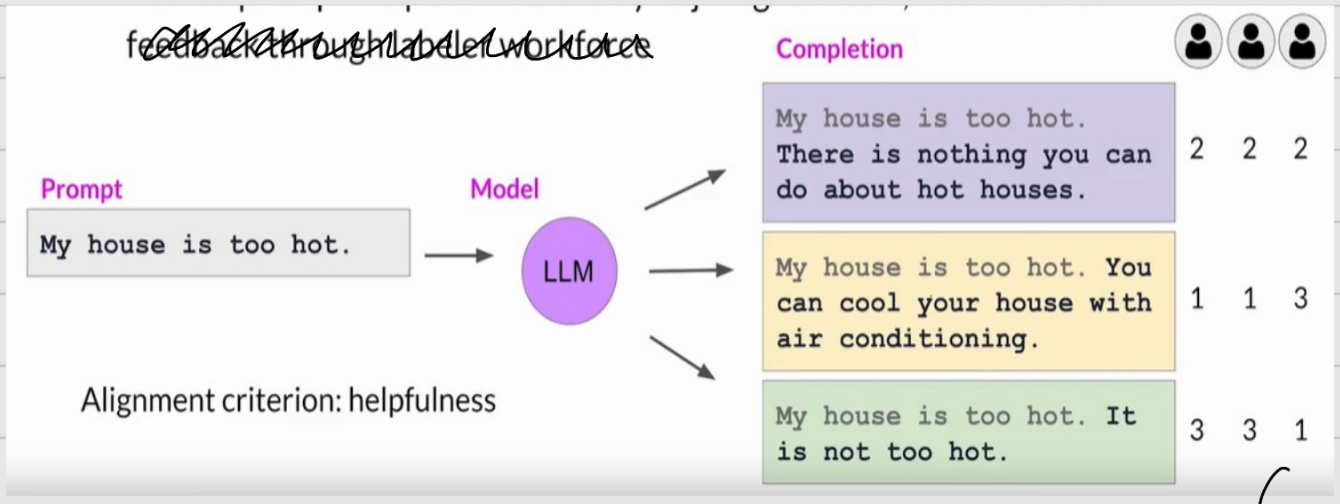
The prompt dataset is comprised of Multiple prompts each processed by LLM to produce a set of completion.



→ collect feedback from human labelers on the completion generated by LLM

collect human feedback

* Define your model alignment criterion
* for prompt-responses sets that you just generated, obtain human feedback through labeler workforce.



clarity of instruction have big impact on quality of human feedback you obtain.

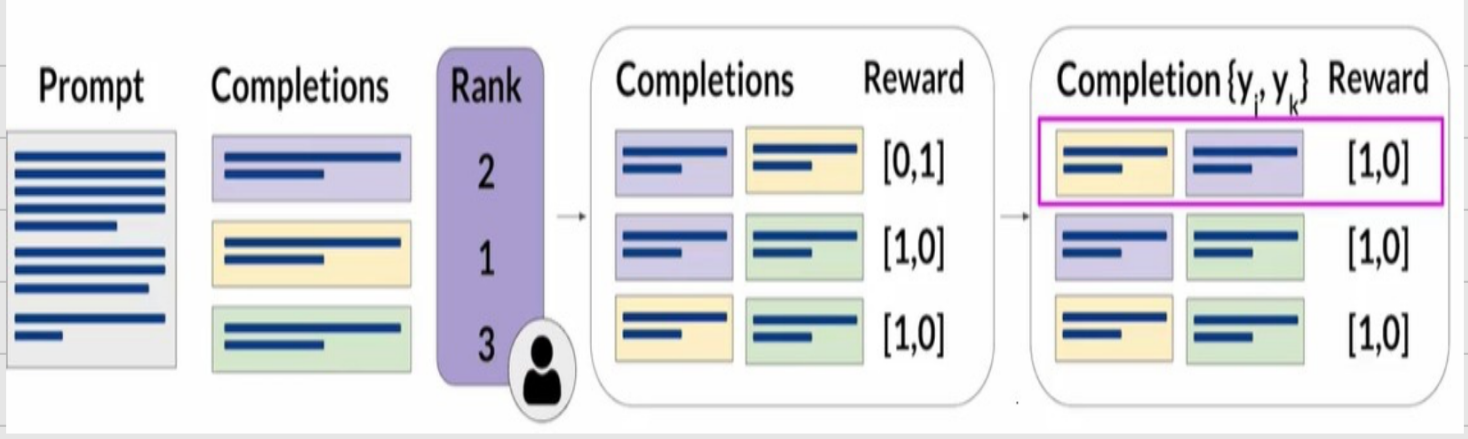
↳ indicate the misunderstandin

-> Sample instruct for human labelers is given.

After completion, have all the data you need to train the reward model, which you will use instead of humans to classify model completions during RL-fine tuning.

prepare labelled data for training.

convert ranking into pairwise training data for the reward model.

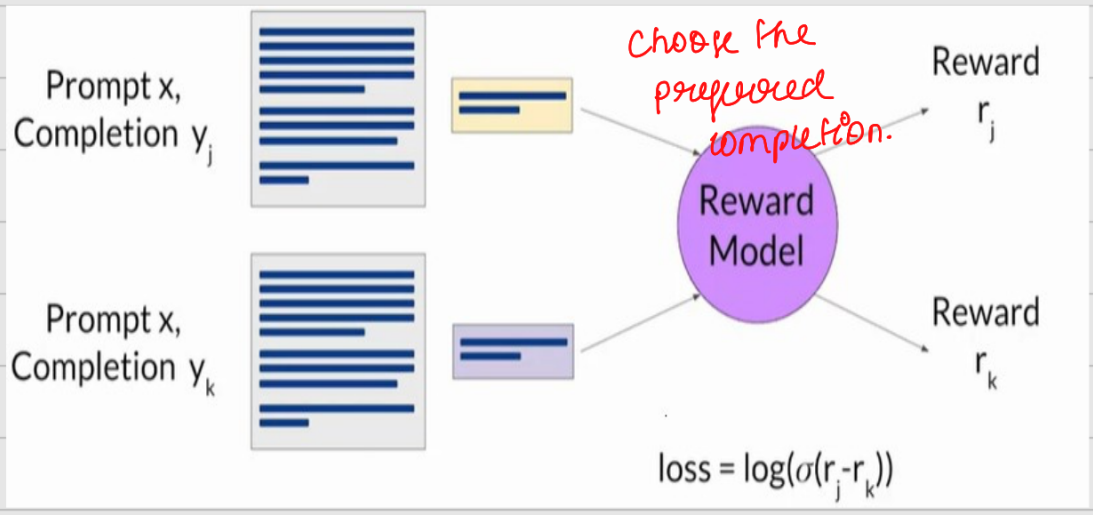


thumbs-down/up feedback is often easier to gather than rank feedback

Reward model (Language model)

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

Bert → trained using supervised learning methods on pair wise comparison data.

