GenAI essentials

-Srimugunthan

Outline

GenAI essentials –Part1

- GenAI-An introduction
- Basic concepts and building blocks
  - RNN
  - LSTM
  - Attention mechanism
  - Transformers
- GenAI models
  - Large language models (GPT models)
- Landscape of open LLMs
  - Implement a chatbot

GenAI essentials –Part2

- Prompt engineering
- Finetuning
- Quantization ,Model pruning and model distillation
- Langchain – Introduction
- RAG & Knowledge graphs
- LLM app design and deployment
- Costs

GenAI essentials-part1

Agenda

1

GenAI- Introduction

GenAI history, usecases , applications and learning path

2

GenAI- Basic building blocks

RNN,,Attention, Transformers

3

GPT and chatGPT models

Architectural details of GPT

4

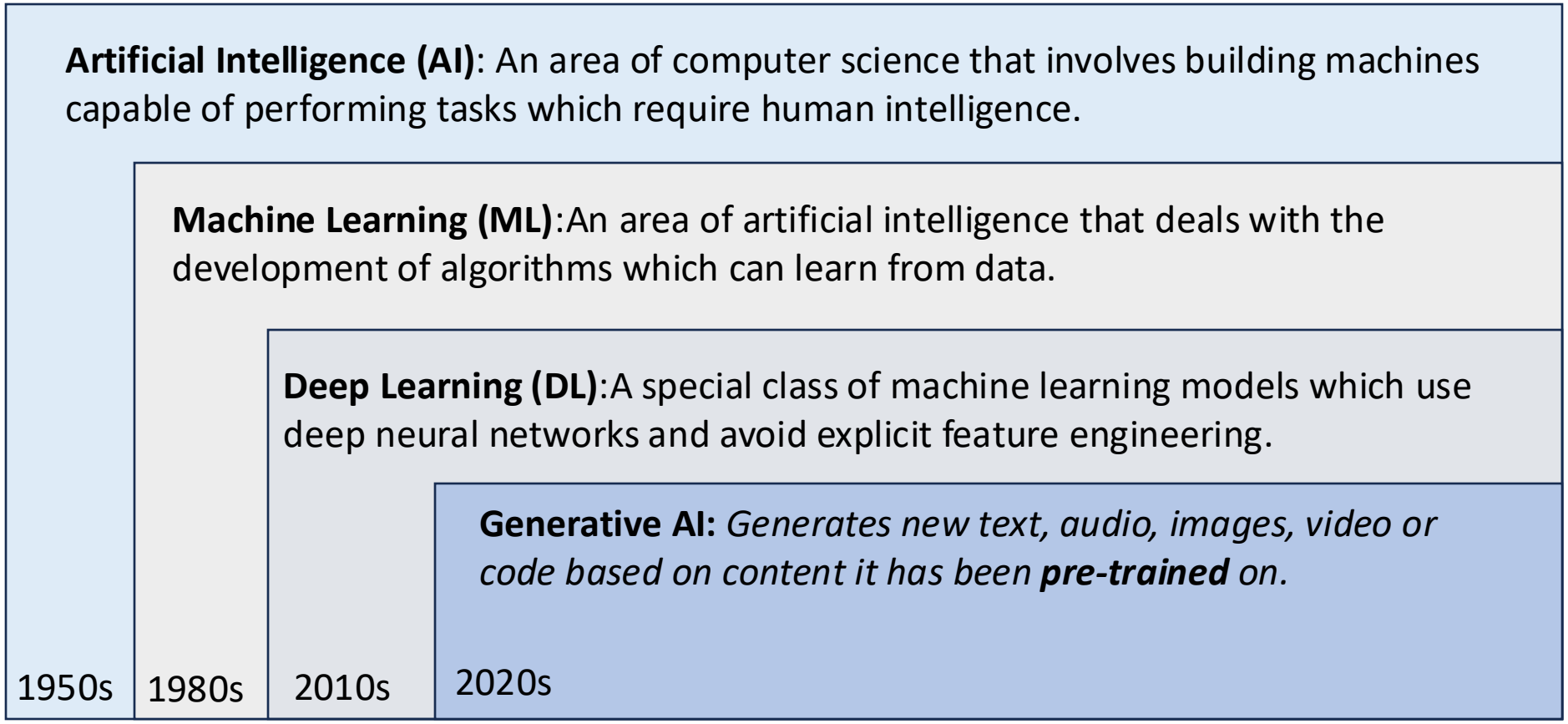
Landscape of open LLMs

Overview of hugging face models

01

GenAI-An introduction

What is Generative AI



https://arxiv.org/pdf/2310.12321

GenAI - Now and Future

AI events post chatGPT

March 2023
GPT-4 Launch and ChatGPT API

May 2023
MusicLM for Text to Music and Midjourney 5.1 Launched

June 2023
Runway's Gen-2 Revolutionizes Text-To-Video

August 2023
DeepMind's AlphaFold Milestone

September 2023
ChatGPT Can See, Hear, And Speak

December 2023
Launch of Google Gemini

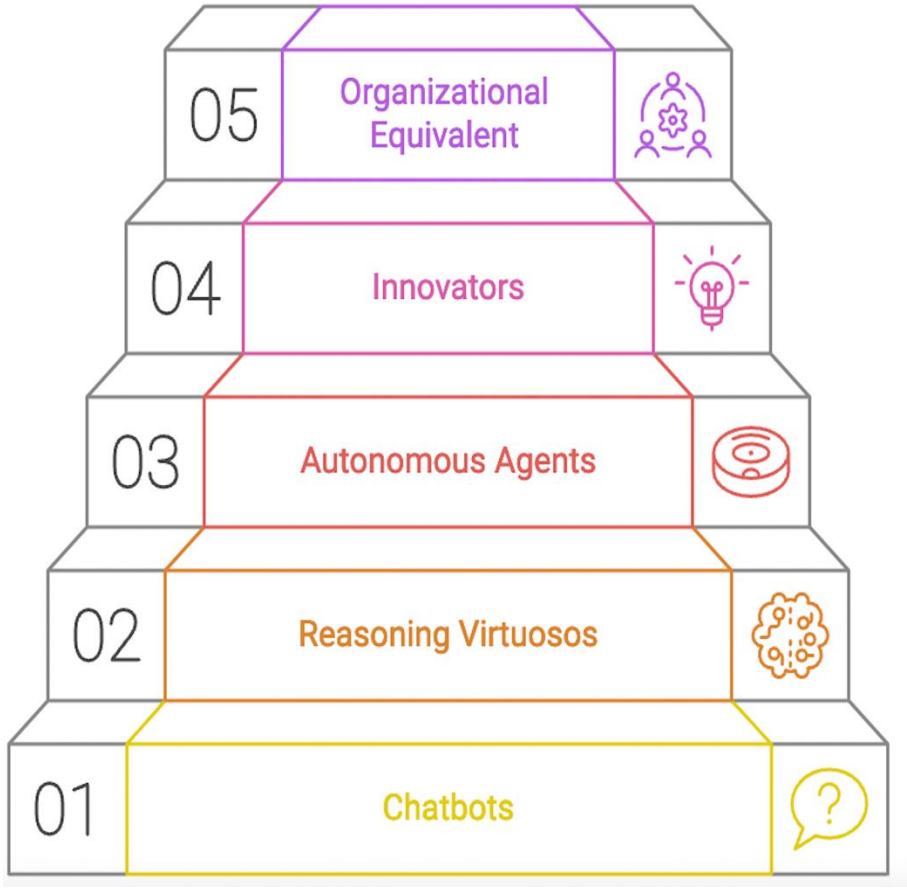
March 2024
EU AI Act Passed

April 2024
Small Language Model Llama 3, Phi3 Released

December 2024
Google Launched Gemini 2.0/Gemini Deep Research

January 2025
Deepseek-r1 Surpasses ChatGPT in Number of Downloads

Open AI AGI blueprint



Collaborating AI agents that can run an organization

AI that can create new ideas and solutions

AI agents that act autonomously without human intervention

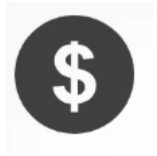
AI capable of reasoning and complex problem solving

Basic AI that interacts with users through text and voice

https://www.forbes.com/sites/jodiecook/2024/07/16/openais-5-levels-of-super-ai-agi-to-outperform-human-capability/

Generative AI usecases

FINANCIAL



- Automated report generation
- Financial Forecasting Narratives
- Synthetic Data Generation
- Fraud Scenario Simulation

RETAIL &E-COMMERCE

- Product Description Generation
- Customer Review
- Summarization
- Chatbots for Shopping
- Demand Forecast Commentary

MANUFACTURING

- Automated Maintenance Logs
- Design Prototyping
- Work Instruction Generation
- Anomaly Explanation

HEALTHCARE

- Clinical Report Drafting
- Medical Literature Summarization
- Synthetic Patient Data
- Drug Discovery Support

TELECOM and MEDIA

- Network Issue Explanation
- Personalized Content Summaries
- Script Writing
- Customer Retention Bots

GenAI Learning path

Deep learning basics

Neural network
Next word prediction with RNN
Image and Vision models
Encoder decoders

→

GenAI Foundations

Transformers
Reinforcement learning with human feedback
Quantization
Pre-tuning and Finetuning
Quantization

→

Using GenAI models

Foundation model API
Hugging face models.,
Ollama, Running models locally
Prompt engineering

→

Image & Audio

Auto encoders, GAN
Stable diffusion

RAG

Retrieval augmented Generation
Vector Databases. RAG application

Multimodal RAG

Knowledge Graph

→

LLM tools

Langchain
Tool
LLamaIndex

Streamlit, openwebui, chainlit

→

LLM ops

Deploying LLM app on cloud

Langserve

→

Autonomous AI agents

AI agent development
Autogen, Langgraph, CrewAI etc

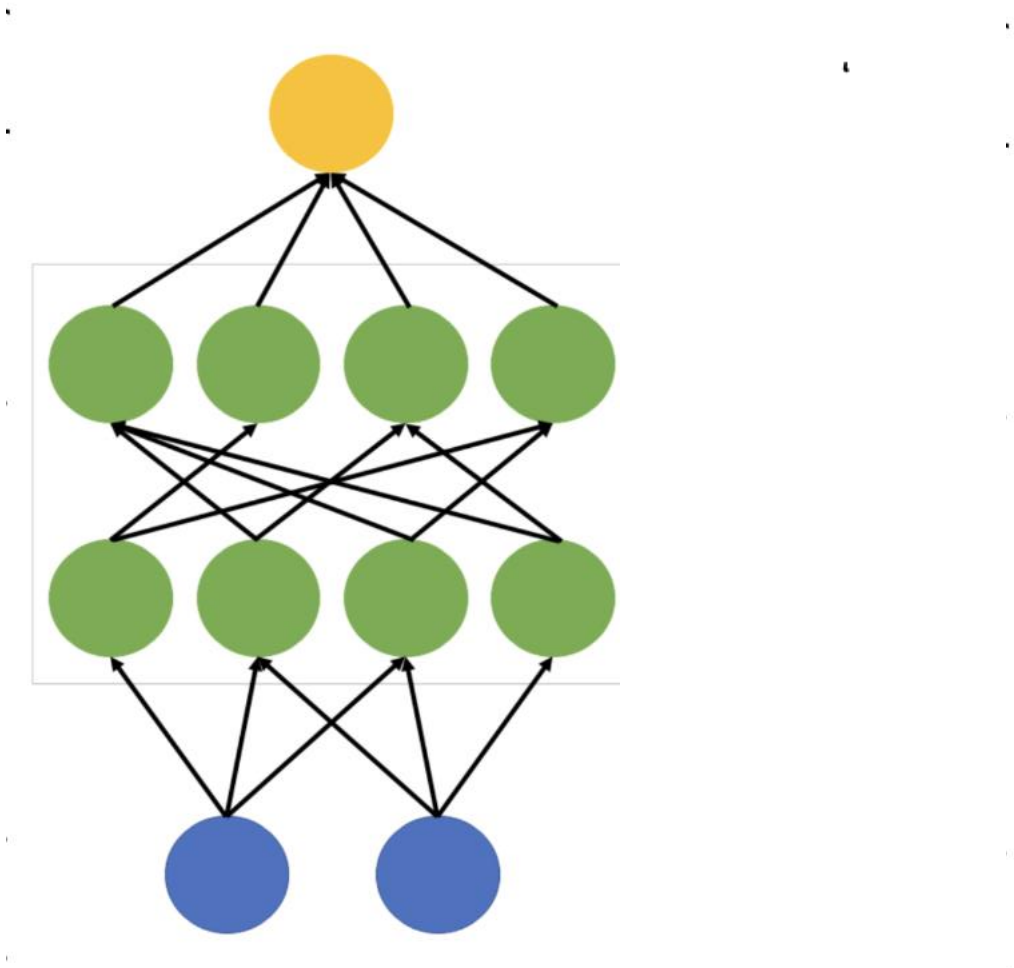
Agentic RAG

02

GenAI basic building blocks

Neural network basics

loss(Pred, Truth) = E



Forward pass

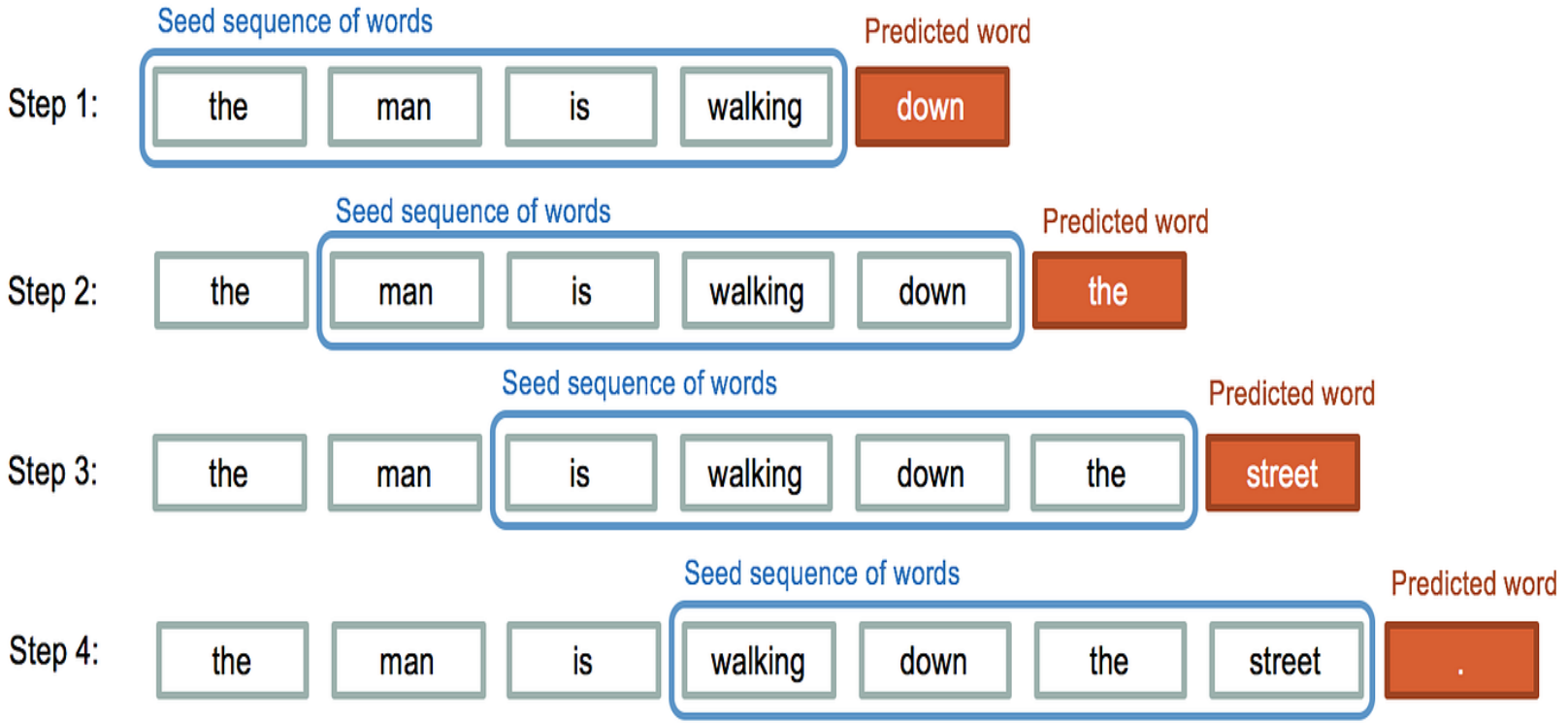
Backpropagation

Gradient

∇

new weight = weight - learning rate*gradient

Next word prediction problem



Goal:
Predict next word given

prob	word
0.001	apple
0.002	banana
0.400	cake
0.012	donut
…	…

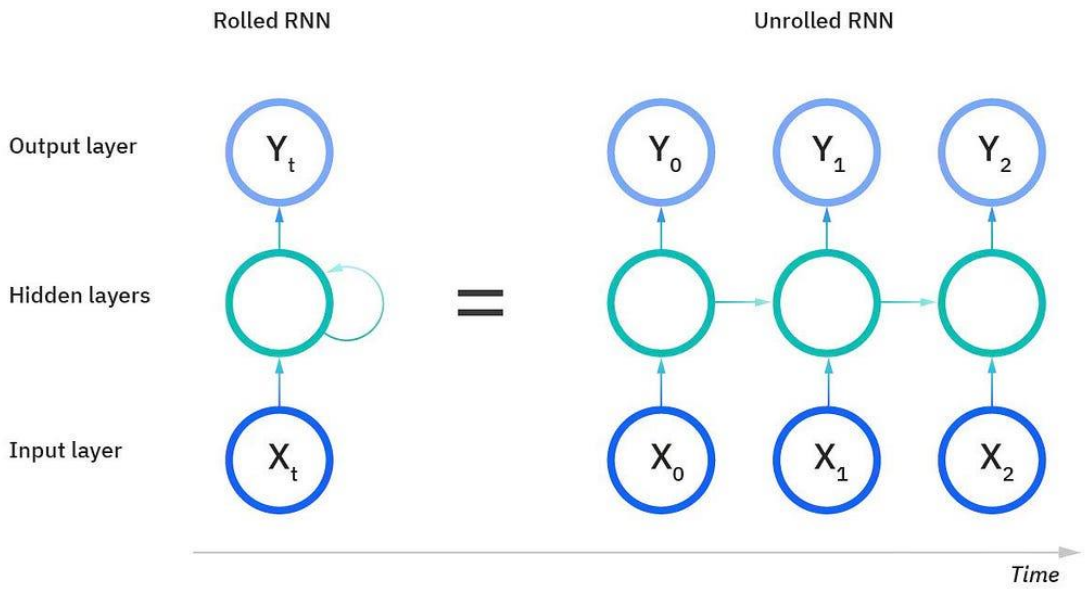
Steps:

1. **Pre-processing**
2. **Model Construction**
3. **Training**
4. **Evaluation**
5. **Prediction**

Recurrent neural network

Early research in the mid-1980s by Jordan explored training neural networks on sequential patterns using recurrent neural networks (RNNs) with "state units" to provide memory. The network learned by predicting the next symbol in a sequence.

- RNNs are designed for sequential data.
- Used in NLP, time series, speech, and more.



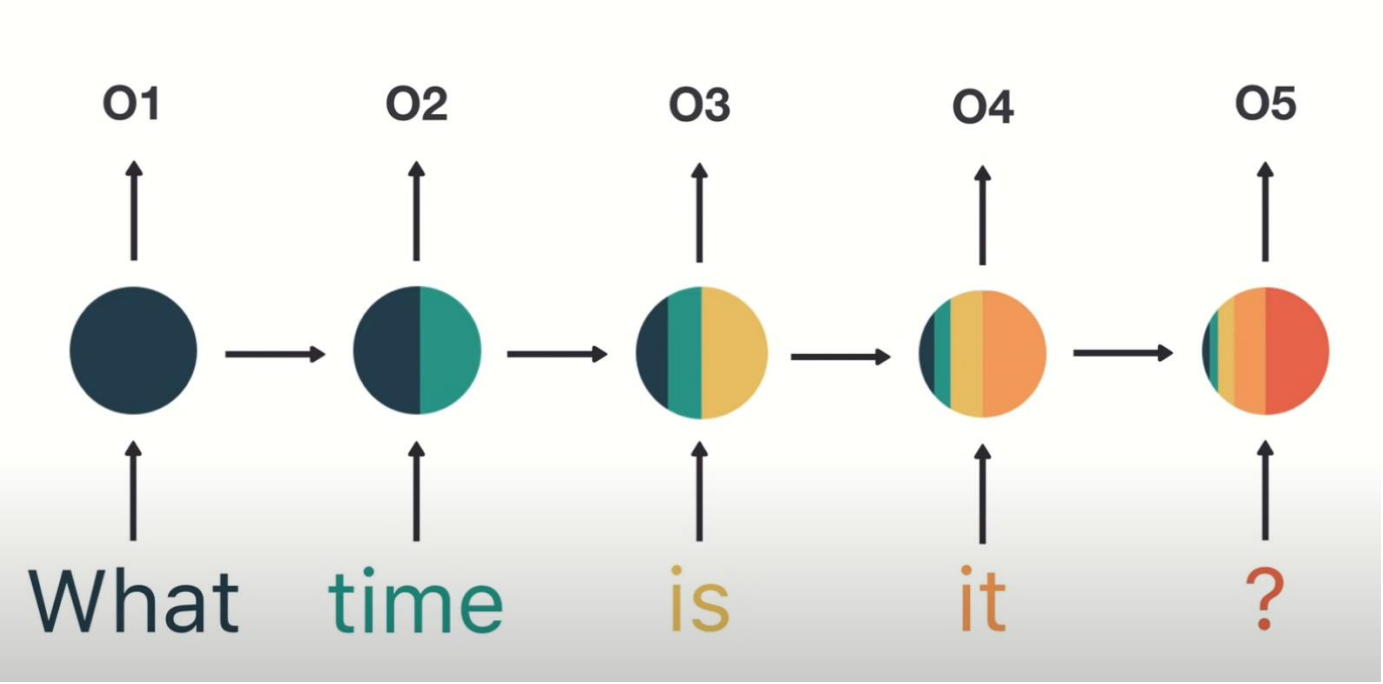
RNNs maintain a **hidden state** that acts as a "memory" of previous inputs in the sequence. This allows them to process sequences of arbitrary length while considering the context from earlier steps.

not only memorize sequences but also generalize them, identifying underlying patterns.

https://medium.com/@RobuRishabh/recurrent-neural-network-rnn-8412b9abd755

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

RNN intuition



- At each time step t, the RNN maintains a **hidden state** (often denoted as h_t).
- This hidden state is a vector that captures information about the sequence up to the current time step. It acts as the network's "memory."
- The hidden state acts as a **compressed representation of the past sequence**. As the network processes subsequent inputs, the hidden state is updated, effectively carrying forward relevant information.

https://medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07b

RNN for next word prediction

https://colab.research.google.com/drive/1qX-lC5gWa26hjM_0jl26lcesX23XEYF7

Limitations of RNN

Sequential Computation

- RNNs process tokens one-by-one, making parallelization difficult.

