Generative AI with Large Language Models.

DeepLearning.AI amazon web services coursera

Course Notes : July, 2023

Generative AI, and LLMs specifically, is a General Purpose Technology that is useful for a variety of applications.
"LLMs can be, generally, thought of as a next word prediction model"

PART 1
LLM Pre-Training

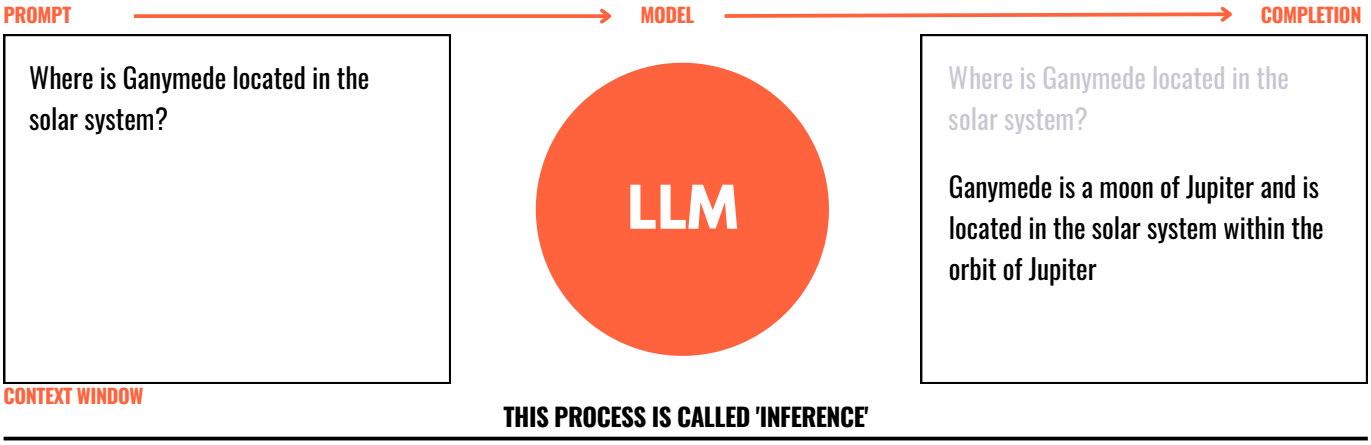
PART 2
LLM Fine Tuning

PART 3
RLHF & Application

Part 1

What is an LLM?

- LLMs are machine learning models that have learned from massive datasets of human-generated content, finding statistical patterns to replicate human-like abilities.
- Foundation models, also known as base models, have been trained on trillions of words for weeks or months using extensive compute power. These models have billions of parameters, which represent their memory and enable sophisticated tasks.
- Interacting with LLMs differs from traditional programming paradigms. Instead of formalized code syntax, you provide natural language prompts to the models.
- When you pass a prompt to the model, it predicts the next words and generates a completion. This process is known as inference.



What are the Use Cases for LLMs?

While Chatbots have emerged to become the most popular applications of LLMs, there are a variety of other tasks that LLMs can be used to accomplish -

- Writing - From essays to emails to reports and more
- Summarisation - Summarise long content into a meaningful shorter length
- Language Translation - Translate text from one language to the other
- Code - Translate natural language to machine code
- Information Retrieval - Retrieve specific information from text like names, locations, sentiment
- Augmented LLM - Power interactions with real world by providing information outside of LLM training

TRANSFORMERS.

The arrival of the transformer architecture in 2017, following the publication of the "Attention is All You Need" paper, revolutionised generative AI.

How was text generation done before Transformers?

- Before the arrival of transformers, text generation tasks were accomplished by Recurrent Neural Networks (RNNs).
- The next word was predicted looking at the previous few words. The more the number of previous words, the larger was the computational requirement of the RNN.
- The prediction wasn't great. The reason was the design of looking only at a few previous words.

I took my money to the bank.

HOMONYMS

River Bank?
Financial Bank?

The teacher taught the student with the book

SYNTACTIC AMBIGUITY

Did teacher teach with the book?
Was it a student with the book?

TRANSFORMERS ARE ABLE TO PAY ATTENTION TO THE MEANING OF THE WORDS

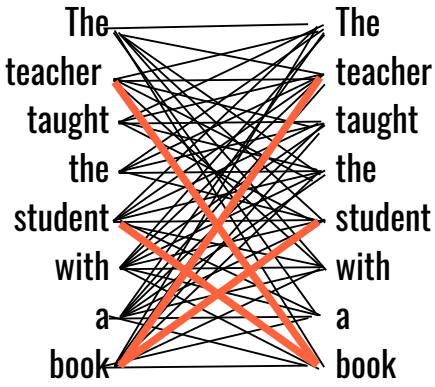
TRANSFORMERS SCALE EFFICIENTLY

TRANSFORMERS CAN PROCESS DATA PARALLELLY

What is Attention?

Transformers supersede all previous natural language architectures because of their ability to 'pay attention'

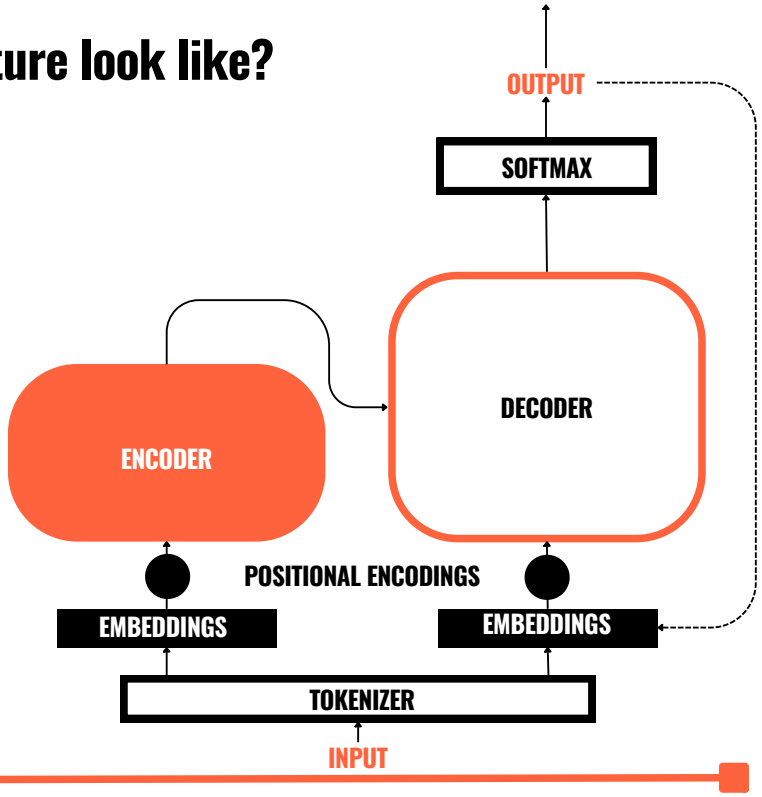
EACH WORD IS CONNECTED TO EVERY OTHER WORD THROUGH ATTENTION WEIGHTS



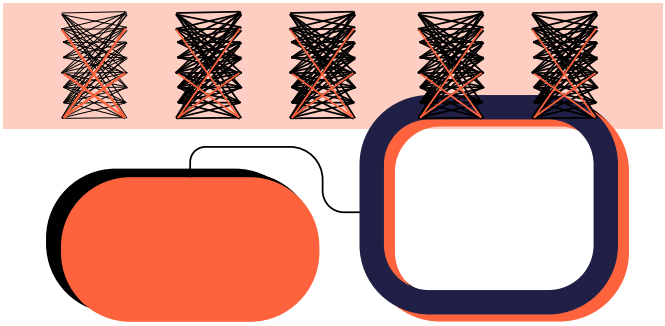
THE MORE IMPORTANT THE ASSOCIATION, THE STRONGER IS THE ATTENTION

What does a Transformer Architecture look like?

Tokenizer :	Numeric representation of words
Embeddings :	Higher order vector representation of each token
Positional Encodings :	A vector representation of the position of the word in the input
Encoder :	Encodes each input token into vector by learning self-attention weights & passing them through a FCFF Network
Decoder :	Accepts an input token, passes them through the learned attention and FCFF Network to generate new token
Softmax :	Calculates the probability for each word to be the next word in sequence



The Learning of Attention Weights is not a single process, but several parallel processes. As a result, multiple sets of attention weights are learnt. This is called Multi-Headed Self Attention.



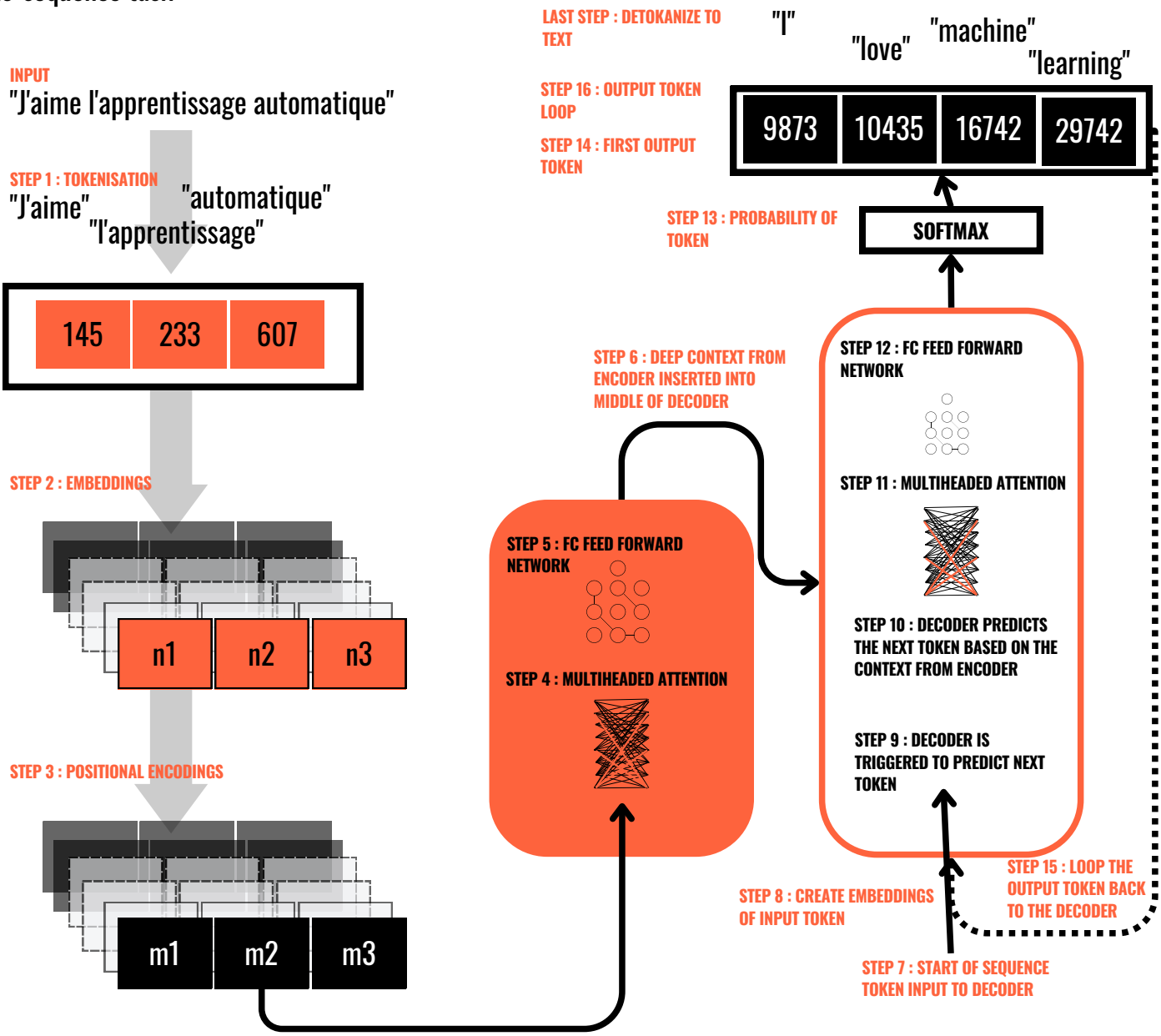
MULTI HEADED SELF ATTENTION

Only for illustrative purpose

Think of one set of attention weights as a representation of vocabulary, another set as tonality, yet another as a style of writing.