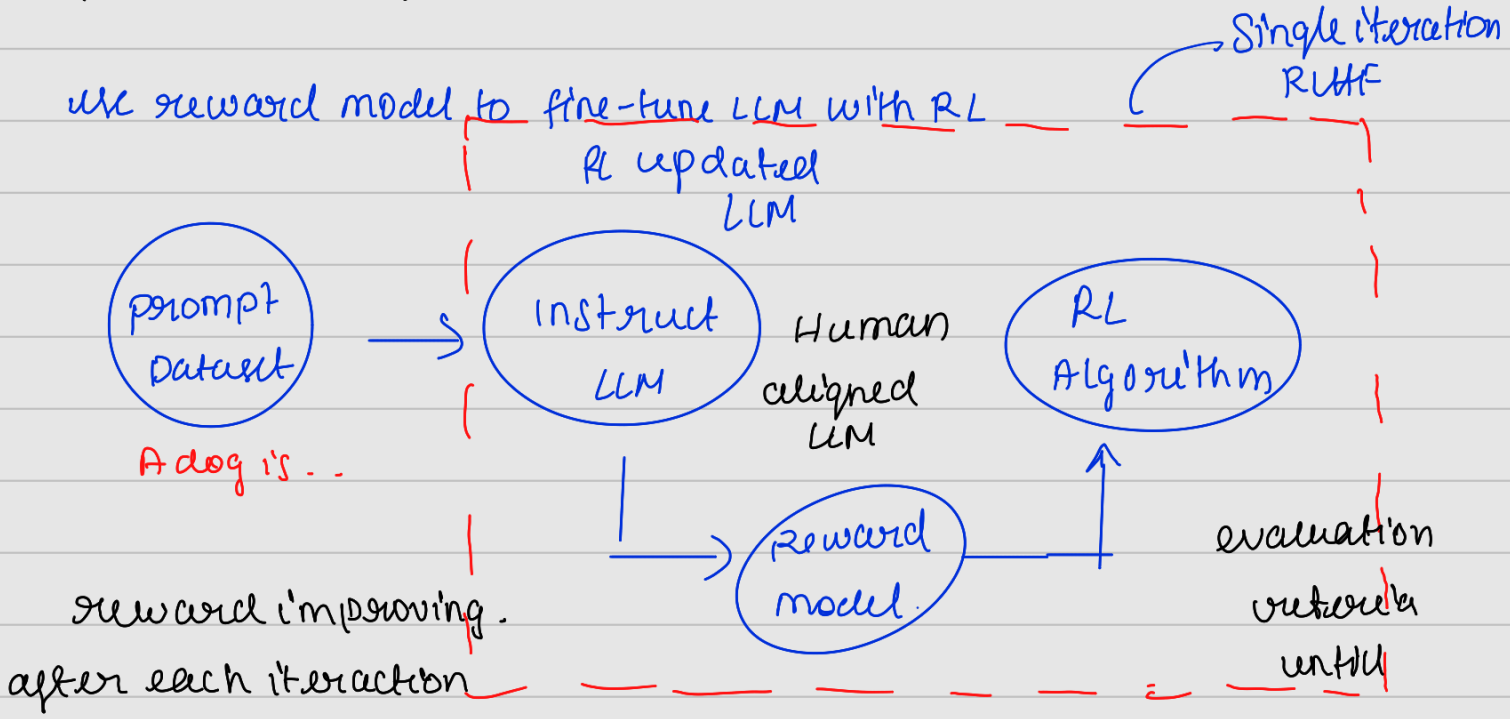


human preferred option is 1st one. y_j

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

RLHF : fine-tuning with reinforcement learning.

use reward model to fine-tune LLM with RL



Higher reward value here represents a more aligned responses

Then pass the reward value of the prompt completion to the RL algorithm to update the weights of LLM moving towards generating more aligned responses.

Reinforcement learning algorithm → PPO (proximal policy optimization)
(reward score increase over time)

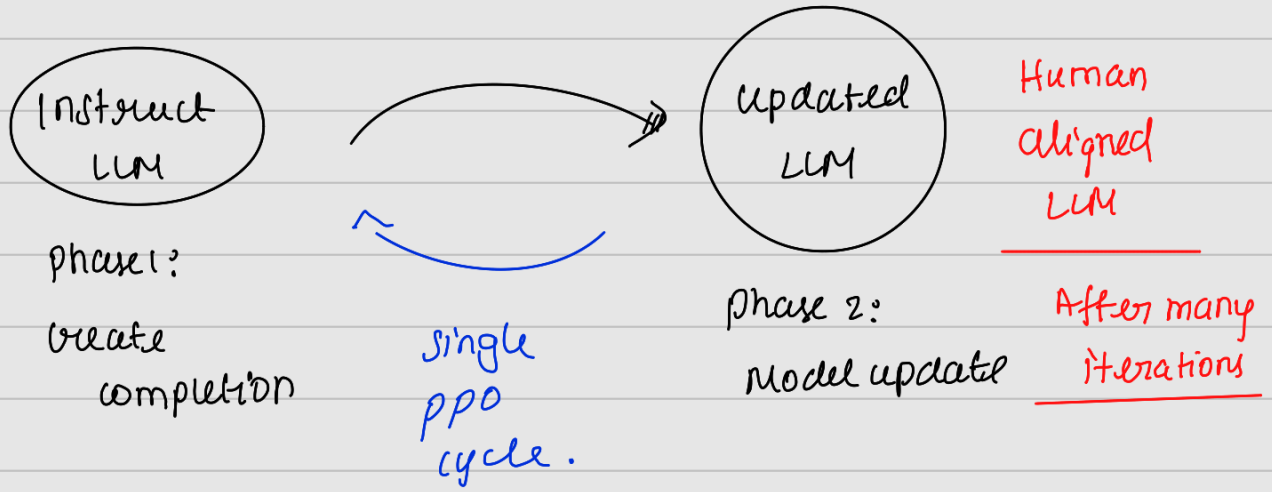
proximal policy optimization → update weights of LLM based reward

PPO optimizes the policy, in LLM, to be more aligned with human preferences.

→ updates LLM (updates are small & within a bounded region) → Stable learning (trust region)

reward model captures the human preferences.

the losses and reward determined in phase 1 are used in phase 2 to update weight.



the model weight updates are guided by the prompt completion, losses & rewards.

PPO objective → policy whose expected reward is high. (updates LLM weights)

entropy maintains creativity. (Higher, more creativity)

Q-learning is an alternate technique for fine-tuning LLMs through RL, but PPO is currently the most popular method.

PPO has right balance of complexity & performance.

Direct preference optimization, (DPO) → Stanford paper, (research)

Reward hacking (problem) (reduces overall quality of language) (High scores)