Long-Range Dependency Issues

- RNNs struggle to retain information over long sequences due to vanishing gradients.

Gradient Instability

- RNNs (especially vanilla ones) suffer from **vanishing/exploding gradients** during training.

Limited Contextual Awareness

- RNNs have a fixed-size hidden state; info gets compressed over time.

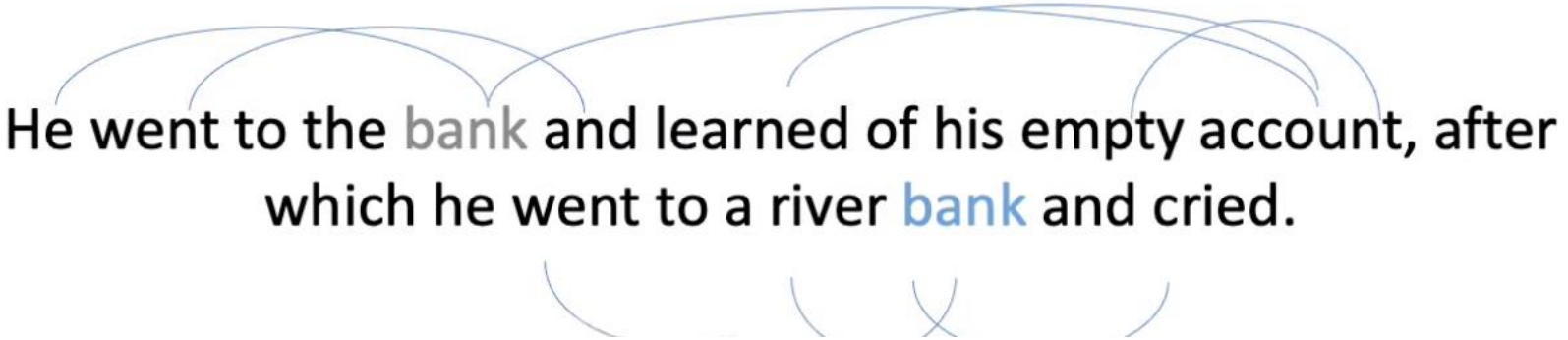
Difficulty Handling Long Sequences

- RNNs degrade in performance with longer input.

Attention mechanism -Intuition

Attention: Based on the idea that not only can all the input words be taken into account in the context vector, but relative importance should also be given to each one of them.

Calculate relationships between words



7 x 7

	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99

•Compute **attention score** for how well each word's **Key** matches the current **Query**.

Attention score calculation

$$\text{Attention Score} = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

- The **dot product** ($Q \cdot K^T$) measures similarity.

- **Softmax** turns scores into probabilities (which words matter most?).

Why This Works So Well

1. Dynamic Relationships:

1. Unlike fixed rules (e.g., grammar), attention adapts based on context.
2. *"Bank"* can attend to *"money"* or *"river"* depending on the sentence.

2. Long-Range Dependencies:

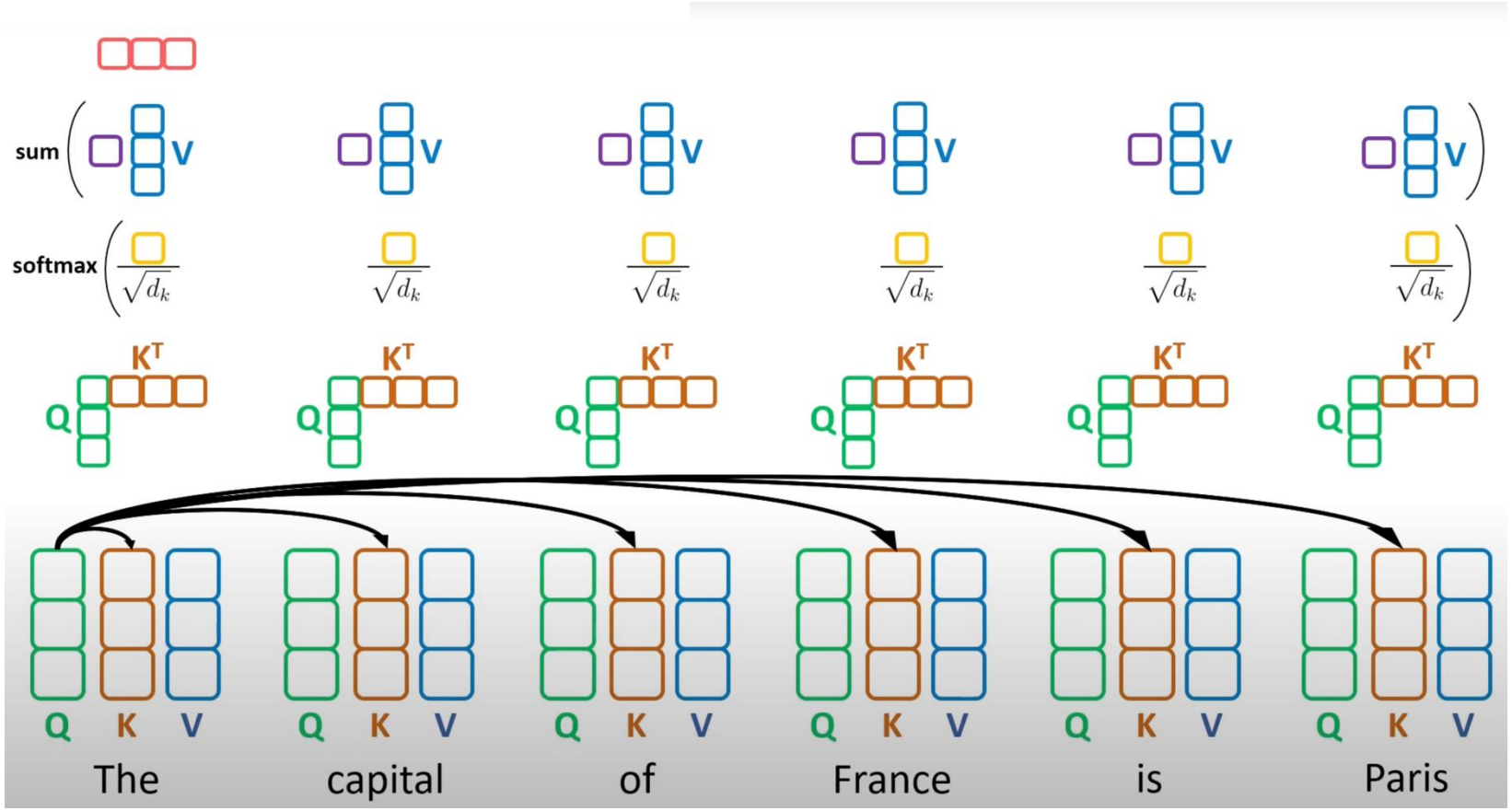
1. Words far apart (e.g., *"cat"* and *"mat"*) can still influence each other.

3. Parallel Computation:

1. All attention scores are computed at once (unlike RNNs, which process sequentially).

Visualizing Attention mechanism

$$Attention(Q, K, V) = softmax(QK^T/sqrt(d)) * V$$



https://www.youtube.com/watch?v=u8pSGp__0Xk

Understanding Attention mechanism

Self-attention (core idea)

Given a sentence like:

> "The cat sat on the mat."

Let's say we're processing "sat" — attention lets the model look at **all other words** and decide which are important to understand "sat" in context.

How?

Each word is turned into:

- **Query (Q)** → What am I looking for?
- **Key (K)** → What do other words offer?
- **Value (V)** → What do I take from them?

Then for each word:

$$Attention(Q, K, V) = softmax(QK^T/sqrt(d)) * V$$

This gives a weighted combination of other words' information — focusing on the **most relevant**.

How This Captures Semantic Relationships
Example: "The cat sat on the mat."

•Suppose we're processing the word **"cat"**:

- Its **Query (Q)** asks: *"Which words are related to me?"*
- **Keys (K)** of other words respond:
  - **"sat"** → *"I'm an action done by animals."* (High match → high attention)
  - **"mat"** → *"I'm something you sit on."* (Moderate match)
  - **"the"** → *"I'm just a filler word."* (Low match → ignored)

Implementing Attention mechanism

https://colab.research.google.com/drive/1tT4e1p4MW5OUqjDKJhVpUvrdarER0aBB

Multihead Attention mechanism

- **Multi-Head Attention:** Instead of performing a single attention operation on large Q, K, and V matrices, multi-head attention breaks down the problem.
- It uses h (number of heads, typically 8) sets of smaller dimension query, key, and value matrices.
- These smaller Qi, Ki, and Vi matrices (where i ranges from 1 to h) are obtained by multiplying the input features X with h different sets of weight matrices (WQi, WKi, WVi).
- **Steps in Multi-Head Attention:**
  - Generate h sets of query (Q1 to Qh), key (K1 to Kh), and value (V1 to Vh) matrices.
  - Perform scaled dot-product attention independently on each (Qi,Ki,Vi) triple. The output of each SDP is a context matrix Headi with dimensionality T×hD, where D is the original dimensionality of the input features.
  - Concatenate the h resulting context matrices (Head1 to Headh) along their last dimension. This results in a matrix Z with dimensionality T×D.
  - Multiply the concatenated matrix Z with another learnable weight matrix WO to produce the final output of the multi-head attention layer.
- **Benefit of Multi-Head Attention:** While having a similar computational cost to single-head attention, multi-head attention can capture context information from different representation subspaces and at different positions within the input sequence.

https://www.youtube.com/watch?v=mmzRYGCfTzc

Self attention vs Cross attention

- **Self-Attention vs. Cross-Attention: Self-Attention:** Used to find relationships between words within a single input sequence. The query (Q), key (K), and value (V) matrices are all derived from the same input sequence X.
- **Cross-Attention:** Used when there are two different input sequences, for example, in translation tasks where one sequence (Y, e.g., German) attends to the other (X, e.g., English). To achieve this, the query matrix (Q) is derived from one sequence (Y), while the key (K) and value (V) matrices are derived from the other sequence (X).
- The subsequent steps after obtaining Q, K, and V are the same for both self-attention and cross-attention.

https://www.youtube.com/watch?v=mmzRYGCfTzc

Attention mechanism-Advantages

No Gradient instability

No Bottleneck of Hidden State

•RNNs compress all prior info into a single hidden state (which loses detail over time).

•Transformers use **attention scores** to keep context explicitly, avoiding the need to squeeze everything into one vector.

Handling Long sequences

Efficiency Tricks for Long Sequences

Transformers: Any token can directly attend to any other token, even if they're far apart (e.g., a verb at the start of a sentence can influence a noun at the end). Instead of forgetting, they **reweight importance**—irrelevant tokens may get low attention scores but aren't explicitly erased

Techniques like sparse attention work for long sequence

Fully parallelizable

Transformers: Process **all tokens simultaneously** in a single forward pass using **self-attention**.

- No recurrence → Full parallelization across the sequence.
- Enables efficient GPU/TPU utilization (critical for large models).

From Attention mechanism to Transformers

Attention

- **Attention** is a way for a model to "look at" different parts of an input sequence when processing a word.
- It lets the model focus on relevant information, even if it's far away in the sequence.

"While processing word X, I should also pay attention to word Y."

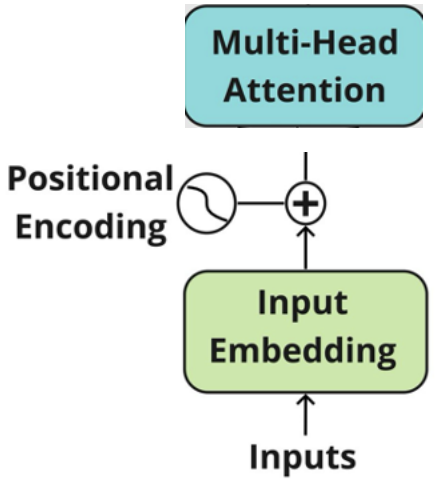
attention needs structure and additional blocks to actually **do something useful** .

- **Multi-head attention** helps learn multiple relationships in parallel.
- **Positional encoding** gives the model a sense of word order (attention doesn't know order by default).
- **Feedforward layers** add non-linearity and transformation power.
- **Layer normalization** and **residual connections** make deep models trainable.

STEP2: Attention mechanism +Add positional encoding

- Attention is permutation-invariant—i.e., it doesn't know the *order* of tokens.

- **Solution:**
  Add **Positional Encodings** to input embeddings to inject information about the position of each word in the sequence.

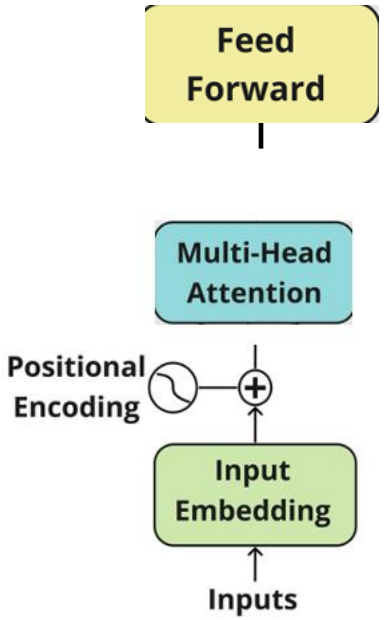


Value Added:
Adds **sequential structure**, helping the model understand token order (e.g., "dog bites man" vs "man bites dog").

STEP3 +Add feed forward layer

- Self-attention captures interactions but lacks **per-word transformation**.

- **Solution:**
  Apply a 2-layer MLP with ReLU to each position independently.



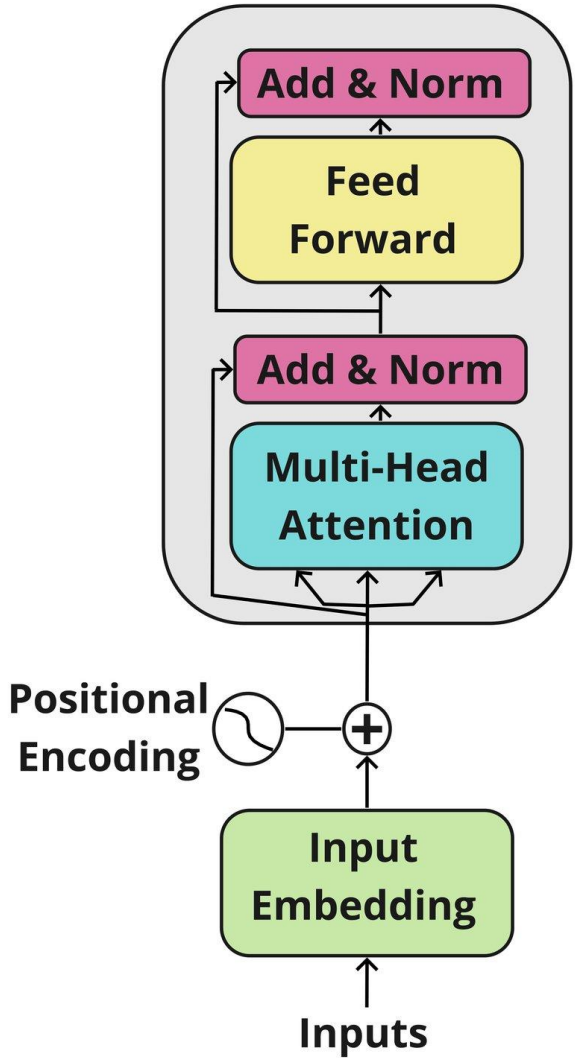
Value Added:
Adds **non-linearity and transformation** capacity.
Attention captures dependencies, but FFN transforms representations at each position to be richer, learn complex mappings
Adds **non-linearity** and transforms features (e.g., turning "cat + sat" into "resting").

STEP4 +Residual connections+Layer norm

- Deep networks suffer from vanishing gradients and training instability.

- **Solution:**

Residual connection: Add the input back after transformation.
Layer normalization: Normalize across features for stable training.



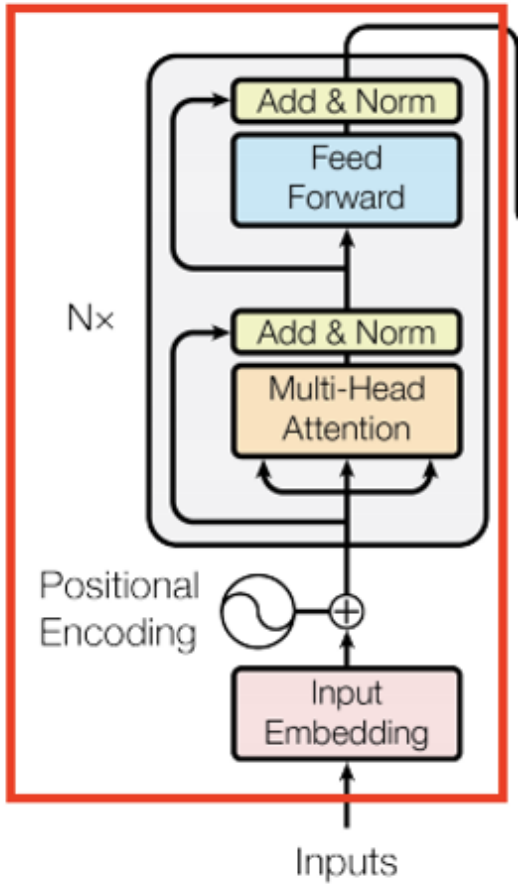
Value Added:
•Residuals help **gradient flow** and **preserve input features**.
•Layer norm stabilizes training and allows deeper stacking.

STEP5: Encoder block

Structure:
•Multi-head self-attention
•Add + Norm
•Feedforward
•Add + Norm

Value Added:
The encoder can now build **rich contextual representations** of the input sequence.
Encoder processes "bank" to mean "financial institution" (not "riverbank") using context.

Can be stacked (depth = more abstraction)



Attention mechanism –Decoder block

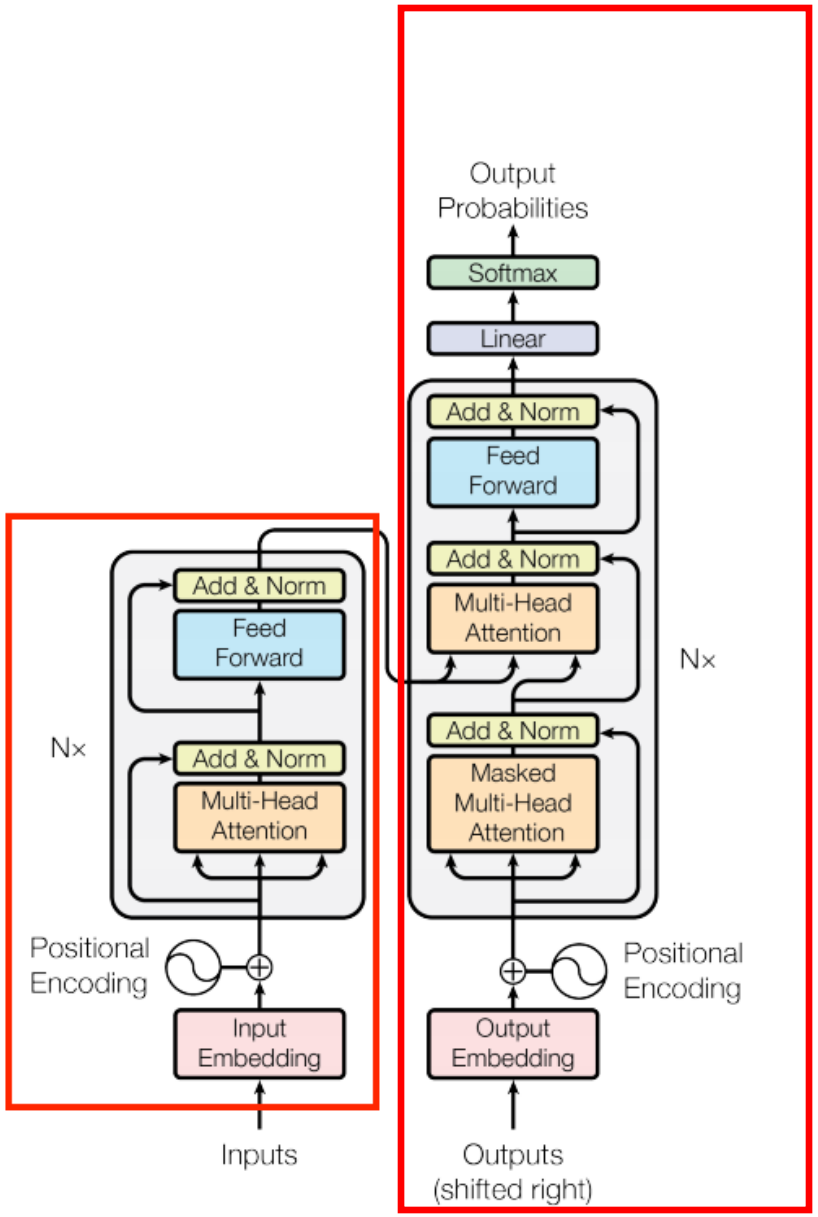
The decoder generates outputs **autoregressively** (word by word).