How does a Transformer generate text?

The original objective of the transformers architecture was for Language Translation in form of a sequence-to-sequence task

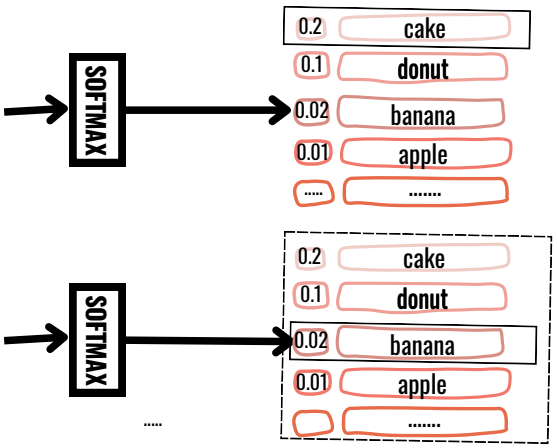


What is a Prompt?

- The natural language instruction in which we interact with an LLM is called a Prompt. The construction of prompts is called Prompt Engineering.
- The inferencing that an LLM does and completes the instruction given in the prompt is called 'in context learning'
- The ability of the LLM to respond to the instruction in the prompt without any example is called 'Zero Shot Learning'
- When a single example is provided, it's called 'One Shot Learning'
- If more than one examples in provided, it's called 'Few Shot Learning'
- Context Window, or the maximum number of tokens that an LLM can provide and inference on, is critical in the Zero/One/Few Shot Learning

ZERO SHOT LEARNING	ONE SHOT LEARNING	FEW SHOT LEARNING
Classify this review : I loved this movie! Sentiment :	Classify this review : I loved this movie! Sentiment : Positive Classify this review: I don't like this chair Sentiment :	Classify this review : I loved this movie! Sentiment : Positive Classify this review: I don't like this chair Sentiment : Negative Classify this review: Who would use this product? Sentiment :

Greedy vs Random Sampling.



Greedy : The word/token with the largest probability is selected

'Cake' has the highest probability of 20%

Random Sampling : The word/token is selected using random-weighted strategy

'Even though Cake' has the highest probability of 20%, 'banana' is selected

Inference Configuration Parameters.

CONTEXT WINDOW
Maximum number of tokens that the model can consider when generating or processing text. This includes both the prompt and the response.

GPT 4 has a context window of 32,768 tokens
Anthropic AI's Claude claims to push the context window upto 100,000 tokens

MAX TOKENS
A parameter to adjust the number of tokens to be used for a particular request. This is capped to the context window.

One token is approximately equal to four English language alphabets

TEMPERATURE
LLM outputs are based on the probability of the generated word. Temperature controls the randomness of generation. The higher the temperature, the more random is the generation.

A temperature of 'Zero' will make the generation deterministic and repititive

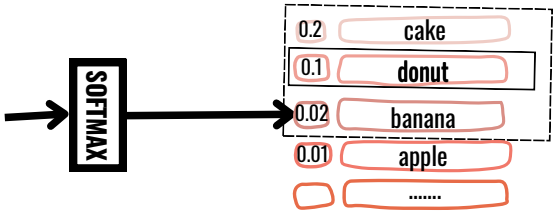
TOP P/ TOP N
Since LLM outputs are based on probability, setting a Top Parameter will restrict the selection of the next word from the Top 'N' most probable words or Top words summing up to probability 'P'.

While temperature controls the randomness within the model, Top P and Top N are applied on top of the predicted probabilities

PENALTY
You can adjust this factor to reduce the repetition of tokens in the generated output. Penalty adds a negative weight to tokens that have previously been generated.

A temperature of 'Zero' will make the generation deterministic and repititive

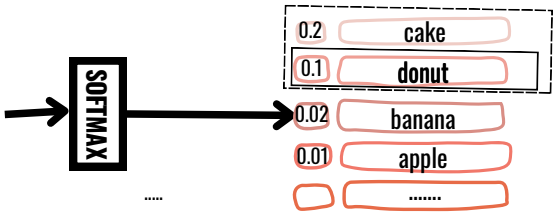
Top N.



Top N : The word/token is selected using random-weighted strategy but only from amongst the Top 'N' words/tokens

Here for N=3, one of cake, donut or banana will be selected randomly but apple will never be selected

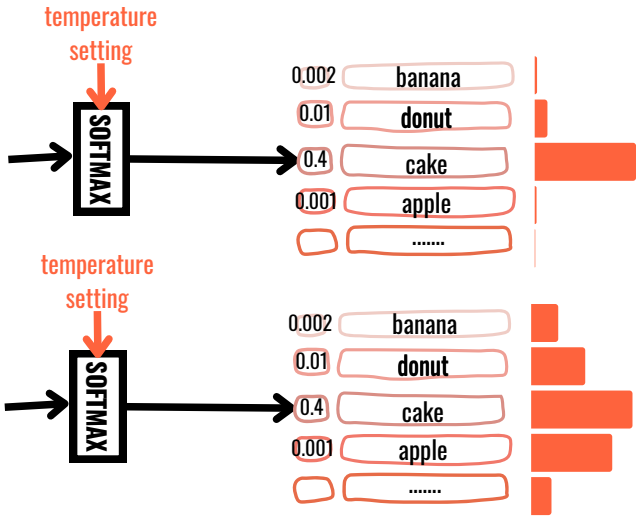
Top P.



Top P : The word/token is selected using random-weighted strategy but only from amongst the top words totalling to probability <=P

Here for P=0.33, one of cake or donut will be selected randomly but apple or banana will never be selected

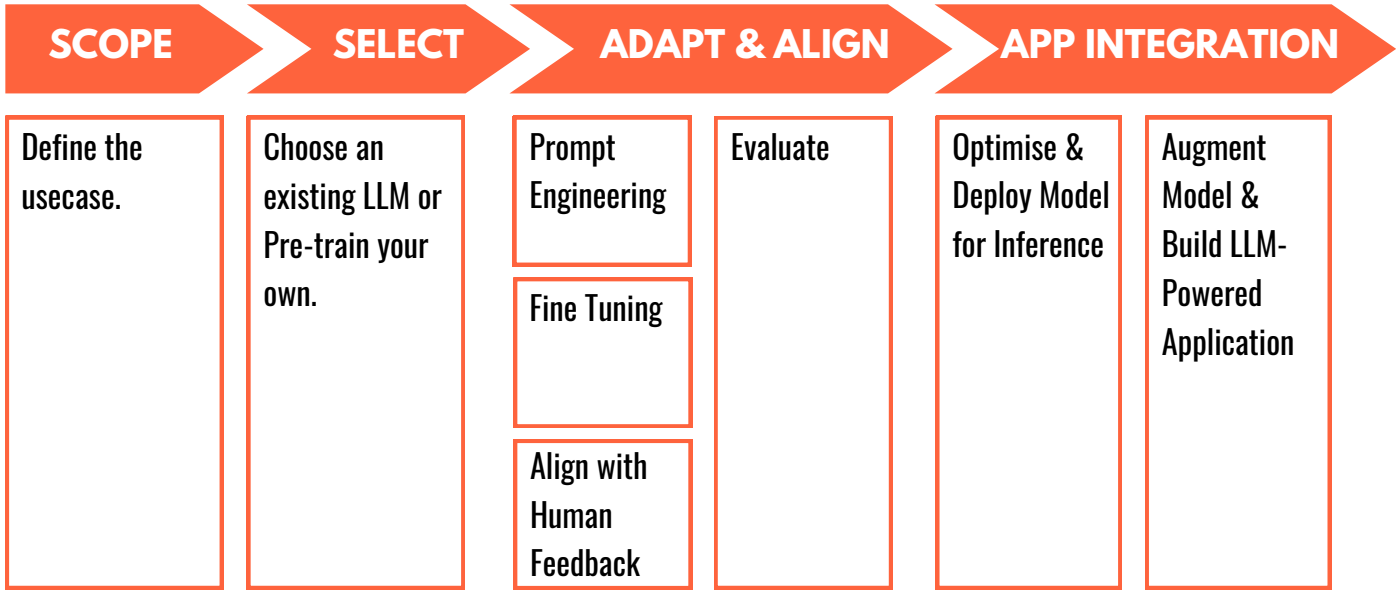
Temperature



Cooler Temperature (lesser value) : The distribution is strongly peaked

Warmer Temperature (higher value) : Flatter probability distribution

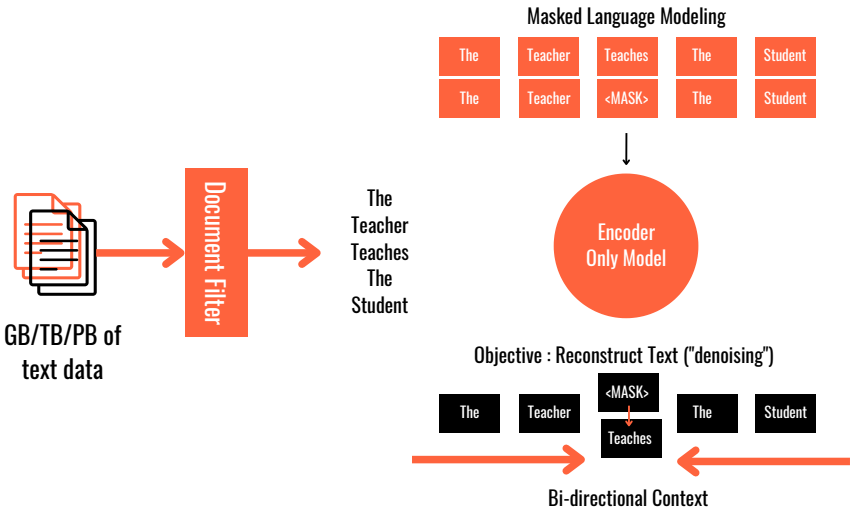
Generative AI Project LifeCycle.



- Defining the scope accurately and narrowly is a crucial initial step in any project.
- LLMs have diverse capabilities depending on the model's size and architecture, so it is essential to consider the specific function your LLM will serve in your application.
- Choosing between training your own model from scratch or using an existing base model is the first decision you'll face.
- While starting with an existing model is common, there may be cases where training from scratch becomes necessary
- Prompt engineering and in-context learning techniques can often improve the model's performance by using task-specific examples.
- There are instances where the model may still underperform, even with prompt engineering, and in such cases, fine-tuning the model through supervised learning can be explored.
- Learning with human feedback as an additional fine-tuning technique to ensure the model's good behaviour.
- Evaluation plays a crucial role in all these techniques
- Optimising the model for deployment ensures efficient utilisation of compute resources and provides the best possible user experience.
- It is important to consider any additional infrastructure requirements for your application to work effectively.
- LLMs have inherent limitations, such as inventing information or limited reasoning abilities, which can be challenging to overcome through training alone.