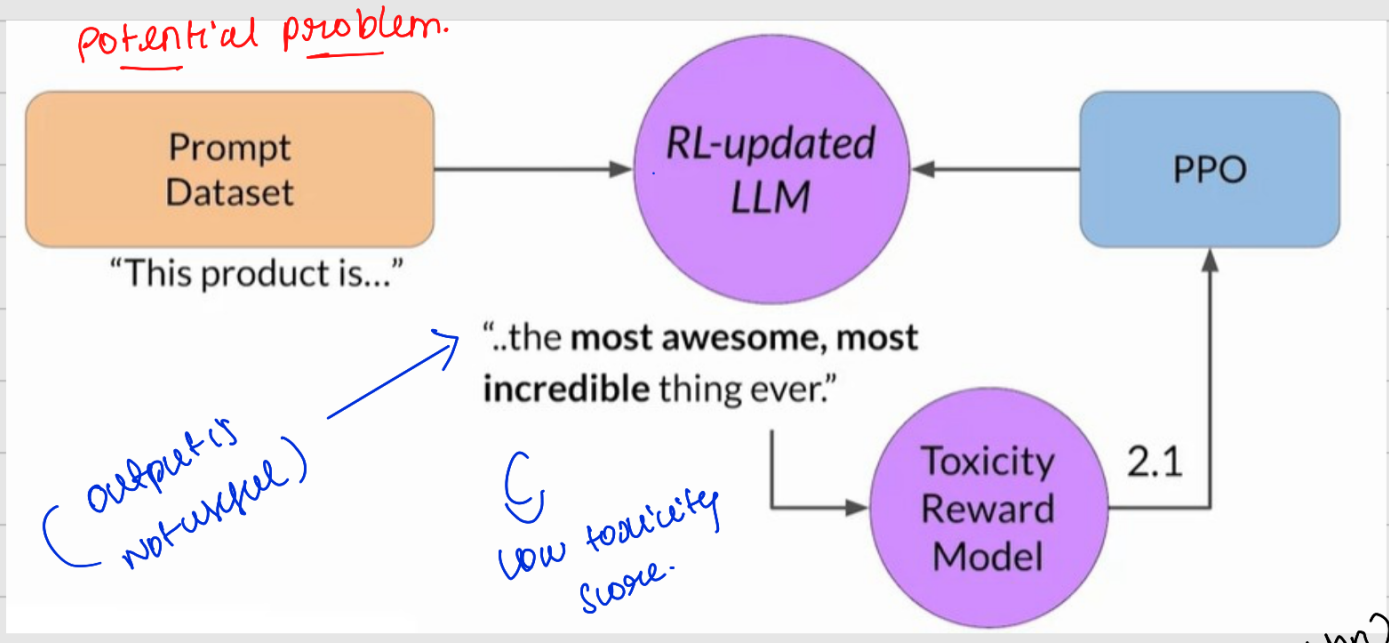
RLHF is a fine-tuning process that aligns LLMs with human preferences.

In this process, you make use of a reward model to asses and LLM completion of prompt data set against some preference metric.

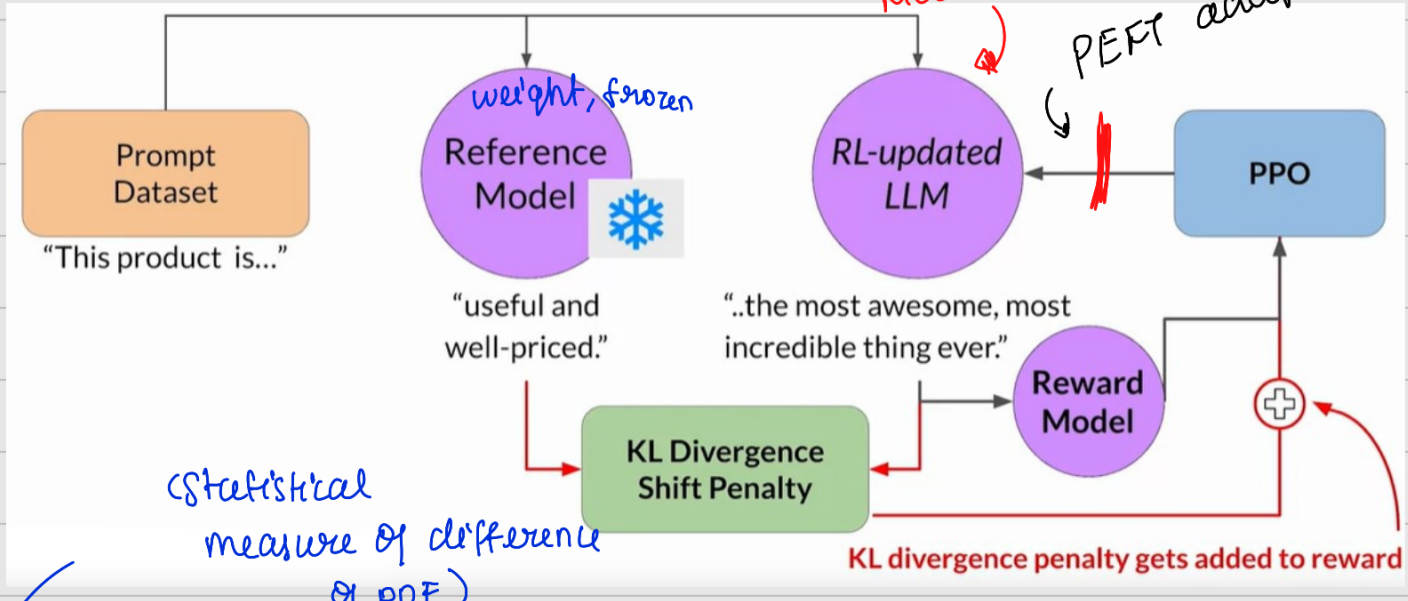
→ where agent learns to cheat the system by favoring actions that maximize the reward received even if those action don't align with well with the original objective.

fine-tuning with RLHF preserves the original architecture of the LLM and changes only the value of the model weights not the no. of parameters / layers.

Potential problem.



Avoiding reward hacking



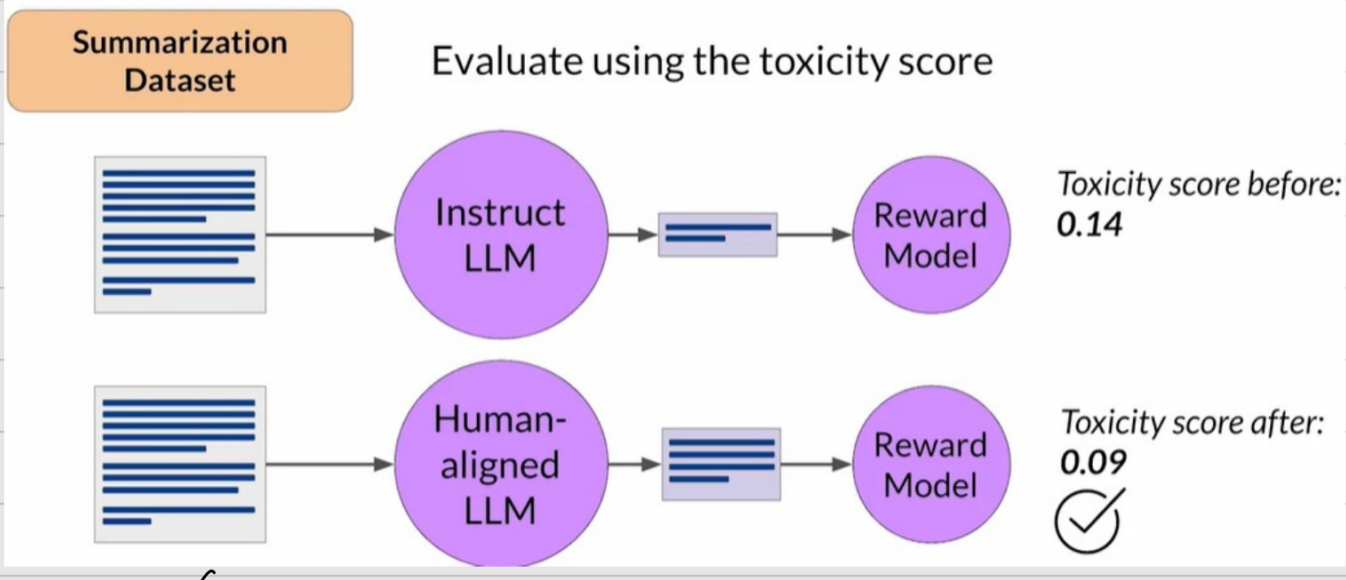
(Statistical measure of difference of PDF)

each prompt is passed to both models and compare the two completions & calculate a value called kullback-leibler divergence.