Masked Self-Attention + Cross-Attention)

Value Added:

•**Masking** enables **autoregressive generation**.

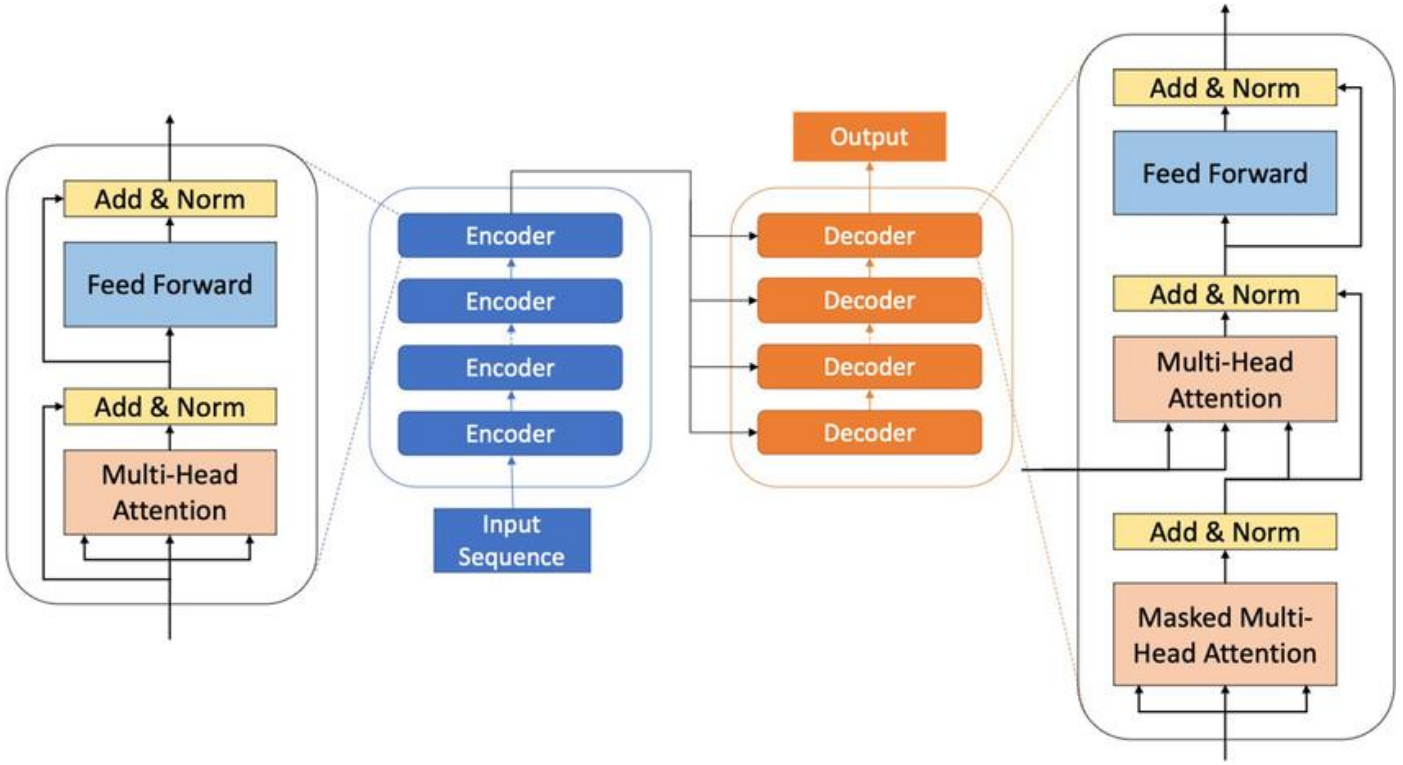
Cross-attention links input and output (essential for seq2seq tasks).



Attention mechanism –> Stacked transformers

Stacking multiple encoder/decoder blocks (e.g., 6 each in the original paper)
•Refine representations **hierarchically**
•Capture increasingly **complex and abstract patterns**
•**Attend over attention** outputs from prior layers, improving reasoning depth

Value Added:
More layers = more expressive power.
Deeper architectures can capture **subtle relationships**, like long-range dependencies, multi-hop reasoning, etc.



Transformers –Architecture summary

Step	Component	Why it's added / What it enables
1	Attention	Contextual understanding, long-range dependencies
2	Positional Encoding	Awareness of word order
3	Feedforward Layer	Non-linear transformation per token
4	Residual + LayerNorm	Stability, faster convergence, better training in deep models
5	Encoder Block	Rich contextual token representations
6	Decoder Block	Sequence generation, conditioned on input
7	Multiple Stacked Blocks	Hierarchical abstraction, increased capacity

DEMO:

Implementing transformers

https://medium.com/data-science/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb

Visualizing transformers

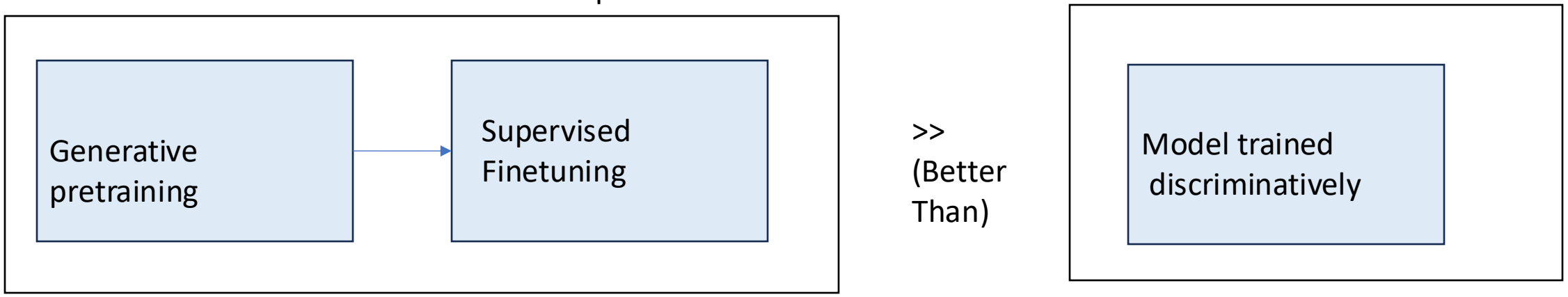
https://poloclub.github.io/transformer-explainer/

03

GPT and chatGPT model

What is Generative Pre-training

For NLP tasks like text classification or sentiment prediction:



Because:

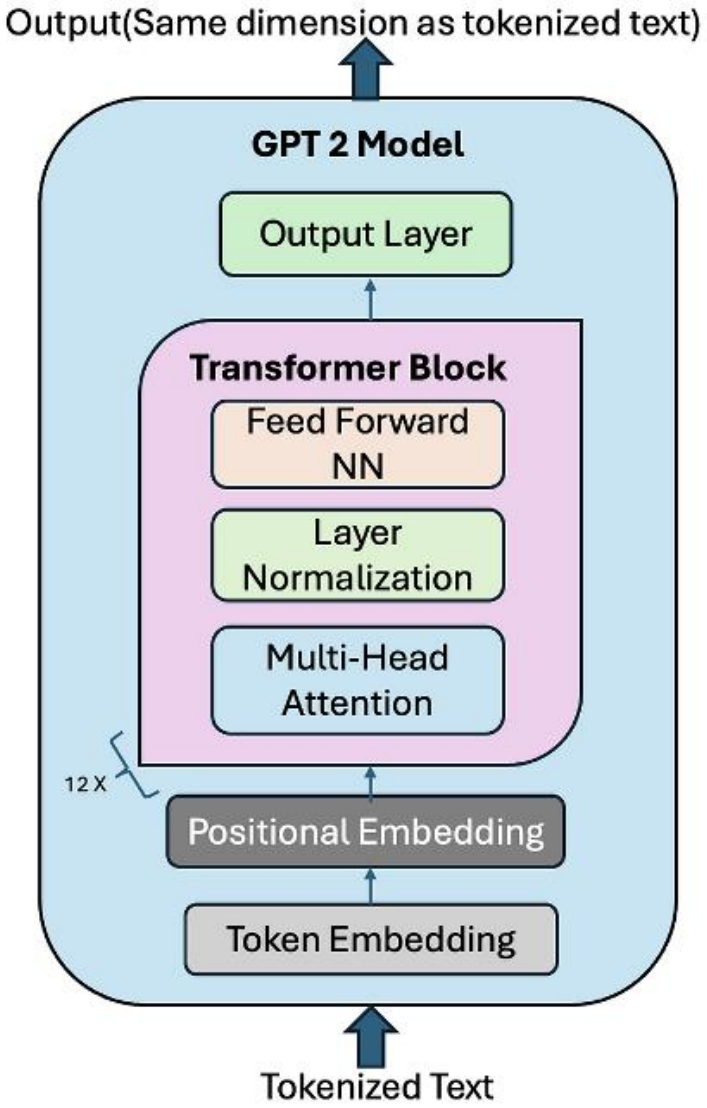
- to accurately predict the next word, the model needs to understand syntax, semantics, context, and even some world knowledge.
- generatively pre-trained model starts with a much **better initial understanding of the language** compared to models trained from scratch or purely discriminatively.

So:

1. First Learn general language representations
2. Follow with Adapt to a specific task with supervised finetuning

"Improving Language Understanding by Generative Pre-Training"
https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT: Generative pretrained transformer



• **Decoder Stack:** GPT primarily utilizes the decoder part of the Transformer architecture.

•**Key Components**:

- **Input Embeddings:** Input tokens are converted into dense vector representations.
- **Positional Encoding:** Adds information about the position of tokens in the sequence, crucial as the Transformer lacks inherent sequential processing.
- **Self-Attention**: Captures contextual relationships between words.
- **Layer Normalization & Residual Connections**: Stabilizes training and improve information flow within the deep network..
- **Feed-Forward Networks**: Processes attention outputs.

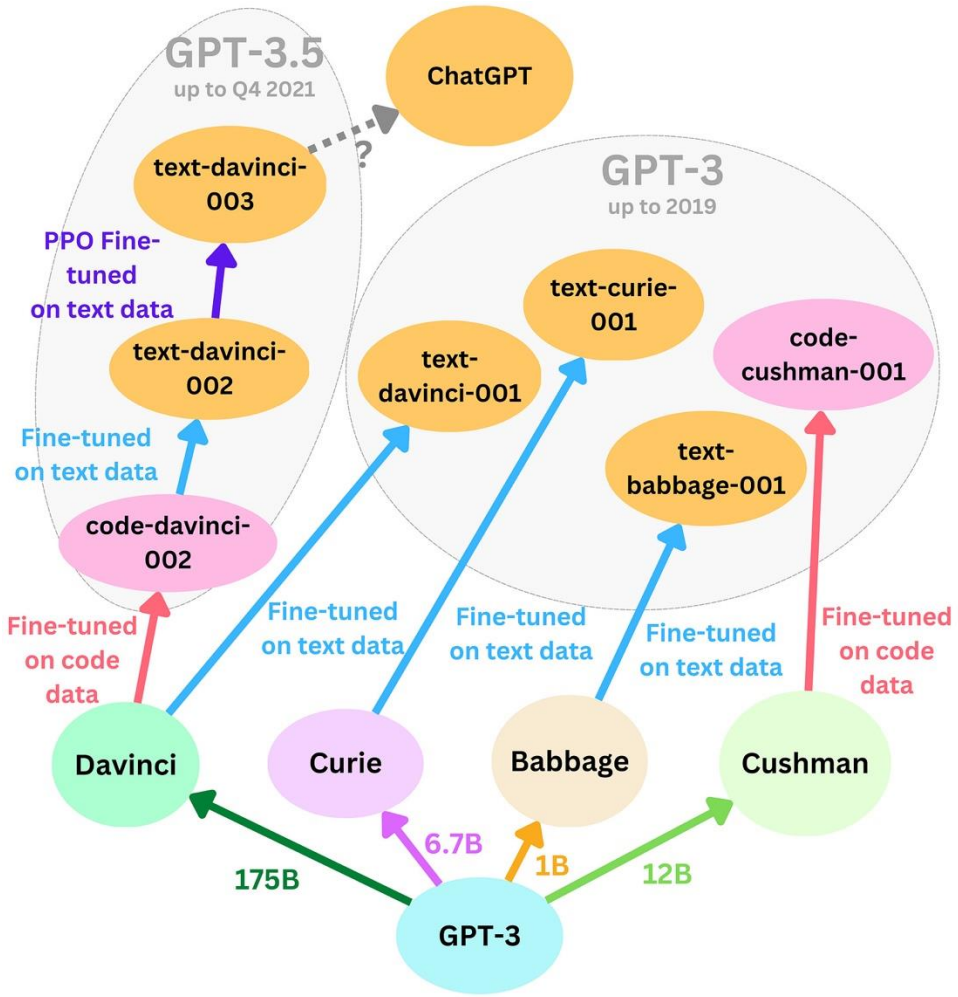
- **Autoregressive Generation:** Text is generated sequentially, one token at a time, with the previously generated tokens serving as context for the next prediction.

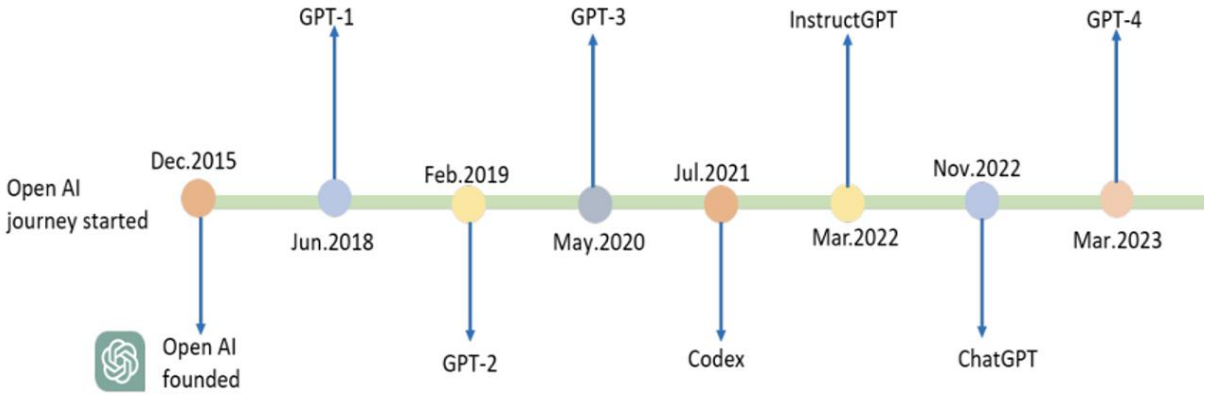
GPT implementation

GPT implementation:
https://colab.research.google.com/drive/195dPYDRjAbqw9P0Y3OWprzrafd6X8mWk#scrollTo=fjjvMifYZf7x

https://medium.com/@vipul.koti333/from-theory-to-code-step-by-step-implementation-and-code-breakdown-of-gpt-2-model-7bde8d5cecda

GPT family of models



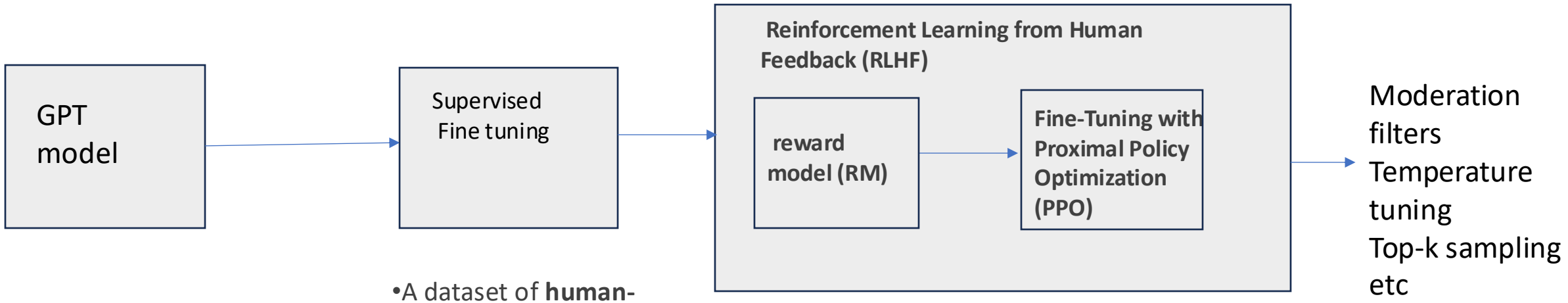


Feature	GPT-1	GPT-2	GPT-3	GPT-4
Year	2018	2019	2020	2023
Paper Title	"Improving Language Understanding by Generative Pre-training"	"Language Models are Unsupervised Multitask Learners"	"Language Models are Few-Shot Learners"	"GPT-4 Technical Report"
Parameters	117M	1.5B	175B	Undisclosed but larger than GPT-3
Architecture	12-layer transformer decoder	48-layer transformer decoder	96-layer transformer	Optimized transformer architecture, multimodal capabilities
Context Window	512 tokens	1024 tokens	2048 tokens	Longer than GPT-3, up to 32,000 tokens in some versions

https://newsletter.theaiedge.io/p/the-chatgpt-models-family

https://medium.com/@vipul.koti333/evolution-of-gpt-models-gpt-1-to-gpt-4-0238ee07a29b

Additional Steps : GPT to “Model for chatGPT”



•A dataset of **human-written conversations** (user prompts + assistant responses) is used to fine-tune GPT in a **supervised** manner.
•The model learns to mimic human-like dialogue patterns (e.g., question answering, follow-up responses).

•Human annotators rank multiple model responses to the same prompt (e.g., "Which response is better?").
•This creates a **reward model (RM)** that predicts human-preferred outputs.

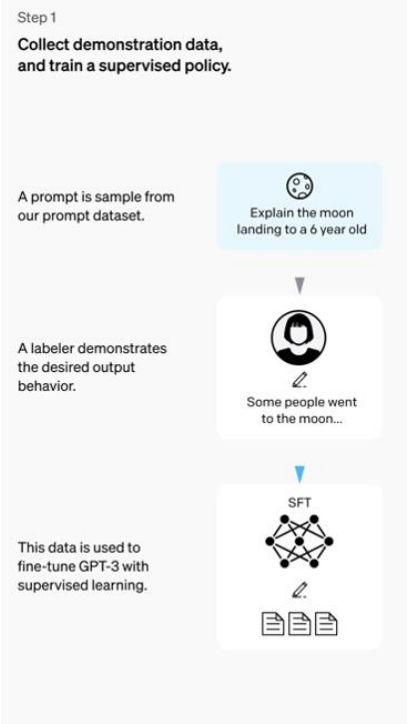
•The base GPT model is further optimized using **reinforcement learning (RL)**.
•The **reward model** guides the policy (GPT) to generate higher-quality, safer, and more engaging responses.
•The model learns to **maximize reward** while staying close to the original policy (to avoid drastic deviations).

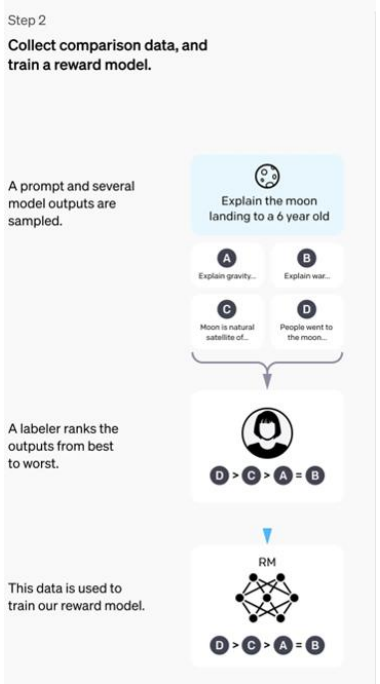
Additional Steps : GPT to “Model for chatGPT”

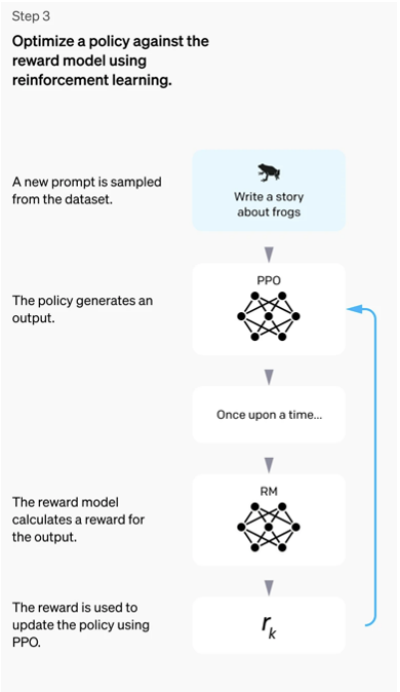
GPT model → Supervised Fine tuning → Reinforcement Learning from Human Feedback (RLHF) → Moderation filters, Temperature tuning, Top-k sampling etc

Reinforcement Learning from Human Feedback (RLHF): reward model (RM) → Fine-Tuning with Proximal Policy Optimization (PPO)

Finetuning and alignment







Demo: RLHF

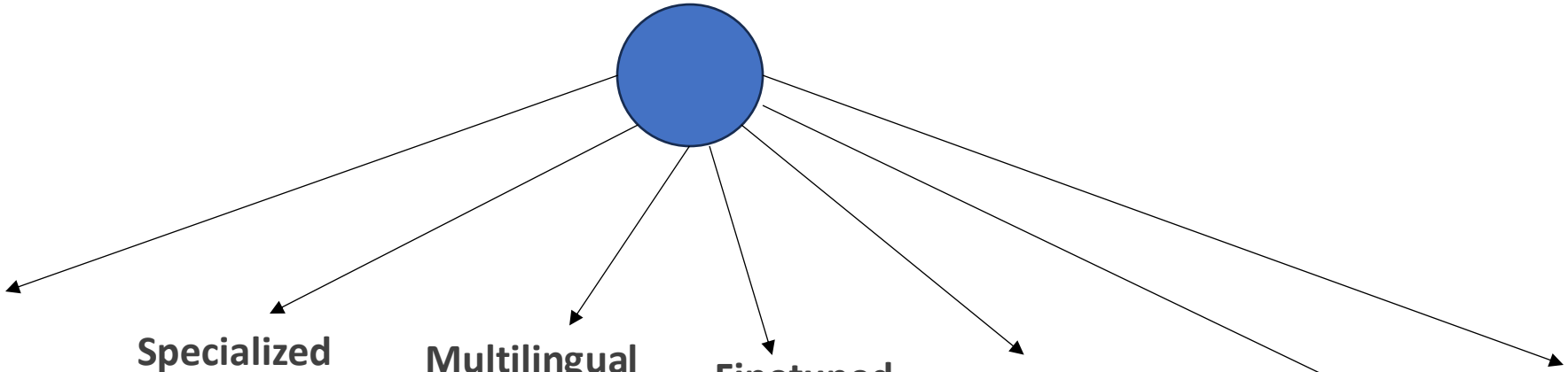
https://colab.research.google.com/github/heartexlabs/RLHF/blob/master/tutorials/RLHF_with_Custom_Datasets.ipynb?authuser=1#scrollTo=D_lQu3Ky0sWb

https://huggingface.co/blog/the_n_implementation_details_of_rlhf_with_ppo

04

Open LLMs landscape

Categorizing open LLMs



Generic LLMs

•**Generic LLMs:** These models are trained on a wide range of text data and can perform various natural language processing tasks, such as text generation, language translation, and sentiment analysis.