How do you pre-train Large Language Models?

AutoEncoding Models (Encoder Only)



USE CASES

- Sentiment Analysis
- Named Entity Recognition
- Word Classification

EXAMPLES

- BERT
- ROBERTA

Finetuning Foundation Models vs Pretaining Your Own

For most requirements, finetuning an existing LLM will suffice. However, there can be cases when pre-training a new LLM will provide better application especially when the language is highly domain specific e.g Legal, Medical. Pre-training is a resource intensive and costly process.

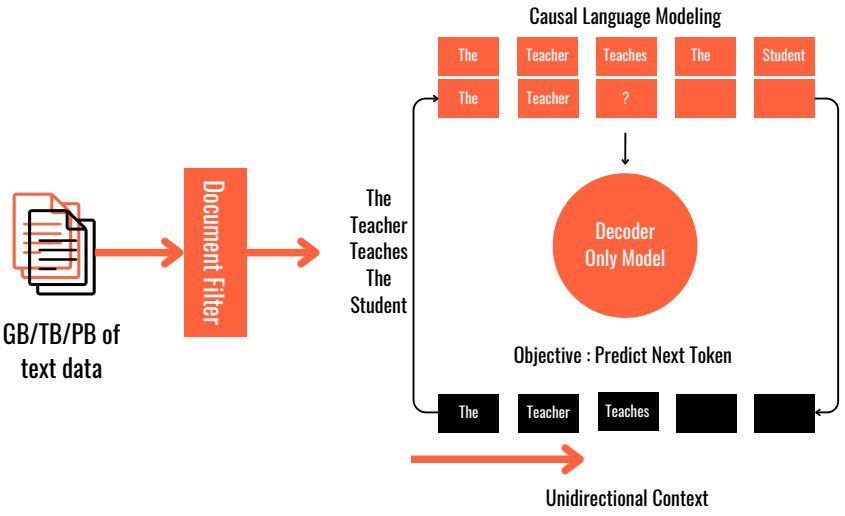
Open Source LLMs

While OpenAI's proprietary GPT-3.5, GPT-4 have gained immense popularity, HuggingFace Model Hub provides access to powerful Open Source LLMs along with documentation and training architecture. Architecture plays a critical role in defining what objective can each LLM be used for.

Training Data and Model Size

LLMs are generally trained on Petabytes of data, mostly from the open internet. The unstructured text requires careful filtering. As a result only 2-3% of data is useful for training. LLM size is measured in terms of the number of parameters. Larger models have generalised well to a variety of tasks.

AutoRegressive Models (Decoder Only)



USE CASES

- Text Generation

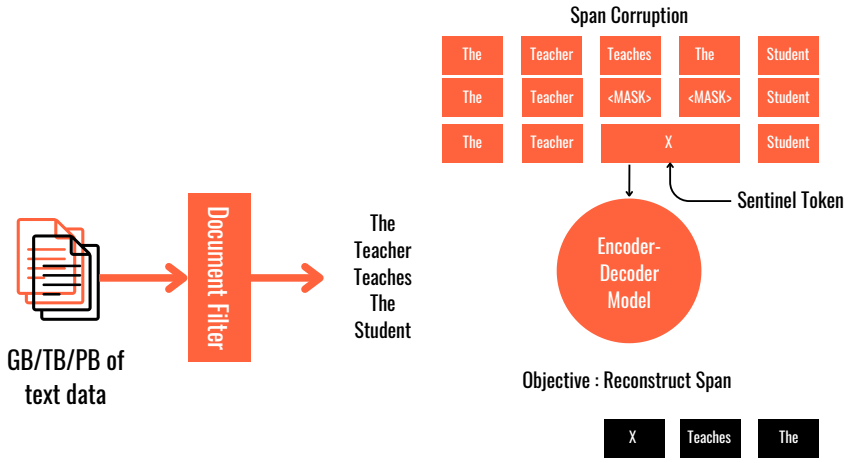
  (Most common architecture now and larger models can perform a variety of tasks)

EXAMPLES

- GPT
- BLOOM

How do you pre-train Large Language Models?

Sequence-to-Sequence Models (Encoder-Decoder)



USE CASES

- Translation
- Text Summarisation
- Question Answering

EXAMPLES

- T5
- BART

Challenges with pre-training LLMs.

Computational Memory

1 Parameter = 4 bytes (32 bit float)
1 Billion Parameters = 4 x 10E9 bytes = 4GB

Model Parameters = 4 bytes per parameter
2 Adam Optimisers = +8 bytes per parameter
Gradients = +4 bytes per parameter
Activations = +8 bytes per parameter