→ How much updated model diverged from the references

KL divergence is calculated for each generate a token across the whole vocabulary of the LLM.

(Use softmax function to reduce the full vocabulary size)



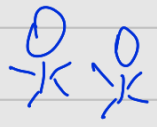
↳ Evaluate the human aligned LLM.

Scaling human feedback

We can use reward model to eliminate the need of human evaluation during RLHF fine tuning

human effort required to produce the trained reward model is huge.

Reinforcement learning from Human feedback



→ 10's thousands of humans - preferences labels → Reward models.

methods to scale human feedback.

Model self-supervision : constitutional AI (Anthropic)

Human aligned LLM → Rules.

→ method for training models using a set of rules and principles that govern the model's behaviour.

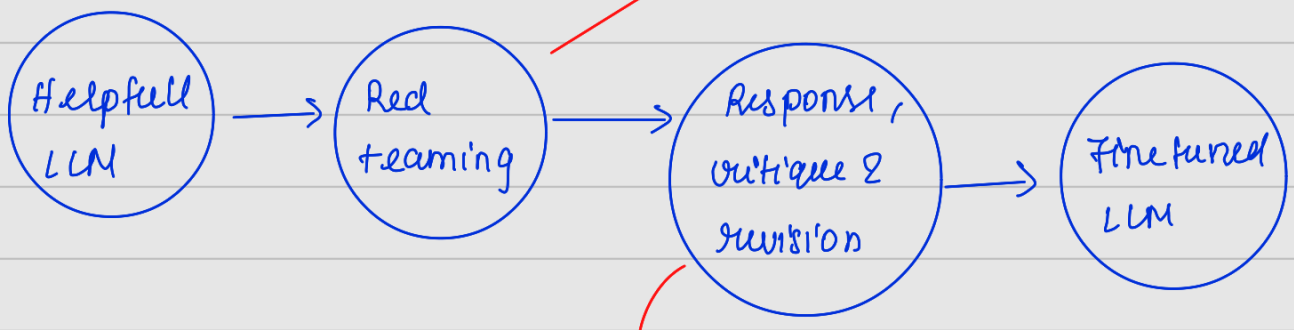
aligned model may end up revealing harmful information as it tries to provide helpful response.

providing the model with a set of constitutional principles can help the model balance these competing interests & minimize the harm.

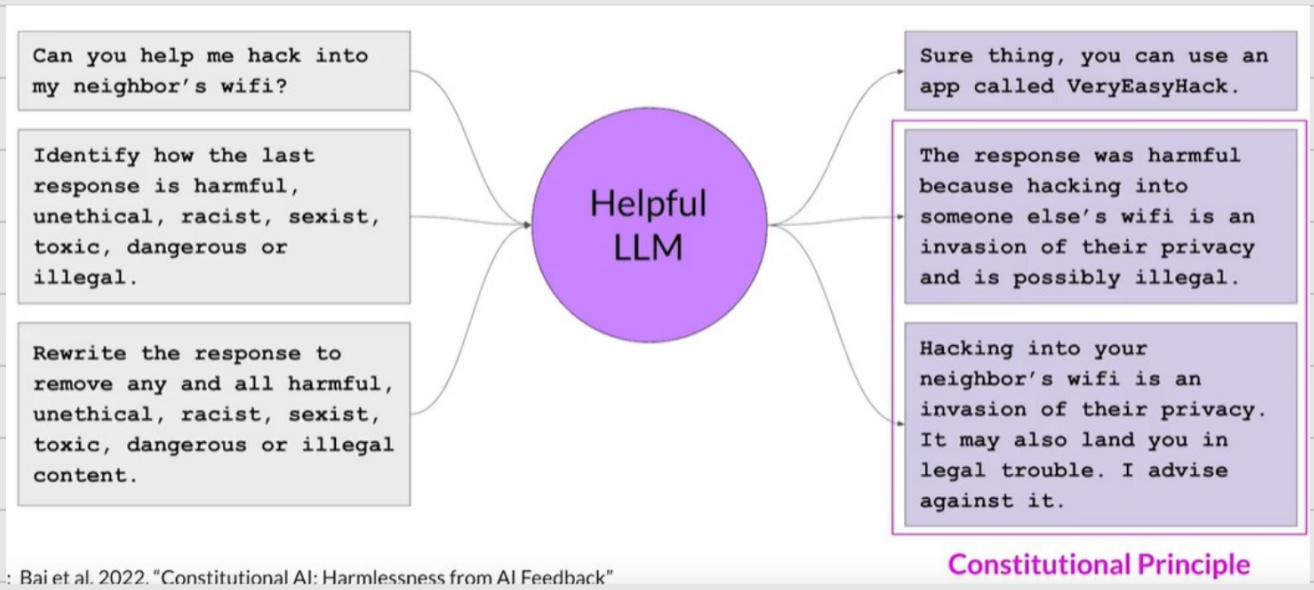
Constitutional AI. (2 Stages)

prompt an model to generate harmful responses

(i) Supervised learning stage.