Specialized LLMS

•**Specialized LLMs:** These models are tailored for specific tasks or domains, such as legal documents, medical records, or code generation. They are trained on datasets relevant to the specific area.

Multilingual LLM

Multilingual LLMs: These models are trained on text data in multiple languages, allowing them to understand and generate texts in different languages

Finetuned LLM

•**Fine-tuned LLMs:** These models are trained on a large corpus of pext data and then fine tuned on a small set of data specific to a particular task or domain. Fine tuning LLM helps optimize the model for specific tasks.

Customized LLM

•**Customizable LLMs:** These models allow users to customize various aspects, such as model architecture hyperparameters and training data, to accommodate their specific needs or preferences better.

Community driven LLM

•**Community-Driven LLMs:** These models are developed and maintained by open source communities, enabling various developers and researchers to collaborate, collaborate, and improve.

Few-shot LLMs

Few shot LLMs: These models are designed to perform well even when fine-tuned with small amounts of labeled data. Examples : T5 with 10 labelled points

Open LLMs: https://github.com/eugeneyan/open-llms

Other ways of Categorizing open LLMs

By openness

Partially Open LLMs (Weights Available, but Restrictions Apply)

Fully Open LLMs (Weights + Architecture + Training Data Disclosed)

Proprietary

By size

Small & Efficient Open LLMs (<1B Parameters)

LLMs (>1B Parameters)

By Domain

Finance

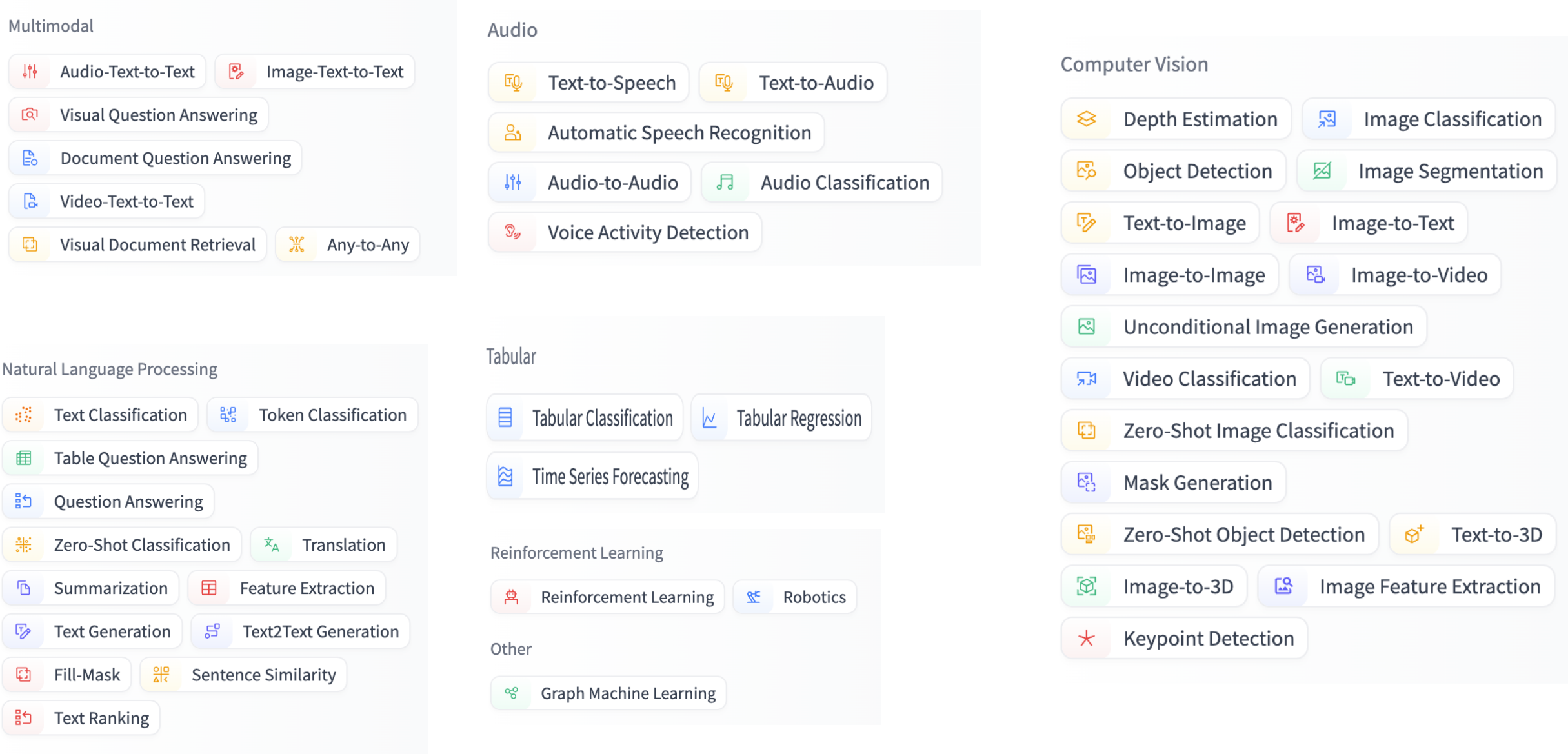
Legal

Medicine

By License

- apache-2.0
- mit
- cc-by-sa-4.0

Hugging face open LLMs: Task specific categories



https://huggingface.co/models

Estimating number of open LLMs

Open LLMs on Hugging Face (Mid-2024)

1. **Base LLMs (Original Models)**
    1. ~**200–300** distinct open-weight base models (e.g., LLaMA, Mistral, Falcon, OLMo, Phi, Qwen, etc.).
2. **Fine-tuned/Derivative Models**
    1. ~**10,000+** LLM variants (adaptations, LoRA fine-tunes, instruct models like WizardLM, OpenHermes, etc.).
3. **Small/Specialized LLMs (<1B params)**
    1. ~**5,000+** tiny/task-specific models (e.g., TinyLlama, GPT-2 clones, distillations).

Total Open LLMs on Hugging Face

- **Conservative estimate**: **5,000–10,000+** (including all variants).
- **Base models only**: **200–300+**.

•The number expected to **double every 6–12 months** as open AI research accelerates.

LLM leaderboard: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/

Domain based LLMs

Task	Hugging face LLM	Description
Finance	FinGPT	Financial LLM for specific tasks like sentiment analysis and market forecasting.
	InvestLM	fine-tuned on LLaMA-65B using a curated instruction dataset related to financial investment. It has shown strong capabilities in understanding financial text and providing helpful responses to investment-related questions.
	FinLlama	Based on the Llama 2 foundational model, FinLlama is fine-tuned for financial sentiment classification, aiding in algorithmic trading applications
•**Medicine**	MedPalm 2	Developed by Google achieves 86% accuracy on the MedQA dataset (
	Llama3-OpenBioLLM-8B	pre-trained audio event classification model
	Llama-medx_v3.1	A widely used multilingual speech-to-text model.
Legal	SaulLM	open-source Large Language Model for Law, with a 7 billion parameter base model
	InLegalBERT	pre-trained on a large corpus of Indian legal documents.

LLMs for NLP tasks

Task	Hugging face LLM	Description
summarization	facebook/bart-large-cnn	widely used BART model fine-tuned for abstractive summarization
translation	Helsinki-NLP/opus-mt-en-fr -	MarianMT model specifically trained for English to French translation
named-entity-recognition	dbmdz/bert-large-cased-finetuned-conll03-english	BERT model fine-tuned on the CoNLL-2003 dataset for English NER.
fill-mask	bert-base-uncased	original BERT base model is often used for fill-mask tasks
•**sentence-similarity**	sentence-transformers/all-mpnet-base-v2	Sentence-BERT model that excels at generating sentence embeddings for similarity tasks.
Code generation	Salesforce/codegen-350M-mono	A CodeGen model trained on a large corpus of Python code.
text-to-sql	google/t5-large-nl2sql -	A T5 model fine-tuned for the task of natural language to SQL translation.
text-to-image	CompVis/stable-diffusion-v1-4	Primarily a diffusion model for image generation
table-question-answering:	google/tapas-base	BERT-based model specifically designed for question answering over tabular data.

LLMs for audio/vision tasks

Task	Hugging face LLM	Description
Image classification	google/vit-base-patch16-224	Foundational vision transformer for image classification
object-detection	facebook/detr-resnet-50	Transformer based approach for object detection
semantic-segmentation	nvidia/segformer-b0-finetuned-cityscapes-512x1024	Segformer based model for semantic segmentation on the cityscapes dataset
image-generation	stabilityai/stable-diffusion-xl-base-1.0	Diffusion based model for image generation
•**visual-question-answering (VQA)**:	bert-vqa-base -	BERT based model for visual Q&A
audio-classification	google/yamnet-1024-cqlogmel-penn	pre-trained audio event classification model
speech-recognition	openai/whisper-small	A widely used multilingual speech-to-text model.
text-to-speech (TTS):	espnet/kan-bayashi_ljspeech_vits	A VITS model trained on the LJSpeech dataset for single-speaker text-to-speech
audio-generation	facebook/musicgen-small	A transformer-based model for music generation.

Evals : LLM Evaluation

EVALUATION CRITERIA:

- How accurate and factual are the model's responses?
- How coherent and fluent is the generated text?
- Can the model follow instructions effectively?
- How well does it perform on specific tasks like question answering, translation, or code generation?
- Does the model exhibit any harmful biases or generate inappropriate content?
- How robust is the model to adversarial inputs?

EVALUATION BENCHMARKS:

- Examples include: **General Language Understanding Evaluation (GLUE) benchmark:** Tests a range of natural language understanding tasks.
- **SuperGLUE:** A more challenging successor to GLUE.
- **MMLU (Massive Multitask Language Understanding):** Tests knowledge across a wide range of academic disciplines.
- **HellaSwag:** Evaluates common-sense reasoning.
- **ARC (AI2 Reasoning Challenge):** Focuses on advanced reasoning.
- **Human Evaluations** (experts or crowd workers rate outputs)
- **Adversarial Testing** (intentionally tricky prompts to expose flaws)
- **Task-Specific Evals** (e.g., code generation tested on **HumanEval**)

LLM metrics

- **BLEU (Bilingual Evaluation Understudy):** Measures the similarity of machine-translated text to human reference translations.
- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** Measures the overlap of n-grams between a generated summary and reference summaries.
- **Perplexity:** Measures how well a language model predicts a sequence of words. Lower perplexity generally indicates a better model.
- **BERTScore:** Leverages pre-trained language models to assess semantic similarity between generated and reference text.

Evals framework: https://github.com/openai/evals
Simple evals: https://github.com/openai/simple-evals

LLM Leaderboards

https://huggingface.co/collections/open-llm-leaderboard/open-llm-leaderboard-best-models-652d6c7965a4619fb5c27a03

https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/

Medical LLM leaderboards: https://huggingface.co/spaces/bigscience/med_llm

Chatbot arena leaderboard: https://huggingface.co/spaces/lmarena-ai/chatbot-arena-leaderboard

Alpacaeval leaderboard: https://tatsu-lab.github.io/alpaca_eval/

Demo: with a hugging face model

https://github.com/srimugunthan/streamlit-chatbot/

Resource requirements for running LLM

GPU (Graphics Processing Unit): The Heart of LLM Setup

•**VRAM (Video RAM) is Crucial:** The more VRAM, the larger the models you can run and the better the performance.

•**Entry-Level Setups:**

- RTX 3060 (12GB VRAM) or RTX 4060 (8GB VRAM).
- Can handle smaller models (e.g., LLaMA 3.8B parameters), especially with quantization.

•**Mid-Range Setups:**

- RTX 3090 or RTX 4090 (both with 24GB VRAM).
- Can handle medium-sized models, potentially quantized versions of larger models (e.g., LLaMA 3.1 70B parameters).

•**High-End Setups:**

- Multiple NVIDIA A100s (80GB VRAM each).
- Required for massive models at full precision (e.g., LLaMA 3.1 405B parameters).

https://www.youtube.com/watch?v=oaV_8ZSFblg

https://www.youtube.com/watch?v=dmg5kZBCYck

https://www.youtube.com/watch?v=c9PL5RXyfU4

CPU (Central Processing Unit): Don't Overlook Its Importance

•Handles tasks like data pre-processing and model loading.

•**Smaller Models:** Intel Core i5 or AMD Ryzen 5 may suffice.

•**Larger Models/Intensive Workloads:** Consider Intel Core i9 or AMD Ryzen 9.

•**Largest Models:** May require server-grade CPUs.

RAM (Random Access Memory): Critical for Data Handling

•**Rule of Thumb:** At least twice as much RAM as your GPU's VRAM.

•**Recommendations:**

- 16GB RAM for GPUs with 8GB VRAM.
- 64GB or even 128GB RAM for setups running larger models.

•Ensures smooth data handling and prevents bottlenecks

Storage: For Fast Model Loading and Data Access

•**Primary Drive:** NVMe SSD with at least 1TB capacity for fast model loading and overall responsiveness. **Additional Storage:** Large HDD for storing multiple models and datasets,

GenAI essentials –part2

Outline

GenAI essentials –Part1

- GenAI-An introduction
- Basic concepts and building blocks
  - RNN
  - LSTM
  - Attention mechanism
  - Transformers
  - Diffusion models
- GPT models
  - Large language models
  - Image generation models
  - Multimodal models
- LLMs for development-landscape
- Implement a chatbot

GenAI essentials –Part2

- Prompt engineering
- Finetuning
- Quantization ,Model pruning and model distillation
- Langchain – Introduction
- RAG & Knowledge graphs
- LLM app design and deployment
- Costs

Agenda

1. **Prompt engineering**
   GenAI history, usecases , applications and learning path
2. **Finetuning**
   RNN,,Attention, Transformers
3. **Reducing model size –Quantization, Distillation, Model pruning**
   Architectural details of GPT
4. **Langchain framework**
   Overview of hugging face models
5. **RAG and knowledge graphs**
   Overview of hugging face models
6. **Architecting and deploying LLM application**
   Overview of hugging face models
7. **Costs**
   Overview of hugging face models

01

Prompt engineering

Prompt engineering

CRISPE (Context, Role, Instructions, Steps, Parameters, Examples).

* **Role/Persona:** Defining the AI's identity (e.g., "You are a professional writer").
•**Task/Objective:** Clearly stating the goal to be achieved.
•**Instructions:** Providing specific guidance on how to accomplish the task, including subcategories for detailed execution.
•**Reasoning Steps:** Explicitly outlining the thinking process, especially important for generative models like GPT-4.1 that are not inherently reasoning models (e.g., "Think step by step, considering...").
•**Output Format:** Specifying the desired structure of the AI's response (e.g., using XML).
•**Examples (Few-Shot Learning):** Providing demonstrations of the expected output to guide the model.
•**Context:** Supplying relevant information for the AI to work with.

Prompting benchmarks:

Needle in a Haystack" benchmark,

"fiction.livebench" benchmark

Tools and communities emerged: **PromptBase**, **Awesome-Prompting**,

Prompt optimization through **AutoPrompt**, **PromptPerfect**, and **LangChain**

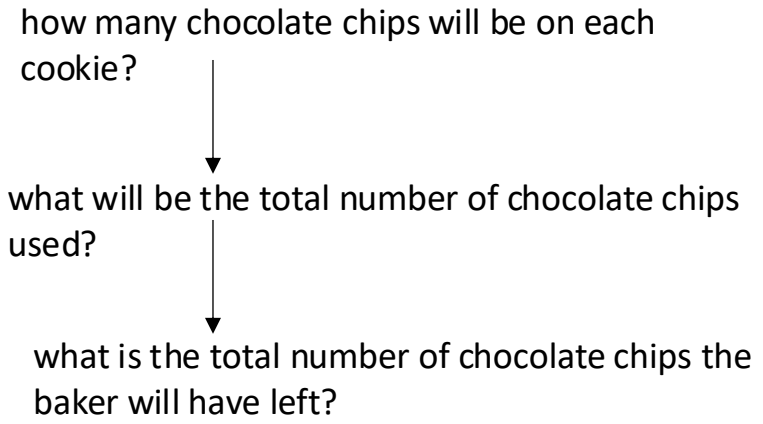
Prompting examples

Roleplaying prompt	**Customer Service Agent:** You are a friendly and helpful customer service agent for an online bookstore. A customer emails you saying their order hasn't arrived after two weeks. They provide their order number: #12345. How do you respond to this customer?
Few shot prompt(Provide 2–3 examples before the task)	**Topic Classification:** Article: New study shows promising results for cancer treatment. -> Topic: Health Article: Stock market experiences significant gains after tech earnings. -> Topic: Finance Article: Local band announces their upcoming summer tour dates. -> Topic: Music Article: Scientists discover a new species of deep-sea fish. -> Topic: ??
Zero shot prompt	**Prompt:** What is the sentiment of the following sentence? "This movie was absolutely fantastic!" **Prompt:** Write a short story about a robot who wants to learn how to paint.

Prompting examples

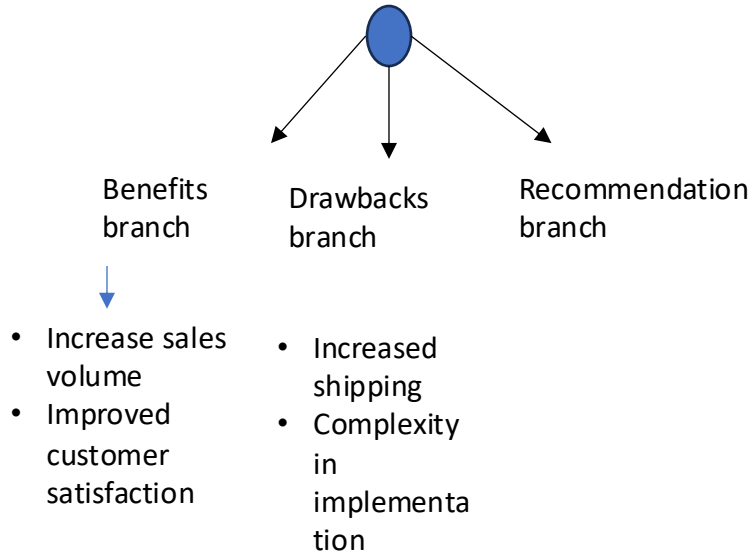
Chain of thought

Prompt:
A baker has 36 chocolate chips and wants to put them equally into 4 cookies. First, how many chocolate chips will be on each cookie? Then, if the baker decides to add 2 more chocolate chips to each cookie, what will be the total number of chocolate chips used? Finally, what is the total number of chocolate chips the baker will have left?



Tree of thought

Prompt:
A small online bookstore is trying to decide whether to offer free shipping on orders over ₹500 to boost sales. Analyze the potential benefits and drawbacks of this strategy. Then, considering these factors, recommend whether they should implement free shipping and justify your recommendation.



Prompting examples

Self consistency prompt	**Prompt:** "A train leaves Station A at 8:00 AM traveling towards Station B, which is 200 kilometers away. The train travels at a constant speed of 80 kilometers per hour. At the same time, another train leaves Station B traveling towards Station A at a constant speed of 120 kilometers per hour. At what time will the two trains meet?
Meta prompt	**Prompt :** You are an expert prompt engineer. Given a task description, your job is to generate an optimized prompt that will produce the most accurate and structured output from a language model like GPT. TASK: Summarize any article into 3 bullet points focusing on key business insights. Generate an optimal prompt for this task.

Prompt engineering: Best practices

1. Use Clear Delimiters
2. Use ALL CAPS for Emphasis or Headers
3. Label Sections Clearly
4. Anchor with Explicit Formats. Set expectations using formats (especially helpful for JSON, SQL, tables, etc.).
5. **Use Bullet Points or Numbered Steps**
6. Add Examples (Few-Shot Learning)
7. Use Role-based Instructions
8. Avoid Ambiguity

Importance of Delimiters:

•Delimiters are used within prompts to segment different sections, each serving a specific purpose.

•Common delimiters include Markdown (e.g., hashtags) and XML-like brackets.

•OpenAI's prompting guide suggests that XML has proven to be the most effective delimiter, especially for system prompts with extensive context and complexity.