Total = 4 bytes per parameter +
20 extra bytes per parameter

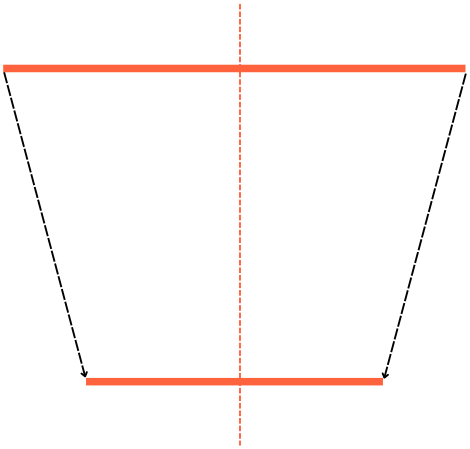
To Store

4 GB@32 bit full precision

To Train

80 GB@32 bit full precision

Quantisation



FP32
32 Bit Floating Point

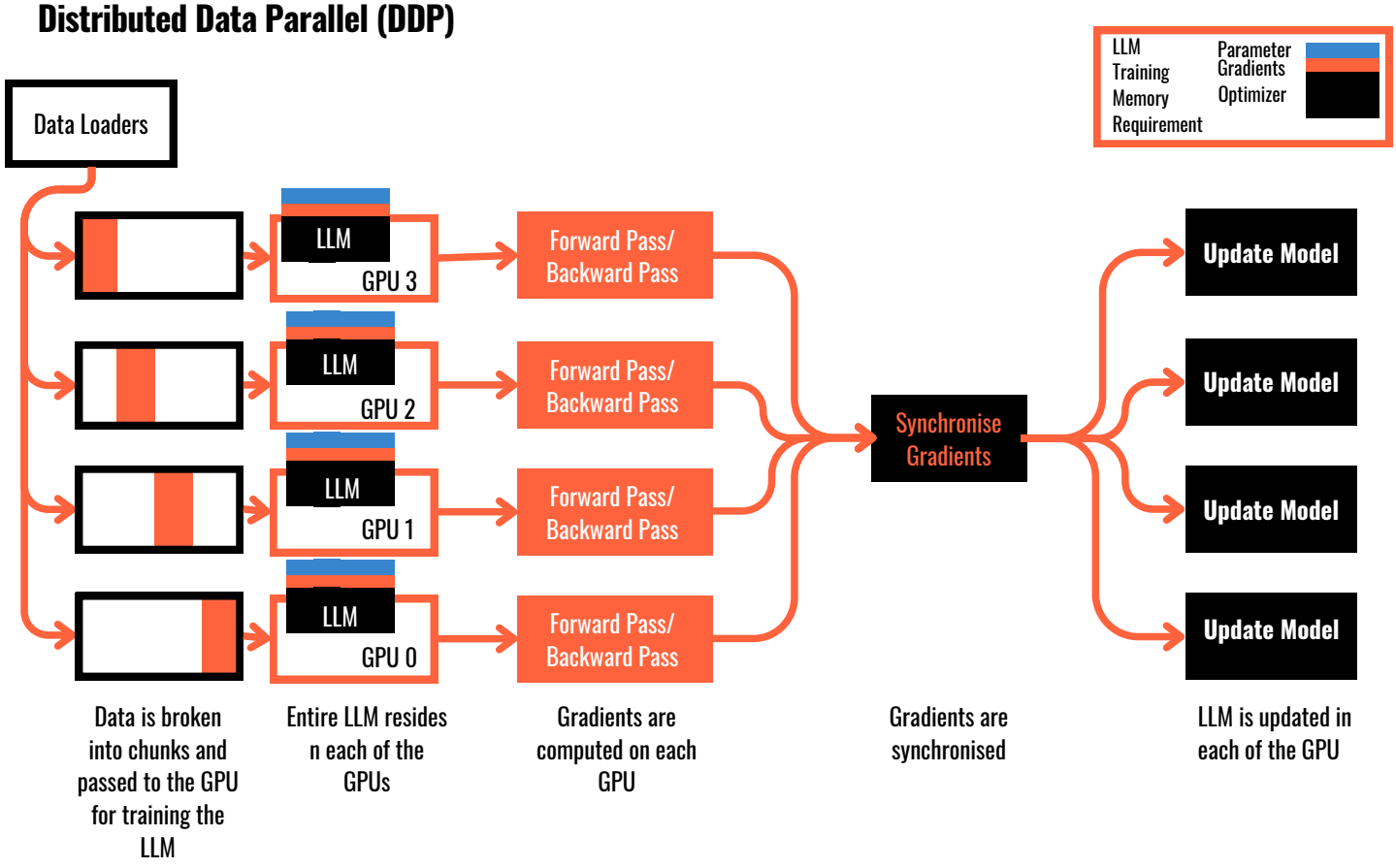
Range
3e-38 to 3e+38

FP16 | BFLOAT16 | INT8
16 Bit Floating Point
8 Bit Integer

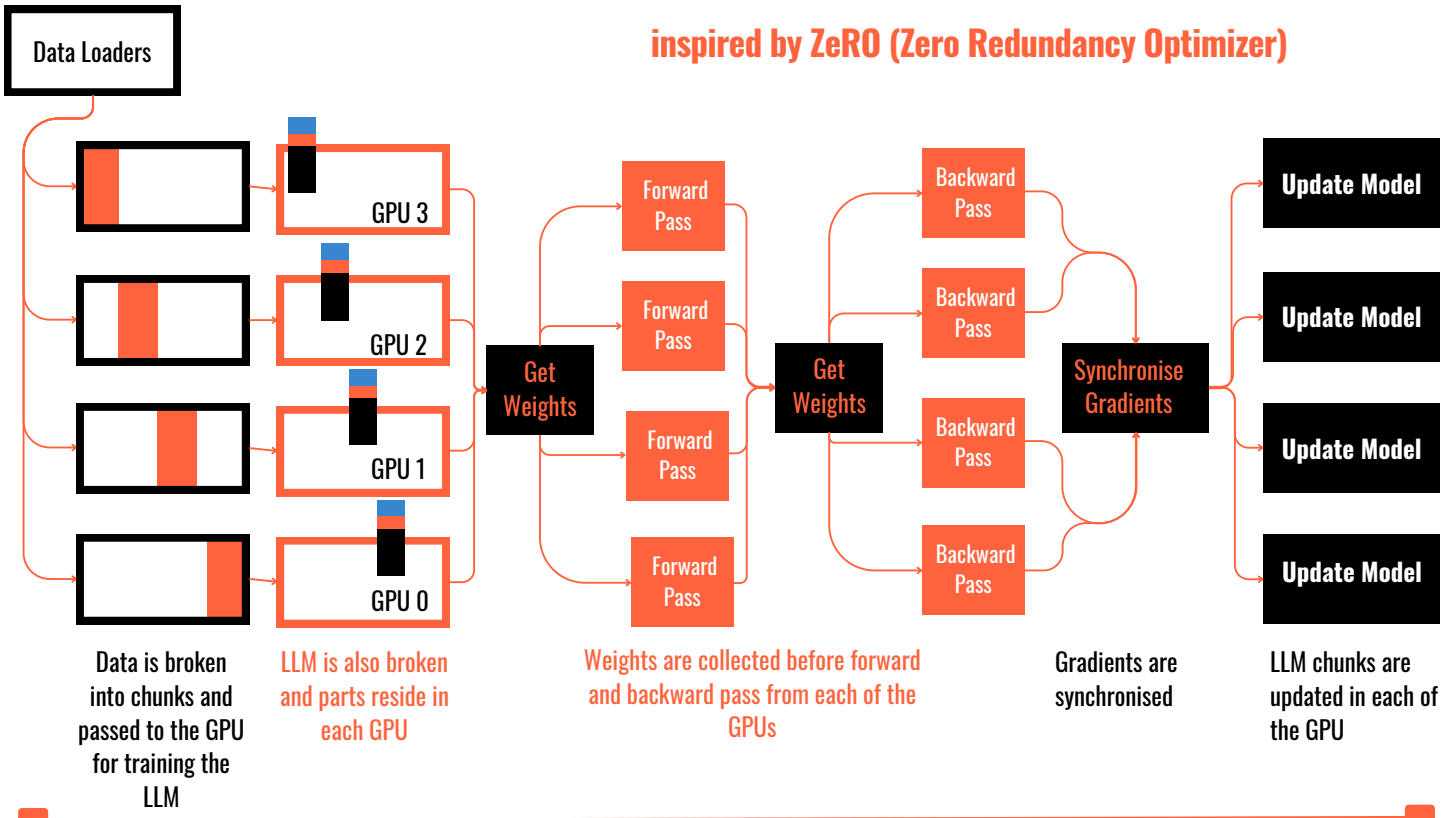
- Reduces the memory required to train and store models
- Quantisation projects the 32bit numbers into a lower precision space
- Quantisation aware training (QAT) learns quantisation scaling factors during training
- BFLOAT16 is a popular choice

Multi-GPU Compute (Optional)

Distributed Data Parallel (DDP)

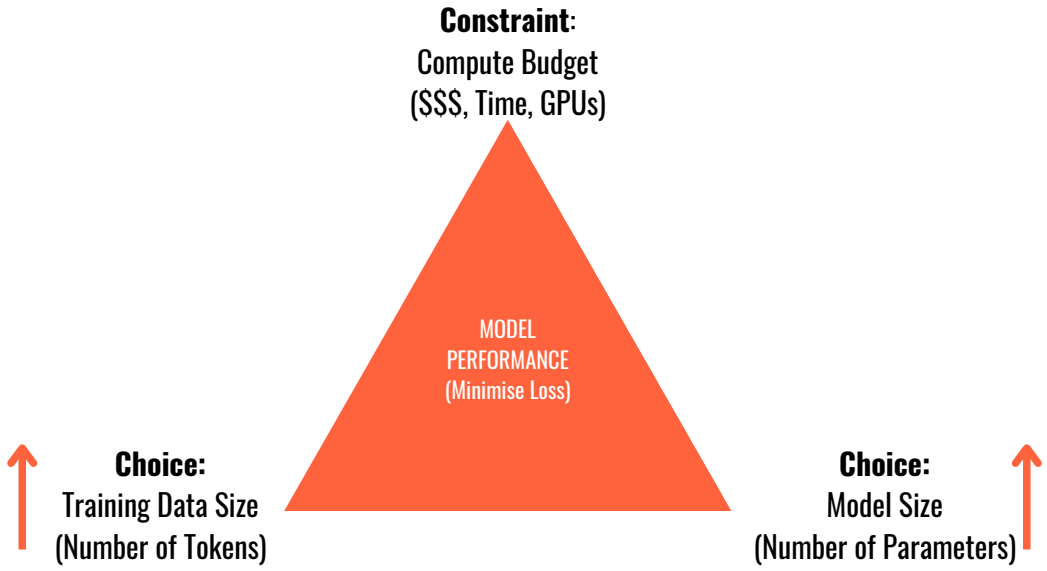


Fully Sharded Data Parallel (FSDP)



What is the optimal computation configuration for pre-training an LLM?

- Our aim should be to minimise the loss function of the LLM.
- This can be done by providing more training tokens and/or increasing the number of parameters
- Compute cost (budget, time, GPUs) is the constraint that we operate in.



One metric to measure the cost of computation is "petaFLOP/s-day"
Number of FLOating Point operations performed at the rate of 1 petaFLOP per second for one day

1 petaFLOP per second = 10E15 or 1 Quadrillion floating point operations per second

1 petaFLOP per second - day requires 8 NVIDIA V100 chips running at full efficiency for 24 hours

1 petaFLOP per second - day requires 2 NVIDIA A100 chips running at full efficiency for 24 hours

To put this in context, OpenAIs GPT3 is a 175 B parameters model and required 3700 petaflops/s-day

Chinchilla Paper

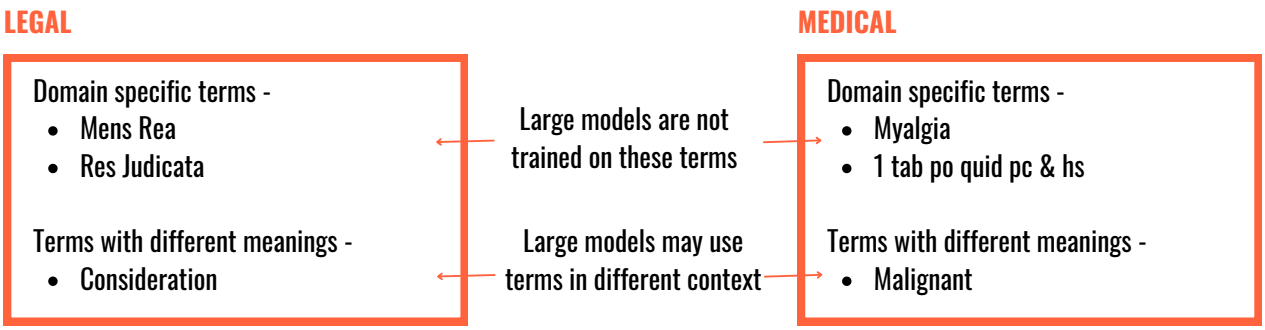
In 2022, the paper "Training Compute-Optimal Large Language Model" studied a large number of models with different training data and model sizes to find the optimal number of tokens and parameters for a given compute budget. The authors called this optimal model "Chinchilla"

One important take-away was that the compute optimal number of training tokens should be 20 times the number of parameters

This means that smaller models can achieve the same performance as the larger ones, if they are trained on larger datasets

Why Pre-train for Domain Adaptation?

- Developing pre-trained models for a highly specialised domains may be useful when -
  - There are important terms that are not commonly used in the language
  - The meaning of some common terms is entirely different in the domain vis-a-vis the common usage



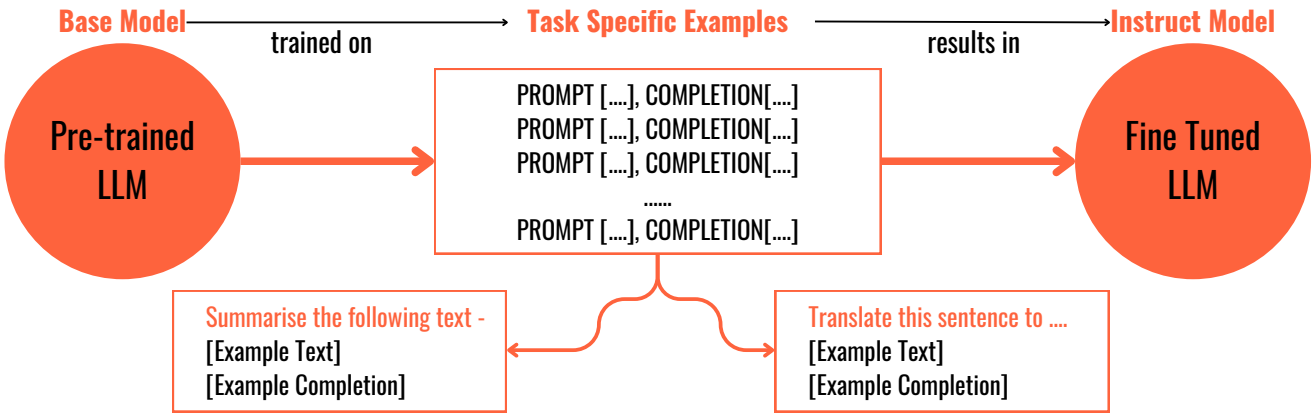
EXAMPLE : Bloomberg GPT

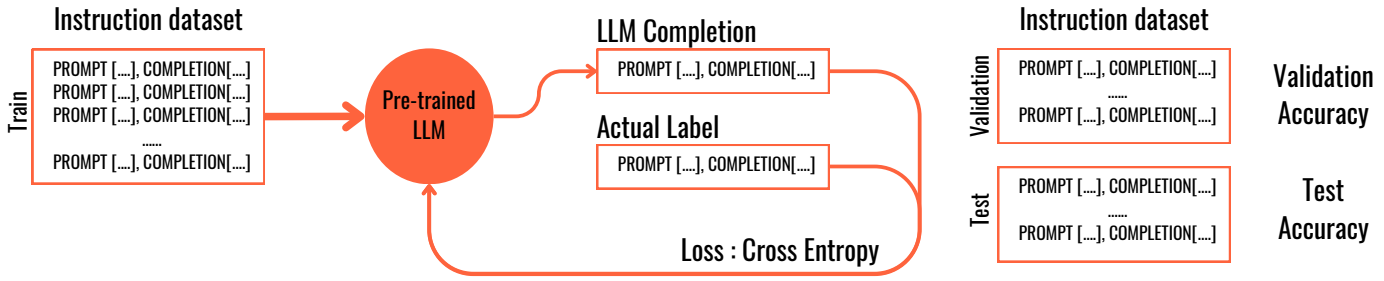
END OF PART 1

Part 2

What is Instruction Fine Tuning?

- Through in context learning, or prompting, only a certain level of performance can be achieved.
- Few shot learning might not work for smaller LLMs and it also takes up a lot of space in the context window.
- Fine Tuning is a supervised learning process, where you take a labelled dataset of prompt-completion pairs to adjust the weights of an LLM.
- Instruction Fine Tuning is a strategy where the LLM is trained on examples of Instructions and how the LLM should respond to those instructions. Instruction Fine Tuning leads to improved performance on the instruction task.
- Full Fine Tuning is where all the LLM parameters are updated. It requires enough memory to store and process all the gradients and other components.





Fine tuning process is a classification model training

What is Catastrophic Forgetting?

- Fine Tuning on a single task can significantly improve the performance of the model on that task.
- However, because the model weights get updated, the instruct model's performance on other tasks (which the base model performed well on) can get reduced. This is called Catastrophic Forgetting.