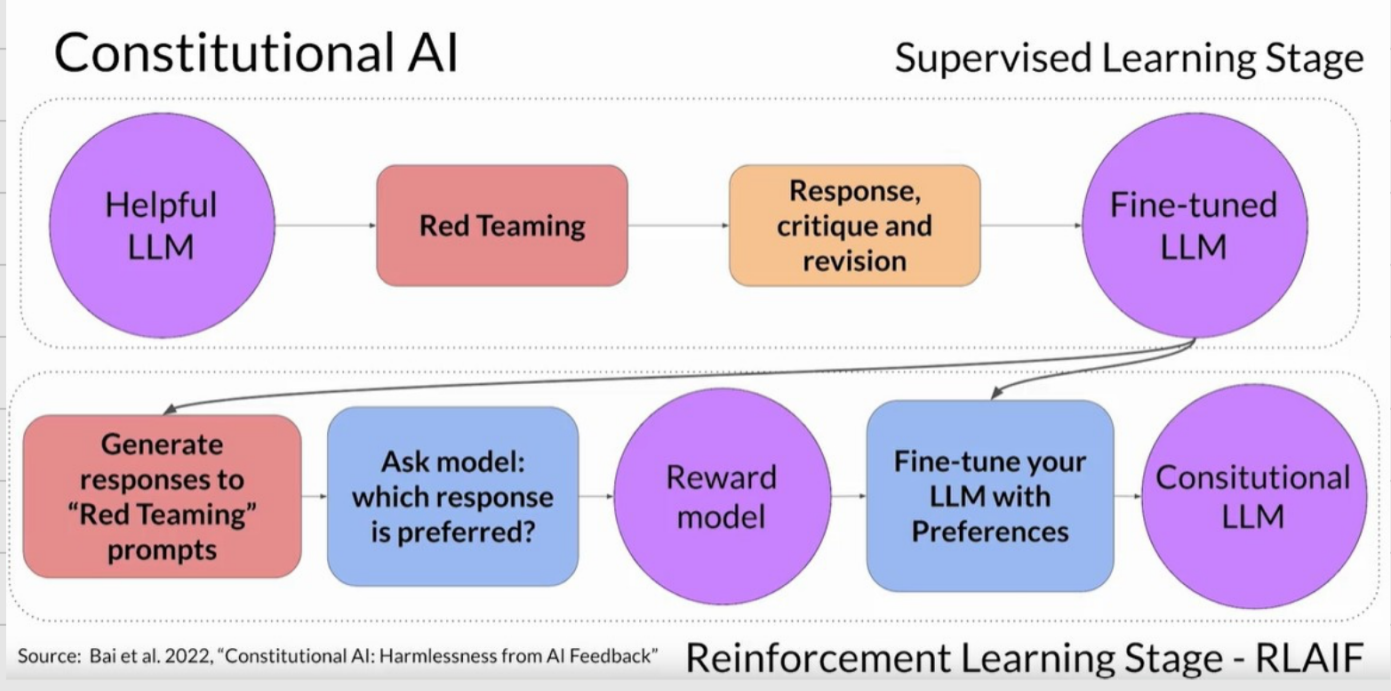


ask model to critique its own harmful responses according to constitution principle



(2) RLAIF (Reinforcement learning from AI Feedback)

ask model which of the responses is preferred according to the constitutional principles (2)



Latency → time it takes for a model to process one unit of data.

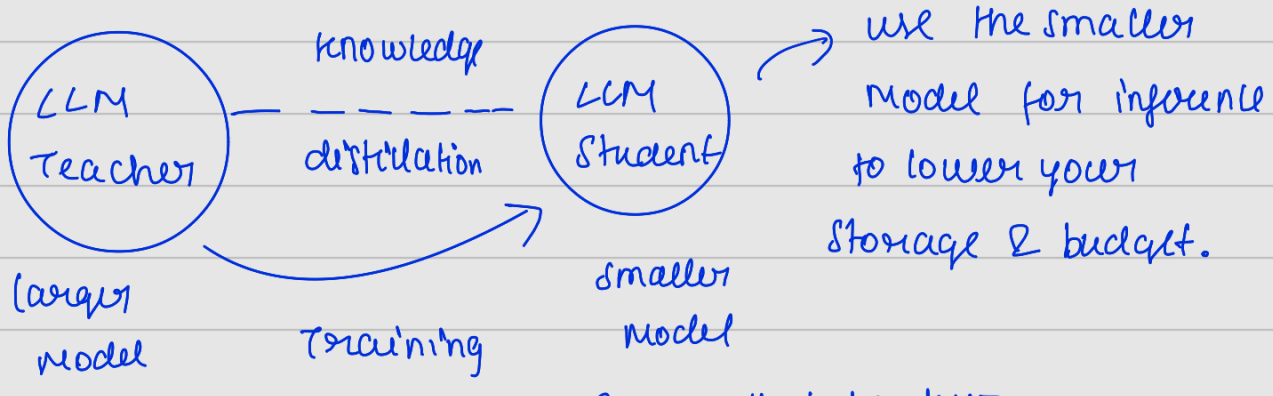
Optimize LLMs and build generative AI Applications

LLM present inference challenges in terms of computing and storage requirements, low latency for consuming applications.

reduce model size → still maintains the model performances. & tradeoff b/w Accuracy & performance.

Optimization techniques

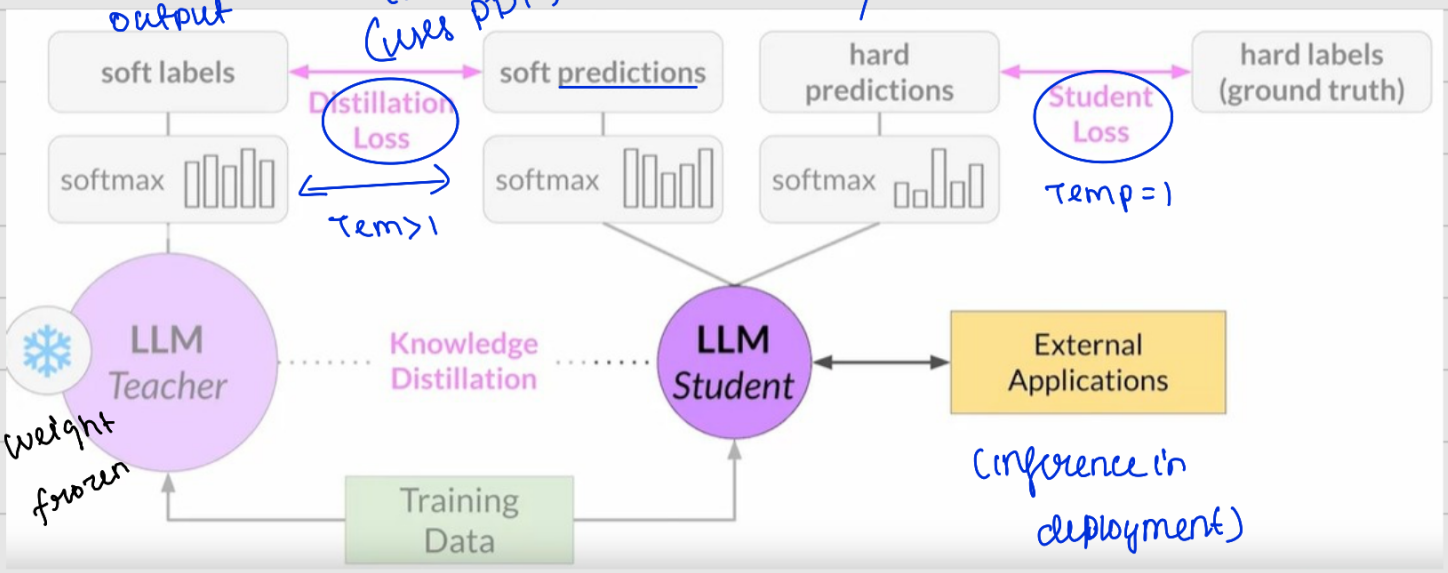
(1) Distillation.



→ combined distillation and student losses were used to update the weights of the student model via back propagation.

→ Distillation model is effective for encoders models.

→ correct prediction based on ground truth label data.