•Interestingly, Anthropic's Claude models have historically performed well with XML, indicating a potential convergence on this format across leading model providers.

GPT-4 prompting- What is changed?

Prompting Varies by Model

GPT-4.1 boasts a substantial 1 million token context window.

Improved Instruction Following:

•Past prompting best practices often involved using unconventional tactics like all caps, threats, or bribes to get models to follow instructions.

•These newer models are better at adhering to straightforward instructions, making those older, more convoluted methods largely unnecessary.

Acceptance of Negating Terms:

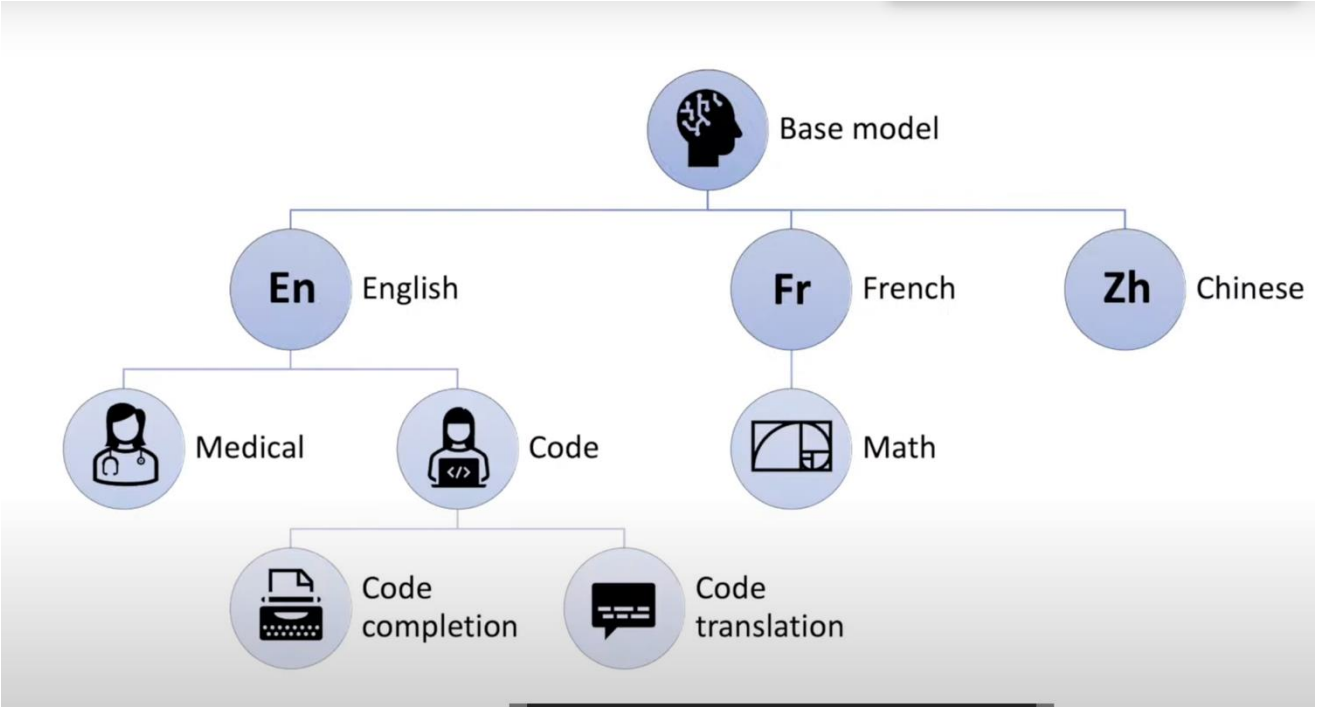
•Previous best practices often advised against using negative terms (e.g., "do not," "never," "not") in system prompts due to the risk of the AI inadvertently performing the prohibited action.

•With the enhanced instruction following of models like GPT-4.1 and Gemini 2.5 Pro, it is now more reliable to use negating terms effectively in prompts.

02

Fine tuning

Finetuning -intro



- Gets lot more data compared to a prompt to get context
- Allows specialization.
  - General purpose LLM -> LLM for medical domain

- Reduces hallucinations

- Greater control. Can deploy the finetuned model locally
- Can reduce cost of compute and latency per request
- Can incorporate guardrails

Finetuning -intro

Fine-tuning is a process where you take a pre-trained model and further train it on a smaller, specific dataset or task so it performs better in that narrower context.

Pretraining →

Base model

Base model → Finetuning → Fine tuned model

Finetuning again → Fine tuned model

Instruction Tuning, Task specific tuning, Domain specific tuning etc

- Full fine tuning: Finetuning changes the weights of the whole model
- Finetuning by transfer learning
- Parameter efficient fine tuning: fine-tuning **only a small subset** of the model's parameters while keeping most of the pre-trained parameters frozen

- Supervised Finetuning

  Instruction Tuning

- Non-Supervised Finetuning

  Self-supervised techniques , Masked language modelling, constastive learning

  Reinforcement Learning from Human Feedback (RLHF) aka Alignment

Instruction Finetuning

- Adapting a pre-trained model to follow task-specific instructions
- Teaches models to generalize from examples formatted as **instruction-input-output triplets**

- **Example:**
  - *Instruction:* "Translate to French."
  - *Input:* "Hello, how are you?"
  - *Output:* "Bonjour, comment ça va ?"

•**Benefits:**

- Improves zero-shot/few-shot task performance.
- Aligns model outputs with human intent.
- Reduces hallucinations by grounding responses in instructions.

•**Use Cases:**

- Chatbots, code generation, content moderation.

- Finetuning example: https://github.com/Ryota-Kawamura/Generative-AI-with-LLMs/blob/main/Week-2/Lab_2_fine_tune_generative_ai_model.ipynb

Parameter efficient Finetuning

Popular PEFT Techniques:

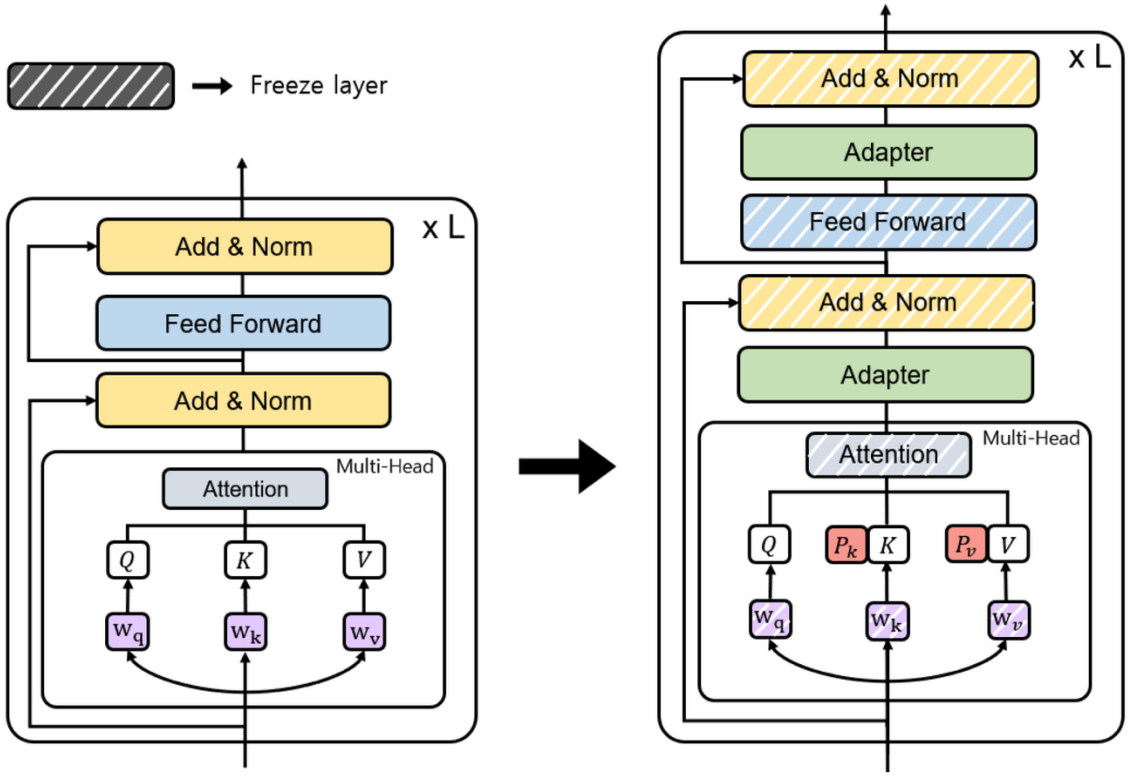
- **Adapter Layers** (e.g., Houlsby Adapters): Insert small trainable modules between layers.
- **LoRA (Low-Rank Adaptation)**: Decomposes weight updates into low-rank matrices.
- **Prefix/Prompt Tuning**: Learns task-specific soft prompts (virtual tokens) to condition the model.
- **BitFit**: Only fine-tunes the bias terms in the model

Parameter efficient finetuning methods for LLMs
https://arxiv.org/pdf/2312.12148

PEFT: Adapter based fine tuning

CORE IDEA:

- introduces small, trainable modules (called "adapters") into a pre-trained neural network
- keeping the original model weights **frozen**
- **Compress task-relevant information** into a smaller space
- **Reuse pre-trained features**



WHY IT WORKS

Neural Tangent Kernel (NTK) Theory: Suggests that over-parametrized models (like LLMs) can adapt to new tasks with **very few parameter changes**—adapters exploit this

WHEN IT WORKS AND WHEN IT DOESN'T:

Scenario	Works Well?	Reason	Better Alternatives
Task aligns with pre-training (e.g., text classification, QA)	✅ Yes	Adapters efficiently tweak existing features without catastrophic forgetting.	—
Small to medium-sized datasets (100-10k samples)	✅ Yes	Fewer trainable parameters reduce overfitting.	—
Multi-task learning (shared adapters)	⚠️ Sometimes	Works if tasks are related; may fail if tasks conflict.	Task-specific adapters (AdapterFusion), LoRA
High domain shift (e.g., medical, legal jargon)	❌ No	Adapters may lack capacity to bridge large gaps.	LoRA + domain-adaptive pretraining
Requires new model capabilities (e.g., Adapting a text-only model (e.g., BERT) to multimodal tasks (e.g., image-text alignment).new modalities, architectures)	❌ No	Adapters only modify features, not add new functions.	Full fine-tuning, Hybrid approaches

PEFT Lora

CORE IDEA:
performs a rank decomposition on the updated weight matrices

for the weight matrices we fine-tune, how expressive should the updates be in terms of matrix rank?

there exists a low Dimension re-parametrization that is as effective for fine-tuning. Larger the model the lower the intrinsic dimension

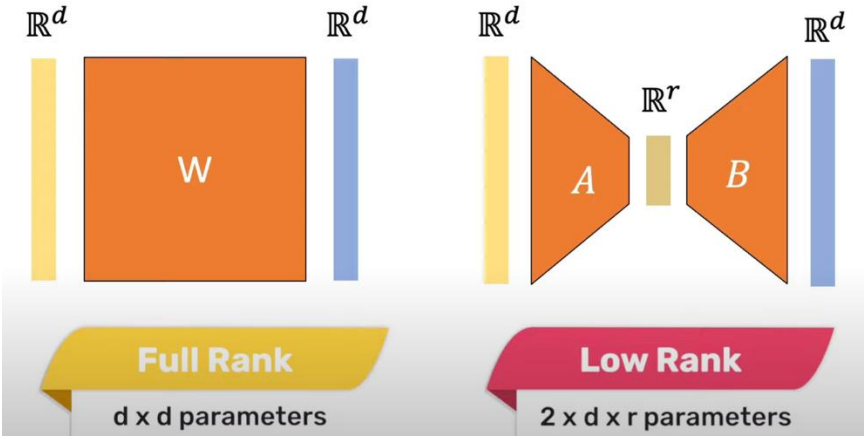
So Downstream tasks don't need to tune all parameters but instead can transform a much smaller set of weights to achieve a good performance

Decreases number of parameters used for finetuning

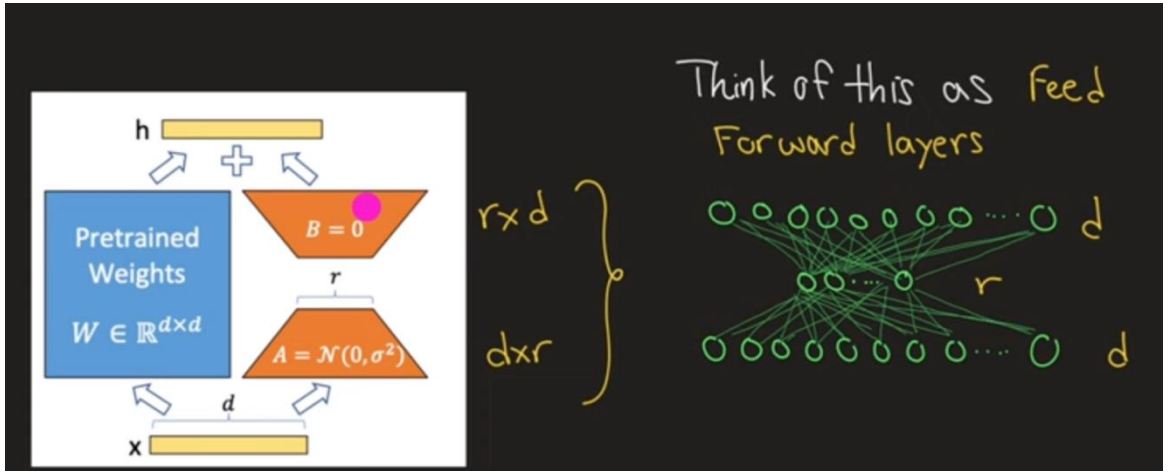
Speed up inference time predictions

https://www.youtube.com/watch?v=DhRoTONcyZE

https://www.youtube.com/watch?v=t509sv5MT0w

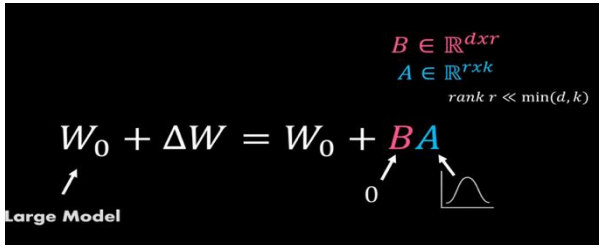


Rank: Number of linearly independent rows or columns



PEFT : How Lora works

- Instead of directly modifying the original weight matrices (W) of the pre-trained model, LoRA injects pairs of much smaller matrices (A and B) into selected layers of the network, typically within the attention mechanism.
- The original weight matrix (W) remains frozen and unchanged. The update to the original weight matrix (ΔW) is represented by the matrix multiplication of B and A . LoRA approximates weight updates (ΔW) using two smaller matrices (A and B) with a low-rank dimension (r ≪ original layer size).



For a weight matrix $W \in \mathbb{R}^{d\times k}$, the update is:

$$\Delta W = BA \quad \text{where} \quad B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$$

- During training, the forward and backward passes only involve the much smaller matrices A and B.
- **Reduced Trainable Parameters**: For rank r*r*, LoRA adds r×(d+k) parameters per layer instead of d×k

Introduces Singular Value Decomposition (SVD) as the mathematical basis: any matrix A can be decomposed into U * S * V^T, where S is a diagonal matrix of singular values.

Singular values are the square roots of the eigenvalues of AA^T.

Eckart-Young theorem proves the applicability of low-rank approximation.

LoRA truncates the SVD of the high-rank ΔW matrix to obtain a low-rank approximation.

This is done by setting the smallest singular values in the diagonal matrix S to zero, keeping only the top k largest singular values.

This results in approximating the high-rank ΔW matrix with the product of two lower-rank matrices, A and B ($\Delta W \approx BA$).

PEFT Lora demo

- LoRA demo: https://colab.research.google.com/drive/1peGphxJFJdxG55y69djU3PlC6biB_SKK?usp=sharing

Soft prompt based tuning

- Soft prompt-based fine-tuning is a parameter-efficient method for adapting large pre-trained language models (PLMs) to downstream tasks without modifying the bulk of the model's weights. Instead of fine-tuning all parameters or using discrete, human-engineered prompts, soft prompts are **learnable continuous embeddings** that are prepended to the input and optimized during training.

Example:

For sentiment analysis, instead of:

•Hard prompt: "Review: {text}. Sentiment: [MASK]",

•Soft prompt: [trainable_emb_1, ..., trainable_emb_k] + [text_embeddings].

The model learns to map these embeddings to the correct sentiment without explicit textual cues.

Variants/Extensions:

•**Prompt Tuning**: The basic form (Lester et al., 2021), where only soft prompts are trained.

•**Prefix Tuning** (Li & Liang, 2021): Extends soft prompts to intermediate layers.

•**SPoT (Soft Prompt Transfer)**: Transfer learned soft prompts across tasks.

https://medium.com/ai-singapore/teaching-opt-to-paraphrase-through-soft-prompt-tuning-4c1a1e852cb8

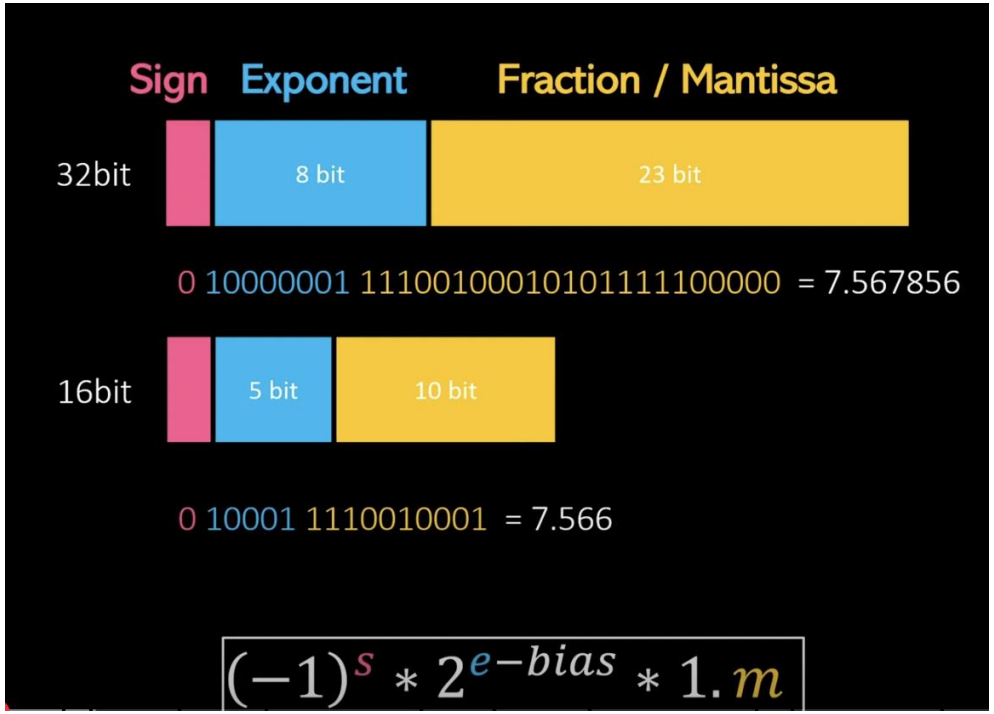
03

Quantisation, Distllation, Model pruning

What is quantisation

a single model checkpoint for the 175 billion parameter variant is 1 Terabyte large

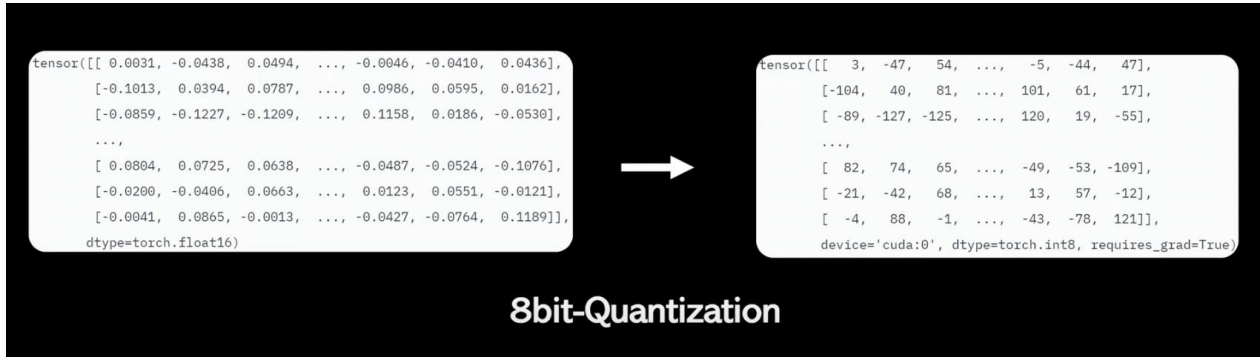
$$Model\ size = size_{datatype} * num\ weights$$



Reduce the size of the model by lowering the precision

Deep learning models consist of weights and activations, typically in float32 format.

Lower precision formats include float16, bfloat16, int8, and int4 (more popular).



Different types of quantisation

Based on How the Mapping to Lower Precision is Done::

1. **Uniform Quantization:**
2. Non-Uniform Quantization:
3. Absolute Max Quantization:
4. Affine (or Zero-Point) Quantization

Based on What is Quantized:

1. Weight Quantization:
2. **Activation Quantization:**
3. Hybrid Quantization:

Based Based on the Target Data Type::

1. **Integer Quantization:**
2. Floating-Point Quantization:
3. Binary Quantization

Based on granularity::

1. Per-Tensor Quantization:
2. Per-Channel Quantization
3. Per-Group Quantization:

Different types of quantisation

Types of Quantization:

•**Downcasting:** Straightforward conversion of float32 weights to float16 or bfloat16 without other techniques.