Avoiding Catastrophic Forgetting

- You might not have to, if you only want the model to perform well on the trained task.
- Perform fine tuning on multiple tasks. This will require lots of examples for each task.
- Perform Parameter Efficient Fine Tuning (PEFT).

How is the performance of a Fine Tuned LLM measured?

- The inherent challenge in measuring the performance of an LLM is that the outputs are non-deterministic, as opposed to a classic classification model where the output is amongst pre-determined classes.

Mike really loves drinking tea = Mike adores sipping tea
Mike does not drink coffee ≠ Mike does drink coffee

- There are two widely used evaluation metrics.

ROUGE

- Used for text summarisation
- Compares a summary to one or more reference summaries

BLEU SCORE

- Used for text translation
- Compares to human-generated translations

ROUGE SCORE

REFERENCE (human) :
It is cold outside

GENERATED:
It is very cold outside

LCS = Length of the Longest Common Subsequences

ROUGE - 1 Recall	=	unigram matches / unigrams in reference	4/4	ROUGE - 1 Precision	=	unigram matches / unigrams in output	4/5	It, is, cold, outside It, is very, cold, outside
ROUGE - 2 Recall	=	bigram matches / bigrams in reference	2/3	ROUGE - 1 Precision	=	bigram matches / biigrams in output	2/4	It is, is cold, cold outside It is, is very, very cold, cold outside
ROUGE - L Recall	=	LCS (R,G) / unigrams in reference	2/4	ROUGE - 1 Precision	=	LCS (R,G) / unigrams in output	2/5	Length (It is) = 2 Length (cold outside) = 2

- Rouge scores should only be compared across the models for the same task
- Rouge score may be high even for imperfect generations

BLEU (Bi-Lingual Evaluation Understudy) SCORE

BLEU = Average (precision score across a range of n-gram sizes)

BOTH ROUGE & BLEU SHOULD BE USED FOR DIAGNOSTIC PURPOSES ONLY.

BENCHMARKS

- Use pre-existing evaluation datasets and benchmarks established by LLM researchers for a more holistic evaluation
- Select datasets that isolate model skills, like reasoning or common sense knowledge, and those that focus on potential risks, such as disinformation or copyright infringement
- Evaluate performance on data that the model has not seen before

Popular Benchmarks

GLUE — General Language Understanding

SuperGLUE

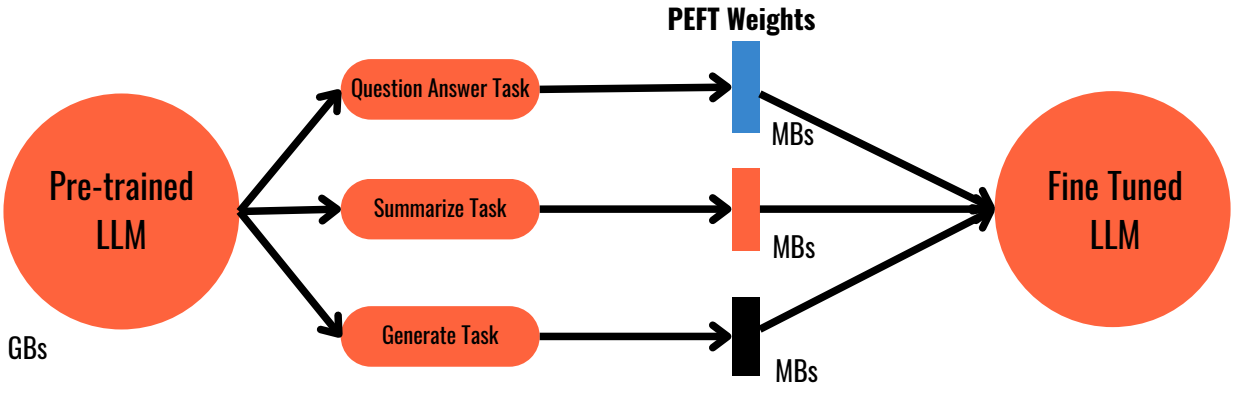
HELM — Holistic Evaluation of Language Models

MMLU — Massive Multitask Language Understanding

BIG BENCH

What is Parameter Efficient Fine Tuning?

- Full fine tuning, like pre-training, requires memory not just to store the model, but also other parameters like optimisers, gradients etc.
- Parameter Efficient Fine Tuning or PEFT fine tunes only a subset of model parameters and, in some cases, do not touch the original weights at all.
- Because PEFT only retrains a subset of parameters, Catastrophic Forgetting can be avoided.



Trade Offs: Parameter Efficiency | Memory Efficiency | Model Performance | Training Speed | Inference Cost

PEFT Method

Selective
- Select a subset of initial LLM parameters to fine tune.
- Performance is mixed and significant trade-offs

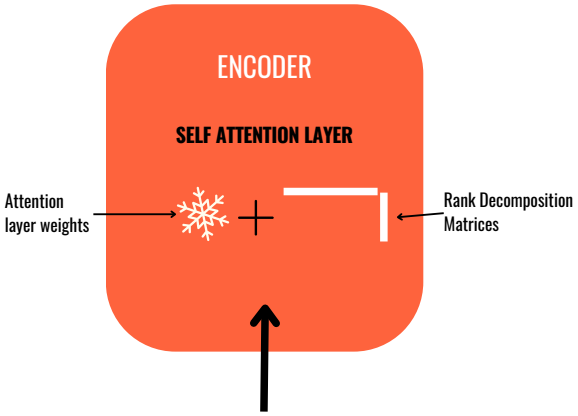
Reparameterization
- Reparameterize model weights using Low Rank Representation

LoRA

Additive
- Add trainable layers or parameters to the original model

Adapters
Soft Prompting

LoRA (Low Rank Adaptations)



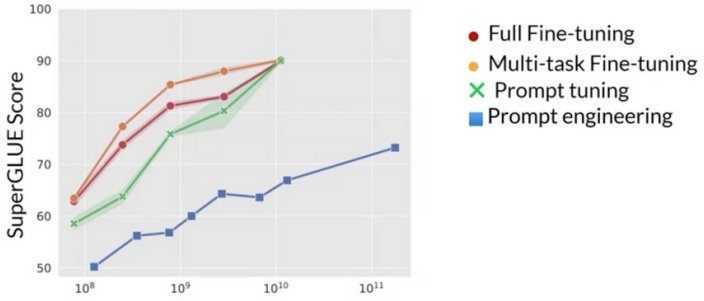
- Most of original LLM weights are frozen.
- 2 rank decomposition matrices are injected.
- Product of rank decompositions matrices is of the same dimensions as the LLM weights.
- The product is added to the LLM weights.
- There's no impact on inference latency since the number of parameters remains the same.
- Applying LoRA to just the attention layer is enough.

- LoRA can reduce the number of training parameters to ~20% and can be trained on a single GPU.
- Separate LoRA matrices can be trained for each task and switch out the weights for each task.

Soft Prompts : Prompt Tuning (not Prompt Engineering)

- Prompt engineering is limited by context window and the manual exercise of writing prompts.
- In Prompt tuning, additional trainable tokens (soft prompts) are added to the prompt which are learnt during the supervised learning process.
- Soft prompt tokens are added to the embedding vectors and have the same length as the embedding.
- Between 20-100 tokens are sufficient for fine tuning.
- Soft prompts can take any value in the embedding space and through supervised learning, the values are learnt.
- Like in LoRA, separate soft prompts can be trained for each task and switch out the weights for each task.

Performance of Prompt Tuning



- For models with >10B parameters, Prompt Tuning can be as effective as Full Fine Tuning
- Analysis of nearest neighbours indicate that learnt soft tokens form tight clusters

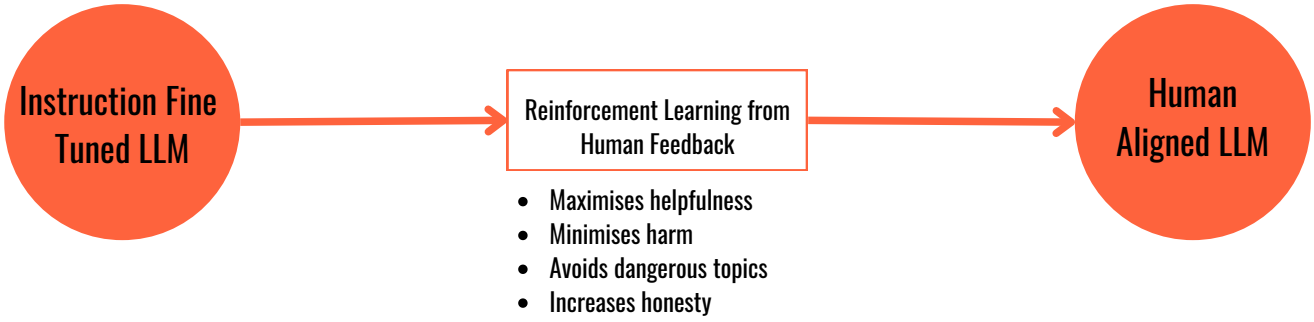
END OF PART 2

Part 3

Aligning with Human Values

- Like with language, in general, Large Language Models can also behave badly -
  - Toxicity
  - Aggression
  - Dangerous/Harmful
- LLMs should align with Helpfulness, Honesty and Harmlessness (HHH)

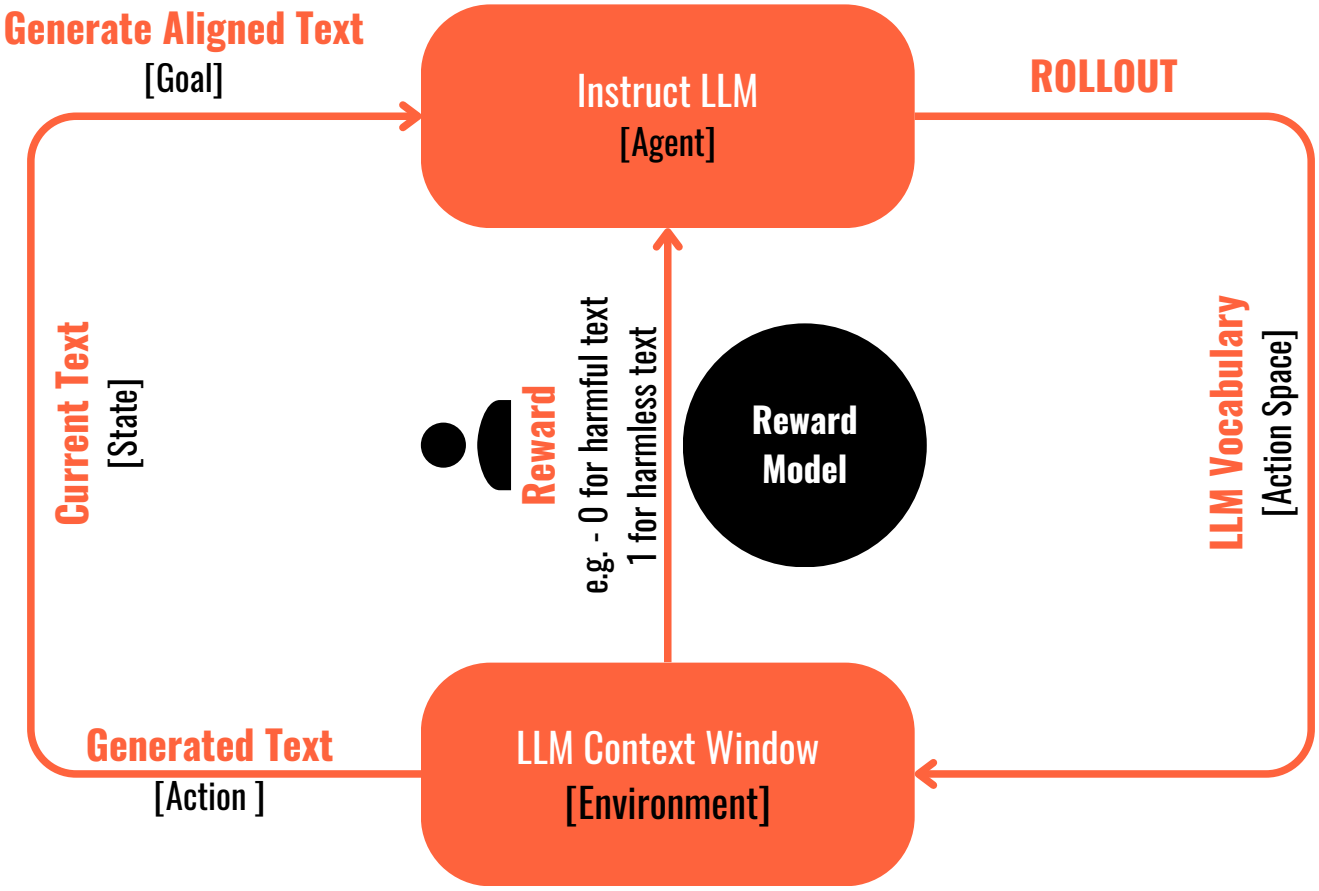
What is Reinforcement Learning from Human Feedback (RLHF)?