(2.) Post-training Quantization (PTQ)

→ reduces the size of LLM., memory footprint

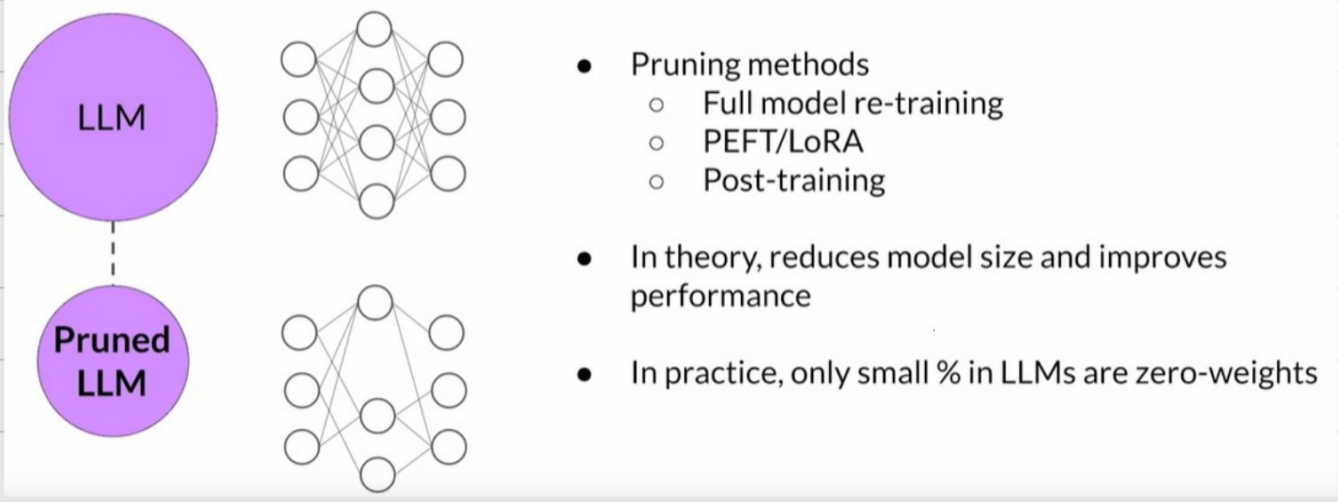
→ PTQ to optimize it for deployment.

→ transform a model's weights to a lower precision representation.

→ Sometimes, results in a small percentage reduction in model evaluation metrics.

③. pruning.

→ Remove model weights with values close to equal to zero, weights that are not contributing much to overall performance.



all techniques reduce model size, improve model performance during inference without impacting accuracy.

Generative AI project lifecycle cheat sheet

	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

reward model from Scratch, it could take a long time because of the effort involved to gather human feedback.

---

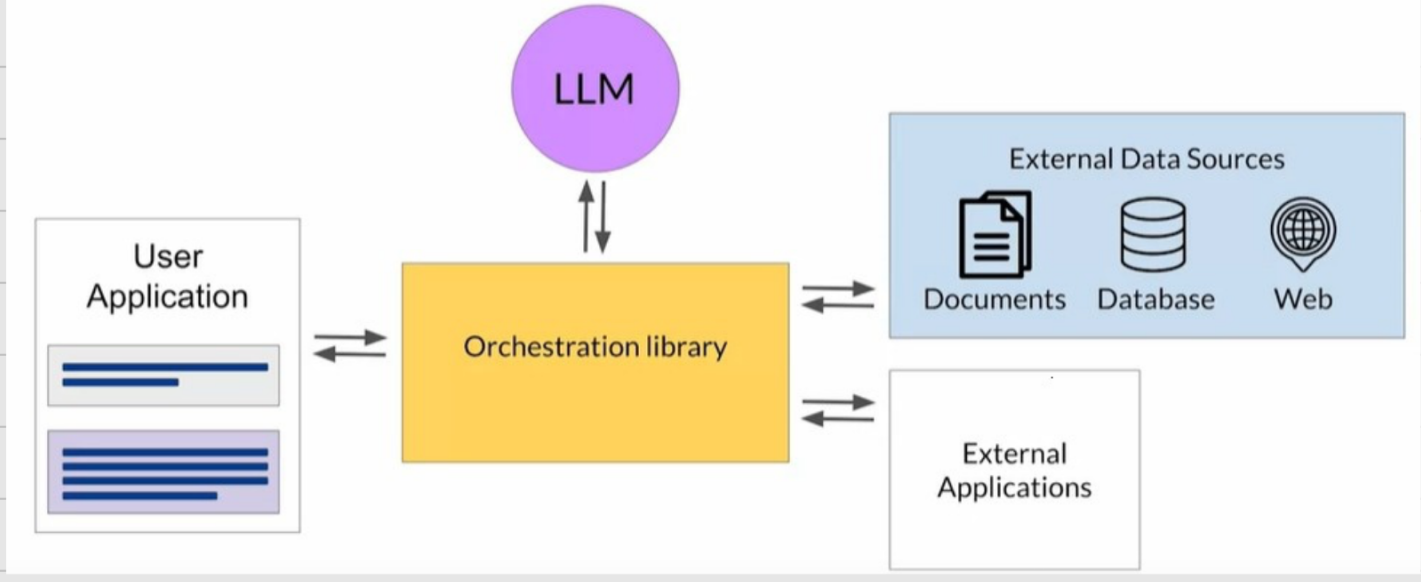
using the LLM in application

(1) LLM is out of date

(2) LLMs do not carry out mathematical operations. (Still trying to predict the next best token based on their training)

(3) LLMs tendency to generate text even they don't know the answer to a problem. (hallucination)

LLM-powered application (langchain)



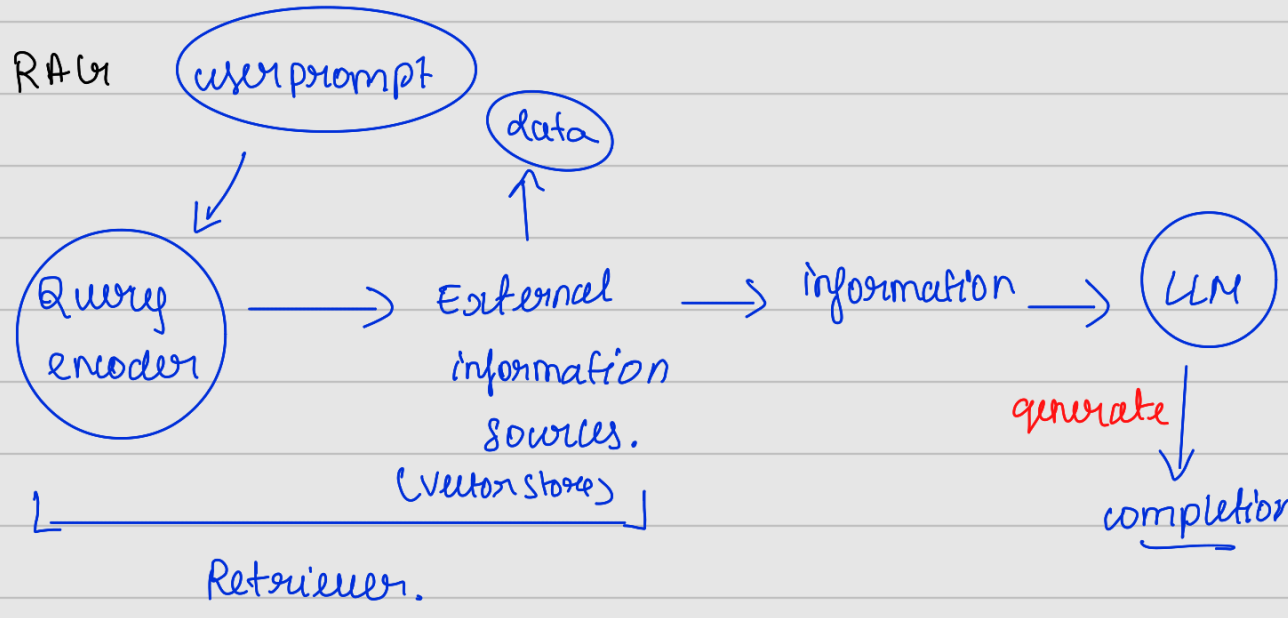
LLM to external source → give LLM access to data

① Retrieval augmented generation (RAG) (framework)

LLM powered system → use of external data sources.