- **Float32:** 32 bits (1 sign, 8 exponent, 23 mantissa), wide dynamic range, high accuracy, suitable for training.
- **Float16:** 16 bits (1 sign, 5 exponent, 10 fraction), reduced dynamic range, faster computations, good for low-power devices, potential accuracy drop in complex tasks.
- **Bfloat16:** 16 bits (1 sign, 8 exponent, 7 fraction), same exponent range as float32 but lower fraction precision. A good balance for reduced memory (half of float32) while preserving accuracy, especially for LLMs.

Integer Weight Only Quantization: Further reduction to 8-bit or 4-bit integers.

•Only weights are quantized; activations often remain at higher precision (e.g., 16-bit).

•Uses scale and zero point to map floating-point ranges to integer ranges.

•**Per-tensor quantization:** Uses a single scale and zero point for the entire tensor.

•**Per-channel quantization:** Computes scale and zero point for each channel individually (better accuracy, slightly more computation).

•**Symmetric quantization:** Zero point is 0.

•**Asymmetric quantization:** Zero point is shifted.

Quantisation demo

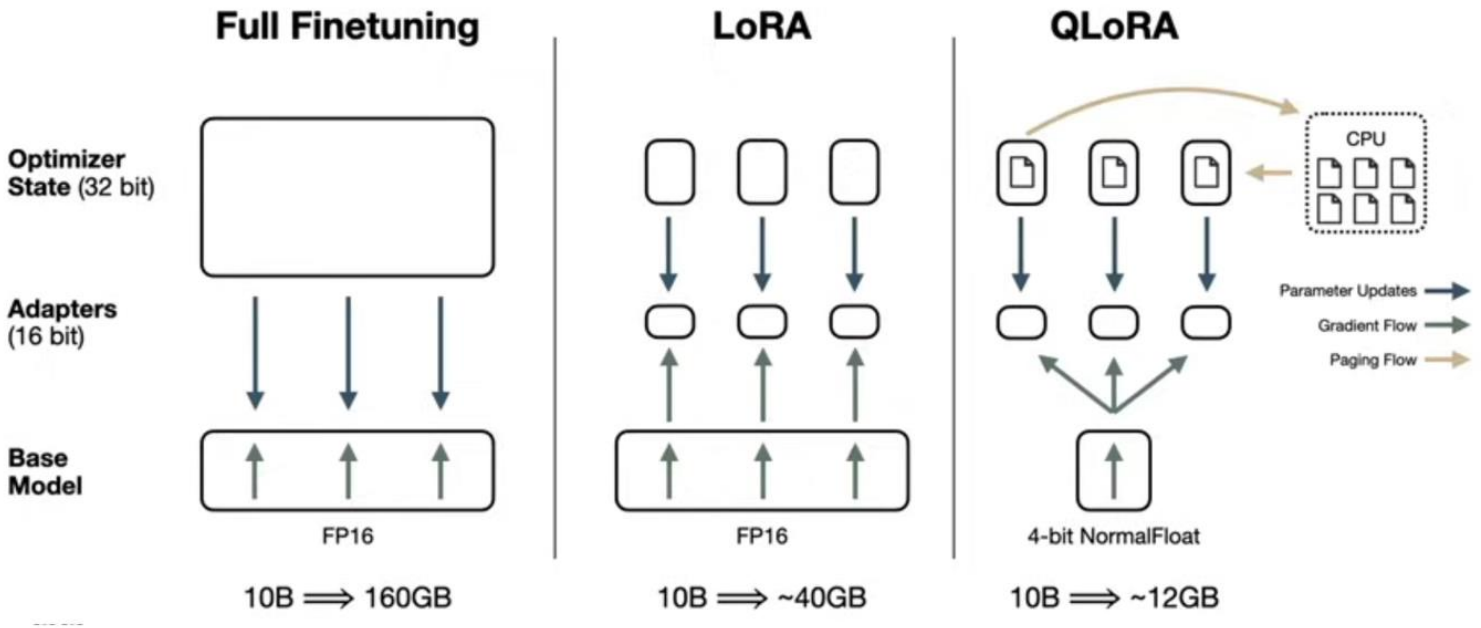
Demo:

https://github.com/r4ghu/llm-quantization/blob/main/LLM%20Quantization.ipynb

Qlora

•QLoRA achieves its memory efficiency through three main innovations: 1) 4-bit Normal Float (NF4) data type 2) Double Quantization 3) Paged Optimizers

4-bit NormalFloat (NF4) Quantization: This is a crucial innovation in QLoRA. It quantizes the frozen pre-trained weights of the LLM to 4 bits using a data type called NormalFloat (NF4).



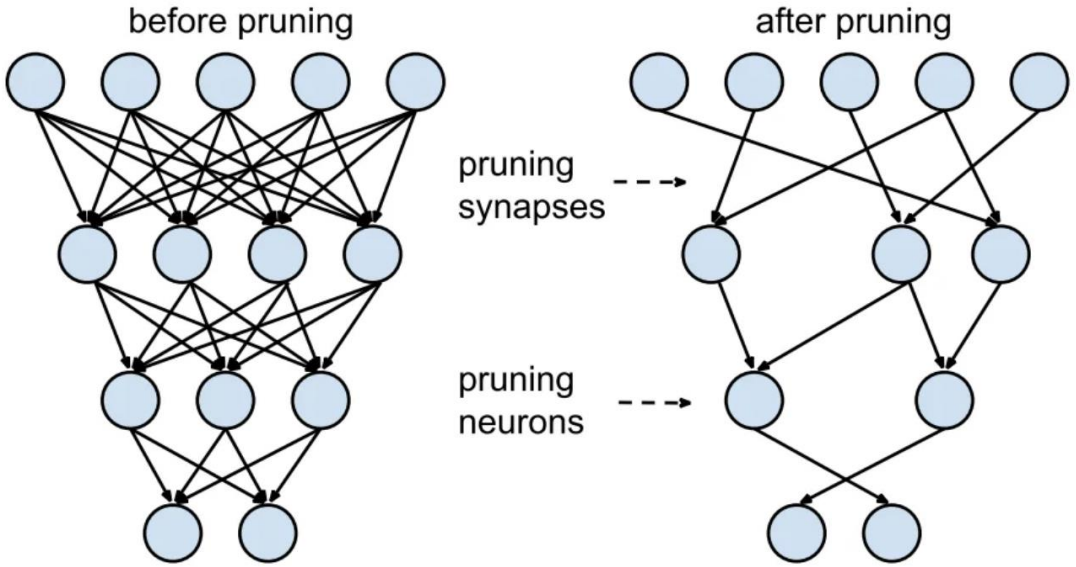
Double Quantization: To further save memory, QLoRA quantizes the quantization constants themselves. These constants are used in the initial 4-bit quantization process.

Paged Optimizers: This technique addresses memory spikes that can occur during the optimizer step in training, especially with large models. QLoRA uses NVIDIA's unified memory feature to allocate paged memory for the optimizer states

High-Precision Computation: Although the pre-trained weights are stored in a 4-bit quantized format, QLoRA performs computations (like matrix multiplications involving the LoRA adapters) in a higher precision, typically 16-bit (like BF16 or FP16).

•QLoRA paper: https://arxiv.org/abs/2305.14314

Model pruning Demo



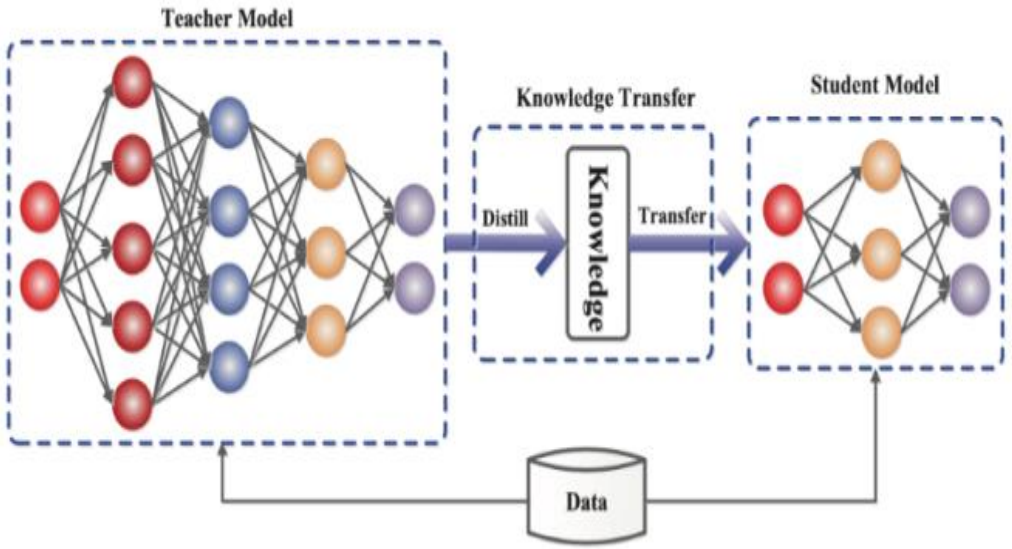
Model pruning involves removing less important connections (weights) or even entire neurons, effectively creating a smaller and more efficient model.

"The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks" (Frankle and Carbin, 2019) demonstrates that neural networks tend to have a specific subset of parameters that are essential for prediction

Model pruning can be performed before the weights are quantised, and the effects of both of these memory reduction techniques will stack.

Pruning Method	Criteria for Pruning
1. Magnitude-Based	Remove neurons/weights with smallest L1/L2 norm.
2. Activation-Based	Prune neurons with lowest average activations.
3. Gradient-Based	Remove neurons with smallest gradient contributions.
4. Taylor Expansion	Approximate loss change using 1st-order Taylor.
5. Regularization (L1)	Train with L1 sparsity, then prune near-zero weights.

Model Distillation Demo



https://www.deepchecks.com/glossary/model-distillation/

STEPS:

1) Train the Teacher Model: Train a high-performance, complex model (teacher) on the training dataset.

2) **Generate Soft Labels (Logits) from Teacher**

Instead of hard labels (one-hot encoded), use **soft labels** (probability distributions) with temperature scaling:

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

zi = logits, T*T* = temperature (hyperparameter).

3. Train the Student Model Using Distillation Loss

- **Distillation Loss (KL Divergence)**: Aligns student's soft predictions with teacher's.

$$\mathcal{L}_{\text{distill}} = T^2 \cdot \text{KL}(q_{\text{teacher}} \| q_{\text{student}}) \qquad \mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}_{\text{hard}} + (1-\alpha) \cdot T^2 \cdot \mathcal{L}_{\text{soft}}$$

4) Fine-Tune the Student Model

https://medium.com/@nminhquang380/knowledge-distillation-explained-model-compression-49517b039429

DEMO: Qlora, model pruning,

Qlora demo: https://colab.research.google.com/drive/1AErkPgDderPW0dgE230OOjEysd0QV1sR?usp=sharing

Model pruning demo: https://github.com/peremartra/Large-Language-Model-Notebooks-Course/tree/main/6-PRUNING

Model diistillation demo https://github.com/peremartra/Large-Language-Model-Notebooks-Course/tree/main/6-PRUNING

https://www.kaggle.com/code/neerajmohan/model-compression-using-knowledge-distillation

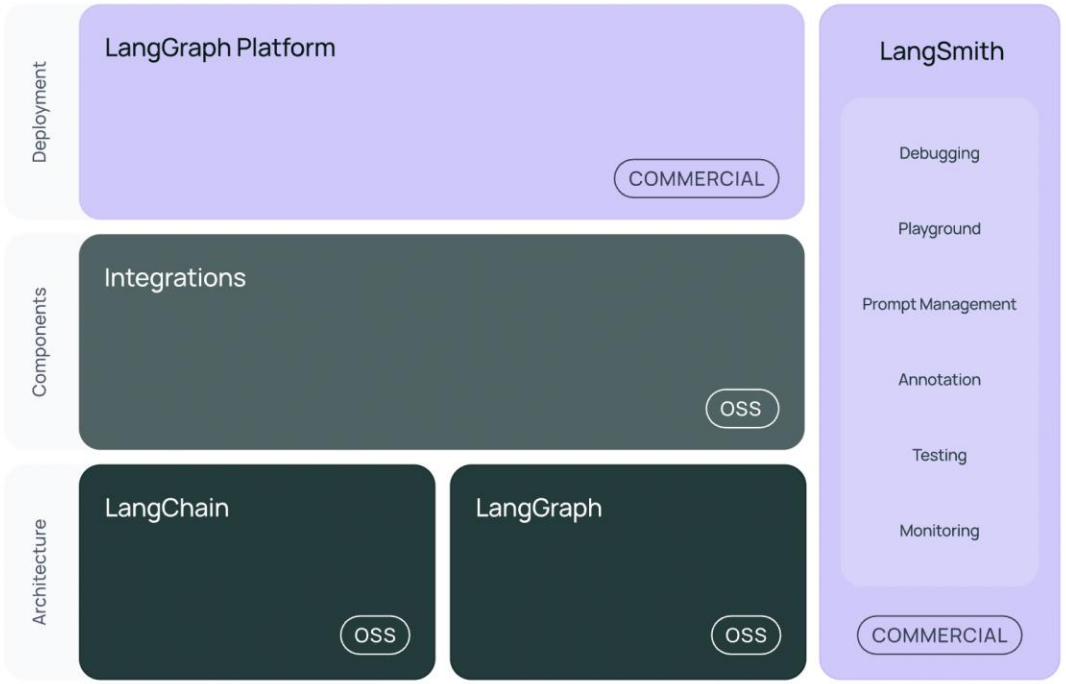
https://www.youtube.com/watch?v=6l8GZDPbFn8

https://www.youtube.com/watch?v=XpoKB3usmKc

04

Langchain

What is Langchain



- An open-source framework for developing applications powered by LLMs.

- toolkit for orchestrating LLM workflows.
- Simplifies the process of integrating LLMs with other data sources and tools.
- Enables the creation of more context-aware, reasoning, and interactive applications.

Building Blocks:

Modular components (Models, Prompts, Chains, Agents, Memory).

•**Models:** Interfaces to various LLMs and Chat Models (e.g., OpenAI, Google Gemini).

•**Prompts:** Templatize and manage inputs to LLMs for consistent and effective interactions.

•**Chains:** sequences of calls to LLMs or other utilities to perform a specific task.

•**Agents:** Use LLMs as a reasoning engine to decide which actions to take based on input, using tools.

•**Memory:** Giving chains and agents the ability to remember previous interactions (statefulness).

•**Indexes:** Structuring documents to facilitate interaction with LLMs, particularly for retrieval.

•**Retrievers:** Systems for fetching relevant documents from an Index based on a query.

•**Output Parsers:** Structuring the output from LLMs into desired formats (e.g., JSON).

https://python.langchain.com/docs/concepts/

Langchain : chatmodels

- High-level abstraction for LLMs optimized for dialogue.
- Wraps models like OpenAI ChatGPT, Anthropic Claude, etc.
- Offers a consistent API to interact with various ChatModel providers. Simplifies managing conversational flow and memory.

```python
from langchain.chat_models import ChatOpenAI
chat = ChatOpenAI(model_name="gpt-4")
```

•**Strategies for Effective Prompting:**
• **System Messages:** Setting the persona, constraints, and goals for the AI.
•**Few-Shot Prompting:** Providing examples of input/output pairs in the message history.
•**Clear Instructions:** Being explicit about the desired output format and task.
•**Iterative Refinement:** Adjusting prompts based on model responses.

```python
messages = [
    SystemMessage(content="Solve the following math problems"),
    HumanMessage(content="What is 81 divided by 9?"),
]
```

Langchain : Prompt templates

- **What is a Prompt Template?** A pre-defined structure for generating prompts.
- Allows for dynamic insertion of values into placeholders.
- Separates the fixed instructions from the variable inputs.

Make prompts **parameterized and testable**

```
from langchain.prompts import PromptTemplate
template = "Tell me a joke about {topic}."
prompt_template = PromptTemplate.from_template(template)
```

Few-Shot Prompt Templates

```
examples = [{"input": "happy", "output": "sunshine"}, ...]
prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=PromptTemplate(...),
    prefix="Convert words to emojis:",
    suffix="Input: {input}\nOutput:"
)
```

Example Applications with Prompt Templates

- **Automated Data Documentation:**
  - Templates to generate descriptions for datasets or features based on metadata.
- **Generating Code Snippets:**
  - Templates for creating boilerplate code for analysis, visualization, or model training based on user requirements.
- **Summarizing Research Papers or Reports:**
  - Templates to extract key findings or generate executive summaries.
- **Creating Synthetic Data:**
  - Templates to define the structure and characteristics of desired synthetic data.

https://github.com/Ryota-Kawamura/LangChain-for-LLM-Application-Development/blob/main/L1-Model_prompt_parser.ipynb

Langchain : Chains

- LangChain allows you to chain together different components.

Example:

- Receive a user query.
- Use a Retriever to find relevant documents from your data (Index).
- Pass the query and retrieved documents to an LLM via a Prompt.
- Use an Output Parser to format the LLM's response.

LangChain Expression Language (LCEL)

A declarative way to compose chains easily.
Provides a clear and concise syntax for building complex workflows.

Offers benefits like:

Streaming,Async capabilities, Parallel execution, Seamless LangSmith tracing

```
from langchain_core.runnables import RunnablePassthrough, RunnableParallel
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

prompt = ChatPromptTemplate.from_template(
   """Answer the question based only on the following context:
{context}

Question: {question}
"""
)

# 4. Define the Output Parser
output_parser = StrOutputParser()

# 5. Construct the LCEL Chain
# This part handles passing the original query to the retriever
# and also making it available for the prompt along with the retrieved documents.
rag_chain = (
   RunnableParallel(
      {"context": retriever, "question": RunnablePassthrough()}
   )
   | prompt
   | llm
   | output_parser
)

# How to use the chain:
# result = rag_chain.invoke("Your user query here")
# print(result)
```

https://github.com/Ryota-Kawamura/LangChain-for-LLM-Application-Development/blob/main/L3-Chains.ipynb

Langchain : memory

- A set of components designed to persist and manage conversational state between turns.
- Allows LLMs to "remember" past interactions. Inject relevant past messages into the current prompt.

```
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferMemory
from langchain.llms import OpenAI

memory = ConversationBufferMemory()
chain = ConversationChain(llm=OpenAI(), memory=memory)

chain.run("Hello, I'm Alice.")
chain.run("What's my name?")
```

The LLM remembers that the user is Alice.

ConversationBufferMemory:

•**Simplest form:** Stores all raw messages in a buffer.

ConversationBufferWindowMemory:

•**Fixed Window:** Stores only the last k interactions (messages or turns).

ConversationSummaryMemory:

•**Condenses History:** Uses an LLM to summarize past conversations.

ConversationSummaryBufferMemory:

•**Hybrid Approach:** Stores recent messages in a buffer and summarizes older ones.

Entity Memory: Remembers specific entities (e.g., names, preferences).

VectorStoreRetrieverMemory (Vector Store Memory):

•**Embeddings & Retrieval:** Stores message embeddings in a vector store and retrieves relevant past messages based on semantic similarity to the current input.

https://github.com/Ryota-Kawamura/LangChain-for-LLM-Application-Development/blob/main/L2-Memory.ipynb

Langchain : Indexes and retrievers

What are Indexers?

•**Definition:** Tools to structure and store data for efficient querying.

•**Key Functions:**

- Convert documents into searchable formats (e.g., vector embeddings).
- Popular methods: Vector stores (FAISS, Pinecone), SQL databases, etc.

•**Example:** Indexing PDFs or web pages for later retrieval.

What are Retrievers?

•**Definition:** Components that fetch relevant data from indexed sources.

•**Key Features:**

- Use queries to return contextually relevant documents.
- Types: Vector store retrievers, TF-IDF, SVM, etc.

•**Example:** Fetching top-3 relevant docs for a user query.

1.Indexing Phase:

1. Documents → Split → Embeddings → Stored in Vector DB.

2.Retrieval Phase:

1. User query → Embedding → Similarity search → Top-K results.

Raw Documents → Indexer → Vector Store → Retriever → LLM Prompt → Output

https://github.com/langchain-ai/langchain/blob/master/docs/docs/tutorials/retrievers.ipynb

Langchain : Output parsers

LLMs return **unstructured text** by default.

- **What are Output Parsers?**
  - Tools to convert raw LLM output into structured formats (e.g., JSON, Python objects).
- **Why Use Them?**
  - Ensure consistency in responses.
  - Integrate with other LangChain components (chains, agents).

Types of Output Parsers

1.Structured Output Parsers → JSON/dictionaries.

2.List Parsers → Extract lists (e.g., ["item1", "item2"]).

3.Datetime Parsers → Standardize dates/times.

4.Enum Parsers → Map responses to predefined options.

5.Custom Parsers → User-defined formats.

6.Pydantic Parsers: Validate against Pydantic models.

```python
from pydantic import BaseModel
from langchain.output_parsers import PydanticOutputParser

class Product(BaseModel):
    name: str
    price: float

parser = PydanticOutputParser(pydantic_object=Product)
```

https://github.com/sugarforever/LangChain-Tutorials/blob/main/LangChain_Output_Parsing.ipynb

Langchain : Callbacks

LangChain applications involve multiple steps: LLM calls, chain executions, tool usage

•**Lack of Visibility:** What exactly is happening *inside* a complex chain?
•Which prompts are being sent?
•What are the intermediate outputs?
•How long did each step take?
•What's the token usage for each component?

A callback is a **hook** into the lifecycle of LangChain operations.
Triggered during:
•Chain start/end
•LLM call start/end
•Tool usage
•Errors and token usage

- A callback can be taken as an event that is integrated with a function and is triggered at different execution points within the function. .