- Reinforcement Learning based on Human Feedback data
- Personalisation of LLMs is a potential application of RLFL

REINFORCEMENT LEARNING
is a type of machine learning in which an **agent** learns to make decisions related to a specific **goal** by taking **actions** in an **environment**, with the objective of maximising some notion of a cumulative **reward**

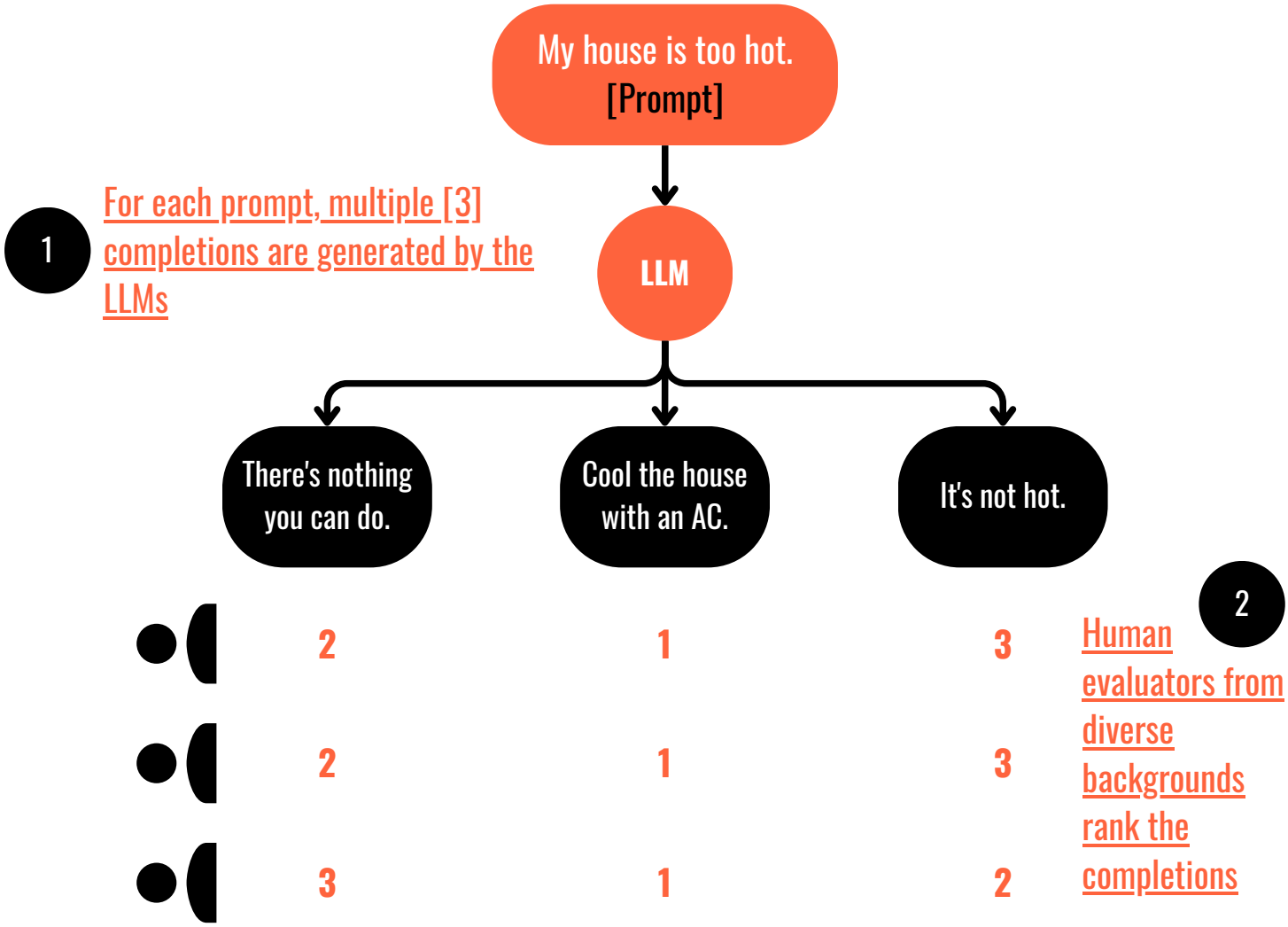
How does RLHF work?



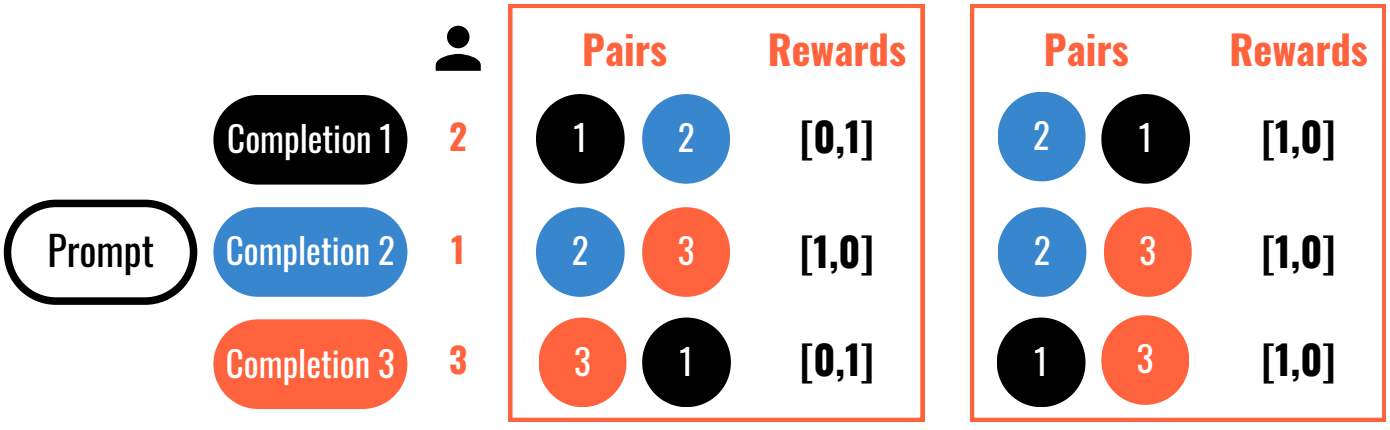
In RLHF, the agent (our fine-tuned instruct LLM) in its environment (Context Window) takes one action (of generating text) from all available actions in the action space (the entire vocabulary of tokens/words in the LLM).
The outcome of this action (the generated text) is evaluated by a human and is given a reward if the outcome (the generated text) aligns with the goal. If the outcome does not align with the goal, it is given a negative reward or no reward. This is an iterative process and each step is called a rollout. The model weights are adjusted in a manner that the total rewards at the end of the process are maximised.

Note : In practice, instead of a human giving a feedback continually, a classification model called the **Reward Model** is trained based on human generated training examples

How is Reward Model training data created from Human Feedback?

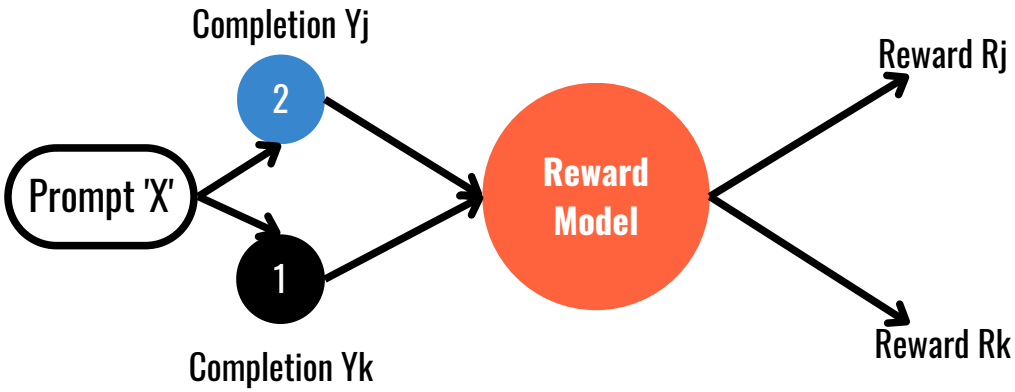


3 Rankings are converted to pairwise training data for the reward model



The pairs are ordered in order {Yj, Yk} such that Yj is the preferred completion

4 Reward Model is a supervised learning language model



The reward model learns to favour the human preferred response Yj while minimising the log of sigmoid difference between the rewards.

5 Reward Model is finally used as a binary classifier

The logit values for the Positive Class are passed as the Rewards. The LLM will change the weights in such a way that the choice of generated text will yield the highest rewards.