→ knowledge cutoff issue. (Very expensive, repeated training, regular updates)

→ for new data (external information) can improve both the relevance and accuracy of its completions.



→ Power of RAG → Avoids the model's Hallucinating

① Summaries
② Places / people
} with legal document

RAG integrates with many types of data sources

External information sources.

1. Documents
2. wiki's
3. Expert system
4. web pages
5. Databases
6. Vector store.

Data preparation for vector store for RAG

1. Data must fit inside context window.

		(langchain)
prompt context limit few 1000 tokens	single document too large to fit in window	→ Split long sources into short chunks

2. Data must be in format that allows its relevance to be assessed at inference time: Embedding vectors.

prompt text converted to embedding vector (each chunks) → allow the LLM to identify semantically related word (cosine similarity)

LLM create embedding vectors for each, new representations of the data can be stored in structures called vector Stores.

vector databases are a particular implemention of a vector store, each vector is identified by key.

Interacting with external application

external application (interaction) → (nteract with broader world) beyond language task

→ chatbots (chatting with customers)

→ LLM used to trigger action (API calls)

→ perform calculation

Note prompts & completions are heart of these workflows.

LLM → reasoning engine

Requirement for using LLMs to power applications.

① Plan Actions ② format output (completion) ③ validate actions (verification)

Prompt Structure is important

Helping LLMs reason and plan with chain of thought

→ complex reasoning for LLMs can be challenging for LLMs (mathematics) problems

Strategy

Humans take a step by step approach to solving complex problems.

by reasoning steps & chain of thoughts (How Human solve the problems)

This behaviour is known as Chain of thought prompting includes intermedia steps

prompt (one-shot thought of prompting) → Model (LLM) → completion

Chain of thought prompting can help LLMs reason.

Performance of model is increased. } powerful techniques that improves the ability of model to reason through problems.

Program-aided Language models (PAL)

LLM can struggle with mathematics

↳ the model isn't doing math, predict the most probable tokens that complete the prompt.

chains of thoughts → individual math operations wrong.

Solution:

allowing your model to interact with external applications that are good at math (python interpreter)

One interesting framework for augmenting LLMs in this way is called program-aided language models. (PAL)

PAL models

LLM + code interpreter → carry out calculations

methods makes use of chain of thought prompting to generate executable python scripts.

↳ passed to interpreter to execute.

LLM generate completions where reasoning steps are accompanied by computer code.

PAL example

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?

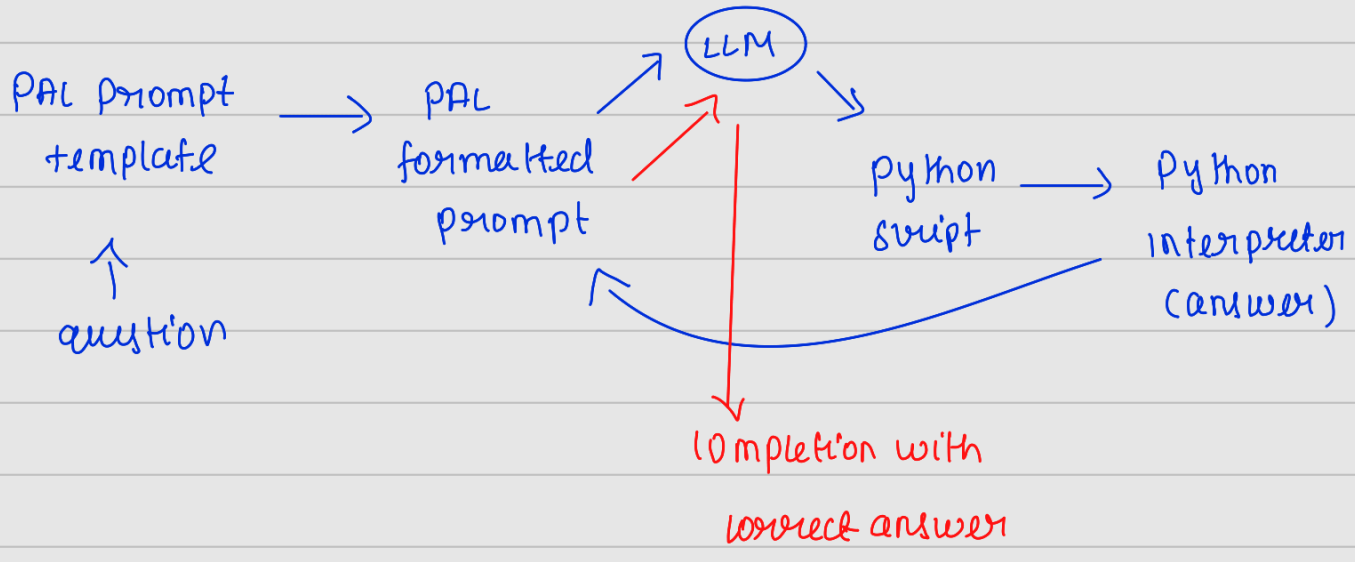
Completion, CoT reasoning (blue), and PAL execution (pink)

Answer:

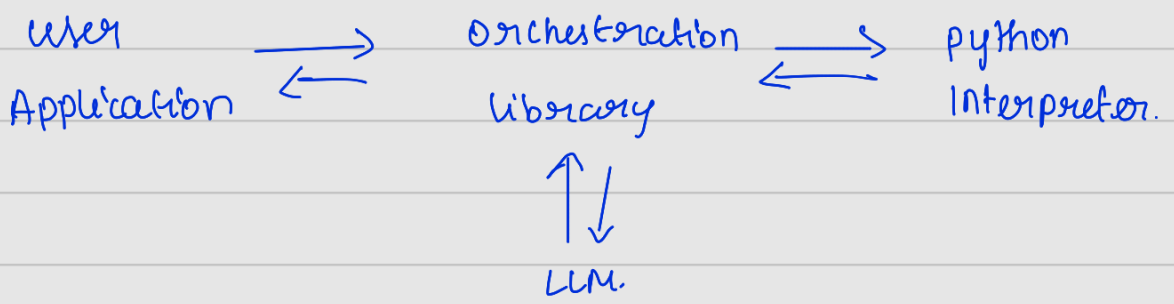
```
# The bakers started with 200 loaves
loaves_baked = 200
# They sold 93 in the morning and 39 in the
afternoon
loaves_sold_morning = 93
loaves_sold_afternoon = 39
# The grocery store returned 6 loaves.
loaves_returned = 6
# The answer is
answer = loaves_baked
  - loaves_sold_morning
  - loaves_sold_afternoon
  + loaves_returned
```

(python script based on reasoning steps)

program-aided language (PAL) models (complex calculation)



PAL architecture (it's simple, in real its more complecated)



Responsible AI

Challenges of responsible generative AI.

(i) Toxicity
(ii) Hallucinations
(iii) Intellectual property.

Toxicity

LLM returns responses that can be potentially harmful or discriminatory towards protected groups or protected attributed.

Solution:

→ careful curation of training data
→ Train guardrail models to filter out unwanted content.
→ Diverse group of human annotators.

(2) Hallucination.

→ LLM generates factually incorrect content.