- **LLM Events:**
  - on_llm_start: Before an LLM call.
  - on_llm_end: After an LLM call (successful).
  - on_llm_error: If an LLM call fails.
  - on_llm_new_token: For streaming responses (each new token).
- **Chain Events:**
  - on_chain_start: Before a chain execution.
  - on_chain_end: After a chain execution.
  - on_chain_error: If a chain execution fails.
- **Tool Events:**
  - on_tool_start: Before an Agent uses a tool.
  - on_tool_end: After a tool execution.
  - on_tool_error: If a tool execution fails.
- **Retriever Events:**
  - on_retriever_start: Before a retriever fetches documents.
  - on_retriever_end: After a retriever fetches documents.
- **Agent Events:**
  - on_agent_action: When an agent decides on an action.
  - on_agent_finish: When an agent has completed its thought process.

e, memory interactions, retrievals.

Observability using LLMonitor :
https://github.com/langchain-ai/langchain/blob/master/docs/docs/integrations/callbacks/llmonitor.md

Langchain : Agents

An **agent** is an LLM-powered entity that:
•Uses tools (e.g., search, calculator, APIs)
•Decides **what to do and when**
•Takes multi-step actions to complete a task

•LangChain agents add reasoning + tool use to LLMs
•AI systems that dynamically decide actions using LLMs as a "brain".
•

Popular Agent Types

•**Zero-shot ReAct:** Uses ReAct framework (Reason + Act).

•**Conversational:** Optimized for chat-based interactions.

•**Self-ask with Search:** Answers sub-questions via search.

•**Custom Agents:** Tailored to specific domains.

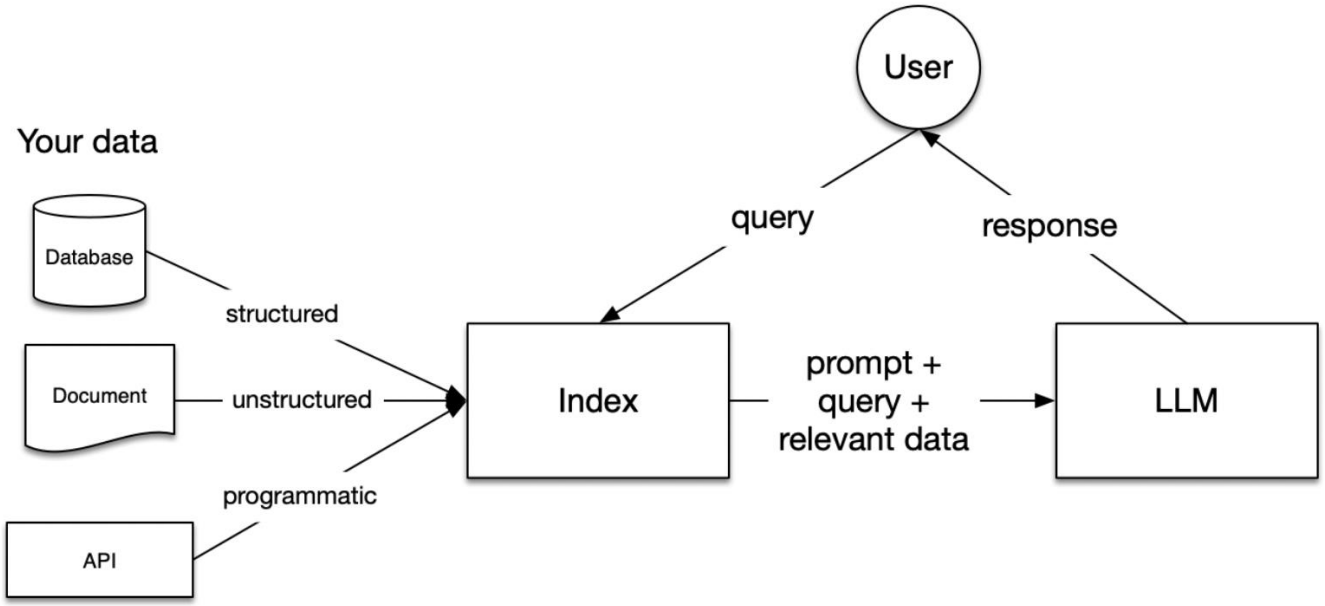
Feature	Chain	Agent
Control Flow	Fixed / Predefined	Dynamic / LLM-decided
Tool Usage	None or limited	Flexible and dynamic
Use Case	Simple tasks	Complex, multi-step reasoning

https://github.com/Ryota-Kawamura/LangChain-for-LLM-Application-Development/blob/main/L6-Agents.ipynb

05

RAG and knowledge graph

What is RAG (Retrieval augmented generation)



Retrieve: Given a user query, find the most relevant information from a vast knowledge base.
Augment: Inject this retrieved information into the LLM's prompt.
Generate: The LLM then uses both the original query and the provided context to formulate a grounded and accurate response.

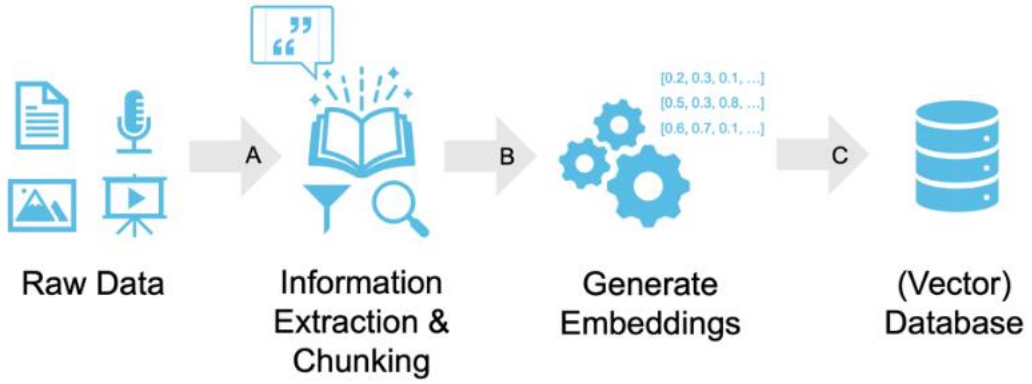
RAG vs finetuning:

RAG (Retrieval-Augmented Generation)	Finetuning
Provide LLM with *external, up-to-date, or domain-specific knowledge* for grounded responses.	Adapt LLM's *style, tone, or specific task performance* using new examples.
External, dynamic knowledge base (vector database, documents, etc.).	New knowledge is "baked into" the model's weights through training.

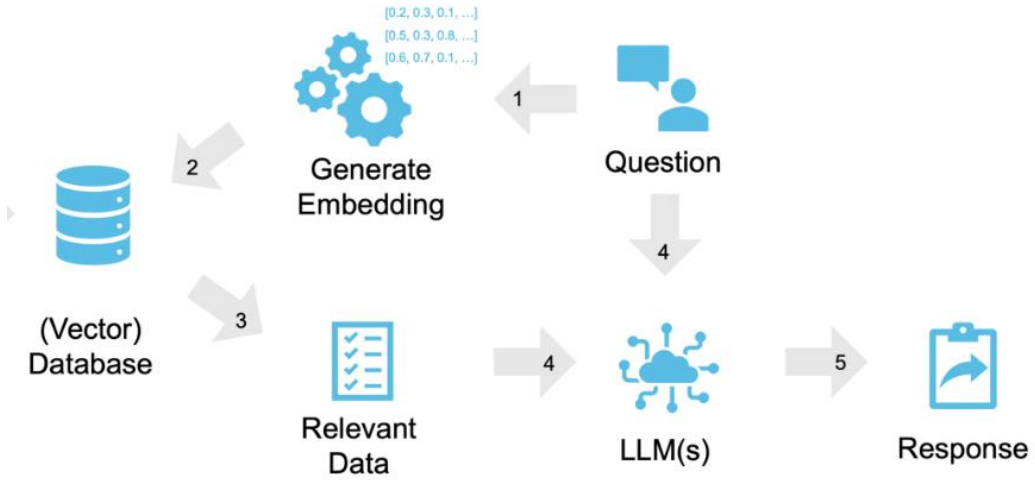
How RAG reduces Hallucinations:

- By including specific context in the prompt, RAG limits the LLM's "imagination,"
- Limits LLM responses to context from retrieved documents
- **Limits reliance on parametric memory**: It reduces dependence on the model's internal (and possibly outdated) training data.

RAG pipeline



Prepare Knowledge Base: Collect all external data.
Chunk & Embed Data: Break data into chunks, create numerical embeddings.
Store Embeddings: Save embeddings in a vector database.



RAG Pipeline Steps

1.Query Input: User question/prompt.

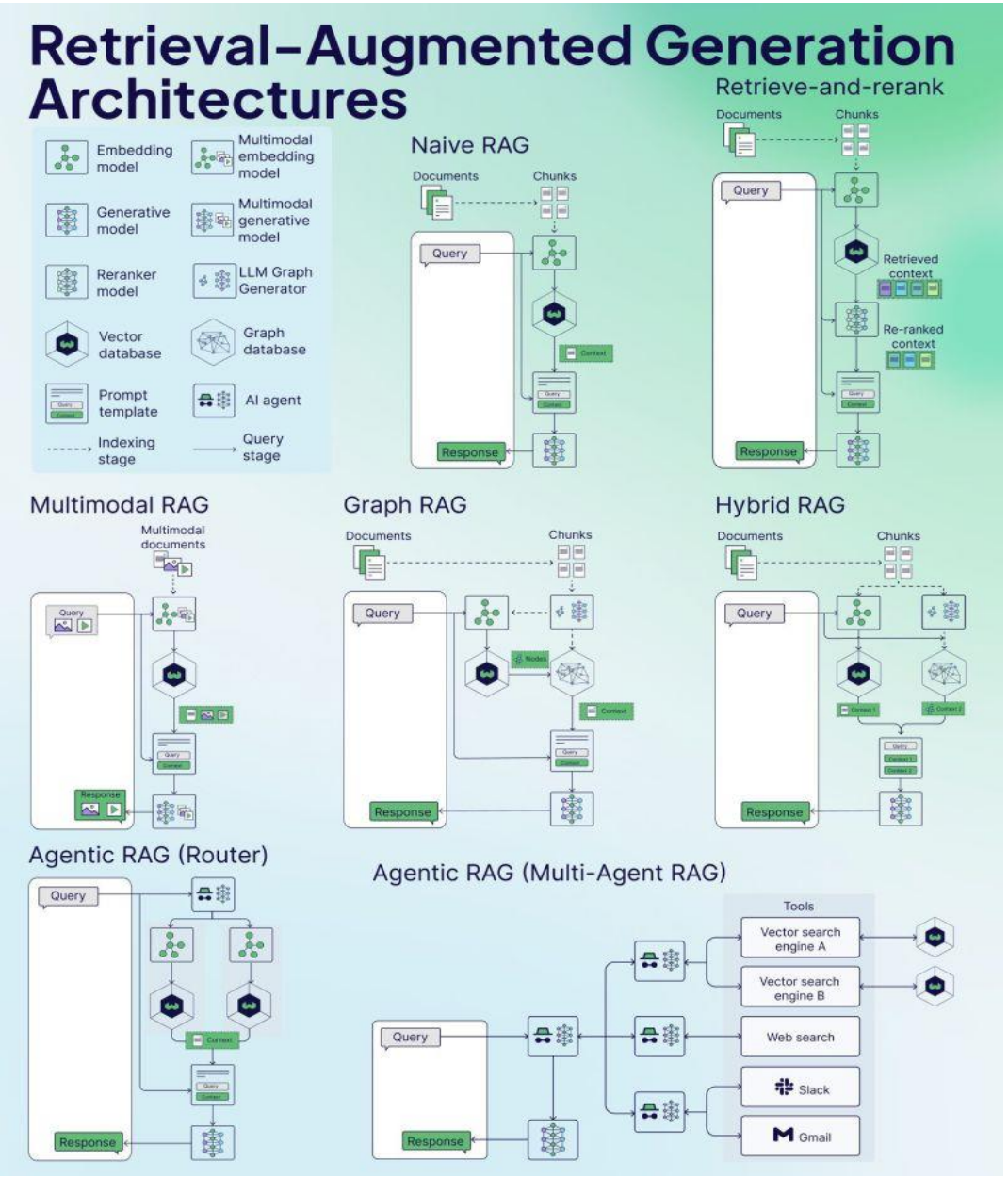
2.Retrieval Phase: Fetch relevant documents/chunks.

3.Augmentation: Combine query + retrieved context.

4.Generation: LLM produces final output.

RAG evaluation: https://medium.com/decodingml/the-engineers-framework-for-llm-rag-evaluation-59897381c326
https://cratedb.com/use-cases/chatbots/rag-pipelines

Types of RAG



✅ **Standard RAG** – The foundational approach. Retrieves relevant context and generates responses—simple but powerful synergy.

🔁 **Corrective RAG** – Iteratively refines outputs using feedback, reducing errors and improving factual consistency.

🔮 **Speculative RAG** – Generates multiple candidate responses in parallel, then selects or combines the best options.

🔗 **Fusion RAG** – Intelligently blends information from diverse sources, leveraging advanced ranking and merging techniques.

🕹️ **Agentic RAG** – Acts autonomously, dynamically adjusting retrieval and generation based on real-time needs.

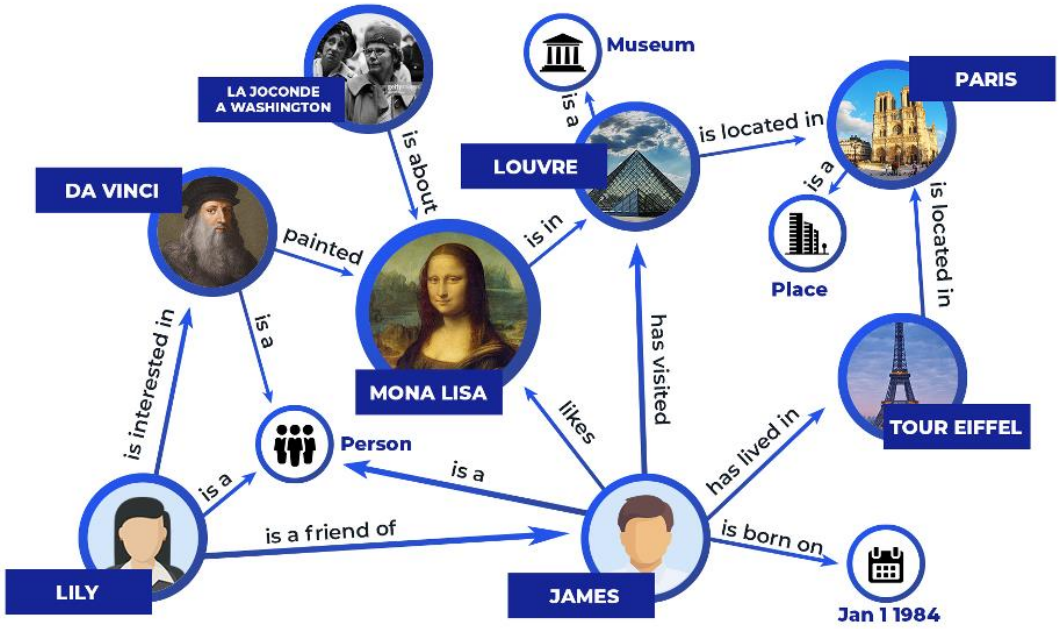
🧠 **Self RAG** – Critiques and refines its own outputs, enhancing accuracy through self-evaluation.

🕸️ **Graph RAG** – Leverages structured knowledge graphs to navigate and reason over complex relationships.

🧩 **Modular RAG** – Flexible architecture with interchangeable components for tailored retrieval and generation.

Knowledge graphs

A **Knowledge Graph** is a structured representation of knowledge that connects entities (e.g., people, places, things) through relationships in a graph format.



Knowledge graph vs RAG:

1.Use a KG for structured reasoning.

2.Use RAG to fill gaps with unstructured data.

•**Structure**:

- Nodes = Entities (e.g., "Albert Einstein," "Theory of Relativity")
- Edges = Relationships (e.g., "developed," "contributed to")
- Often stored in graph databases (e.g., Neo4j) or semantic triples (subject-predicate-object).

A) **LLM-Augmented Knowledge Graph** Use LLMs to **build**, **enrich**, or **query** a knowledge graph. **Example**: Extract entities/relations from text using an LLM, then store them in a KG.

B) Knowledge Graph-Augmented LLM: Use a KG to **improve LLM outputs** (e.g., for factual accuracy). **Example**: Retrieve KG facts before generating an answer (similar to RAG but with structured data).

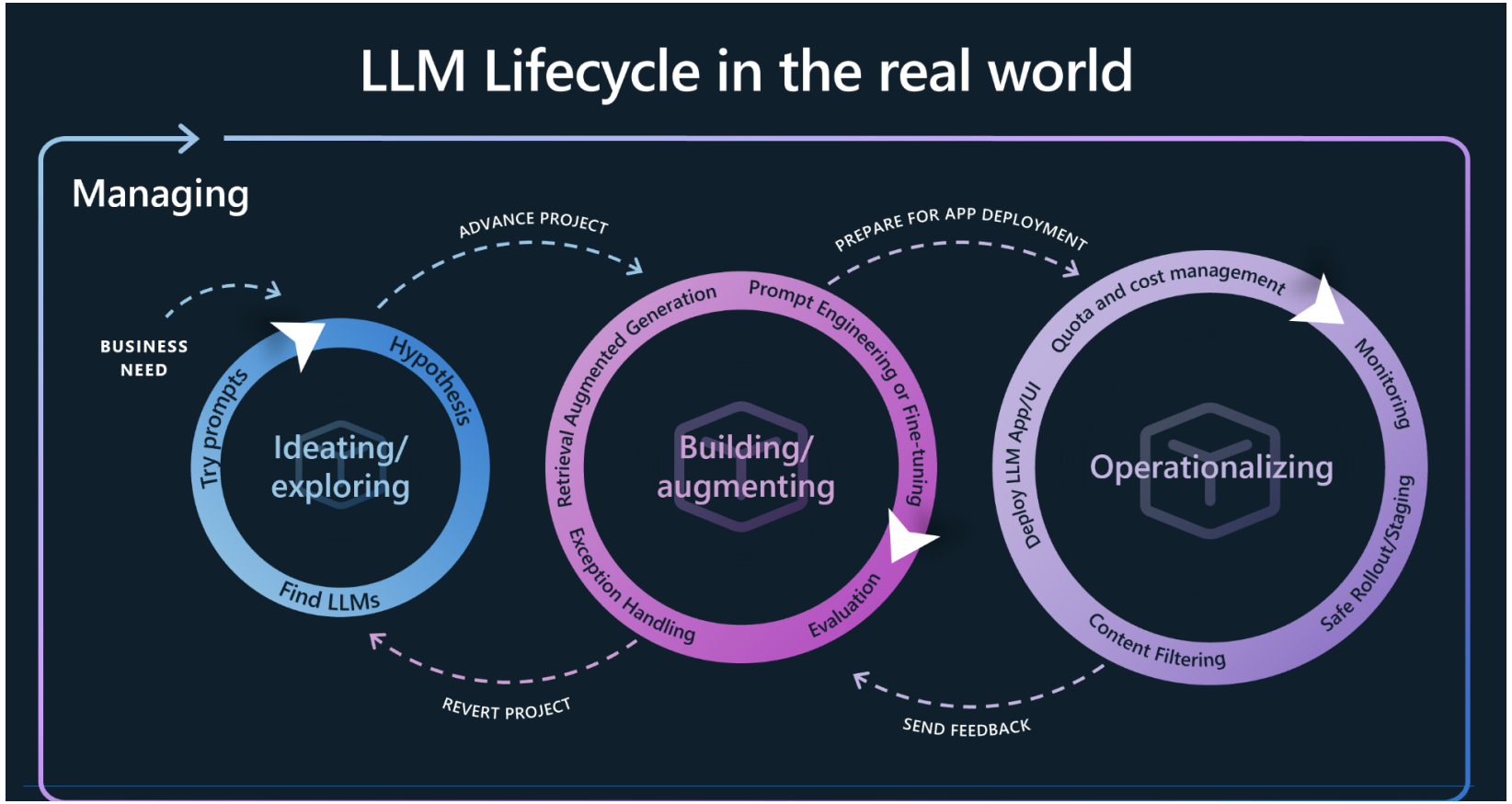
DEMO: RAG and Knowledge graph

RAG in Langchain:	https://medium.com/@vipra_singh/building-llm-applications-introduction-part-1-1c90294b155b
Deploy Langchain apps on google cloud	https://codelabs.developers.google.com/codelabs/build-and-deploy-a-langchain-app-on-cloud-run#0
Knowledge graph demo	https://github.com/tanchongmin/TensorFlow-Implementations/blob/main/Tutorial/LLM%20with%20Knowledge%20Graphs.ipynb https://medium.com/data-science/how-to-convert-any-text-into-a-graph-of-concepts-110844f22a1a