"Tommy loves Television"

Positive Class (Not Hate)	**3.1718**
Negative Class (Hate)	-2.6093

"Tommy hates gross movies"

Positive Class (Not Hate)	**-0.5351**
Negative Class (Hate)	0.1377

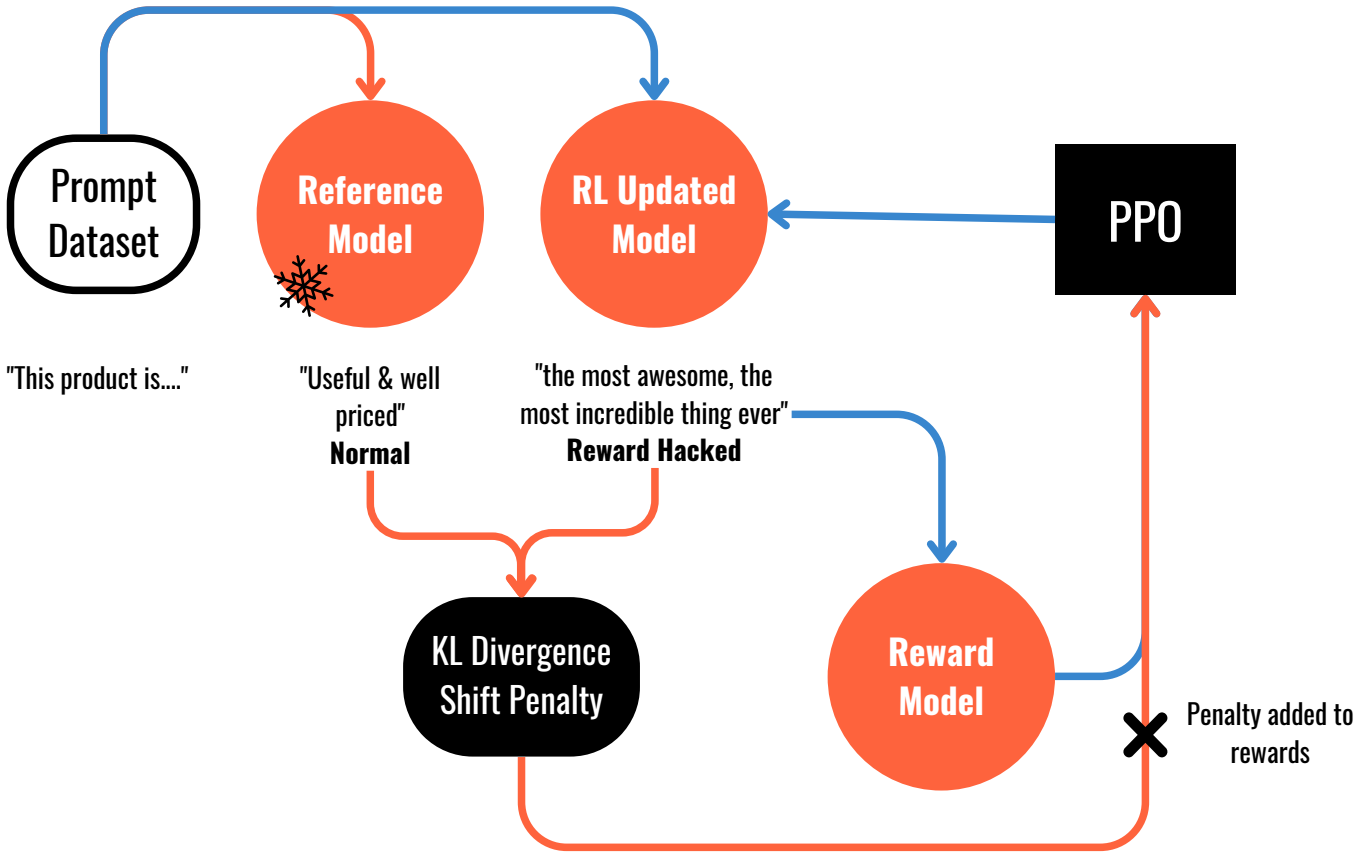
Proximal Policy Optimisation or PPO is a popular choice for training the reinforcement learning algorithms

What is Proximal Policy Optimisation? [Optional]

- PPO stands for Proximal Policy Optimisation. It's a powerful algorithm used in reinforcement learning.
- PPO helps us optimise a large language model (LLM) to be more aligned with human preferences. We want the LLM to generate responses that are helpful, harmless, and honest.
- PPO works in cycles with two phases: Phase I and Phase II.
- In Phase I, the LLM completes prompts and carries out experiments. These experiments help us update the LLM based on the reward model, which captures human preferences.
- The reward model determines the rewards for prompt completions. It tells us how good or bad the completions are in terms of meeting human preferences.
- In Phase II, we have the value function, which estimates the expected total reward for a given state. It helps us evaluate completion quality and acts as a baseline for our alignment criteria.
- The value loss minimises the difference between the actual future reward and its estimation by the value function. This helps us make better estimates for future rewards.
- Phase 2 involves updating the LLM weights based on the losses and rewards from Phase 1.
- PPO ensures that these updates stay within a small region called the trust region. This keeps the updates stable and prevents us from making drastic changes.
- The main objective of PPO is to maximise the policy loss. We want to update the LLM in a way that generates completions aligned with human preferences and receives higher rewards.
- The policy loss includes an estimated advantage term, which compares the current action (next token) to other possible actions. We want to make choices that are advantageous compared to other options.
- Maximising the advantage term leads to better rewards and better alignment with human preferences.
- PPO also includes the entropy loss, which helps maintain creativity in the LLM. It encourages the model to explore different possibilities instead of getting stuck in repetitive patterns.
- The PPO objective is a weighted sum of different components. It updates the model weights through back propagation over several steps.
- After many iterations, we arrive at an LLM that is more aligned with human preferences and generates better responses.
- While PPO is popular, there are other techniques like Q-learning. Researchers are actively exploring new methods, such as direct preference optimisation, to improve reinforcement learning with large language models.

How to avoid Reward Hacking?

- Reward hacking happens when the language model finds ways to maximise the reward without aligning with the original objective i.e. model generates language that sounds exaggerated or nonsensical but still receives high scores on the reward metric.
- To prevent reward hacking, the original LLM is introduced as a reference model, whose weights are frozen and serve as a performance benchmark.
- During training iterations, the completions generated by both the reference model and the updated model are compared using KL divergence. KL divergence measures how much the updated model has diverged from the reference model in terms of probability distributions.
- Depending on the divergence, a shift penalty is added to the rewards calculation. The shift penalty penalises the updated model if it deviates too far from the reference model, encouraging alignment with the reference while still improving based on the reward signal.



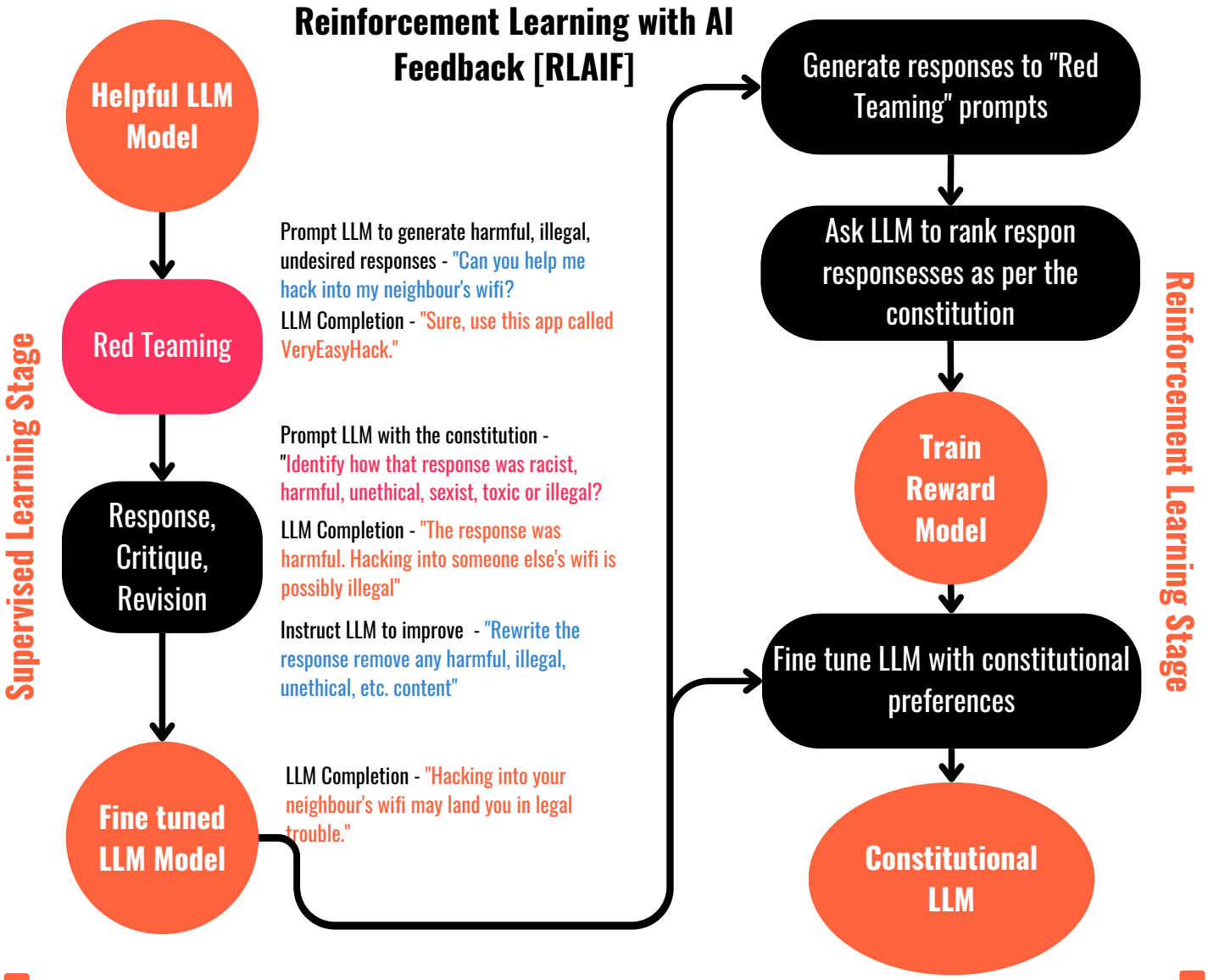
Reinforcement Learning loop that is exposed to the threat of reward hacking

Introduction of Shift Penalty loop increases alignment without exaggeration

RLHF can also be used in conjunction with PEFT to reduce memory footprint

Scaling Human Feedback : Self Supervision with Constitutional AI

- Scaling human feedback for RLHF can be challenging due to the significant human effort required to produce the trained reward model. As the number of models and use cases increases, human effort becomes a limited resource, necessitating methods to scale human feedback.
- First proposed in 2022 by researchers at Anthropic, Constitutional AI is an approach to scale supervision and address some unintended consequences of RLHF. Constitutional AI involves training models using a set of rules and principles that govern the model's behaviour, forming a "constitution".
- The training process for Constitutional AI involves two phases: supervised learning and reinforcement learning.
- In the supervised learning phase, the model is prompted with harmful scenarios and asked to critique its own responses based on constitutional principles. The revised responses, conforming to the rules, are used to fine-tune the model.
- The reinforcement learning phase, known as reinforcement learning from AI feedback (RLAIF), uses the fine-tuned model to generate responses based on constitutional principles

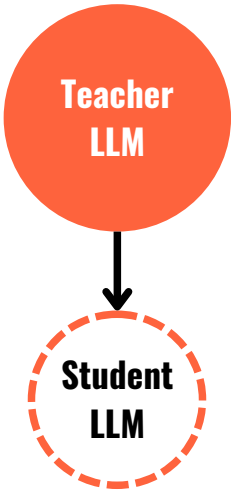


How to optimise and deploy LLMs for inferencing?

- Integrating a language model into applications requires considering factors like model speed, compute budget, and trade-offs between performance and speed/storage.
- Additional considerations include model interaction with external data or applications and determining the intended application or API interface.