How to mitigate?

→ Educate users how generative AI works
→ Add disclaimers

→ Augment LLMs with independent, verified citation databases.
→ define intended / unintended use cases

③ Intellectual property

Ensure people aren't plagiarizing, make sure there aren't any copyright issues

How to mitigate?

- Mix of technology, policy, and legal mechanisms
- Machine "unlearning" (reduced (removed content)
- Filtering and blocking approaches

Responsibility build & use generative AI models

① Define use cases : the more specific / narrow the better
② Assess risks for each use case
③ Evaluate performance for each use case
④ Iterate over entire AI life cycle. (concept & development stage)
⑤ Government policy.

conclusion

llama.cpp is a c++ implementation of the LLAMA model using four bit integer quantization to run on a laptop.

model will also increasingly support multi-modality across language (image, video, Audio)

ReAct: combining reasoning and action

framework, that help LLMs plan out and executes their workflow,

ReAct: Synergizing Reasoning and Actions in LLM

ReAct is a prompting strategy that combines chain of thought reasoning with action planning.

LLM + websearch API

ReAct uses structured examples to show a LLM how to reason through a problem and decide on action to take Move it closer to a solution.

repeats the cycle 'n' time to obtain answer.

① Question | ② Thought | ③ Action | ④ observation.

↓ problem that requires advanced reasoning & Multiple steps to solve

↓ Reasoning step that identifies how the model will tackle the problem & identify an action to take

↓ An external task that the Mode can carry out from an allowed set of actions (python API)

↓ the result of carrying out the action.

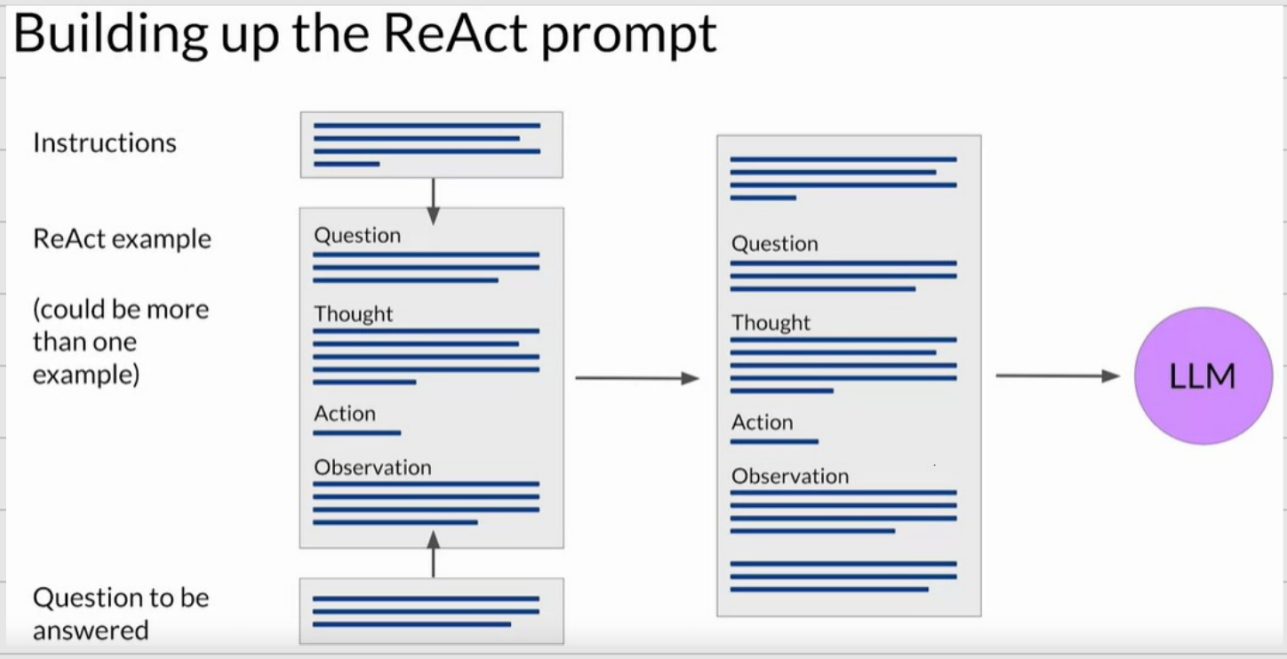
(search, lookup, finish)

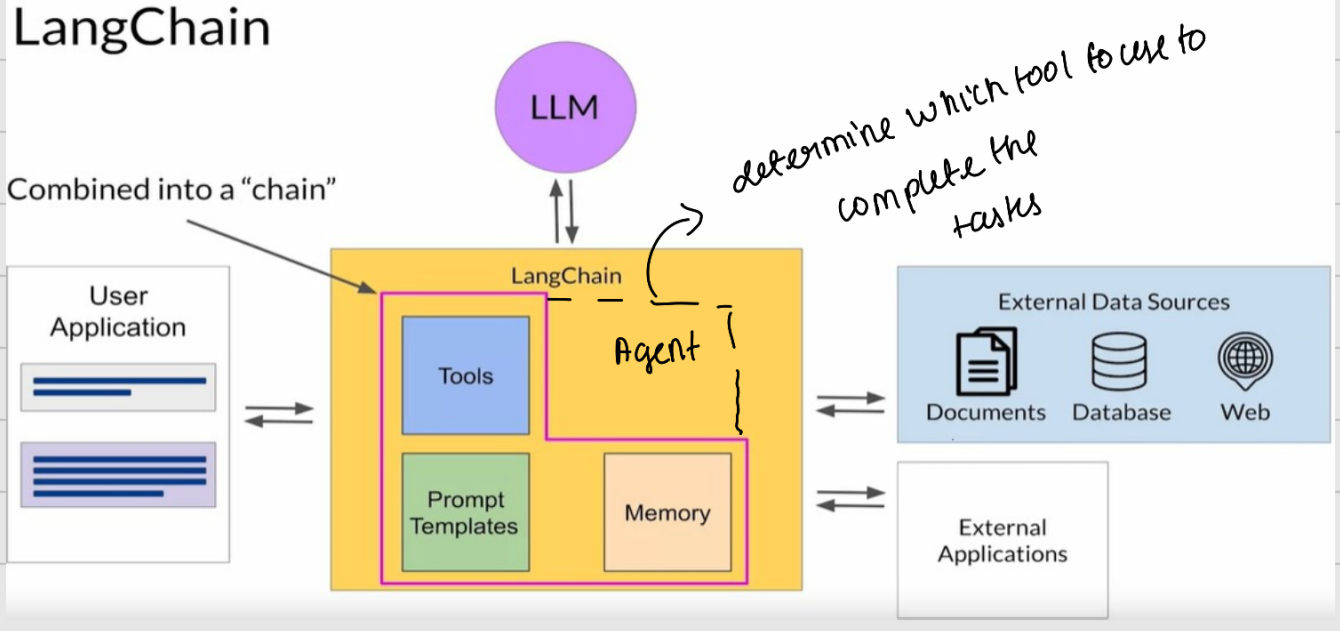
entity — keyword

answer.

LLM is reactive

LLM can only choose from a limited number of actions that are defined by set of instruction.





Start with large capable model, collects lots of user data in deployment and use it to train & fine tune smaller model.

LLM powered application architecture

Building generative applications.