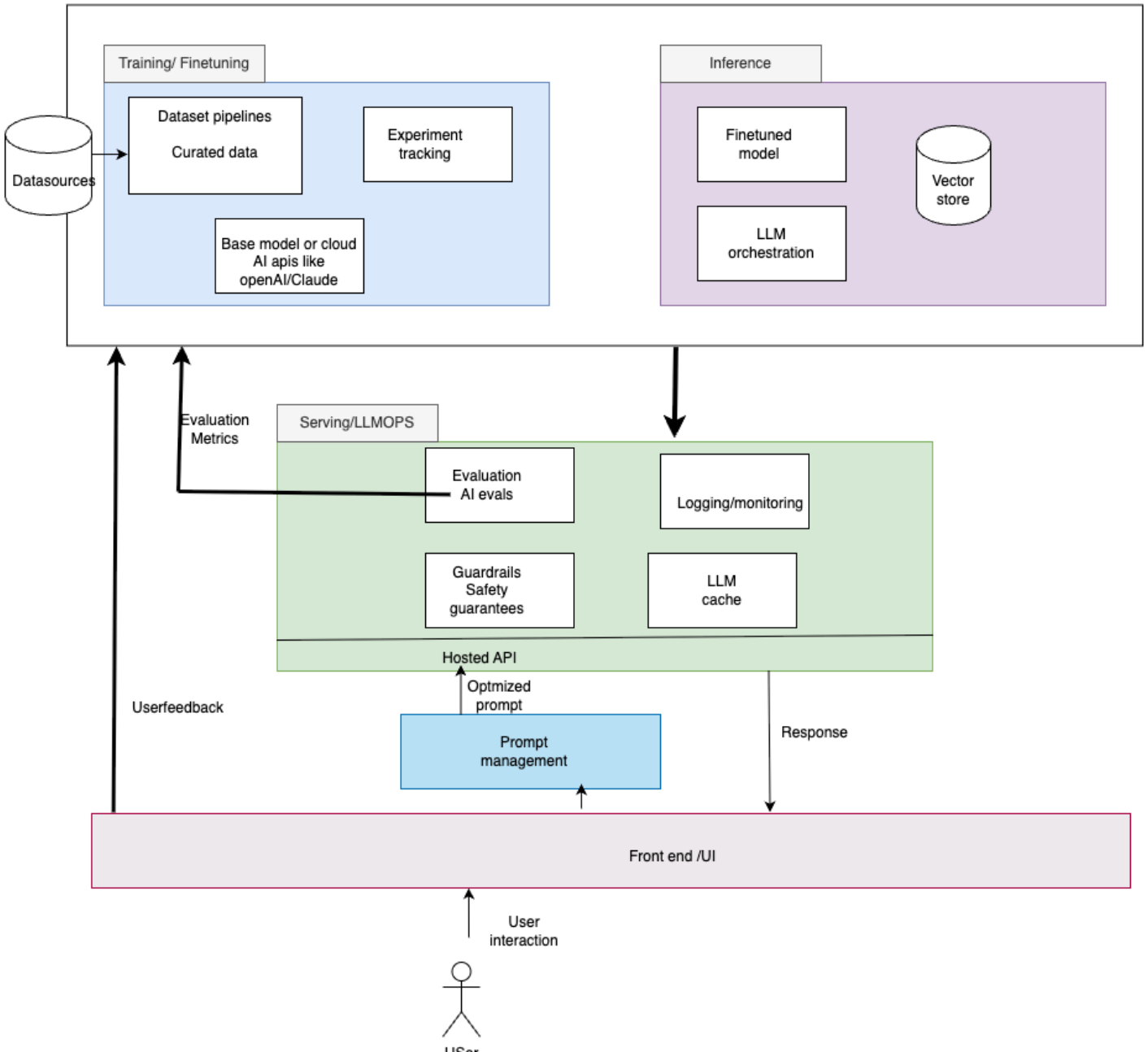
06

LLM app-Design and deployment

LLM app lifecycle



LLM app architecture



LLM app Tools

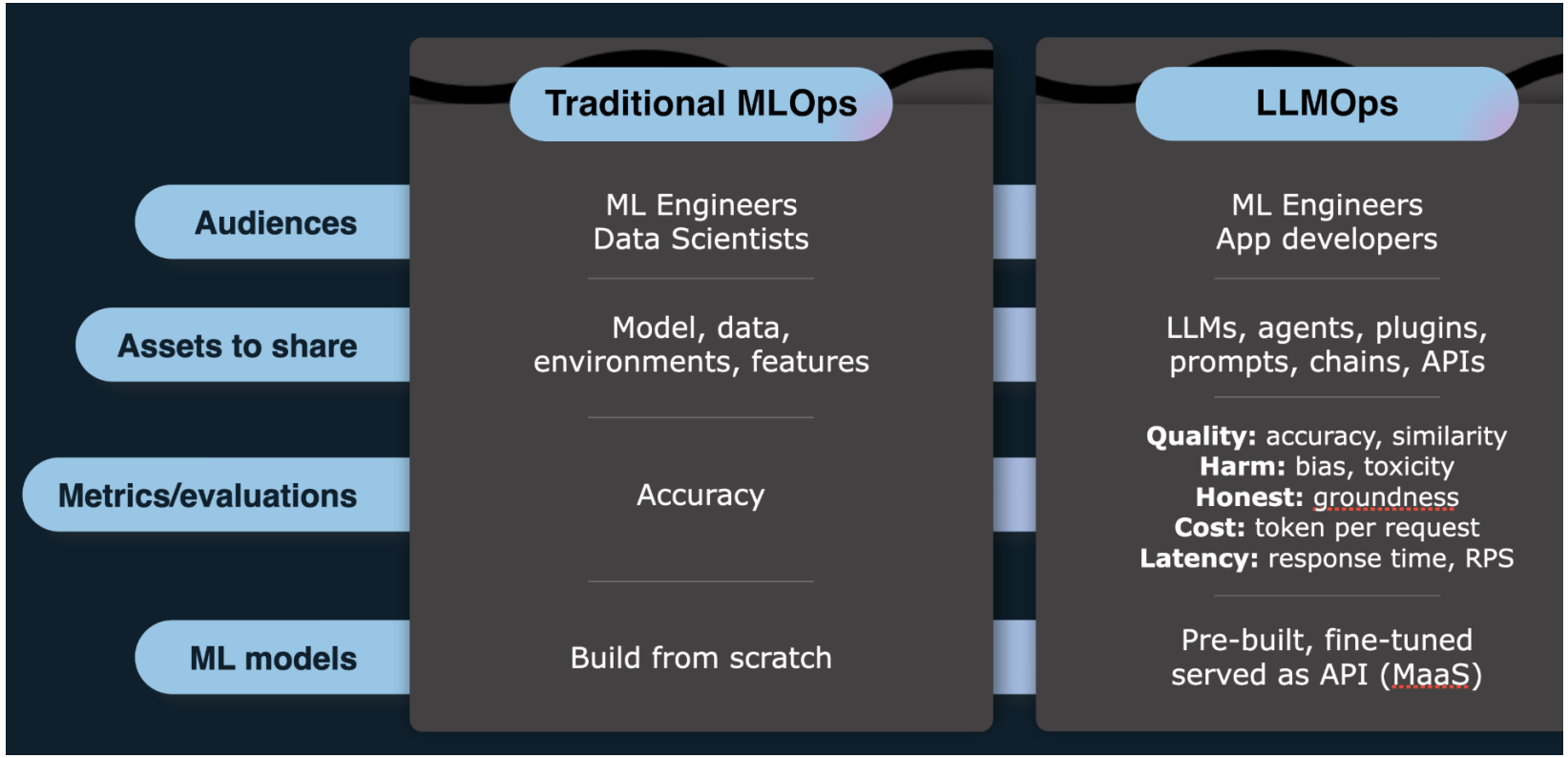
Data pipelines	Databricks	Airflow	Unstructured	
Embedding model	OpenAI	Cohere	Hugging Face	
Vector database	Pinecone	Weaviate	ChromaDB	pgvector
Playground	OpenAI	nat.dev	Humanloop	
Orchestration	Langchain	LlamaIndex	ChatGPT	
APIs/plugins	Serp	Wolfram	Zapier	
LLM cache	Redis	SQLite	GPTCache	
Logging / LLMops	Weights & Biases	MLflow	PromptLayer	Helicone
Validation	Guardrails	Rebuff	Microsoft Guidance	LMQL
App hosting	Vercel	Steamship	Streamlit	Modal
LLM APIs (proprietary)	OpenAI	Anthropic		
LLM APIs (open)	Hugging Face	Replicate		
Cloud providers	AWS	GCP	Azure	
Opinionated clouds	Databricks	Anyscale	Mosaic	

Teams for building a LLM product

AI/ML team	Platform Infrastructure team	Product team
Model finetuning, Model inference Prompting, RAG Implement safety, guardrails, Evals Labelling, curation	Devops, cloud Infrastructure , observability Model orchestration and model serving, real time inference Scale and reliability	Roadmap, features, Prioritisation Cross-team co-ordination

Frontend UI team	Security and Q/A team	Support team
Web front end Devs	Ensure Security , compliance and privacy Evaluation, Safety and Quality assurance	Customer , support

MLOPS vs LLMOps



https://github.com/microsoft/generative-ai-for-beginners/tree/main/14-the-generative-ai-application-lifecycle

AI evals

- Early versions of Copilot were not very functional.
- Evals were crucial in driving the product's improvement, providing signal on what was working and what wasn't.
- Consistent progress against evals by engineers led to significant product advancements.

Category	Purpose	Techniques and Tools	
Automated Benchmarking	Measure performance on fixed tasks.	Static datasets (MMLU, GSM8K), dynamic evaluation, few/zero-shot testing, metric scoring (accuracy, BLEU).	HELM, EleutherAI LM Harness, OpenLLM Leaderboard.
Human Evaluations	Assess subjective quality & alignment.	Likert scales, pairwise comparisons, Turing tests, expert reviews.	Amazon Mechanical Turk, Scale AI, Surge AI.
Adversarial Testing (Red Teaming)	Expose vulnerabilities (bias, jailbreaks).	Prompt injection, toxicity probes, OOD testing, gradient attacks.	CheckList, HateCheck, OpenAI Moderation API.
Real-World Deployment Checks	Validate in production.	A/B testing, shadow mode, continuous monitoring.	Prometheus, Grafana, Weights & Biases.
Specialized Evaluations	Domain/safety-specific tests.	Truthfulness (TruthfulQA), multimodal (CLIPScore), agentic (WebArena).	Custom rule engines, simulated environments.
Emerging Trends	Improve scalability/realism.	LLM-as-a-judge (GPT-4 grading), dynamic benchmarks (Dynabench), user feedback.	Dynabench, OpenAI Evals.

DEMO: AI evals

https://github.com/peremartra/Large-Language-Model-Notebooks-Course/tree/main

MMLU:

https://colab.research.google.com/github/LiuYuWei/llm_model_evaluation/blob/main/llm_evaluation_mmlu.ipynb

LLM_as judge:

https://colab.research.google.com/github/huggingface/cookbook/blob/main/notebooks/en/llm_judge.ipynb

LLM evaluation metrics
https://github.com/hari04hp/LLM-functionalities/blob/main/LLM_evaluation.ipynb

https://medium.com/@bavalpreetsinghh/rag-and-llm-evaluation-metrics-9cfe004d5bc3

https://mlflow.org/docs/latest/llms/llm-evaluate/index.html

RAG evaluation: https://medium.com/decodingml/the-engineers-framework-for-llm-rag-evaluation-59897381c326

07

Costs

LLM scaling laws

When you are training LLMs from scratch, its really important to ask these questions prior to the experiment-

1.How much data do I need to train LLMs from scratch?

2.What should be the size of the model?

The answer to these questions lies in scaling laws.

Scaling laws determines how much optimal data is required to train a model of a particular size.

In 2022, DeepMind proposed the scaling laws for training the LLMs with the optimal model size and dataset (no. of tokens) in the paper ***Training Compute-Optimal Large Language Models.***These scaling laws are popularly known as Chinchilla or Hoffman scaling laws

It states that

The no. of tokens used to train LLM should be 20 times more than the no. of parameters of the model.

1,400B (1.4T) tokens should be used to train a data-optimal LLM of size 70B parameters. So, we need around 20 text tokens per parameter.

What is the Cost to Train a large language model from scratch

100B parameter models need **300–1000B tokens** of high-quality text.
Preprocessing and curation costs for this data (especially filtered or proprietary datasets) can add **$0.1M – $1M**.

Training a dense 100B parameter model from scratch typically requires:

- **~10^23 FLOPs** (floating-point operations)
- Using **H100 GPUs**, this translates to **thousands of GPU-hours**
- At ~$1–$2/hour per H100 (cloud pricing), the **compute cost alone is $1M–$5M**

To train a **100B parameter dense model from scratch**:

- **$2M–$10M** on cloud infrastructure (possibly less with MoE, compression, or on-prem)

- Add **staffing, R&D, and data curation**, and the **total project cost could be $5M–$20M**

What is the Cost to Train a large language model from scratch

Factor	Estimate	Notes
Compute (GPU/TPU)	$1M – $6M	Based on 10^23 FLOPs, assuming H100 GPUs at $1–2/hr
Data Curation & Storage	$0.1M – $1M	High-quality tokenized datasets (300B–1T tokens)
Engineering + R&D	$0.5M – $2M	ML engineers, infra setup, DevOps
Training Infrastructure	$0.3M – $1M	Distributed training setup, networking, I/O
Total Cost	**$2M – $10M+**	May exceed $20M for cutting-edge models (e.g., GPT-4 dense equivalent)

Model	Parameters	Year	Estimated Training Cost
GPT-3	175B (dense)	2020	~$4.6M
PaLM	540B (dense)	2022	~$9M–$23M+
MT-NLG	530B (dense)	2021	~$10M
DeepSeek-V2	236B (MoE, 21B active)	2024	**~$2.3M – $3M (est.)**
GPT-4	~1.8T (MoE, ~280B active est.)	2023	**>$50M** (speculative)
Claude 3 Opus	>100B (undisclosed)	2024	~$10M+ (est.)

https://chatgpt.com/share/681a5962-c61c-800b-9bf7-7c7464677e00

Comparing API cost of Gemini vs ChatGPT vs Deepseek

1. Pre-trained Model Costs:

•**Token-based pricing:** Many LLM providers, like OpenAI, charge based on the number of input and output tokens used. For example, a model like GPT-4 can cost $30.00 per million tokens for input, and $0.50 per million tokens for output, according to Qwak.

•**Managed LLM Providers:** Platforms like Qwak offer managed LLMs, where you pay for the model and the infrastructure it runs on.

•**Example:** GPT-4 costs 6 cents per 1,000 tokens, which is roughly 240 words.

This refers to the cost of generating a response from the LLM based on a prompt. It's measured in tokens, where one token is roughly 3/4 of a word

Cost of self hosting LLM for inference vs ChatGPT API

https://www.reddit.com/r/LocalLLaMA/comments/1jzeo0l/the_real_cost_of_hosting_an_llm/

Self hosting is better: https://www.e2enetworks.com/blog/why-self-hosting-small-llms-are-cheaper-than-gpt-4-a-breakdown

https://sawerakhadium567.medium.com/is-hosting-your-own-llm-cheaper-than-openai-8a9a4dc76c6a

https://www.e2enetworks.com/blog/why-self-hosting-small-llms-are-cheaper-than-gpt-4-a-breakdown

https://www.linkedin.com/pulse/how-much-does-cost-self-host-llm-comprehensive-maxim-yemleninov-cufoe/

https://www.linkedin.com/pulse/true-cost-hosting-your-own-llm-comprehensive-comparison-binoloop-l3rtc/

https://blog.lytix.co/posts/self-hosting-llama-3

Finetuning costs

•**A100 (40GB):** ~1–2/hour (cloud pricing)

Estimated Costs (Cloud-Based Fine-Tuning)

Model Size	Dataset Size	Hardware	Estimated Cost
1B params	10M tokens	1x A100	20–100
7B params	100M tokens	1x A100	200–500
13B params	1B tokens	4x A100	1,000–3,000
70B params	10B tokens	8x H100	10,000–50,000

Cost-Saving Techniques

•**Parameter-Efficient Fine-Tuning (PEFT)** (e.g., LoRA, QLoRA) reduces GPU memory needs (~50–80% cheaper).

•**Mixed Precision Training** (FP16/BF16) speeds up training.

•**Gradient Checkpointing** reduces VRAM usage.

•**Spot Instances** (AWS/GCP) can cut costs by 60–90%.

Example: Fine-Tuning LLaMA 7B with QLoRA

•**Hardware:** 1x A100 (40GB) **Dataset:** 100K examples (~1B tokens) **Time:** ~24 hours

•**Estimated Cost:** ~50–200 (using spot instances).

For **enterprise-scale fine-tuning** (e.g., GPT-3 scale), costs can exceed **100*K*–1M+**.

Finetuning with PEFT costs

1. Small Models (1B–7B Parameters)

•**Example Models:** LLaMA-7B, Mistral-7B

•**Dataset:** 10K–100K instructions (~100M tokens)

•**Hardware:** 1x A100 (40GB)

- **Full Fine-Tuning:** ~10–24 hours → **50–50–200**
- **LoRA/QLoRA:** ~2–10 hours → **10–10–50**

•**A100 (40GB):** ~1–2/hour (cloud pricing)

Cost-Saving Scenarios (Using PEFT)

Model Size	Method	Dataset Size	Hardware	Cost Range
7B	QLoRA	10K examples	1x A100	10–**50**
13B	LoRA	50K examples	2x A100	100–**300**
70B	QLoRA	100K examples	4x H100	500–**2,000**

3. Large Models (70B+ Parameters)

•**Example Models:** LLaMA-70B, Falcon-180B

•**Dataset:** 100K–1M+ instructions (~1B+ tokens)

•**Hardware:** 8x H100 (or A100 80GB)

- **Full Fine-Tuning:** 3–7 days → **10,000–10,000–50,000**
- **LoRA/QLoRA:** ~24–48 hours → **1,000–1,000–5,000**

Cost of LLM app

The example assumptions include:
•Typical input tokens per prompt: 200
•Typical output tokens per prompt: 800 (four times the input)
•Number of analysts using the application: 100
•Prompts per analyst per day: 30
•Working days per year: 252
Based on these assumptions, the estimated annual cost per analyst for GPT-4 is $162, and for 100 analysts, it's $16,200.
For the GPT-4-32k model, the estimated annual cost per analyst is $324, and for 100 analysts, it's $32,400.
An example of a 140-token input prompt is shown. Note that different models calculate tokens differently.
The calculation is broken down:
•Tokens per analyst per day (input + output): 200 + 800 = 1000 tokens/prompt * 30 prompts = 30,000 tokens
•Total tokens per day for 100 analysts: 30,000 tokens/analyst * 100 analysts = 3,000,000 tokens
•Cost is calculated based on the input and output token prices.
The example calculation shows $18 for input tokens and $144 for output tokens per day for 100 analysts, totaling $162 per day.
The provided framework can be used by replacing the assumptions and the model's

Cost calculators

https://mem0.ai/llm/calculator

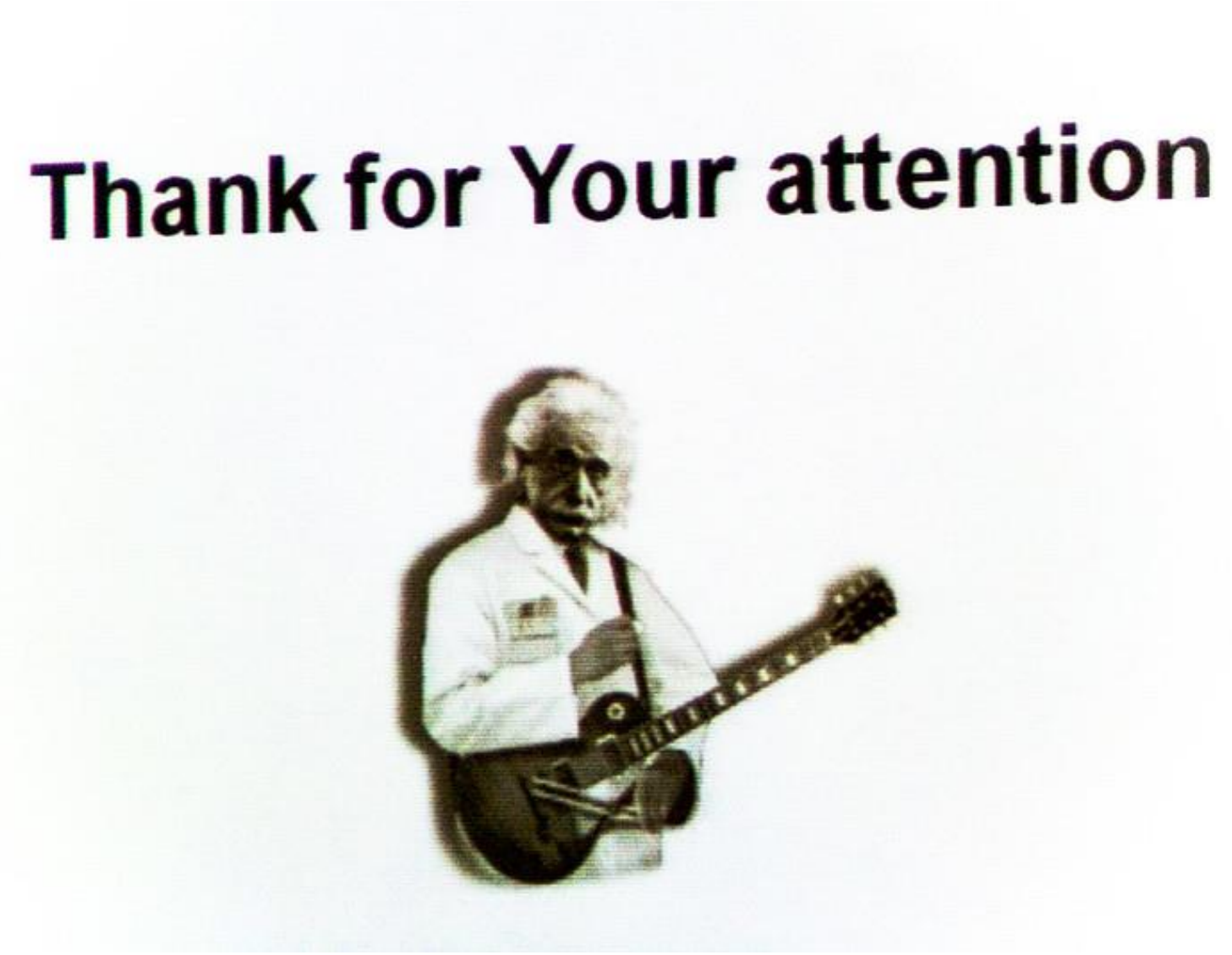
https://www.rtb-ai.com/llm-cost-calculator

https://machinehack.com/llms-cost-estimator?type=gpu

https://custom.typingmind.com/tools/estimate-llm-usage-costs

https://github.com/isEmmanuelOlowe/llm-cost-estimator

Thanks



LinkedIn
connect

https://www.linkedin.com/in/srimugunthan-dhandapani/