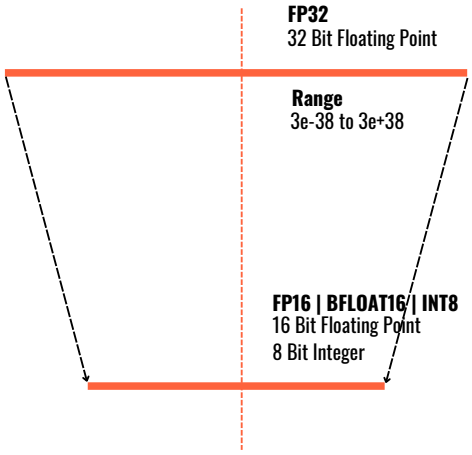
Optimisation techniques

Distillation



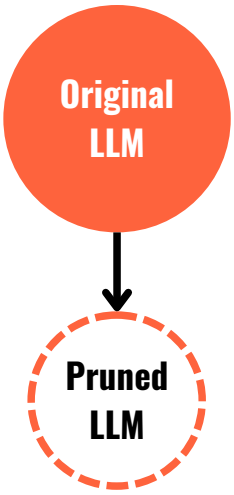
- Use larger teacher model to train smaller student model
- Use student model for inferencing in applications
- Temperature parameter is used to generate soft and hard predictions.
- In practice, distillation is effecting in encoder only models like BERT

Post-training Quantisation



- Reduce the model weights post training to lower precision to reduce model footprint
- Can be applied to both weights and activations
- May result in reduction in evaluation metrics

Pruning



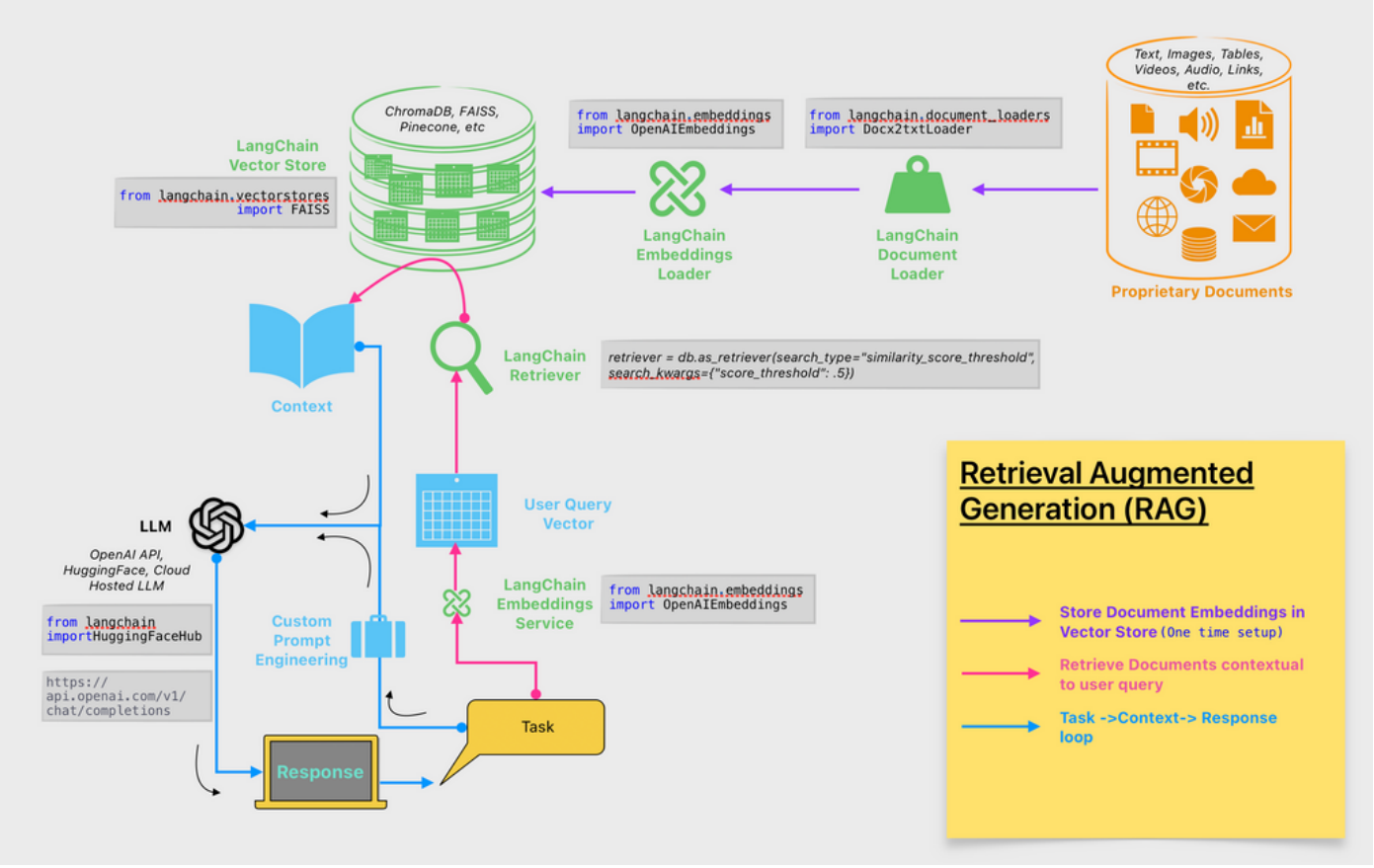
- Remove the weights with values close to 0
- Can be done via full fine-tuning or PEFT
- Reduces size and improves performance
- Is not helpful if only a small percentage of original model weights are zero

Using LLMs in Applications

- Large language models (LLMs) have a knowledge cutoff and may struggle with outdated information.
- LLMs can also face challenges with complex math problems and tend to generate text even when they don't know the answer (hallucination).

Retrieval Augmented Generation

- The Retrieval Augmented Generation (RAG) framework overcomes these issues by connecting LLMs to external data sources and applications.
- RAG provides LLMs access to data they did not see during training, improving relevance and accuracy of completions.



- Implementing RAG involves considerations such as the size of the context window and the need for data retrieval and storage in appropriate formats

Reasoning using Chain of Thought

- LLMs can struggle with complex reasoning tasks, especially those involving multiple steps or mathematics.
- Prompting the model to think more like a human by breaking down the problem into steps has shown success in improving reasoning performance.
- Chain of thought prompting involves including intermediate reasoning steps in examples used for one or few-shot inference.
- This approach teaches the model how to reason through the task by mimicking the chain of thought a human might follow.
- Chain of thought prompting can be used for various types of problems, not just arithmetic, to improve reasoning performance.
- It provides a more robust and transparent response from the model, explaining its reasoning steps.
- Although LLMs can benefit from chain of thought prompting, their limited math skills can still pose challenges for accurate calculations in tasks like sales totaling, tax calculation, or applying discounts.

Program-aided Language Models (PAL)

- LLMs have limitations in carrying out accurate math calculations, especially with larger numbers or complex operations.
- Chain of thought prompting can help LLMs reason through problems, but it may not solve the issue of inaccurate math operations.
- PAL (Program-Aided Language Models) is a framework that pairs LLMs with external code interpreters to perform calculations and improve accuracy.
- PAL uses chain of thought prompting to generate executable Python scripts that are passed to an interpreter for execution.
- The prompt includes reasoning steps in natural language as well as lines of Python code for calculations.
- Variables are declared and assigned values based on the reasoning steps, allowing the model to perform arithmetic operations.
- The completed script is then passed to a Python interpreter to obtain the answer to the problem.
- PAL ensures accurate calculations and reliable results, especially for complex math problems.
- The process can be automated by using an orchestrator, a component that manages the flow of information and interactions with external data sources or applications.
- The orchestrator interprets and executes the plan generated by the LLM, which includes writing the script for the external interpreter to run.
- In more complex applications, the orchestrator may handle multiple decision points, validation actions, and interactions with various external resources.

ReAct : Reasoning and Action

- ReAct combines chain of thought reasoning with action planning in LLMs.
- It uses structured examples to guide the LLM's reasoning and decision-making process.
- Examples include a question, thought (reasoning step), action (pre-defined set of actions), and observation (new information).
- Actions are limited to predefined options like search, lookup, and finish.
- The LLM goes through cycles of thought, action, and observation until it determines the answer.
- Instructions are provided to define the allowed actions and provide guidance to the LLM.

LangChain

- LangChain provides modular components for working with LLMs in applications.
- It includes prompt templates for various use cases, memory to store LLM interactions, and tools for working with external datasets and APIs.
- Pre-built chains optimised for different use cases are available for quick deployment.
- Agents, such as PAL and ReAct, can be incorporated into chains to plan and execute actions.
- LangChain is actively developed with new features being added, allowing for fast prototyping and deployment.

Other Considerations

- The ability of the LLM to reason and plan actions depends on its scale.
- Larger models are generally more suitable for advanced prompting techniques like PAL and ReAct.
- Smaller models may struggle with highly structured prompts and may require additional fine-tuning.
- Starting with a large model and collecting user data in deployment can potentially train a smaller, fine-tuned model for better performance.

LLM Application Architecture

Users

Systems

Application Interface - Web, Mobile App, APIs etc.

Tools & Frameworks like LangChain, ModelHub etc.

Information Sources
- Documents
- Databses
- Web

LLM Models (Optimised)

Generated Outputs
Feedback on Outputs

Infrastructure - for training, fine-tuning, serving, application components etc.

Key Components of LLM Powered Applications

Responsible AI

- Toxicity: Toxic language or content that can be harmful or discriminatory towards certain groups. Mitigation strategies include curating training data, training guardrail models to filter out unwanted content, providing guidance to human annotators, and ensuring diversity among annotators.

- Hallucinations: False or baseless statements generated by the model due to gaps in training data. To mitigate this, educate users about the technology's limitations, augment models with independent and verified sources, attribute generated output to training data, and define intended and unintended use cases.

- Intellectual Property: The risk of using data returned by models that may plagiarise or infringe on existing work. Addressing this challenge requires a combination of technological advancements, legal mechanisms, governance systems, and approaches like machine unlearning and content filtering/blocking.

Generative AI Project Lifecycle Cheatsheet

	Training Duration	Customisation	Objective	Expertise
Pre-training	Days/ Weeks/ Months	• Architecture • Size • Vocabulary • Context Window • Training Data	Next token preditction	High
Prompt Engineering	Not required	• Only prompt customisation	Increase task performance	Low
Fine tuning / Prompt tuning	Minutes/ Hours	• Task specific tuning • Domain specific data • Update model / adapter weights	Increase task performance	Medium
RLHF/ RLAIF	Minutes/ Hours + data collection for reward model	• Train reward model [HHH goals] • Update model / adapter weights	Increase alignment with human preferences	Medium - High
Compression/ Optimisation/ Deployment	Minutes/ Hours	• Reduce model size • Faster inference	Increase inference performance	Medium

Acknowledgements

Ever since the transformers architecture gained popularity, large language models have been in-focus. From the era of skepticism in 2018/2019 to the explosive hype with the release of chatGPT in 2022, the evolution has been nothing short of magical.

Up until recently, the access to the knowledge was also largely inaccessible for regular people. This course by deeplearning.ai and AWS is another step in the democratisation of this technology. Like other courses by deeplearning.ai in the LLM and Generative AI space, this course is simple, practical and useful.

I'd like to thank -

- **Coursera** for being the influencer and contributor in my lifelong learning.
- **Dr Andrew Ng**, for making learning ML and data science for developers and enthusiasts unintimidating, simple and practical.
- The **team at deeplearning.ai** for making yet another course with such fine detail and practical labs.
- **Antje Barth, Chris Fregly, Shelbee Eigenbrode and Mike Chambers** for teaching me this very special course.
- All my **colleagues and friends** who endeavour to learn, discover and apply technology everyday in their effort to make the world a better place.

With Lots of Love,
Kim

I talk about :
#AI #MachineLearning #DataScience
#GenerativeAI #DataProducts #Analytics
#LLMs #Technology #Education #EthicalAI

Let's connect: