

# Session 1

# GenAI overview and

# Transformers – Part 1

Palacode Narayana Iyer Anantharaman

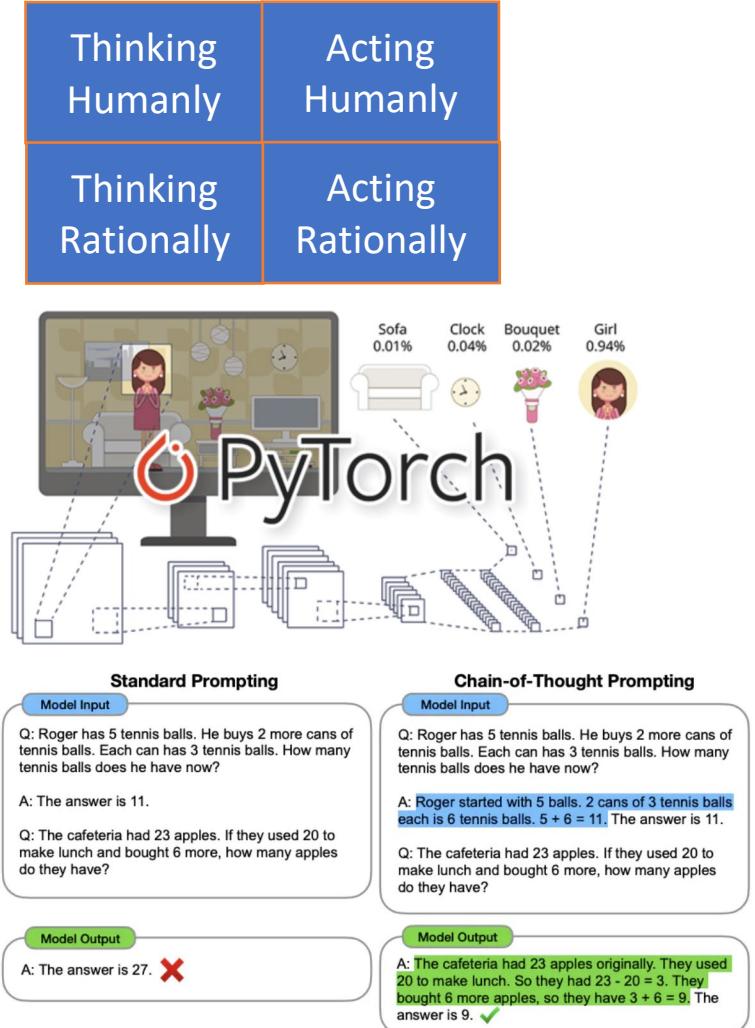
9<sup>th</sup> Aug 2024

# Welcome!

- Generative AI is a disruptive technology that is likely to shape the future. Ever since ChatGPT was released, this field has exploded.
  - Probably more impactful than what Web caused when it evolved through '90s.
- Practical applications of this is immense and are across the industry
- With the advent of core foundation models of Generative AI, e.g. GPT, the software development metaphor is undergoing a change alongside with new developer tools
- This new eco system has opened up a huge opportunity for the start ups as well as the established players.
- Our goal is to review these emerging trends viewing from the prism of technology. Specifically, we look at the landscape of LLM's, some use cases, review the application architectures and under the hood concepts.

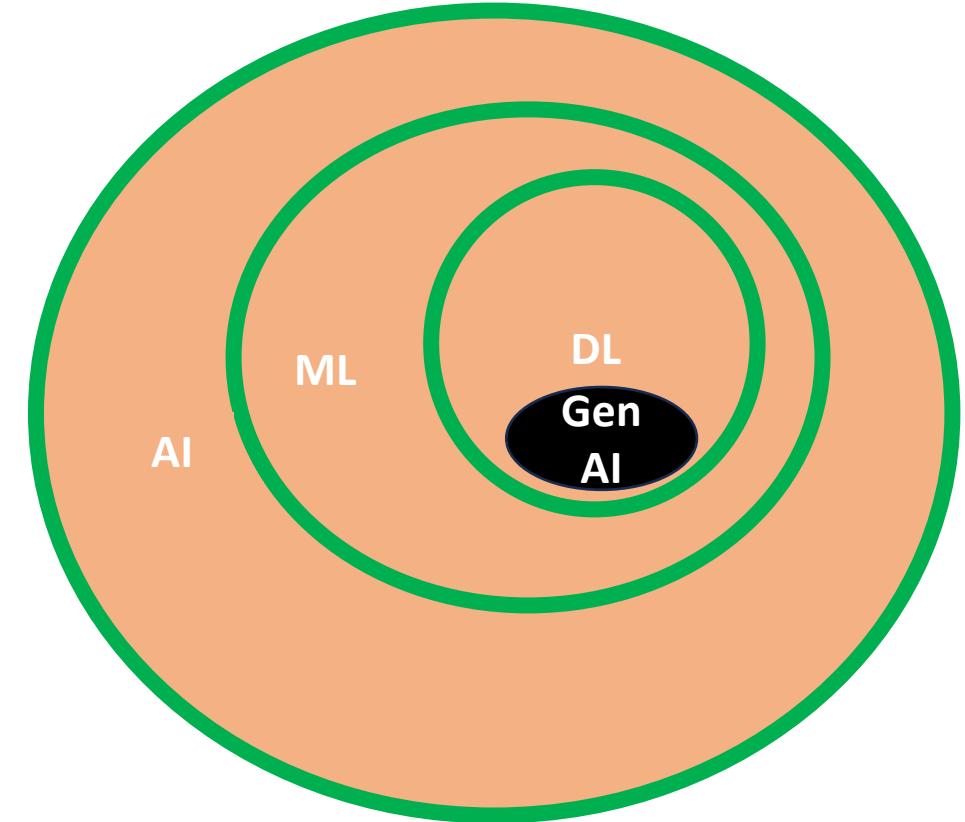
# AI, ML, DL and Generative AI

- Goal of the AI is to build a machine that is as capable as a human
  - Ability to think, reason and act like humans, humanly and rationally  
(Ref: Russel and Norvig)
- Machine Learning is about learning from data without being explicitly programmed
- Deep learning models are specialized ML architectures that rely on a large number of layers, complex models
- ML/DL Techniques constitute “narrow” AI or “weak” AI
- Generative AI, not only can generate new content but also go beyond fixed narrow functionality



# Artificial Intelligence, Machine Learning and Deep Learning

- The goal of AI is to build **human-like intelligence** on machines
- ML is a core approach to achieve this goal.
  - Key idea behind ML: **Learning from data**, ML is narrower in scope relative to AI
- DL is a suite of techniques that form a sub set ML
  - ML includes a broad variety of techniques like Probabilistic Graphical Models, Decision Trees, Neural Networks etc. The models can be shallow or deep.
  - Deep learning uses a **large number of computing layers** stacked vertically (output of one feeds in to the input of the next).
- Generative AI builds on top of DL architectures like Transformer networks, Diffusion networks etc.



# What AI can do?

English ▾ 🔊 🔉 ↔ Tamil ▾ 🔊 🔉

Artificial Intelligence is an exciting course [Edit](#)

செயற்கை நுண்ணிவு உற்சாகமான போக்காகும்  
Ceyarkai nunnarivu uracakamāṇḍu pōkkākum

English ▾ 🔊 🔉 ↔ Kannada ▾ 🔊

Artificial Intelligence is an exciting course

ಕೃತಕ ಬುದ್ಧಿಮತ್ತೆ ಒಂದು ಉತ್ತೇಜಕ ಹೋನ್‌  
Kṛtaka bud'dhimatte ondu uttējaka kōrs



Transfer 1m to my paypal  
now or i will kill your cat



"Alexa, ask DineTime what restaurants are near me."

"Alexa, ask DineTime for the wait at Red Robin."

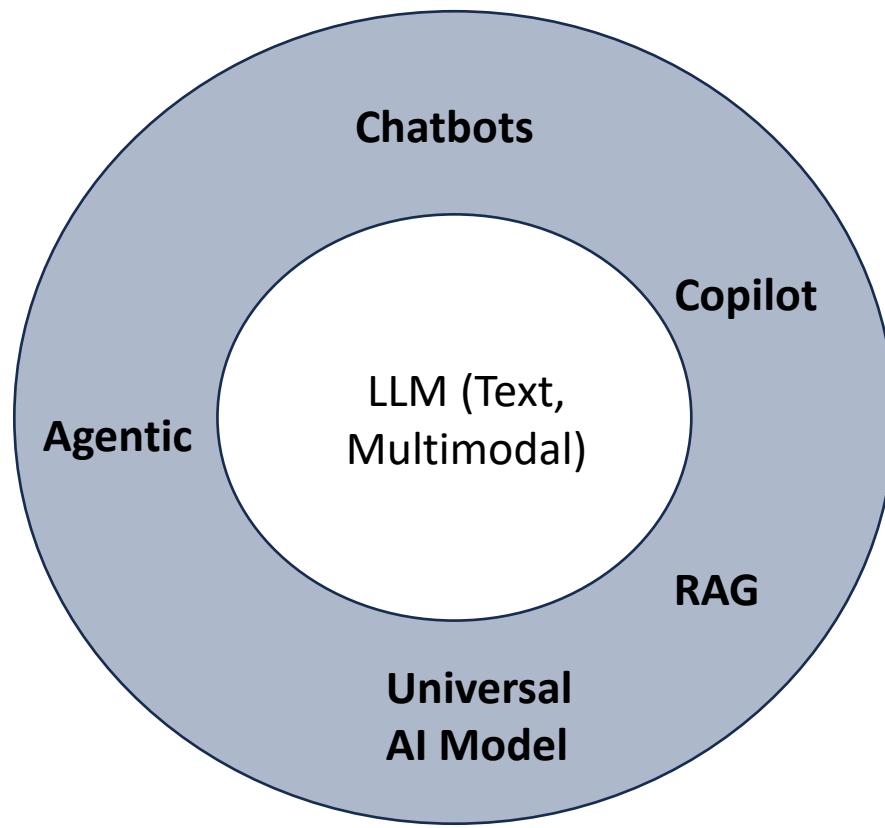


"Alexa, tell DineTime to add my name to the waitlist at Red Robin."

# Gen AI, LLM, Multimodal LLM

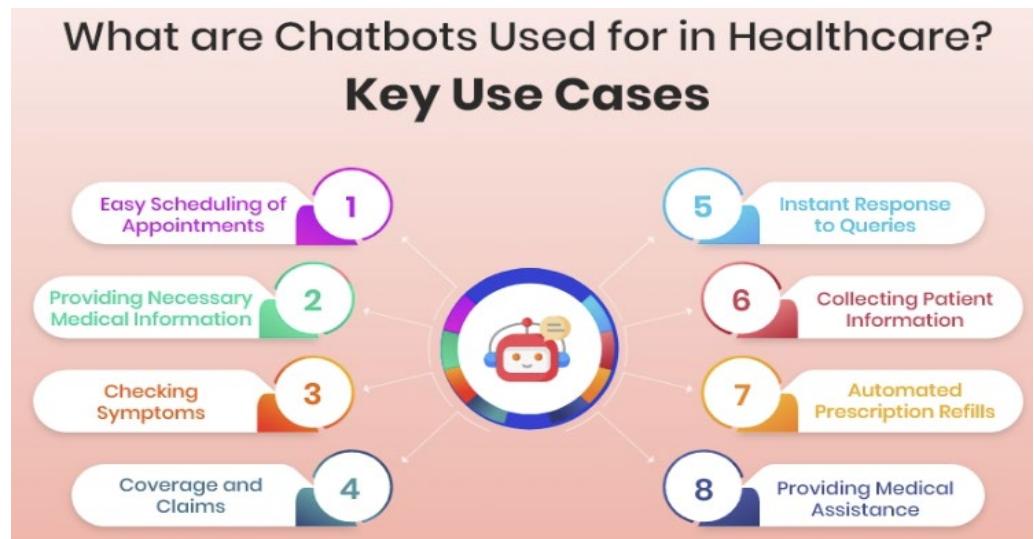
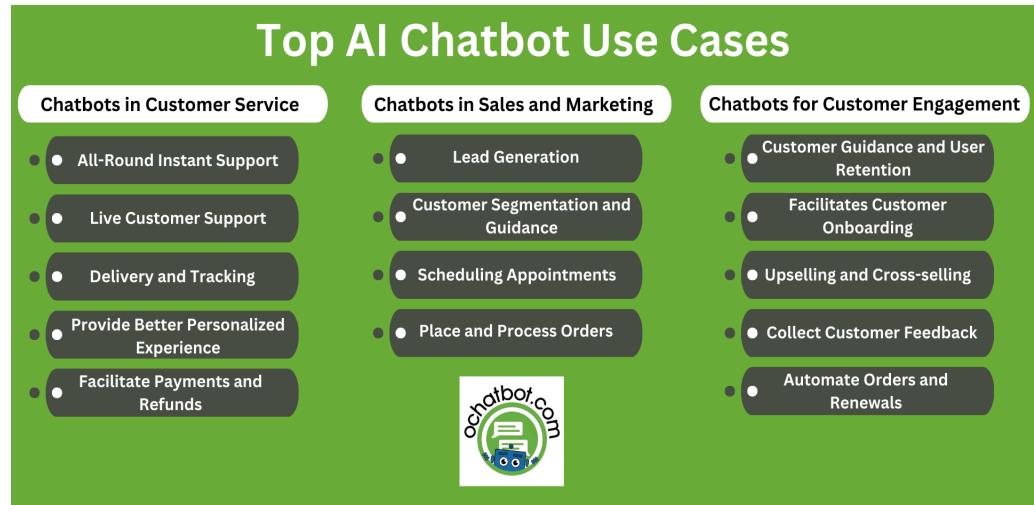
- Generative AI refers to generation of content using a AI program
- Content can be: Text (Structured, Unstructured, Semi Structured), Source code, Images, Videos, Audio, etc.
- Gen AI is built over deep learning models: for example, a generative adversarial network (GAN) is a form of Gen AI.
- Generative AI is a broader term that encompasses any form of content generation through an AI program that includes Large Language Models.

# LLM Application Patterns



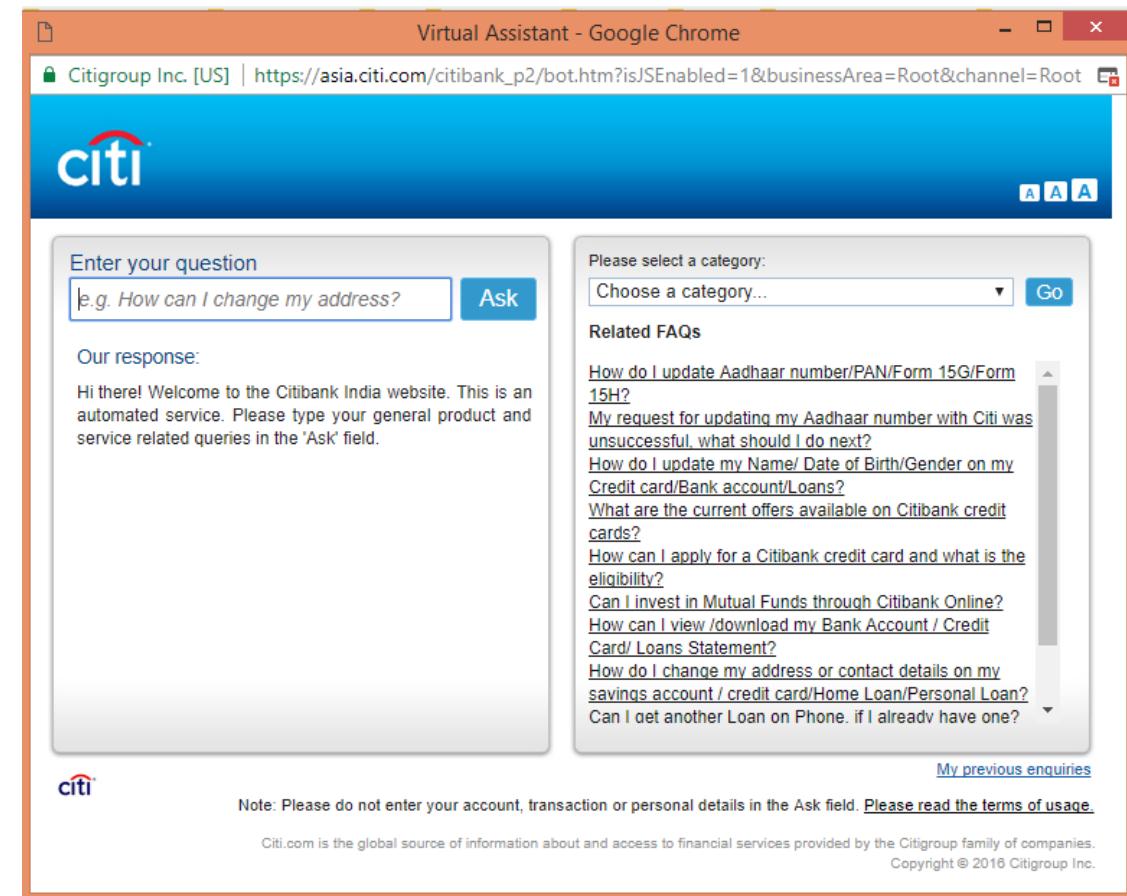
# Pattern#1: Chatbots

- Chatbots have immense power that have the ability to cut costs and improve productivity.
- LLMs embody a massive amount of knowledge besides having the natural ability to generate natural language text.
- Typical use cases:
  - Online shopping
  - Customer Support
  - Sales and Marketing
  - Fixing Appointments
  - Banking
  - Transportation: e.g. Booking Reservations

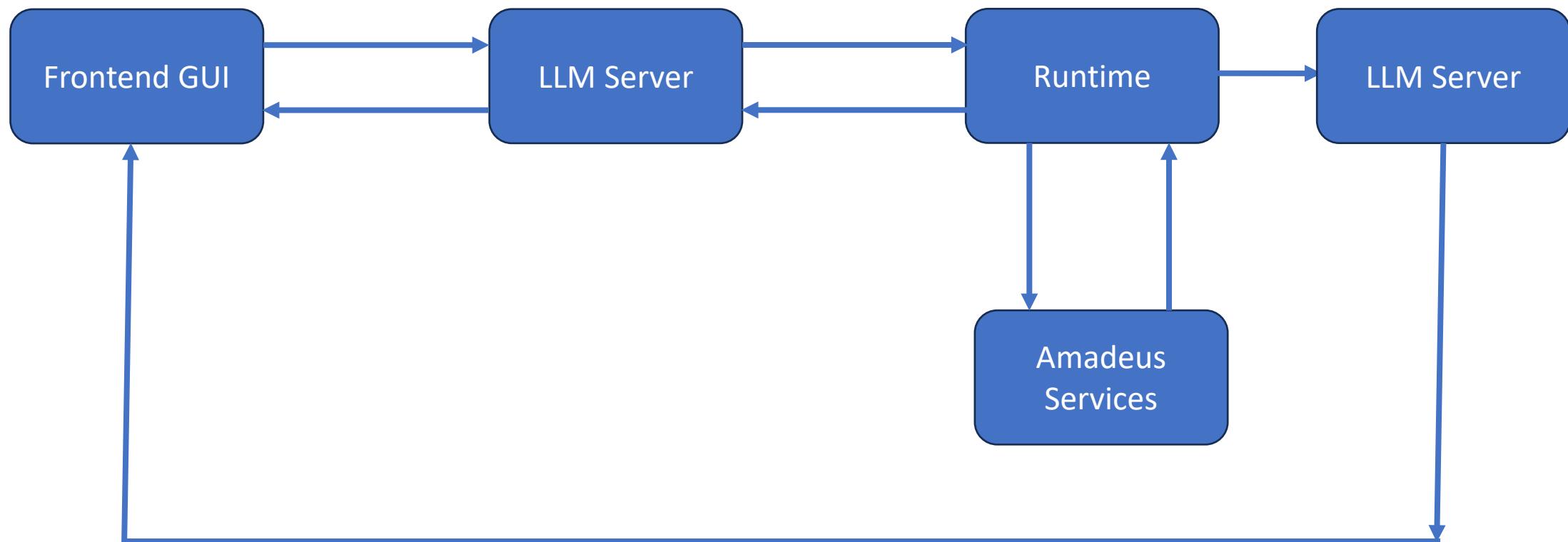


# Limitations of Conversational Agents

- While Chatbots are being used in the industry for over a decade, they are more viable only after the advent of LLMs
  - Closed domain solutions existed (e.g. bot for banking applications), there weren't products that performed satisfactorily well in terms of a flexible conversation
  - Most solutions were based on heuristic rule engines, fixed templates and so on and open domain conversational agents that performed well were hardly reported.
  - Probably for the first time, Google Lamda and ChatGPT demonstrated the ability to support wide open domain conversations, opening up a whole lot of opportunities



# Case Study: Chat Agent for Airline Reservation



# Amadeus API

## Overview of the API flow



### Flight Offers Search

[ Base URL: test.api.amadeus.com/v2 ]

Before using this API, we recommend you read our [Authorization Guide](#) for more information on how to generate an access token.

Please also be aware that our test environment is based on a subset of the production, if you are not returning any results try with big cities/airports like LON (London) or NYC (New-York).

#### Schemes

HTTPS

### Shopping

**POST** /shopping/flight-offers Return list of Flight Offers based on posted searching criteria.

**GET** /shopping/flight-offers Return list of Flight Offers based on searching criteria.

```
from amadeus import Client, ResponseError

amadeus = Client(
    client_id='REPLACE_BY_YOUR_API_KEY',
    client_secret='REPLACE_BY_YOUR_API_SECRET'
)

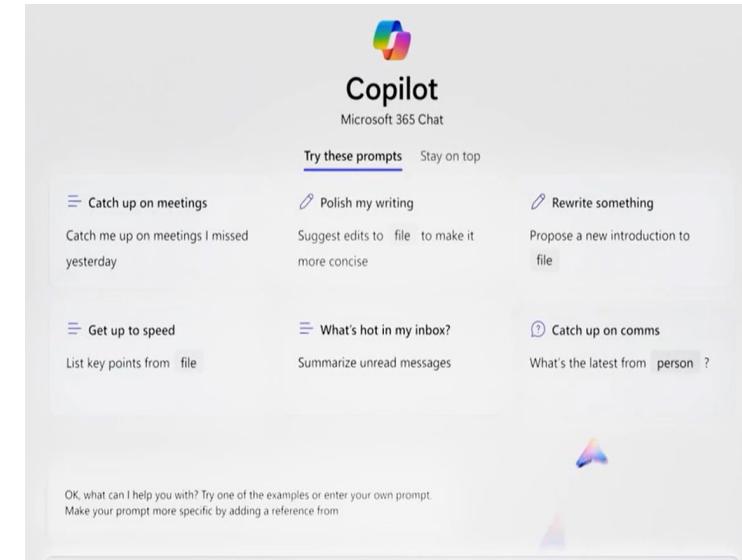
try:
    response = amadeus.shopping.flight_offers_search.get(
        originLocationCode='MAD',
        destinationLocationCode='ATH',
        departureDate='2024-11-01',
        adults=1)
    print(response.data)
except ResponseError as error:
    print(error)
```

# Exercise (Possible topic for the hackathon)

- Write a 1 page spec on an Airline/Cab/Hotel reservation system
- Use Amadeus API to get the relevant data
- Implement a chatbot driven by an open source LLM that implements the spec
- Test, Evaluate and deploy

# Pattern#2: Co Pilots

- Co Pilots are AI assistants intended to accomplish certain specific tasks.
  - Github copilot: Generates code, test cases etc.
  - Microsoft 365 copilot: Integrated into Microsoft Office applications, this LLM assists with tasks like generating text, creating summaries, and analyzing data within Word, Excel, and other Office tools.
  - Adobe has also come up with assistants pertaining to its products suite. E.g. Acrobat.



- Product development teams can consider implementing copilots for their products to enhance productivity. These are LLMs embedded in to the product.

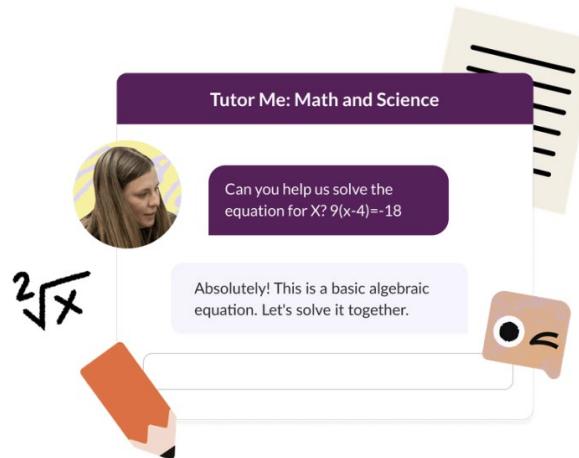


**Generative Fill**

Use a brush to remove objects or paint in new ones.

- Challenges:
  - Apprehensions of skill reduction among users, job cuts
  - Data privacy, security

# Examples of Copilot



## Copilot in OneNote

Summarize and organize text, generate ideas, and create lists with Copilot as your notetaking partner in OneNote.

C	D	E	F	G	H
Product	Discount Band	Units Sold	Manufacturing Price	Sale Price	Gross Sale
Biodegradable Cleaning Products	Low	4345	\$10	\$352	\$1,512
Biodegradable Cleaning Products	Low	4245	\$10	\$352	\$1,488
Biodegradable Cleaning Products	None	4124	\$10	\$352	\$1,448
Biodegradable Cleaning Products	Medium	3954	\$10	\$352	\$1,328
Biodegradable Cleaning Products	Low	4952	\$10	\$127	\$626
Recycled Can	Low	4903	\$5	\$127	\$635
Natural Personal Care Products	Low	1954	\$270	\$302	\$591
Biodegradable Cleaning Products	High	1990	\$220	\$302	\$664
Compostable Food Packaging	Low	1254	\$3	\$380	\$4,680
Natural Personal Care Products	Medium	1465	\$224	\$302	\$4,488
Sustainable Snack Packaging	Medium	1410	\$120	\$302	\$3,624
Natural Personal Care Products	Medium	1369	\$224	\$302	\$3,624
Compostable Food Packaging	Medium	1052	\$3	\$380	\$3,960
Compostable Food Packaging	Low	980	\$312	\$380	\$3,728
Sustainable Snack Packaging	Medium	1298	\$120	\$302	\$3,960
Compostable Food Packaging	None	954	\$3	\$380	\$3,624
Biodegradable Cleaning Products	Low	2785	\$110	\$127	\$3,455

A screenshot of Copilot in Microsoft Edge. The page shows an "About us" section for a company called Fosilium. Copilot is prompting the user to generate ideas, make to-do lists, or summarize the page. It also asks about key strengths for an advertising campaign and provides a "Working on your request..." status bar.

A screenshot of a news article from THE TIMES OF INDIA. The headline is "SAP India key in building GenAI copilot Joule". The article discusses SAP's role in developing the GenAI copilot Joule. The interface includes a weather widget for Bengaluru (25°C) and navigation links for various cities and news sections.

## SAP India key in building GenAI copilot Joule

# Demo – Amazon Q Developer in VS Code

- Amazon Q Developer is a copilot product
- Developers who use a supported IDE like VSCode can install Amazon Q developer extension plugin and use it while coding.
- Code suggestions, code review etc make the developer more productive.
- Limitations: Accuracy, code quality

# Exercise (Possible topic for the hackathon)

- Copilot powered products are highly suitable for product companies, such as Adobe, who already have many worldclass products.
- Adobe already has released copilot features in some of its products.
- Can you think of copilot driven features in the products you are working on and discuss how it will improve the user's productivity?
- As a specific idea, build your own Adobe Acrobat Assistant, extending the features of the current implementation.
  - Performance improvement
  - Multiple document support
  - Multimodal Support

# Pattern#3: Chatting With Data: RAG

- Suppose we need to work with our private data or we need to use data that are not “known” to the LLM, we can consider RAG approach
  - If we have a large number of documents (e.g PDFs, Excels, CSVs, MS Word, Web pages) and we want to interact with data in natural language.
- Retrieval Augmented Generation is a key technique that allows such interactivity.
- Can be used for mostly for Q&A and summarization
- High quality RAG systems are hard to build, recently many advancements have emerged to mitigate this.



**Andrea** 12 minutes ago

@Sonna What legal opinions or memos do we have that provide compliance guidance for remote worker regulations?



1



2 replies



**Sonna** APP 12 minutes ago

Consulting with my digital oracle...



**Sonna** APP 12 minutes ago

We have several documents that provide guidance on compliance with remote worker regulations, including internal memos, opinions from outside counsel, and drafts of the revised policy.

## References

Memo - Legal Considerations for Remote Work Regulations

Outside Counsel Opinions on Remote Work Policy Compliance

Revised Remote Work Policy - Draft

# Why Retrieval Augmented Generation (RAG)?

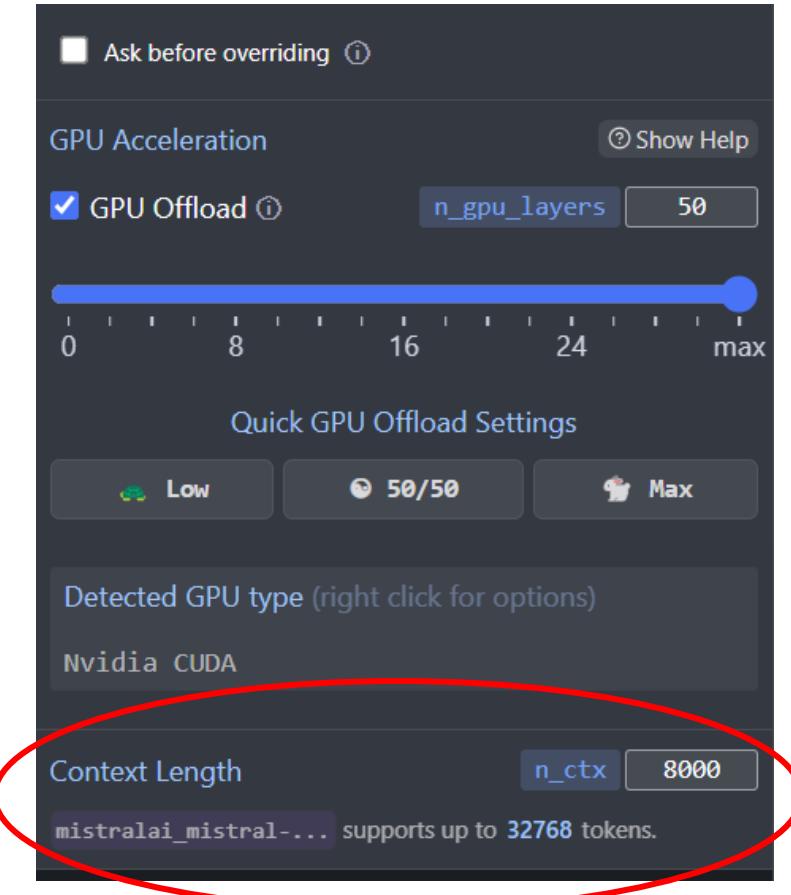
- The datasets that were used for LLM training are not live and real time.



- Training datasets are usually public information. Private data are not included in foundational LLMs. Supporting custom data is a major requirement for the LLMs. For example:
  - Proprietary code, design documents
  - Company confidential financial documents
- One way to provide live data to LLM is by using the context window. But this has limitations.

# Context Limitations

- LLM's are unaware of concepts outside of their training set
- Filling gaps in knowledge with assumptions
- Very hard to teach LLM's about new concepts



# Naïve RAG: Illustration

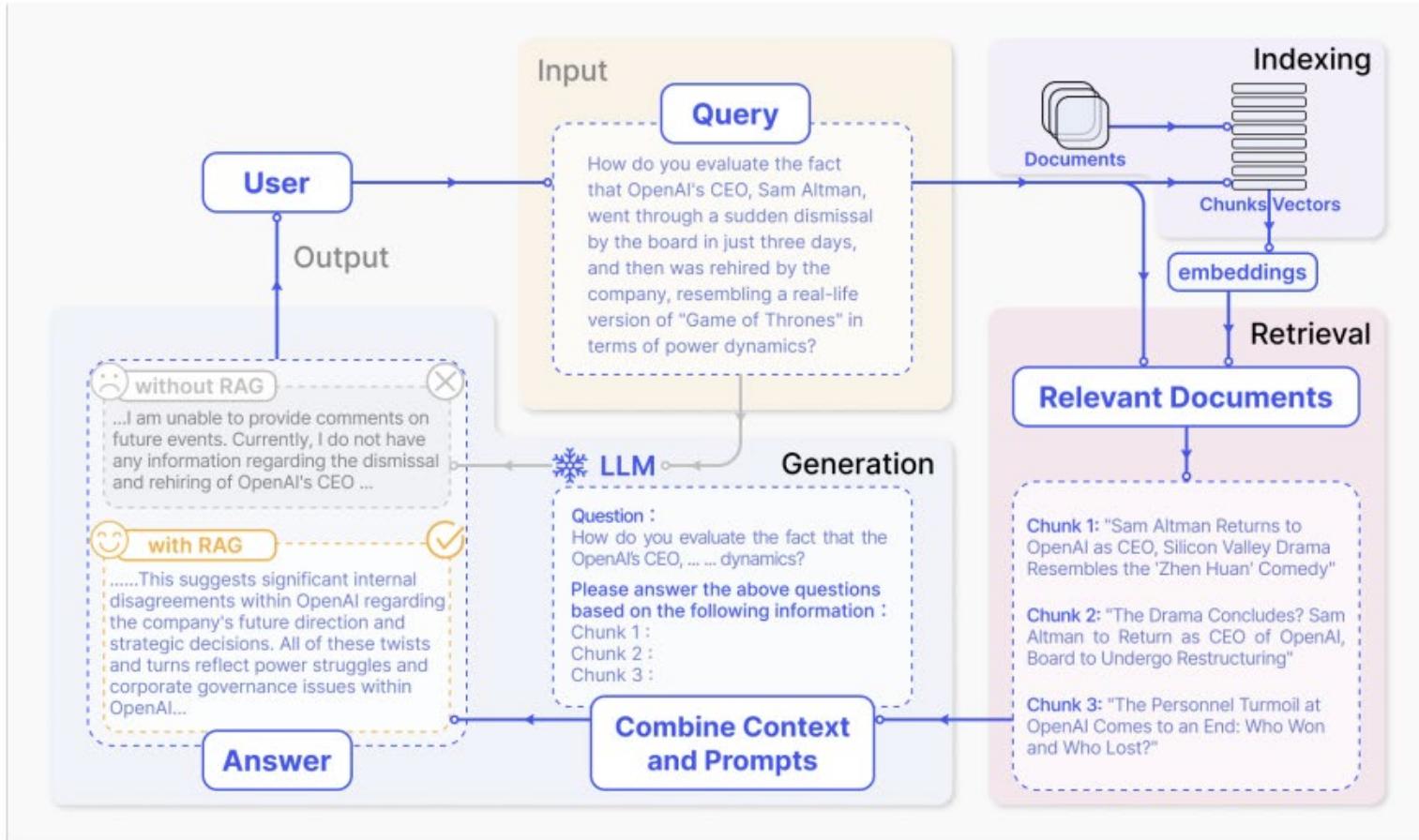


Fig. 2. A representative instance of the RAG process applied to question answering. It mainly consists of 3 steps. 1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity. 3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer.

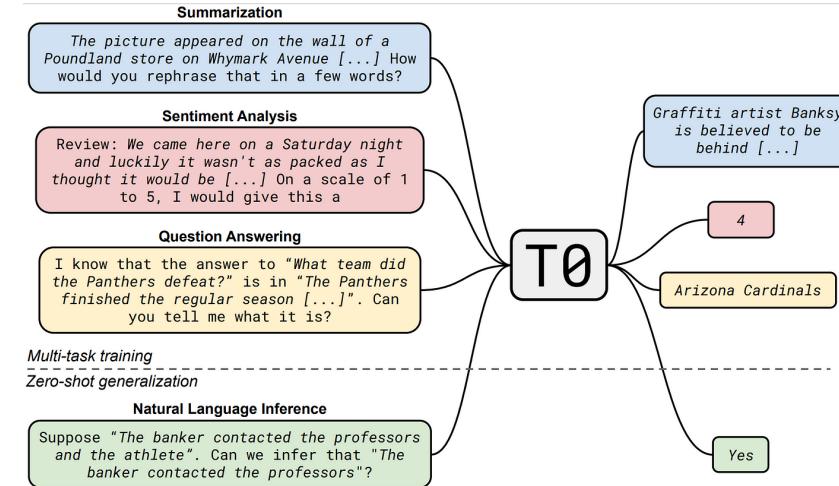
Fig Credits: RAG for LLM Models – A survey, Gao et al.

# Demo: Using RAG to Q&A on pharma companies

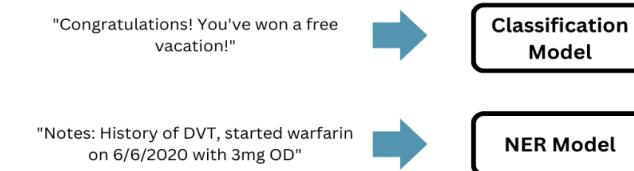
- Look at Adobe assistant in Acrobat for an example
- We ingest some sample documents pertaining to 2 pharma companies: Microlabs and Torrent
- We accept a question in natural language from the user and use the RAG technique to get the answer from the ingested documents.

# Pattern#4: LLM as a Universal DL Model

- The traditional ML/DL models solve narrowly defined tasks: e.g. Spam detection, Sentiment Analysis, Summarization, etc. based on individual models.
- LLMs can not only generate new content but also can perform discriminative classification tasks
- LLMs can be used as a single AI model that can be prompted suitably to behave like multiple high performant AI models. The same system can do a variety of tasks, for instance, co-reference resolution, machine translation, classification besides doing generative tasks
- Benefits: LLMs can be more accurate, training-free zero shot performance. Challenges are that LLMs can be inaccurate for tasks that are less common.



## Traditional NLP



## LLM Method



# Quora Questions Dataset: Duplicate detection

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-	0
2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when $[math]23^{24}[/math]$ is divided by 24,23?	0
4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0
5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What doe	1
6	13	14	Should I buy tiago?	What keeps childern active and far from phone and video games?	0
7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1
8	17	18	When do you use ä, instead of ä—?	When do you use "&" instead of "and"?	0
9	19	20	Motorola (company): Can I hack my Charter Motorola DCX3400?	How do I hack Motorola DCX3400 for free internet?	0
10	21	22	Method to find separation of slits using fresnel biprism?	What are some of the things technicians can tell about the durability and	0
11	23	24	How do I read and find my YouTube comments?	How can I see all my Youtube comments?	1
12	25	26	What can make Physics easy to learn?	How can you make physics easy to learn?	1
13	27	28	What was your first sexual experience like?	What was your first sexual experience?	1
14	29	30	What are the laws to change your status from a student visa to a green card in the US, ho	What are the laws to change your status from a student visa to a green c	0
15	31	32	What would a Trump presidency mean for current international masterâ€™s students on	How will a Trump presidency affect the students presently in US or plan	1
16	33	34	What does manipulation mean?	What does manipulation means?	1
17	35	36	Why do girls want to be friends with the guy they reject?	How do guys feel after rejecting a girl?	0
18	37	38	Why are so many Quora users posting questions that are readily answered on Google?	Why do people ask Quora questions which can be answered easily by G	1
19	39	40	Which is the best digital marketing institution in banglore?	Which is the best digital marketing institute in Pune?	0
20	41	42	Why do rockets look white?	Why are rockets and boosters painted white?	1
21	43	44	What's causing someone to be jealous?	What can I do to avoid being jealous of someone?	0
22	45	46	What are the questions should not ask on Quora?	Which question should I ask on Quora?	0
23	47	48	How much is 30 kV in HP?	Where can I find a conversion chart for CC to horsepower?	0
24	49	50	What does it mean that every time I look at the clock the numbers are the same?	How many times a day do a clockâ€™s hands overlap?	0
25	51	52	What are some tips on making it through the job interview process at Medicines?	What are some tips on making it through the job interview process at Fc	0
26	53	54	What is web application?	What is the web application framework?	0
27	55	56	Does society place too much importance on sports?	How do sports contribute to the society?	0
28	57	58	What is best way to make money online?	What is best way to ask for money online?	0
29	59	60	How should I prepare for CA final law?	How one should know that he/she completely prepare for CA final exam	1
30	61	62	What's one thing you would like to do better?	What's one thing you do despite knowing better?	0
31	63	64	What are some special cares for someone with a nose that gets stuffy during the night?	How can I keep my nose from getting stuffy at night?	1
32	65	66	What Game of Thrones villain would be the most likely to give you mercy?	What Game of Thrones villain would you most like to be at the mercy of	1

# Exercise

- Using the prompt below, perform duplicate detection using GPT-4o and Llama 3 8B models and discuss the results.

You are given a pair of questions as below. If one question is a duplicate of the other, respond "duplicate", otherwise respond "unique". Provide a rationale for your classification.

question#1: What is the step by step guide to invest in share market in India?

question#2: What is the step by step guide to invest in share market?

question#1: How do I read and find my YouTube comments?

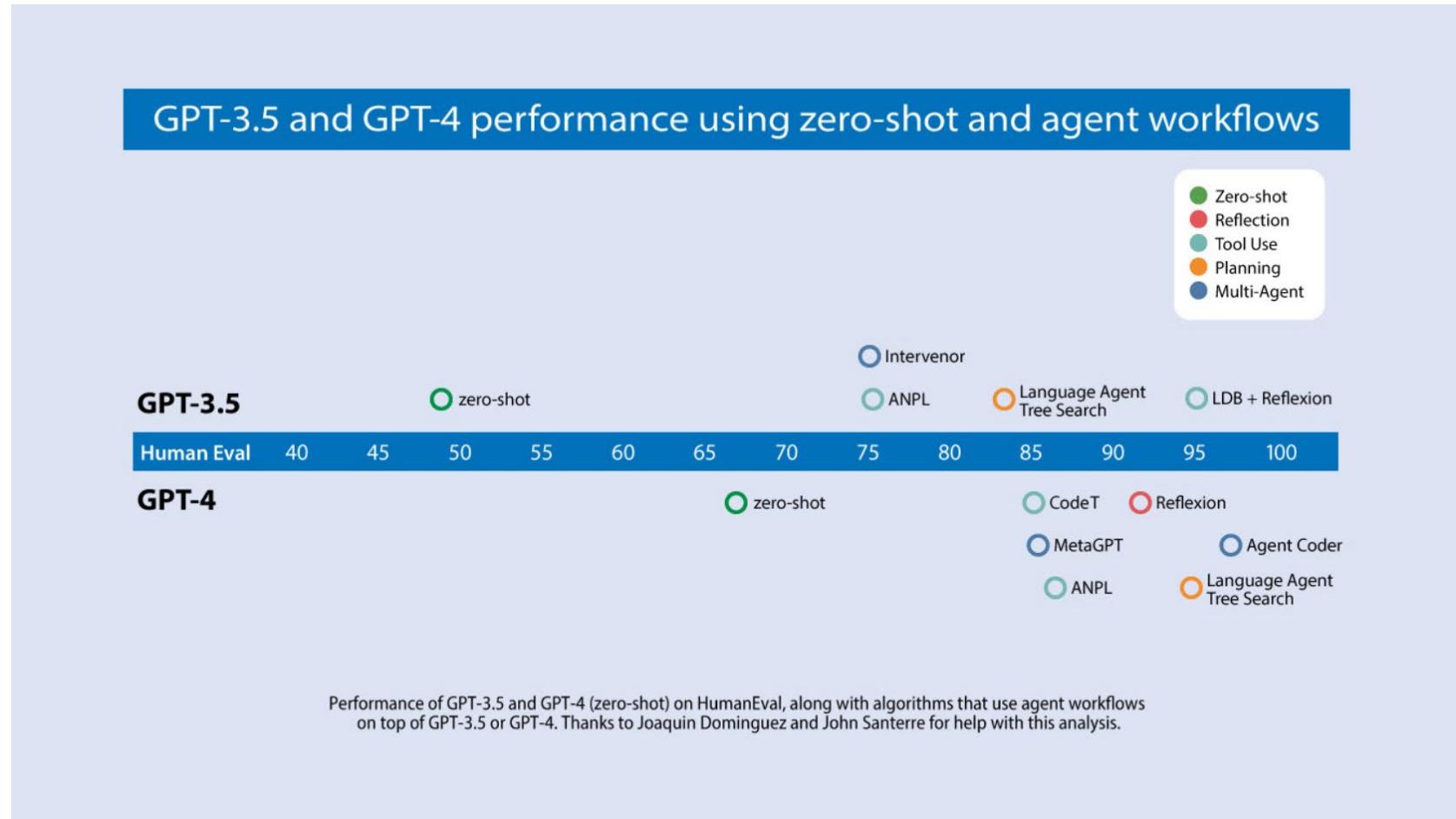
question#2: How can I see all my YouTube comments?

# Pattern#5: Agentic Workflows

- Agentic workflows are considered the next big thing in GenAI by many, including Prof Andrew Ng, Stanford University.
- Agentic workflow is an application pattern that uses agents to solve a larger problem
  - Agentic workflow allows implementation of a real world problem by dividing the problem in to multiple pieces, each performed by an agent.
- Agents are “conversable”, can be provided with: LLM, functions, tools and other assets
- Agents can be formed in a variety of topologies, such as a chain of agents, tree of agents and so on. They communicate using natural language text messages, lending a great degree of flexibility.

# LLM Performance: Zero shot versus Agentic

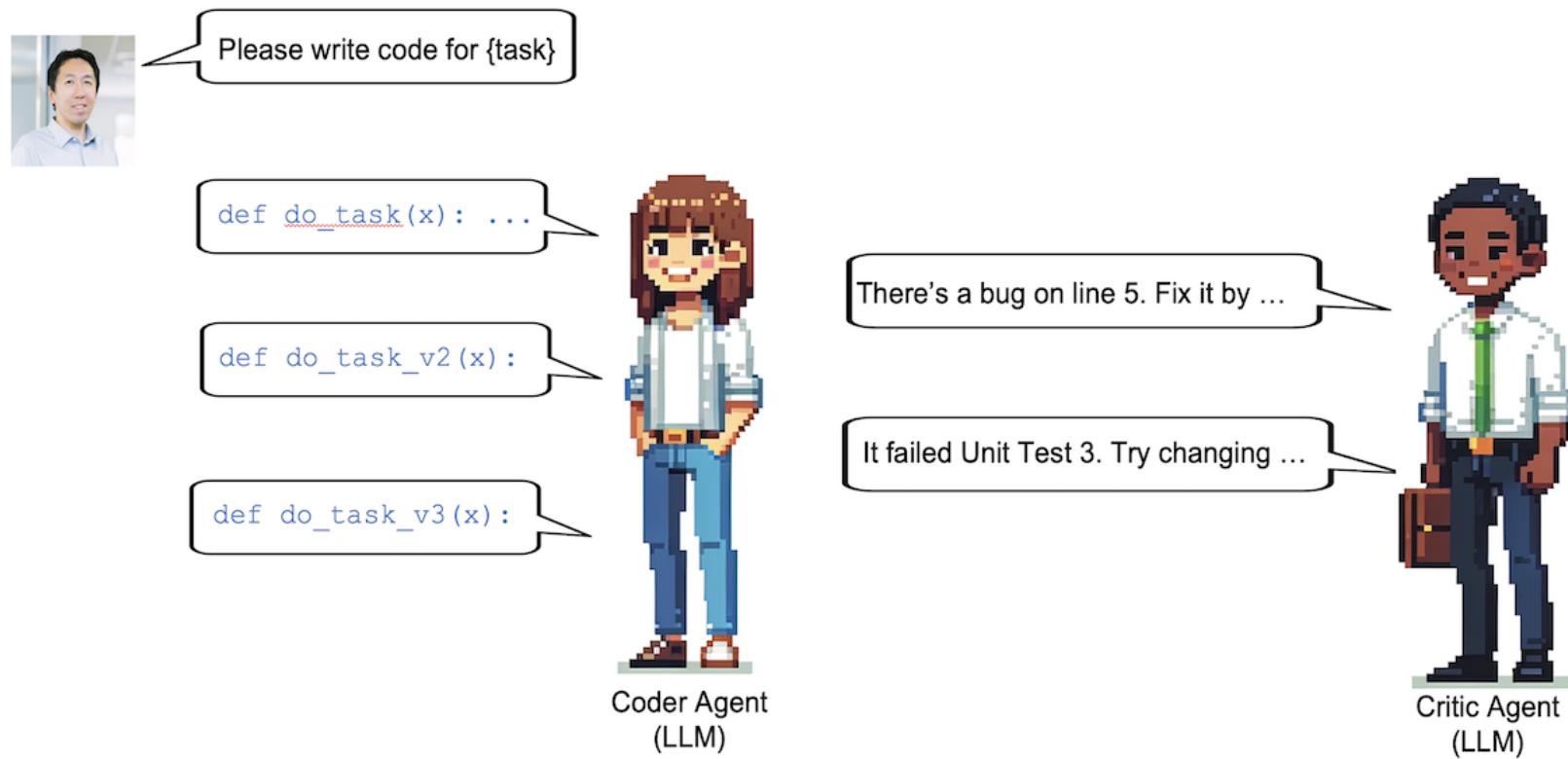
(Ref:  
Prof Andrew Ng, "The Batch")



- Zero shot performance of GPT 3.5 is about 48.1%
- GPT-4 Zero shot is about 67%
- Agentic performance of GPT 3.5 achieves up to 95.1%

# Example: Reflection Pattern for Error correction

## Agentic Design Patterns: Reflection



- LLM is given a task to perform through a prompt
- LLM provides a response for the given prompt
- A critic examines the response and provides feedback. This is the next level prompt.
- LLM revisits the previous response and provides a modified response

# Finetuning

# Why should we finetune?

- LLMs can be prompted to answer our questions or process our inputs
- Our inputs can have very diverse sizes, from a one line question to something that is an entire database.
- Solving such requirements with only prompt engineering has to confront the context limitations
- Finetuning allows us to put more data in to the LLM: External knowledge to Parameterized
- Finetuning also helps LLMs in acquiring domain specific knowledge. E.g. Certain chemical names, abbreviations, terminology used widely in a pharma company.

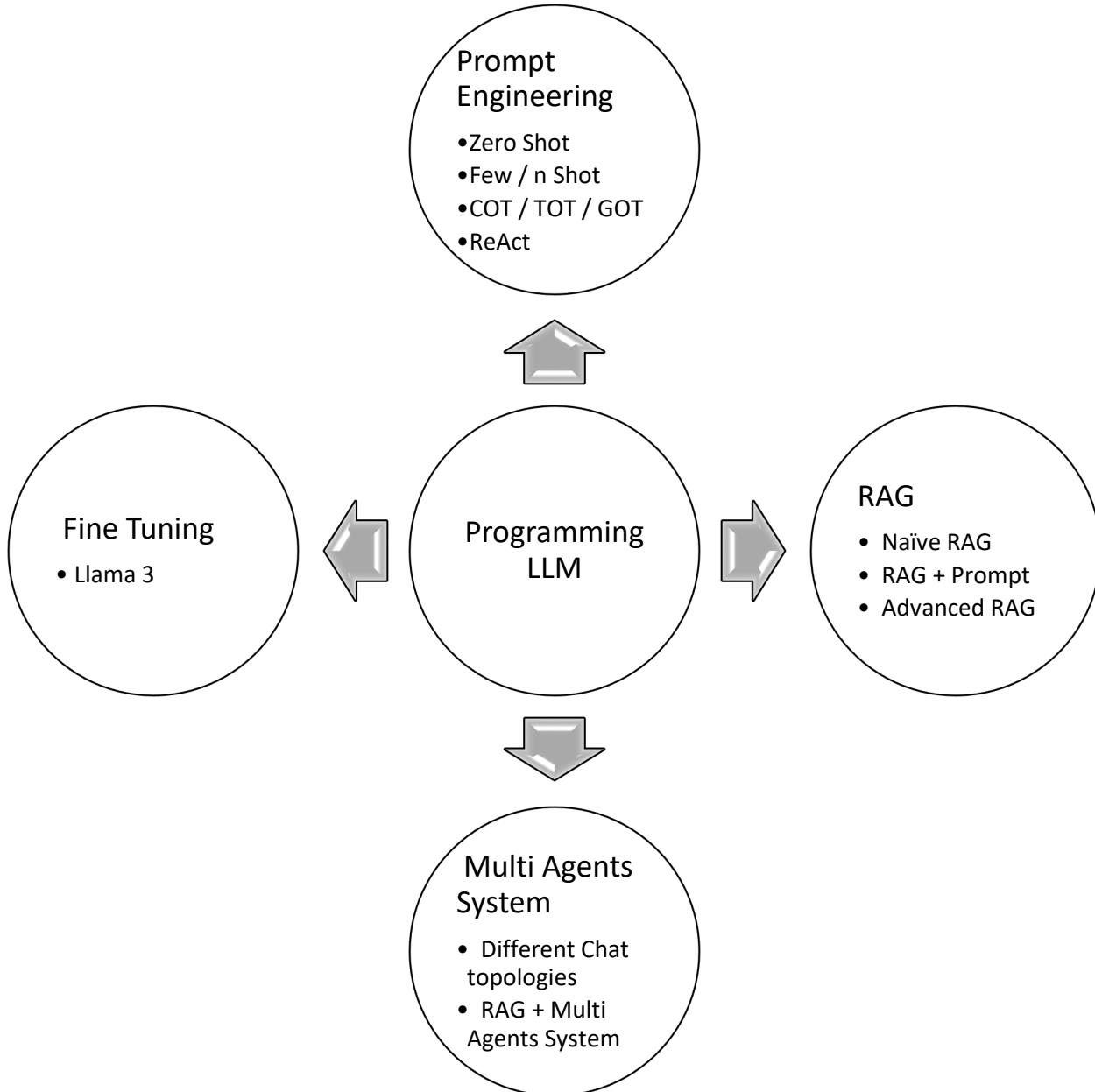
# Example: Finetuned Model for Airline Reservation

- Create a dataset:
  - Inputs: Natural language question
  - Outputs: API call invocation code with right function calls, parameters
- Choose a suitable foundation model
- Perform finetuning
- Evaluate
- Test and deploy

# Application Patterns and Design Options

Application Patterns	Design Patterns
Chatbots	Prompt engineering, Finetuned Models (e.g. chatting on a special domain)
Copilot	Finetuned Models, Agents, RAG, Prompt Engineering
RAG	RAG, Prompt engineering
Universal AI Model	Prompt Engineering, Finetuned Models
Agentic Workflows	Prompt engineering, Agent based design

# Programming The LLM



**LLMs can be programmed for**

- Generation
- Prediction
- Action

**Tools**

- Langchain
- Autogen
- ChromaDb
- Unsloth
- LMStudio
- Streamlit
- Gradio
- HuggingFace API

# Stable Diffusion



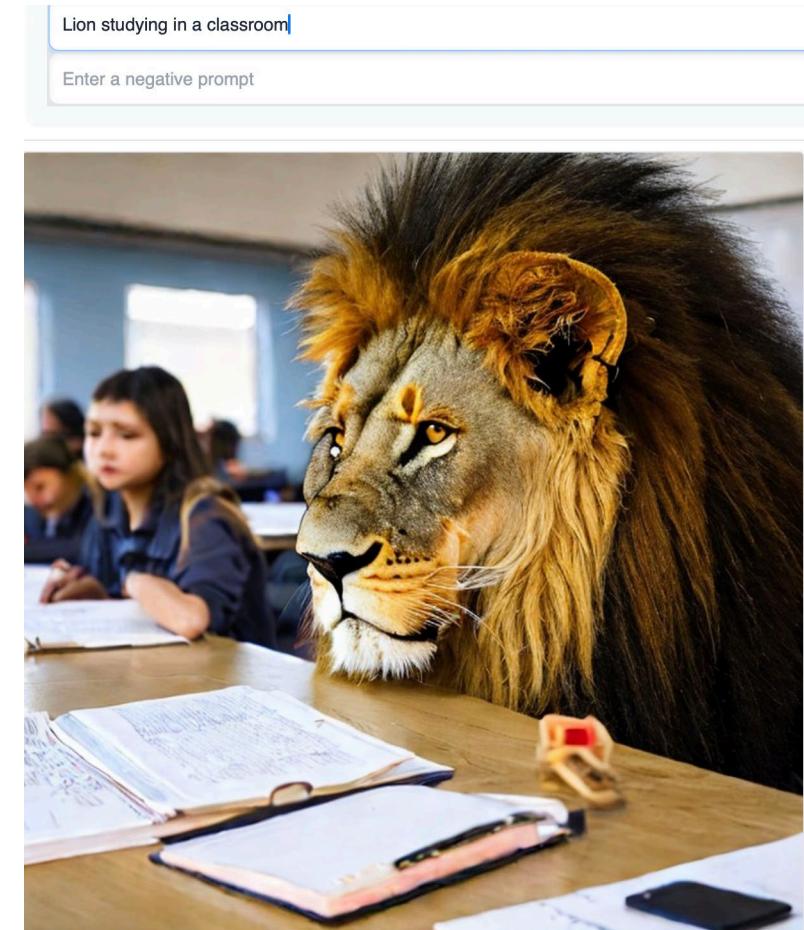
<https://huggingface.co/spaces/stabilityai/stable-diffusion>

<https://huggingface.co/stabilityai/stable-diffusion-2-1>



# Multimodal LLMs

- Models such as OpenAI DALL-E 2, Stable Diffusion take a text based prompt as input and generate image as output.
- Meta's AudioCraft generates high quality audio and music from text description
- Such models that work with multiple modalities such as text, images, audio are called multimodal LLMs
- As these generate content, they are part of Generative AI.

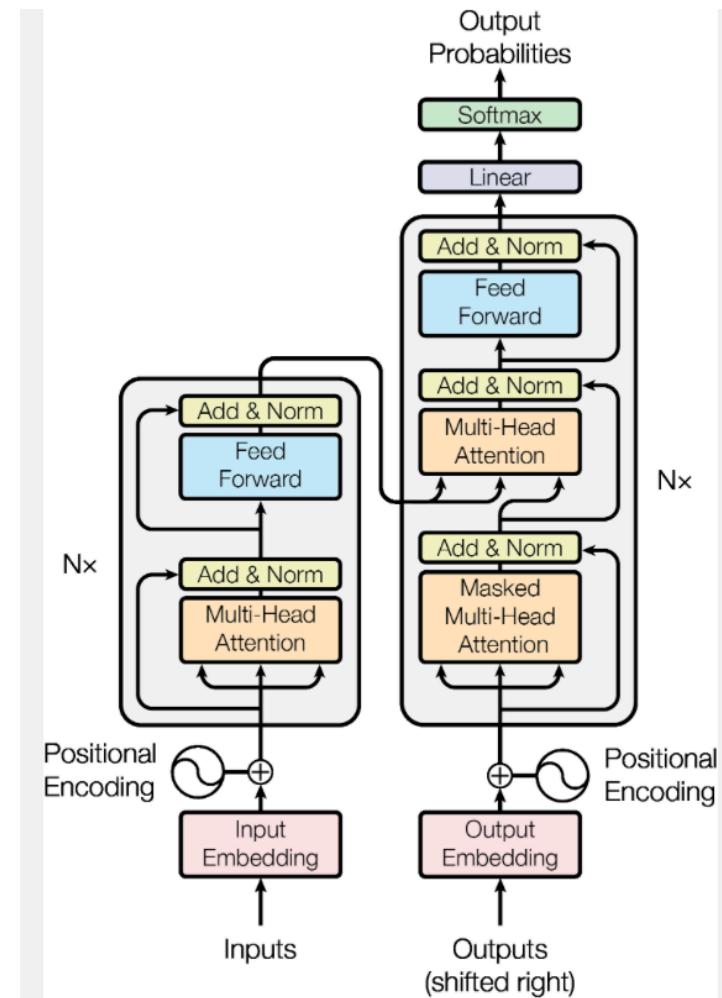


# Recent Trends: Small LLMs for Edge devices

- Several use cases of LLM require the model to be run on local machines
- Custom domain specific LLMs are needed for verticals: e.g. Finance, Pharma, sports
- In near future, mobile devices running LLMs on-device are envisioned.
- Challenges:
  - Memory footprint of LLMs
  - Compute requirements: High end CPU, GPUs
  - Battery Power
- Solution: Small LLMs that can run on systems with memory < 8 GB
  - Microsoft Phi 3
  - Google gemma 2B

# LLM Inference Settings

- Configuration parameters at the inference time (not at the training time)
- Max new tokens: limits the number of tokens model will generate in the decoder
- Top K: top k ranking labels
- Top p: labels that add up to p
- Temperature: randomly pick with uniform probability



# Using HuggingFace

Hugging Face Search models, datasets, users...

Models Datasets Spaces Posts Docs Solutions Pricing

TheBloke/Mistral-7B-Instruct-v0.2-GGUF like 120

Text Generation Transformers GGUF mistral finetuned text-generation-inference arxiv:2310.06825 License: apache-2.0

Model card Files and versions Community 6 Train Deploy Use in Transformers

Provided files

Name	Quant method	Bits	Size	Max RAM required	Use case
<a href="#">mistral-7b-instruct-v0.2.Q2_K.gguf</a>	Q2_K	2	3.08 GB	5.58 GB	smallest, significant quality loss - not recommended for most purposes
<a href="#">mistral-7b-instruct-v0.2.Q3_K_S.gguf</a>	Q3_K_S	3	3.16 GB	5.66 GB	very small, high quality loss
<a href="#">mistral-7b-instruct-v0.2.Q3_K_M.gguf</a>	Q3_K_M	3	3.52 GB	6.02 GB	very small, high quality loss
<a href="#">mistral-7b-instruct-v0.2.Q3_K_L.gguf</a>	Q3_K_L	3	3.82 GB	6.32 GB	small, substantial quality loss
<a href="#">mistral-7b-instruct-v0.2.Q4_0.gguf</a>	Q4_0	4	4.11 GB	6.61 GB	legacy; small, very high quality loss - prefer using Q3_K_M
<a href="#">mistral-7b-instruct-v0.2.Q4_K_S.gguf</a>	Q4_K_S	4	4.14 GB	6.64 GB	small, greater quality loss
<a href="#">mistral-7b-instruct-v0.2.Q4_K_M.gguf</a>	Q4_K_M	4	4.37 GB	6.87 GB	medium, balanced quality - recommended

# Hands On #1: Installing and loading LLMs Locally

- Download and install LM Studio
- Select the LLM using GUI
- Download and set it up
- Set up the parameters
- Use the LLM in chat mode or as a server
- Refer: [README.txt](#) along with the python files that was shared earlier

# LLM Limitations

- Only know what they were trained on
- Context size is limited
- Bad scaling with increasing context
- Hallucinations
- Hard to evaluate results
- Expensive for high volumes of data

A What is Tree of Thought prompting

 "Tree of Thought" prompting is not a widely recognized term or concept as of my last knowledge update in January 2022. It's possible that this term may have been coined or introduced in a specific context or field after that date. If you have more context or details about what "Tree of Thought" prompting refers to, please provide additional information, and I'll do my best to assist you with the specific information you're looking for.

A What are the earnings of Microlabs Pharma in India in 2023?

 I apologize, but I don't have access to real-time data, including specific financial information for companies. My knowledge is based on information available up to January 2022, and I cannot provide you with the earnings of Microlabs Pharma in India or any other company for the year 2023.

To obtain the most up-to-date financial information for Microlabs Pharma in India for 2023, I recommend checking financial news sources, the company's official reports, or financial databases that provide this information.

Thank You All!

# Session 2

# Deep Learning and Transformers

Palacode Narayana Iyer Anantharaman

16<sup>th</sup> Aug 2024

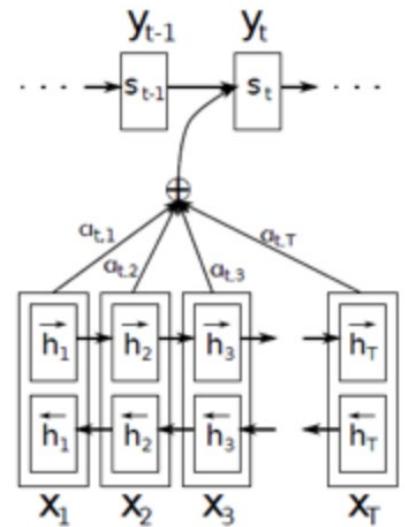
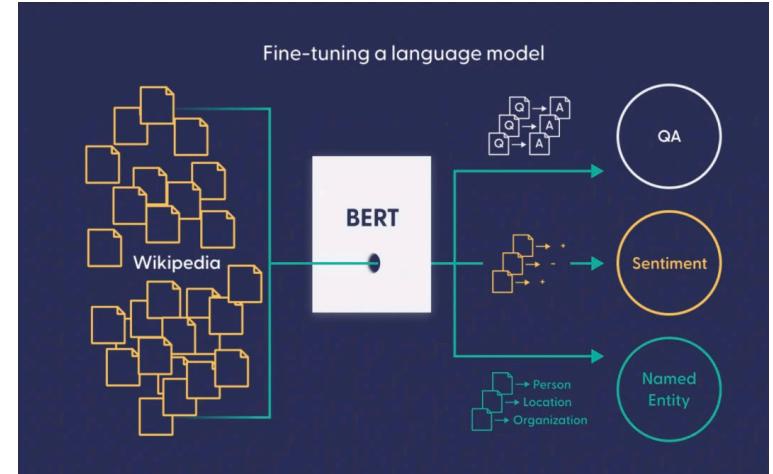
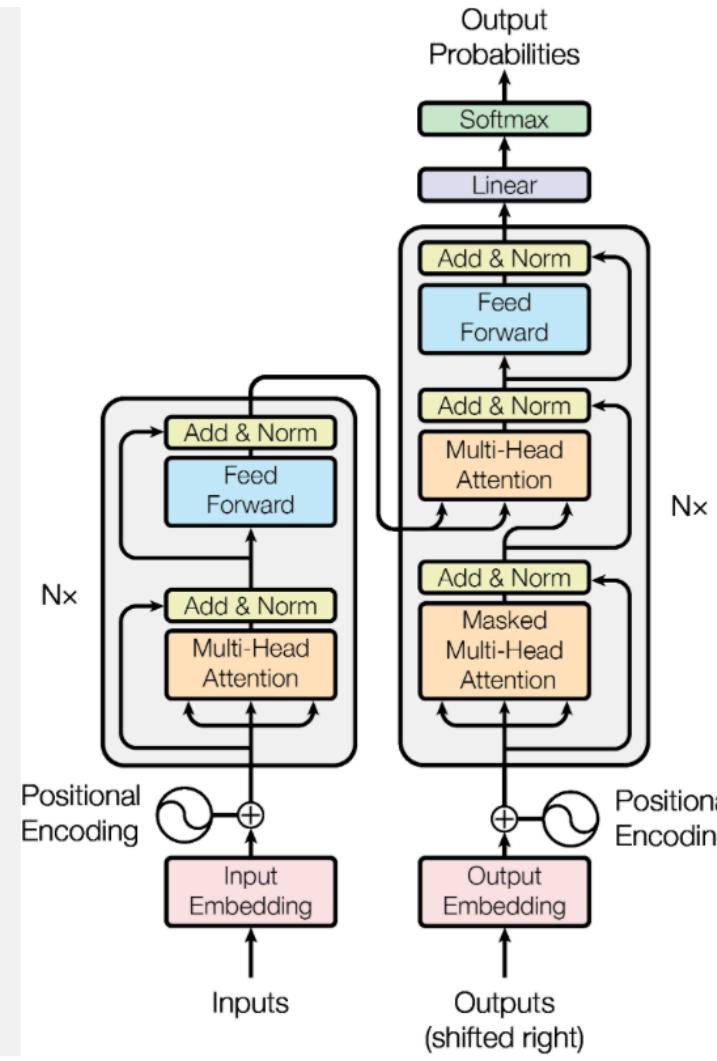
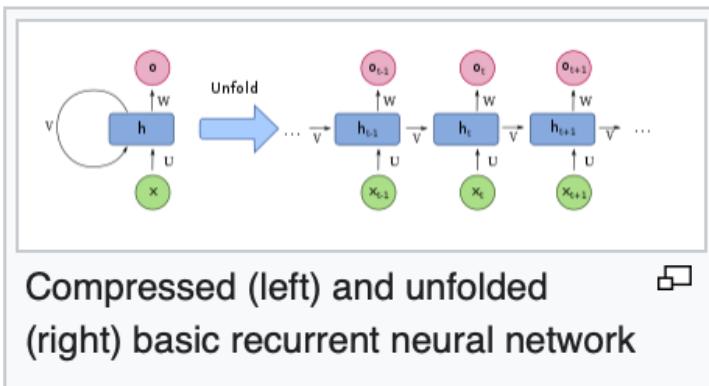
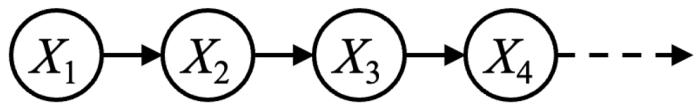
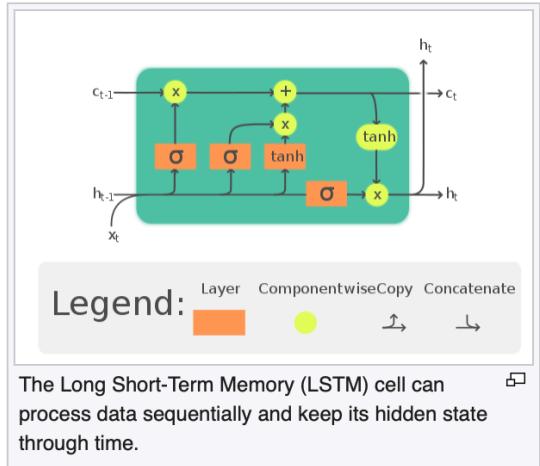
# Transformer Visualization Demo

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

# How LLMs are trained?

- Pretraining: The goal is to learn language patterns, grammatical constructs, common phrases, facts, some bit of reasoning, etc.
- Finetuning: The goal is to specialize the base pretrained model on specific tasks, in particular instruction following behavior
- (Optional) RLHF: Build a human compatible responses, avoid toxic content, enable honest, bias-less and factual answers

# Key Foundational Technologies



# Models, Tools and Technologies



Hugging Face

TheBloke/Mistral-7B-Instruct-v0.1-GGUF like 241

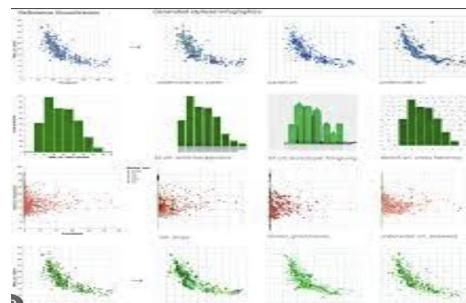
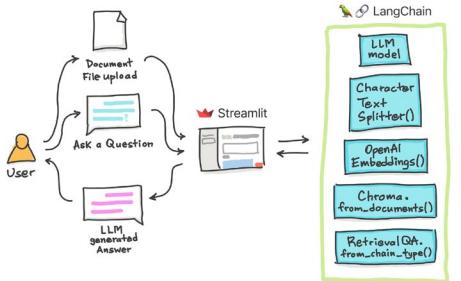
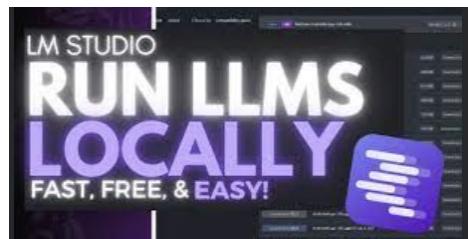
Text Generation Transformers mistral finetuned text-generation-inference License: apache-2.0

Model card Files and versions Community 8

DON'T PANIC! THEBLOKE AI

Chat & support: TheBloke's Discord server Want to contribute? TheBloke's Patreon page

TheBloke's LLM work is generously supported by a grant from [andreessen horowitz \(a16z\)](#)



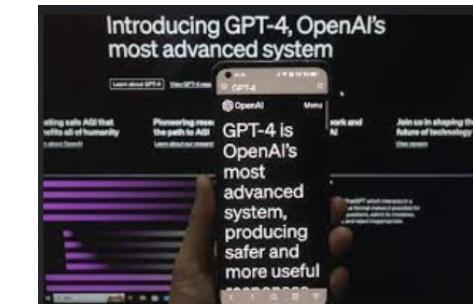
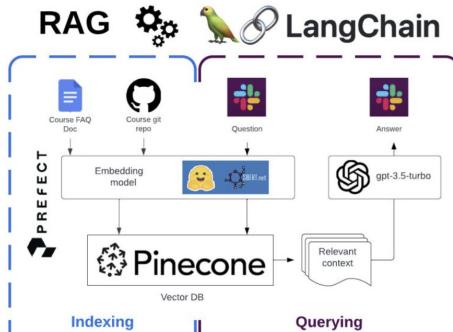
Hugging Face  
llamaindex (LlamaIndex)



Vector  
Databases

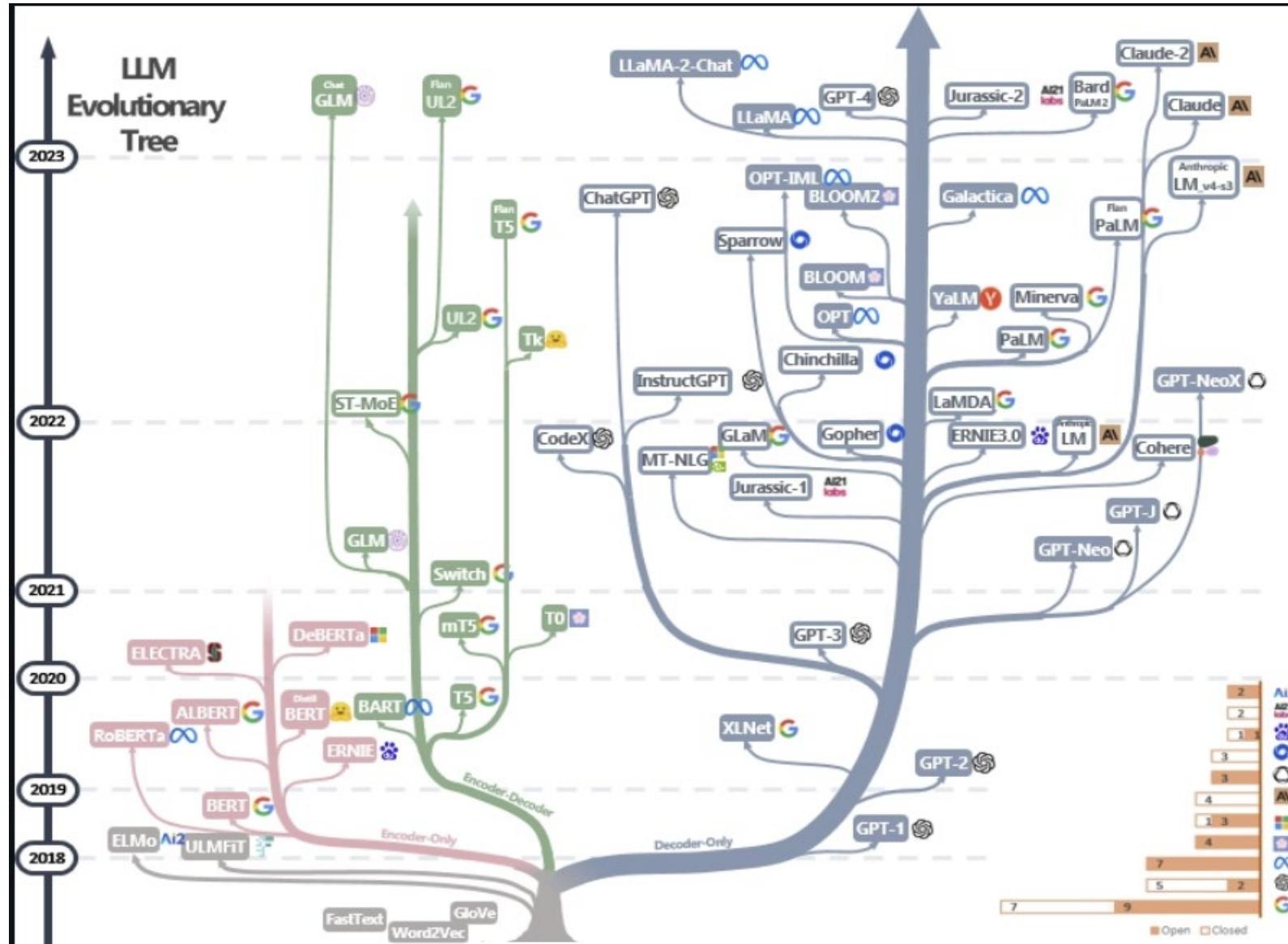
## FAISS

Scalable Search With Facebook AI



LLM is a huge **Transformer** (Encoder | Decoder | Both) Model  
that is **trained** with massive amount of data for one or more  
**NLP tasks.**

# Transformers/LLM Timeline



GenAI products can be:

- Proprietary Models
- Open Source Models

Open source models are mostly **open source weights**.

The data used to train the models or the source code of model and training algorithms may not be public.

Licenses may permit usage only for research purposes

Some models permit more permissible **commercial usage**.  
(Llama 2, Mistral 7B, Llama 3)

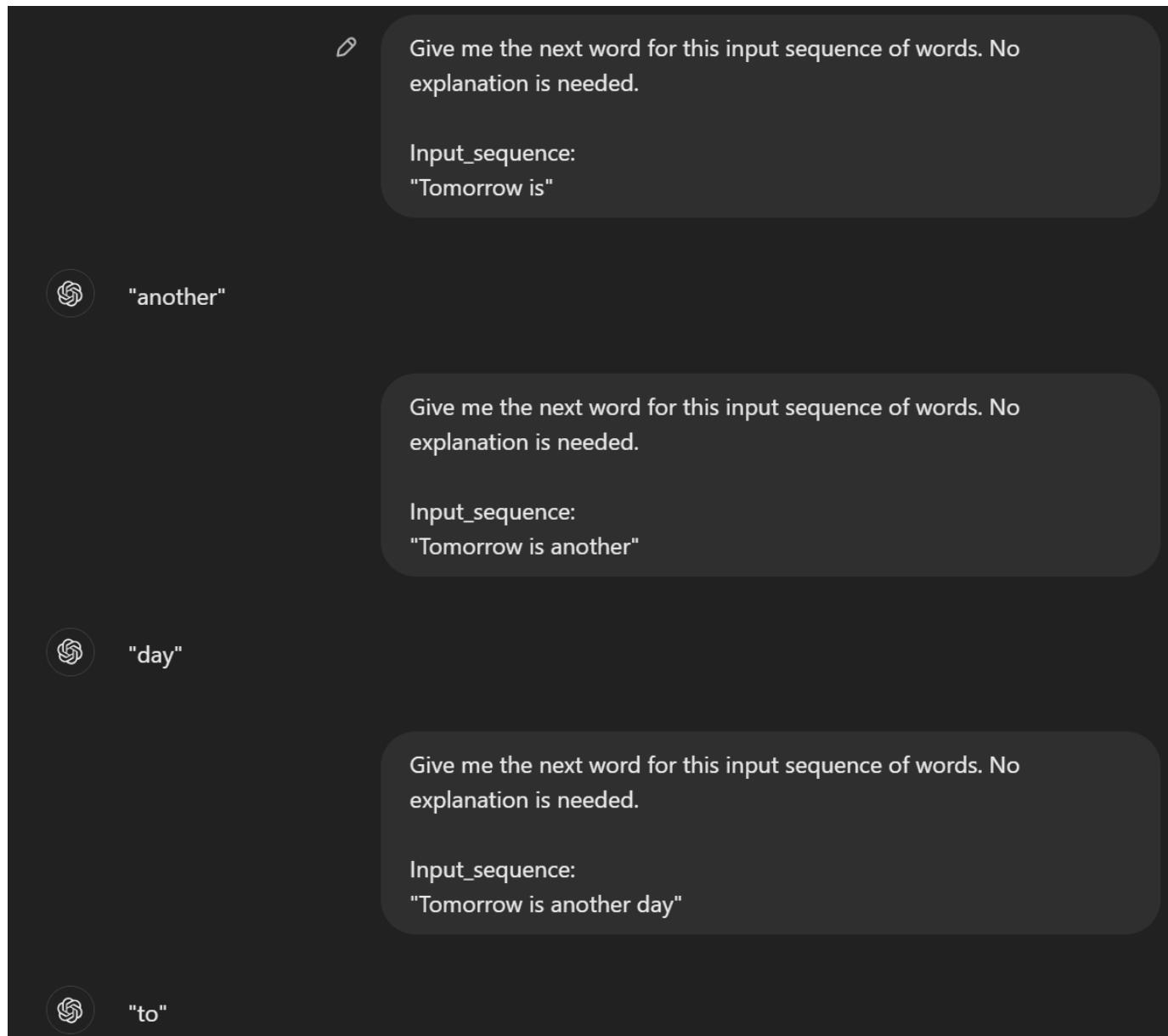
Finetuned models derived from foundation models specialize the functionality

# Key Topic #1 : Language Models

# LMs and LLMs

- Language Model (LM) is a term that has its roots in Natural Language Processing
- LM computes the probability of the next token, given context consisting of current and previous tokens.
  - E.g. Given two context tokens (“bread”, “and”), most of us may predict the next word to be “butter” as the phrase “bread and butter” is quite common.
  - Similarly, one can complete phrases: cout << , SELECT \*, etc
- LMs can generate text. That is, given the context tokens, an LM can generate future words. **This is called an autoregressive model.**
  - GPTs are autoregressive

# Illustration of an Autoregressive generation (GPT)



- Autoregressive models are trained on a huge corpus of data, say all Wikipedia contents, web scraped text, public documents etc.
- The training data is formed by “tokenizing” sentences to words (or sub words) to form different subsequences.
- For each subsequence, the next word in that subsequence constitutes the target for the purpose of training.
- This is a form of unsupervised training, sometimes also termed as semi supervised.
- This training data can be used to train a sequence model (such as a RNN, LSTM or a Transformer)

# Semi Supervised Learning

- Training a LM is a semi supervised learning task
- Start with a huge corpus of text data. Tokenize each sequence (say, a sentence) and further tokenize it into basic tokens (sub words).
- Form a dataset such that the word  $w_{i+1}$  is the target for the subsequence  $w_0 \dots w_i$
- Example: A special flight was arranged after the players were stranded in a hotel for three days after the victory.
  - The above example has 19 words excluding special tokens like SOS, EOS. We can form as many sequences and define targets.
  - Inputs: ["A", "A special", "A special flight", "A special flight was", ...]
  - Targets: ["special", "flight", "was", "arranged", ...]

# Large Language Models (LLM)

- LLMs are based on similar principle as LMs – that is predict a word, given context words.
- LLMs are termed “large” due to the size of the model in terms of number of parameters and also the number of tokens that are used to train the model.
- An LLM like ChatGPT takes in a sequence as input and produces a sequence of tokens as output. Thus, LLMs are sequence to sequence models like transformers.
- GPT stands for Generative Pretrained Transformer, where the pretraining is done on LM objective.
- A “small” LLM like Tiny Vicuna 1B model has 1B parameters and trained using hundreds of billions or even trillions of tokens, while a model like GPT-4 is rumored to have 1.76 Trillion parameters.

# The Language Model Problem

- The LM problem is to assign a probability to a word sequence

- $P(w_1, w_2, \dots, w_n)$
- Example:  $P(\text{tomorrow}, \text{is}, \text{a}, \text{holiday})$



- This may also be stated as a problem of computing the probability of each word given its preceding context.

- $P(w_i | w_1, w_2, \dots, w_{i-1})$
- Example:  $P(\text{holiday} | \text{tomorrow}, \text{is}, \text{a})$

- Language Models can generate text (Auto regressive)

The key question is, how do we implement a machine that takes the context and provides a meaningful completion?

# Language Model Implementation

- Lexical Based: n-gram models
  - Limited Context (Bigram, Trigram)
- Neural Networks based (Bengio, 2003)
  - A small window of context (e.g. 3 words of context)
- Deep networks based (RNN, LSTM)
  - A larger context window (tens of tokens)
- **Transformers based**
  - Much bigger context (typically 4K or above), captures correlations between tokens more effectively. BERT and GPT are the 2 popular transformer based architectures.

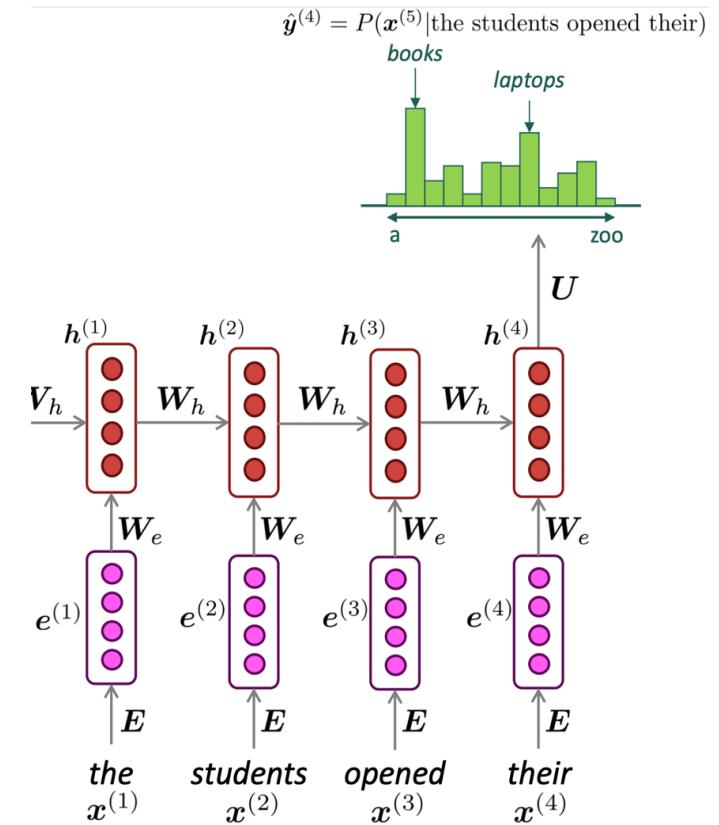
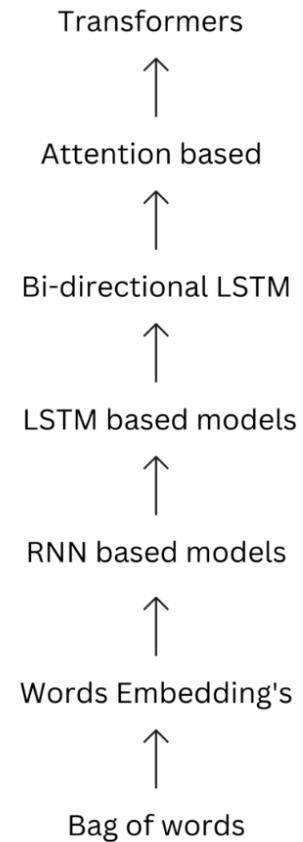
# N-gram Model

e.g. Have a good day!

- Unigram: Words in the given sequence are independent of each other
- Bigram: A word at position t depends only on the one which is at position t-1 and given this, it is independent of all other words in the sequence. (First order Markov assumption)
- Trigram: A word at position t depends only on the one which is at position t-1 and the one at position t-2 and given these, it is independent of all other words in the sequence. (Second order Markov assumption)
- These assumptions are needed to make LM computation tractable

# LM Implementation using DL Models

- Language Models can be implemented using deep learning networks, such as RNN, LSTM, Transformer networks
- These are covered later in this presentation



# Language Models and Embedding

- Embedding provides a numerical representation of a token, typically as a vector.
- A token could be a word, sub word, special words, or even characters. For images, a token may represent a patch  $m \times n$  of an image.
- Typically, this embedding is obtained by training a deep learning model on a LM objective.
- Thus, embedding is one of the applications of Language Modeling
- Other applications of LM include speech recognition, text completion, NLP tasks like Machine Translation and so on.

# Transformers Based Embedding

- The State Of The Art (SOTA) embedding models are Transformer based.
- BERT and GPT architectures created the initial breakthroughs
- The success of such models were key to the evolution leading to ChatGPT
- We cover the technical architecture of BERT/GPT after discussing Transformers

# Tokenization and Embedding (takeaways are highlighted in gray)

- Natural Language text is represented in lexical form by characters, words, sentences, paragraphs etc.
- We could view these lexical units as an ordered sequence of tokens.
  - Tokens could be characters, sub words, words, etc.
- **The process of splitting a natural language text in to tokens is called tokenization.**
- **Embedding Model transforms each token to an embedding vector**, so that they can be further processed by a DL Model.
- Tokenization is not an unique process, usually it depends on the model we are using. E.g. BERT token set is different from that of GPT 4

# HuggingFace Example

- While we focus mostly on text, the data creation process for any modality (image, video, audio) have some common traits.
  - Text, use a [Tokenizer](#) to convert text into a sequence of tokens, create a numerical representation of the tokens, and assemble them into tensors.
  - Speech and audio, use a [Feature extractor](#) to extract sequential features from audio waveforms and convert them into tensors.
  - Image inputs use a [ImageProcessor](#) to convert images into tensors.
  - Multimodal inputs, use a [Processor](#) to combine a tokenizer and a feature extractor or image processor.
- HuggingFace provides a set of classes to handle this.

# Autotokenizer

(Ref: [https://huggingface.co/docs/transformers/en/autoclass\\_tutorial](https://huggingface.co/docs/transformers/en/autoclass_tutorial))

```
>>> from transformers import AutoTokenizer  
  
>>> tokenizer = AutoTokenizer.from_pretrained("google-bert/bert-base-uncased")
```

Then tokenize your input as shown below:

```
>>> sequence = "In a hole in the ground there lived a hobbit."  
>>> print(tokenizer(sequence))  
{'input_ids': [101, 1999, 1037, 4920, 1999, 1996, 2598, 2045, 2973, 1037, 7570, 10322, 4183, 1012, 102],  
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

# Demo: BERT Tokenizer, Embedding

```
[ ]: import torch

[ ]: # from pytorch_pretrained_bert import BertTokenizer, BertModel
from transformers import BertTokenizer, BertModel

[ ]: tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased') # ("bert-base-uncased")

[ ]: txt = "I love Adobe Photoshop"
txt1 = "[CLS] " + txt + " [SEP]"
tokenized_text = tokenizer.tokenize(txt1)

[ ]: with open("vocab.txt", "w", encoding="utf-8") as f:
    for token in tokenizer.vocab.keys():
        f.write(token + "\n")

[ ]: list(tokenizer.vocab.keys())[48947]

[ ]: # Map the token strings to their vocabulary indeces.
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)

# Display the words with their indeces.
for tup in zip(tokenized_text, indexed_tokens):
    print('{:<12} {:>6}'.format(tup[0], tup[1]))

[ ]: # Mark each of the tokens as belonging to sentence "1".
segments_ids = [1] * len(tokenized_text)

print (segments_ids)

[ ]: # Convert inputs to PyTorch tensors
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

[ ]: # Load pre-trained model (weights)
model = BertModel.from_pretrained('bert-base-uncased',
                                    output_hidden_states = True, # Whether the model returns all hidden-states.
                                    )

# Put the model in "evaluation" mode, meaning feed-forward operation.
model.eval()
```

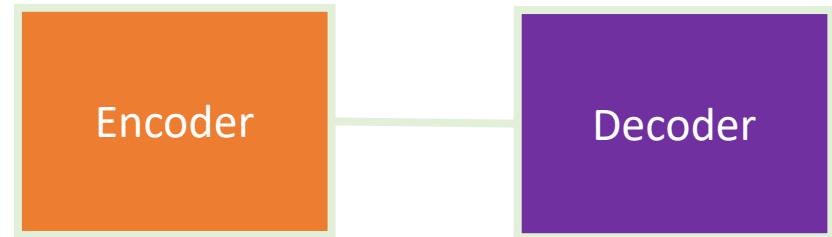
# Hands On: Autotokenizer

- Task#2 Objective: Familiarize yourself with Tokenizers
  - (a) Use the Autotokenizer from HuggingFace and create an instance of Llama 3 tokenizer.
  - (b) Tokenize using `tokenizer.tokenize()` method for the text:

“Amoxicillin is used to treat a wide variety of bacterial infections. This medication is a penicillin-type antibiotic.”
  - (c) Determine the following by writing a suitable program:
    - Vocabulary size of Llama 3 tokenizer
    - Special tokens used in a given sentence e.g. in BERT [CLS] and [SEP] are used.
    - Dimensions of the embedding matrix

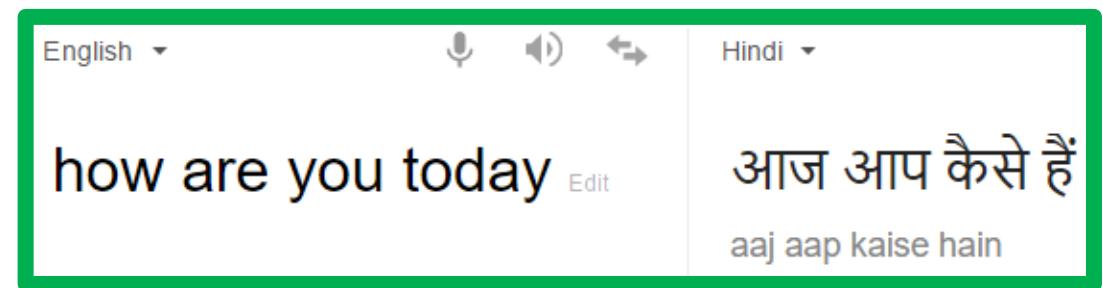
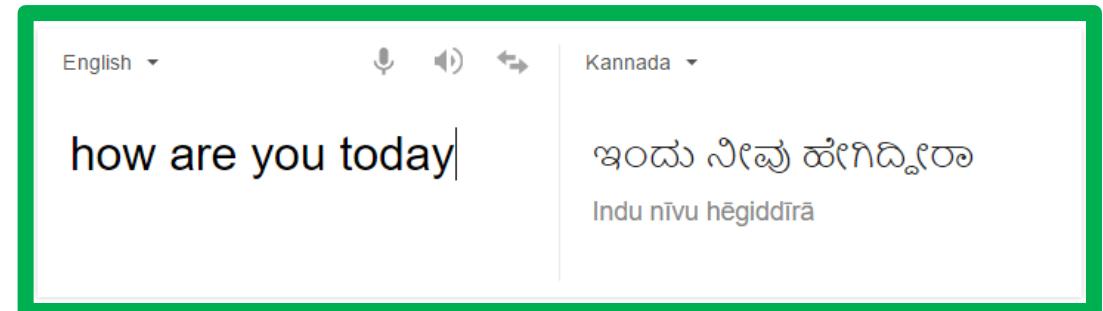
# Encoder Decoder Design

- Use 2 RNN's, one for encoding and the other decoding
  - Example: Machine Translation
- The activations of the final stage of the encoder is fed to the decoder
- Useful when the output sequence is of variable length and the entire input sequence can be processed before generating the output



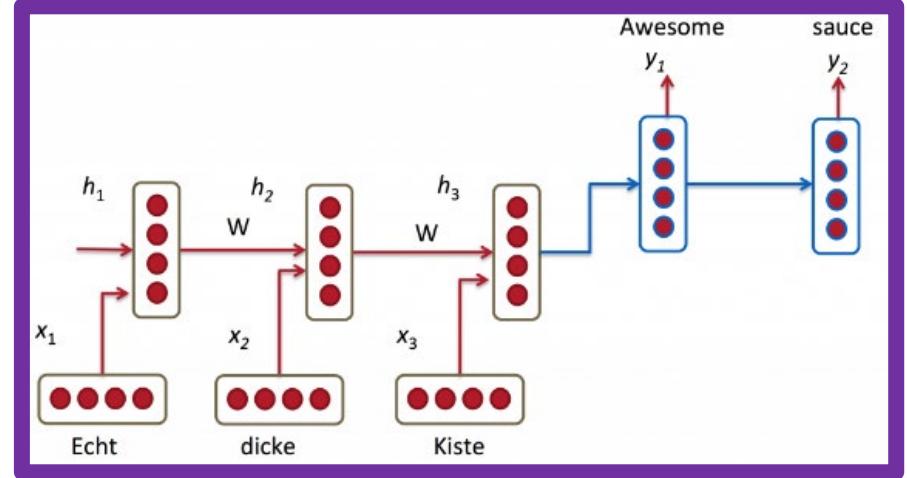
# What are attention networks?

- Suppose we have a sequence of input and we are to produce a sequence of outputs for this.
  - E.g. The inputs could be a sequence of words, could also be image segments
- Attention mechanisms focus on a narrow region of the input when making a classification decision for the output sequence.
  - E.g. In a machine translation, when outputting a word, the attention network might give higher weightage to certain words in the input.



# Neural Machine Translation

- The source sentence is encoded as a single fixed length vector which is then translated in to the target language sentence.
- The encoding process is equivalent to learning a higher level meaning of the sentence, which can't be effectively captured using techniques like n-grams
- The length of input sequence is arbitrary, it is not always effective to encode sentence with same length vector.
  - Suppose you ask an LLM to summarize a big paragraph, is it efficient to use a single vector as the context?
  - Given a large paragraph and a question as shown in the figure on the right, can the model write the required answer?
- Attention mechanisms address the issue by using the hidden states of every step in the encoder to compute an attention vector for the decoder.



The Tribhuvan Museum contains artifacts related to the King Tribhuvan (1906-1955). It has a variety of pieces including his personal belongings, letters and papers, memorabilia related to events he was involved in and a rare collection of photos and paintings of Royal family members. The Mahendra Museum is dedicated to king Mahendra of Nepal (1920-1972). Like the Tribhuvan Museum, it includes his personal belongings such as decorations, stamps, coins and personal notes and manuscripts, but it also has structural reconstructions of his cabinet room and office chamber. The Hanumandhoka Palace, a lavish medieval palace complex in the Durbar, contains three separate museums of historic importance. These museums include the Birendra museum, which contains items related to the second-last monarch, Birendra of Nepal.

When did Tribhuvan die?

# Attention Mechanism

- When using the attention mechanism we make use of the hidden vectors computed at each time step of the encoder and hence the decoding doesn't need to depend on one single hidden vector
- The decoder “attends” to different parts of the source sentence at each step of the output generation.
- The model learns what to attend to based on the input sentence and what it has produced so far.
  - E.g. In the example of English to Kannada translation, the last word “today” translates in to the first word “indu” in Kannada. The attention model learns to focus on “today” when generating “indu”

# Attention Mechanism

- $X_i$  are the input words, processed by the encoder.  $Y_i$  are the outputs, the translated words
- The output  $Y_t$  depends not only on the previous hidden state value but also on the weighted combination of all input states
- $\alpha$ 's are weights that determine how much of a given input word should be considered in computing the output at a given time step
- $\alpha$ 's are normalized to 1 using a softmax

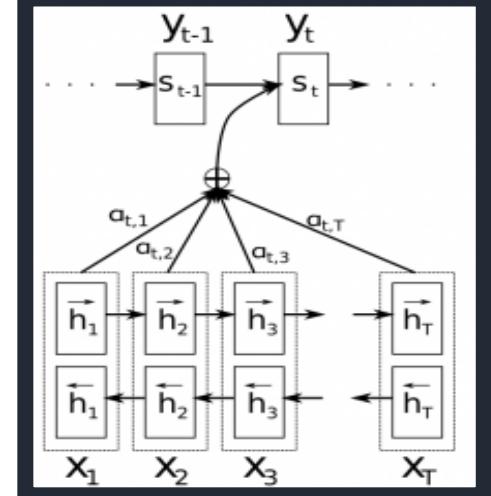
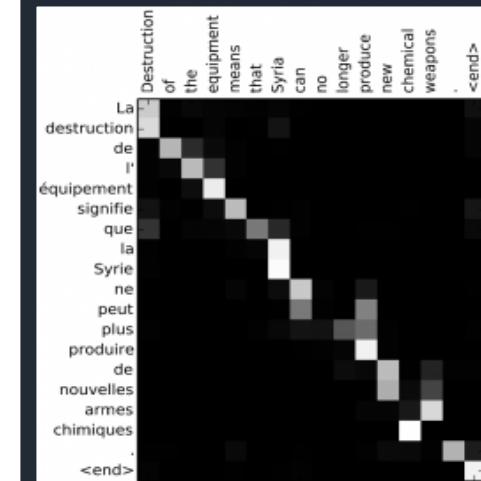
## NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau

Jacobs University Bremen, Germany

KyungHyun Cho Yoshua Bengio\*

Université de Montréal



# Illustration of Self Attention – Coreference Resolution

The animal didn't cross the street because it was too tired .

The word "it" is highlighted in blue. Light blue arrows originate from "it" and point to each word in the sentence: "animal", "didn't", "cross", "the", "street", "because", "it", "was", "too", "tired", and the final period.

The animal didn't cross the street because it was too wide .

The word "it" is highlighted in blue. Light blue arrows originate from "it" and point to each word in the sentence: "animal", "didn't", "cross", "the", "street", "because", "it", "was", "too", "wide", and the final period.

# Self Attention for NLP Tasks

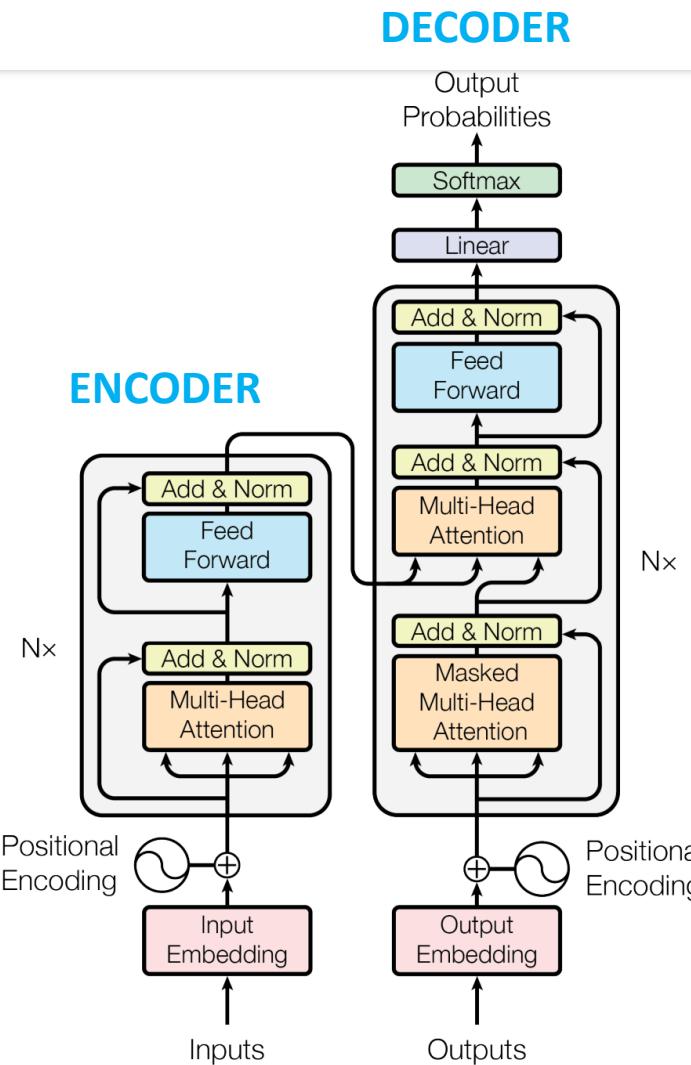
- Coreference Resolution
  - Sachin played well. The legend got the “man-of-the-match” award.
- Anaphora/Cataphora resolution
  - Sachin played well and he was awarded with the trophy.
  - As he was preparing to face the first ball from the fiery fast bowler Dennis Lillie, Sunil Gavaskar appeared an embodiment of concentration.
  - When he wears that blue jersey he will be a different Hardik Pandya because we know he can score those runs and take those wickets
- Word sense disambiguation
  - Ganga was furious today as there were heavy rains due to the cyclone.

# Part 2: Transformers - A Swiss Army Knife for Language Tasks

# Transformers are transformative

- Transformer Networks are excellent at capturing patterns in long sequences
  - Self Attention capturing pairwise influences between tokens in a sequence, mitigation of vanishing gradient issues
- They are scalable to huge datasets
  - Model capacity due to a multilayer architecture, Ability to process tokens in parallel
- Extends beyond NLP applications
  - CNNs capture correlation of pixels in a small kernel. Vision Transformers capture long range dependencies. Imagine the ability to take in to account the influence of a pixel at the top left of an image with those at the bottom right.
- ImageNet moment for NLP: You can download a pretrained model and finetune with custom data.
  - Several papers published with corresponding pretrained models that can be finetuned.
- Model hubs like Hugging Face's enabled easy and consistent access to models and datasets.
  - Number of transformer based models published in Hugging Face exploded.

# Architecture



- Transformer is an implementation of encoder-decoder architectural pattern.
- The encoder maps an input sequence of symbol representations  $(x_1, \dots, x_n)$  to a sequence of continuous representations  $z = (z_1, \dots, z_n)$ .
- Given  $z$ , the decoder then generates an output sequence  $(y_1, \dots, y_m)$  of symbols one element at a time.
- At each step the model is auto-regressive consuming the previously generated symbols as additional input when generating the next.

# Building Blocks

- Input Embedding
- Positional Encoding
- Encoder Stack (A list of Encoders)
  - Encoder
    - Multi Head Attention (h-heads)
      - Scaled dot product attention
    - Feed Forward Network
    - Layer Norm

- Output Embedding
- Positional Encoding
- Decoder Stack (A list of decoders)
  - Decoder
    - Masked Multi Head Attention (h-heads)
      - Scaled dot product attention
    - Multi Head Attention (h-heads)
      - Scaled dot product attention
    - Feed Forward Network
    - Layer Norm
- Output Stage
  - Linear Layer (logits)
  - Softmax

# Dimensions and Terminology

- Inputs to the transformer are a sequence of **tokens** (say, words or sub words from a piece of text) that are in a symbolic form (lexical)
  - Text => Tokenize => sub\_words
- Tokens are to be represented using an **embedding**
  - Sub\_word =>  $d_{\text{model}}$  dimensional vector
- We denote the dimension of embedding same as model dimension:  $d_{\text{model}}$
- We view the notion of Attention as querying a piece of memory organized as key-value pairs: (**Query, Keys, Values**). So, we need to derive these Q, K, V vectors from input vector by projecting on to  $d_k$ ,  $d_k$ ,  $d_v$  space. E.g. from 512 dimensions to 64 dimensions.
- We denote the dimensions of Query, Key, Value to be:  $d_k$ ,  $d_k$ ,  $d_v$
- In the original paper,  $d_{\text{model}} = 512$ ,  $d_k = 64$ ,  $d_v = 64$  and number of heads in the multi head = 8.
- The number of layers in the Encoder (also Decoder) stack = 6

# Embedding

- Embedding associates each token with a vector, termed embedding
- A good embedding should preserve the meaning of the token that should adapt well to the context in which it is used
- A key hyperparameter is the embedding dimension, we set it to  $d_{model} = 512$

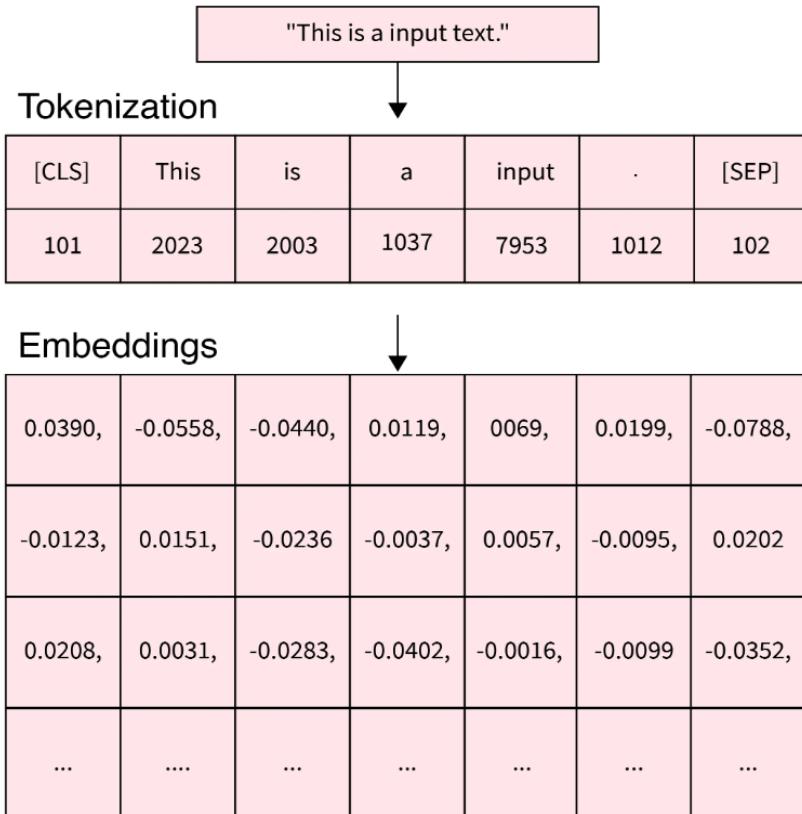
Enter text: Creating embeddings from tokens is the first step for transformers

```
{'input_ids': [101, 4526, 7861, 8270, 4667, 2015, 2013, 19204, 2015, 2003, 1996, 2034, 3357, 2005, 19081, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}'[CLS]', 'creating', 'em', '##bed', '##ding', '##s', 'from', 'token', '##s', 'is', 'the', 'first', 'step', 'for', 'transformers', '[SEP]']
```

```
'adaptive': 19293, 'towns': 4865, 'engineers': 6145, '##hta': 22893, '##raf': 27528, 'version': 254 'aluminium': 14794, '[unused24]': 25, 'dominating': 21949, 'extremes': 28800, 'creaked': 28068, '省' '##dh': 16425, 'census': 2883, 'midst': 12930, 'republicans': 10643, 'burkina': 23089, 'lexie': 241 'conventions': 12472, '34th': 20460, 'goldman': 17765, 'discretion': 19258, 'contrast': 5688, 'cooke' 'ed': 2920, 'grandson': 7631, 'probation': 19703, 'continues': 4247, 'commission': 3222, 'shall': 46 'body': 7955, 'excellent': 6581, 'shortened': 12641, '##fia': 22749, 'penultimate': 25512, '##nir': 2 '': 22852, 'stationed': 8895, 'instructional': 23219, 'enlarged': 11792, '##bat': 14479, 'lauderdale' 'os': 18395, 'core': 4563, 'braden': 27232, 'clutching': 14197, 'experimental': 6388, 'occasionally': 0, 'strap': 16195, 'annexed': 13291, 'aj': 19128, 'jarrett': 24503, '##cha': 7507, 'luca': 15604, 'm' '##tunis': 16255, '##will': 16256, '##book': 17291, '##writing': 20070, '##sail': 17250, '##adventure': 16160, '##'}
```

Vocabulary Size = 30522

# Embedding Illustrated



## Embedding

```
class Embedder(torch.nn.Module):  
    def __init__(self, vocab_size, d_model):  
        super().__init__()  
        self.embed = torch.nn.Embedding(vocab_size, d_model)  
  
    def forward(self, x):  
        # [123, 0, 23, 5] -> [[..512..], [...512...], ...]  
        return self.embed(x)
```

# POS Encoding

- Transformers do not have implicit position information, as the input tokens can be processed in parallel. At the same time, we need to know their position in order to preserve the sequence.
- Hence we encode the position explicitly
- Positional encoding needs to represent a pattern that can be learned by the model.
- POS encoder returns the position as a vector. We need the length of the vector to be  $d_{model}$

## Positional encoding

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

```
: import math

class PositionalEncoder(torch.nn.Module):

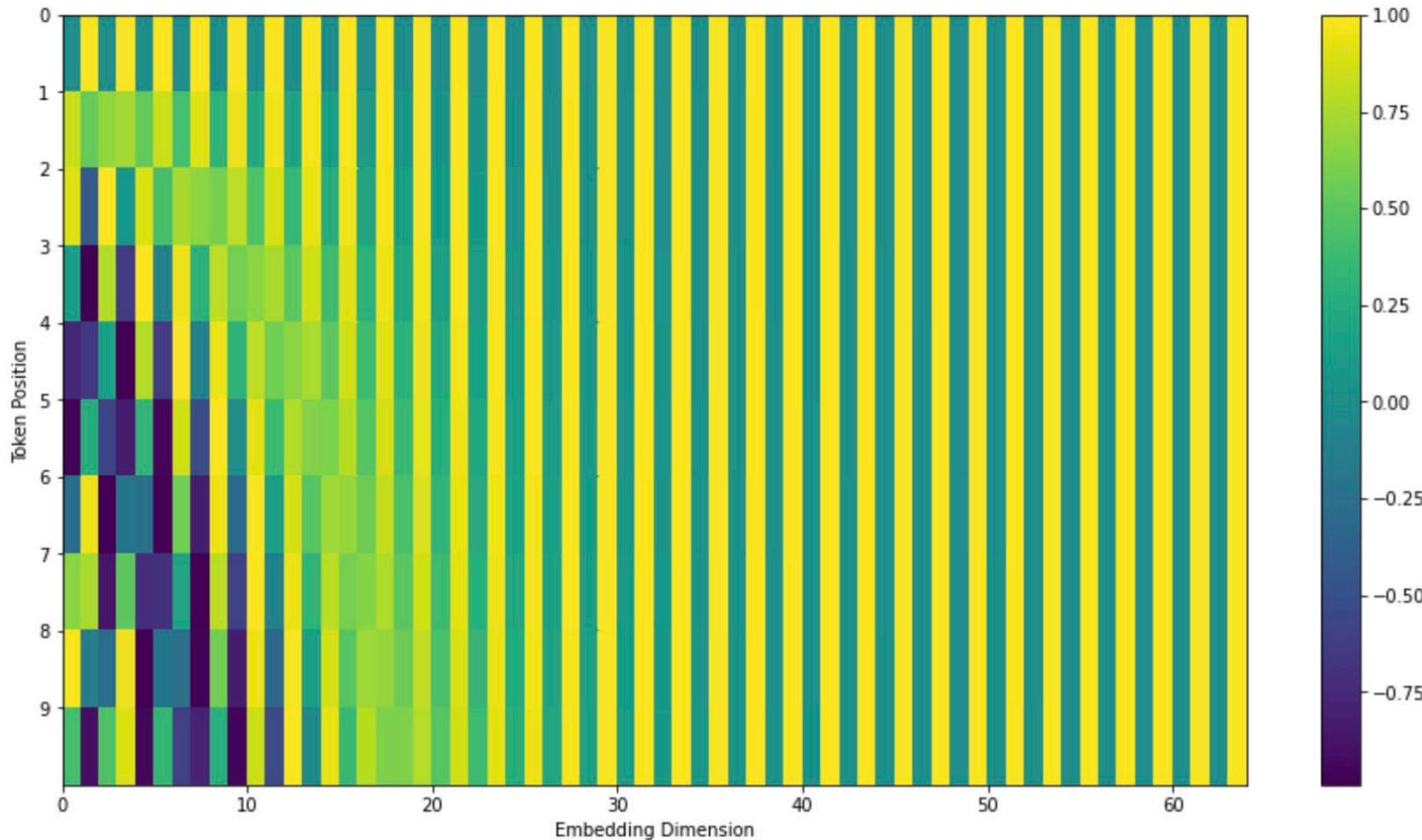
    def __init__(self, d_model, max_seq_len=80):
        super().__init__()
        self.d_model = d_model

        # create constant positional encoding matrix
        pe_matrix = torch.zeros(max_seq_len, d_model)

        for pos in range(max_seq_len):
            for i in range(0, d_model, 2):
                pe_matrix[pos, i] = math.sin(pos/10000**((2*i)/d_model))
                pe_matrix[pos, i+1] = math.cos(pos/10000**((2*i)/d_model))
        pe_matrix = pe_matrix.unsqueeze(0)      # Add one dimension for batch size
        self.register_buffer('pe', pe_matrix)   # Register as persistent buffer

    def forward(self, x):
        # x is a sentence after embedding with dim (batch, number of words, vector dimension)
        seq_len = x.size()[1]
        x = x + self.pe[:, :seq_len]
        return x
```

# POS Encoding Illustrated



# LayerNorm and BatchNorm

- **LayerNorm** normalizes across the features of a single input vector:

Input:  $\mathbf{x}_i = [x_{i1}, x_{i2}, x_{i3}, \dots, x_{id}]$

$$\text{Normalization: } \mu_i = \frac{1}{d} \sum_{j=1}^d x_{ij}, \quad \sigma_i^2 = \frac{1}{d} \sum_{j=1}^d (x_{ij} - \mu_i)^2$$

$$\text{Output: } y_{ij} = \gamma \cdot \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta$$

- **BatchNorm** normalizes across the batch for each feature:

Input:  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$

$$\text{Normalization: } \mu_j = \frac{1}{N} \sum_{i=1}^N x_{ij}, \quad \sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{ij} - \mu_j)^2$$

$$\text{Output: } y_{ij} = \gamma \cdot \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} + \beta$$

- There are different types of normalizations that are used when training a ML model.
  - Feature Normalization, Layer Normalization, Batch Normalization

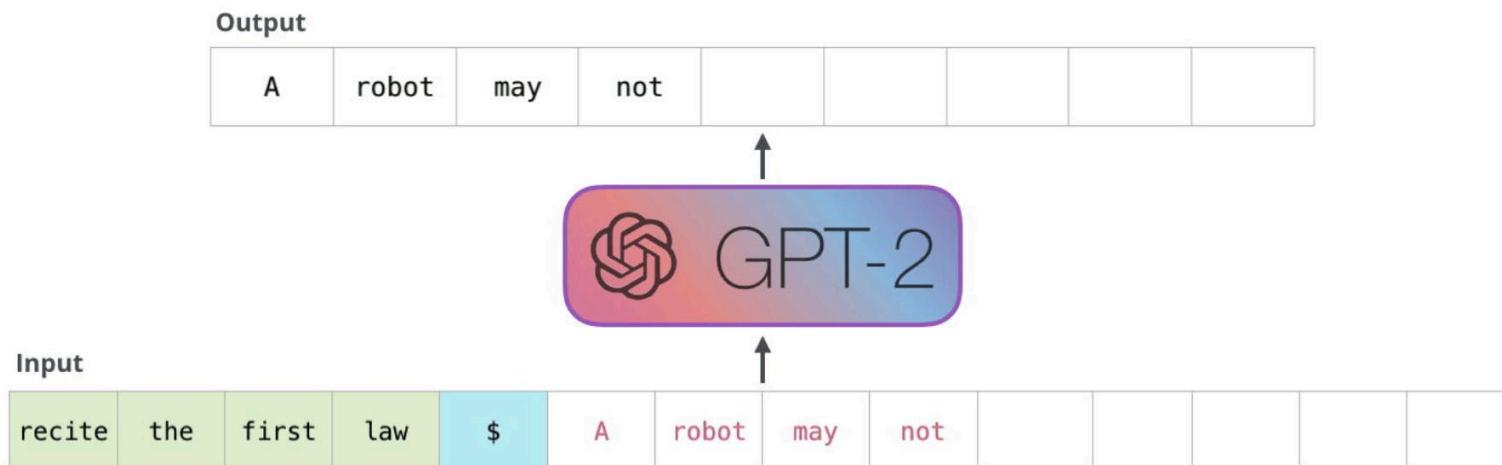
- Transformers use Layer Normalization.

- Normalizations help improve the training stability and enable faster convergence

# Example for illustrating Self Attention

Consider the text:

- First Law of robotics: A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- Second law of robotics: A robot must obey the orders given to **it** by human beings except where **such orders** would conflict with the **First Law**.
- Example Prompt Instruction:



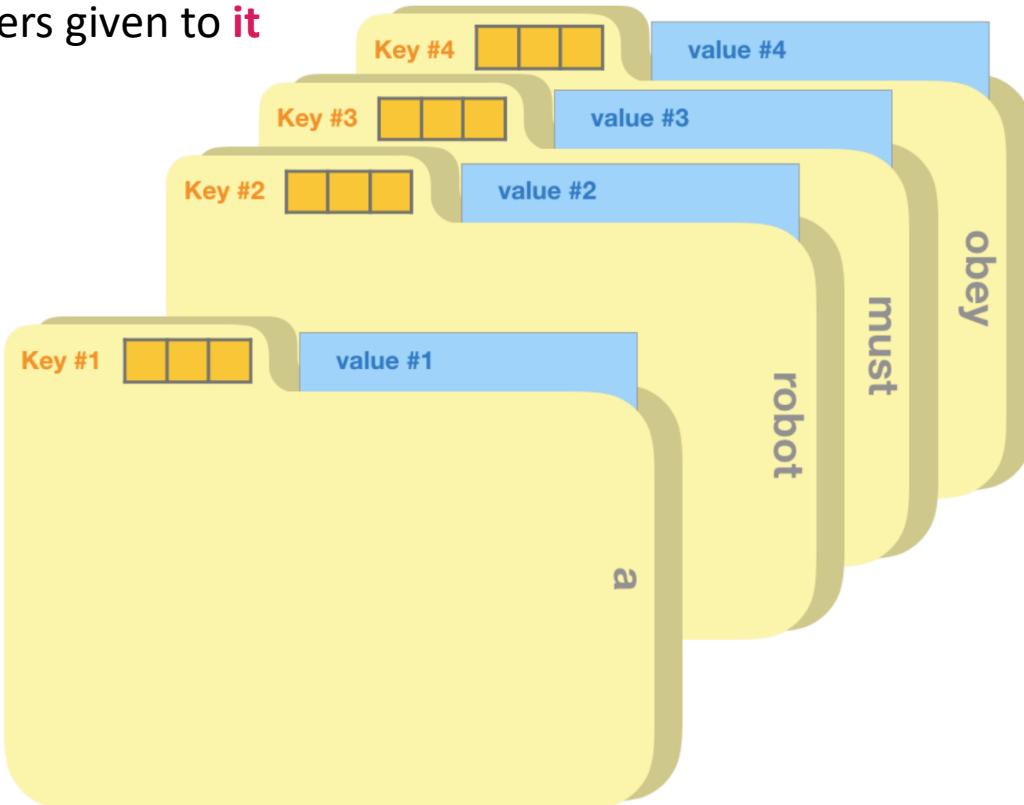
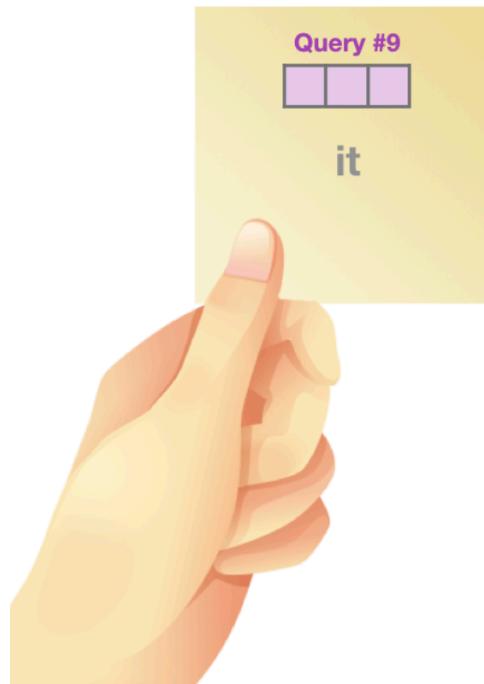
# Example for illustrating Self Attention

- Correlations between tokens in a given sequence
  - Multiple perspectives of looking at these correlations
  - Representing the information in multiple levels of abstraction.
- Consider the text:
- The constitution of India says: "India that is Bharat". Are Indians citizens of Bharat also?
  - To answer the above we need to understand multiple concepts, e.g. Indians are citizens of India, India and Bharat are interchangeable etc.
  - How to learn and generalize across multiple concepts?

# Query, Key, Value

- **Query:** The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
- **Key:** Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
- **Value:** Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

A robot must obey the orders given to **it**



$$\text{Memory} = \{ f_k(k_1): f_v(v_1), f_k(k_2): f_v(v_2), \dots \}$$

$$\text{Query} = f_q(k_i)$$

Taking the dot product between query and memory, we can determine the weight of each token for the given query.

# Feed Forward Network: Notice the linear projections

## Feed Forward layer

```
class FeedForward(torch.nn.Module):
    def __init__(self, d_model, d_ff=2048, dropout=0.1):
        super().__init__()

        self.linear_1 = torch.nn.Linear(d_model, d_ff)
        self.dropout = torch.nn.Dropout(dropout)
        self.linear_2 = torch.nn.Linear(d_ff, d_model)

    def forward(self, x):
        x = self.dropout(F.relu(self.linear_1(x)))
        x = self.linear_2(x)
        return x
```

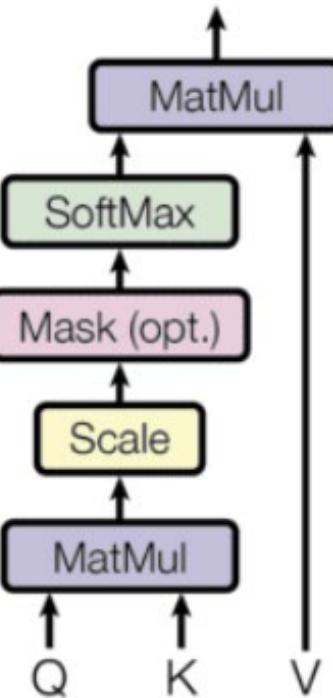
# Computing Self Attention

Q/K	hello	how	are	you
hello				
how				
are				
you				

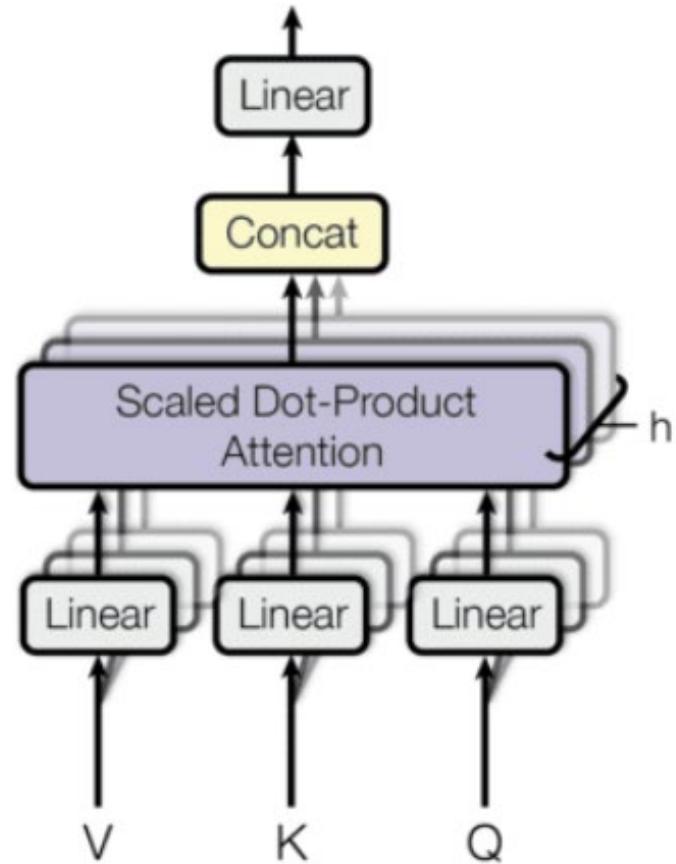
- Our goal is to capture the correlations between each token with every other token in the given sequence
- We take each token, project it on to a query plane, key plane, value plane: That is, provide 3 different representations
- We compute the strength of each query vector with each key vector and scale the value vectors using this strength

# Scaled Dot Product Attention

Scaled Dot-Product Attention



Multi-Head Attention



# What is Self-Attention?

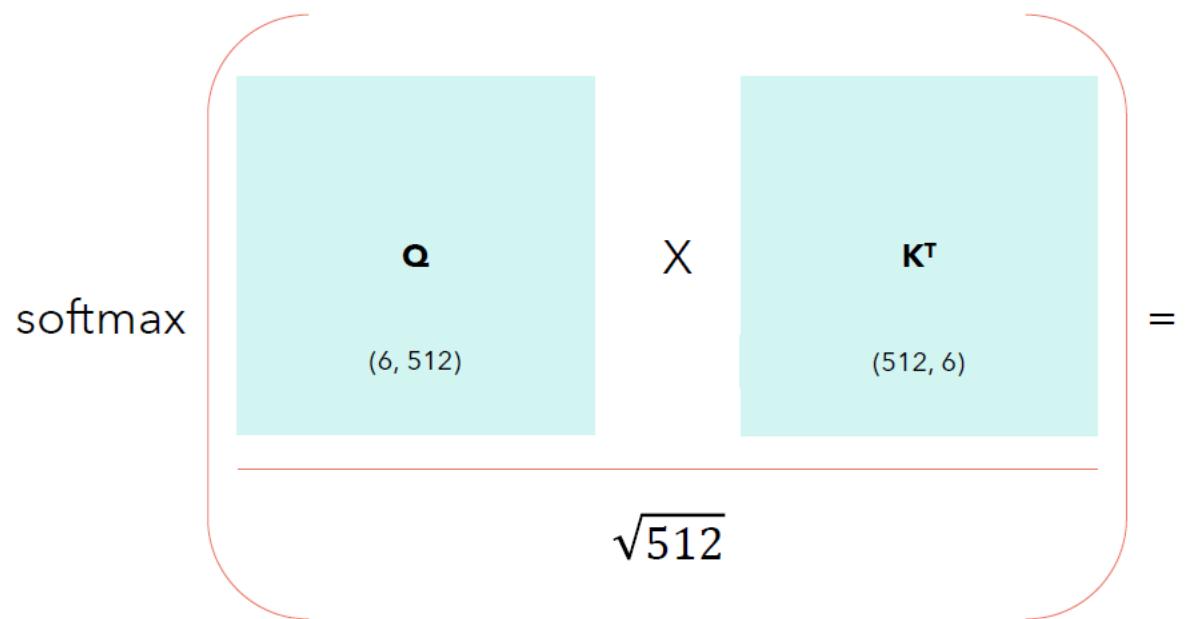
Fig Credits: Umar Jamil

D<sub>k</sub> is 64 in the original paper

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

In this simple case we consider the sequence length **seq** = 6 and **d<sub>model</sub>** = **d<sub>k</sub>** = 512.

The matrices **Q**, **K** and **V** are just the input sentence.



\* for simplicity I considered only one head, which makes  $d_{\text{model}} = d_k$ .

	YOUR	CAT	IS	A	LOVELY	CAT	$\Sigma$
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	<b>1</b>
CAT	0.124	0.278	0.201	0.128	0.154	0.115	<b>1</b>
IS	0.147	0.132	0.262	0.097	0.218	0.145	<b>1</b>
A	0.210	0.128	0.206	0.212	0.119	0.125	<b>1</b>
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	<b>1</b>
CAT	0.195	0.114	0.203	0.103	0.157	0.229	<b>1</b>

\* all values are random.

(6, 6)

# How to compute Self-Attention?

Fig Credits: Umar Jamil

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

(6, 6)

$$\begin{matrix} \times & \mathbf{V} & = & \text{Attention} \\ & (6, 512) & & (6, 512) \end{matrix}$$

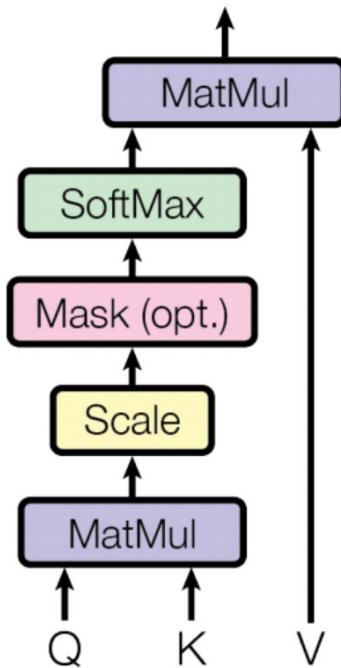
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Each row in this matrix captures not only the meaning (given by the embedding) or the position in the sentence (represented by the positional encodings) but also each word's interaction with other words.

# Scaled Dot Product Attention

## Model layers

### Scaled Dot-Product Attention layer



```
import math
import torch.nn.functional as F

# Given Query, Key, Value, calculate the final weighted value
def scaled_dot_product_attention(q, k, v, mask=None, dropout=None):
    # Shape of q and k are the same, both are (batch_size, seq_len, d_k)
    # Shape of v is (batch_size, seq_len, d_v)
    attention_scores = torch.matmul(q, k.transpose(-2, -1))/math.sqrt(q.shape[-1]) # size (batch_size, seq_len, seq_len)

    # Apply mask to scores
    # <pad>
    if mask is not None:
        attention_scores = attention_scores.masked_fill(mask == 0, value=-1e9)

    # Softmax along the last dimension
    attention_weights = F.softmax(attention_scores, dim=-1)

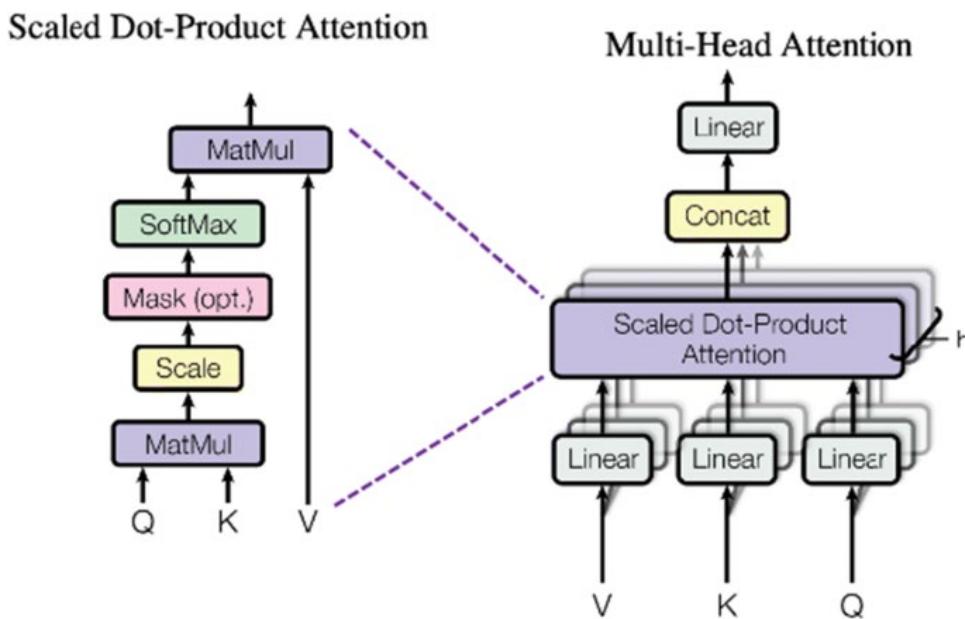
    if dropout is not None:
        attention_weights = dropout(attention_weights)

    output = torch.matmul(attention_weights, v)
    return output
```

# Multi-head Attention

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

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1 \dots \text{head}_h)W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$



```
# Input: Sequence of token embeddings X
Q = Xw^Q # Queries
K = Xw^K # Keys
V = Xw^V # Values

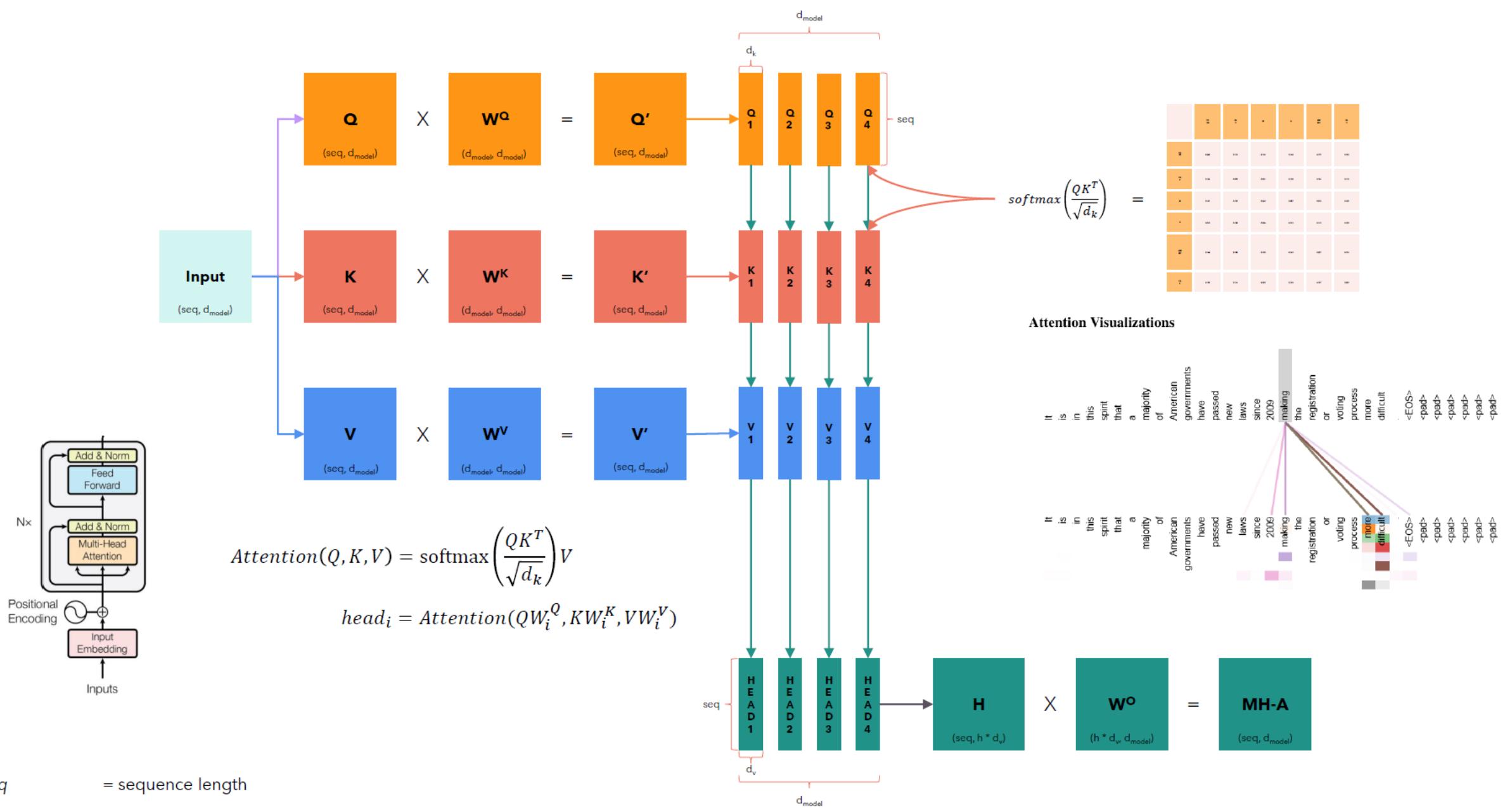
# Attention scores
scores = Q @ K.T / sqrt(d_k)
attention_weights = softmax(scores)

# Weighted sum of values
context = attention_weights @ V

# Multi-head attention (multiple heads concatenated)
multi_head_output = concat([head1, head2, ..., headN]) @ W^O

# Feed-forward network
ffn_output = relu(multi_head_output @ W1 + b1) @ W2 + b2

# Add & Norm
output = LayerNorm(input + ffn_output)
```



**seq** = sequence length

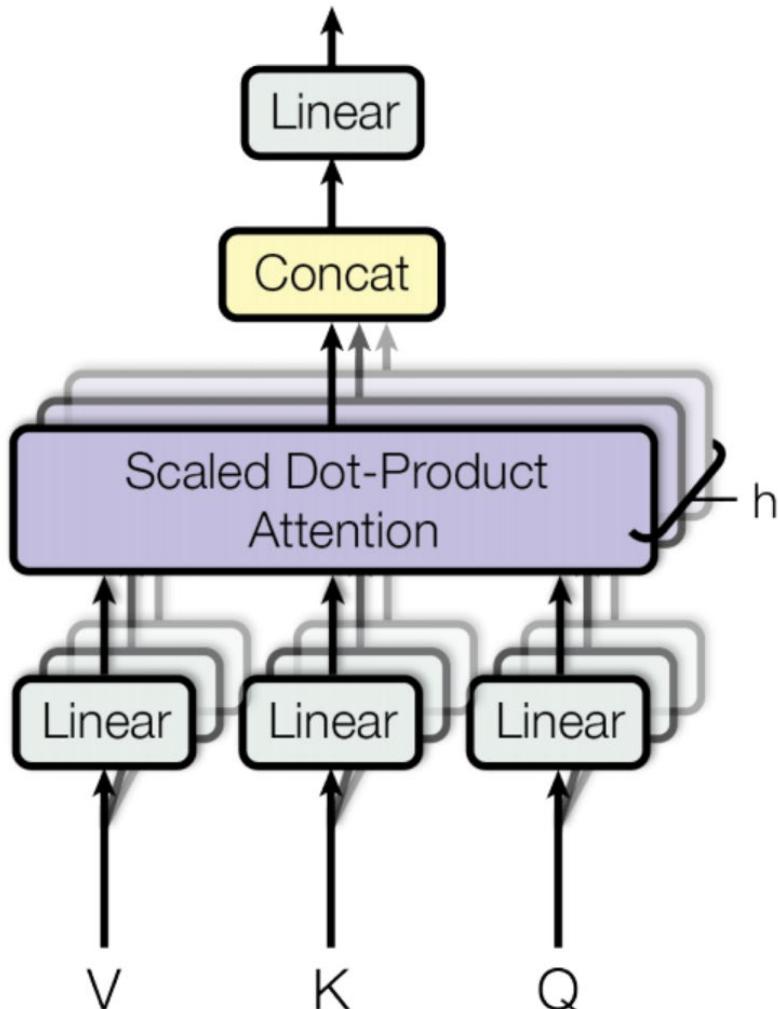
**$d_{\text{model}}$**  = size of the embedding vector

**h** = number of heads

**$d_k = d_v = d_{\text{model}} / h$**

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1 \dots \text{head}_h)W^O$$

# Multi Head Attention



```
class MultiHeadAttention(torch.nn.Module):
    def __init__(self, n_heads, d_model, dropout=0.1):
        super().__init__()

        self.n_heads = n_heads
        self.d_model = d_model
        self.d_k = self.d_v = d_model//n_heads

        # self attention linear layers
        # Linear Layers for q, k, v vectors generation in different heads
        self.q_linear_layers = []
        self.k_linear_layers = []
        self.v_linear_layers = []
        for i in range(n_heads):
            self.q_linear_layers.append(torch.nn.Linear(d_model, self.d_k))
            self.k_linear_layers.append(torch.nn.Linear(d_model, self.d_k))
            self.v_linear_layers.append(torch.nn.Linear(d_model, self.d_v))

        self.dropout = torch.nn.Dropout(dropout)
        self.out = torch.nn.Linear(n_heads*self.d_v, d_model)

    def forward(self, q, k, v, mask=None):
        multi_head_attention_outputs = []
        for q_linear, k_linear, v_linear in zip(self.q_linear_layers,
                                                self.k_linear_layers,
                                                self.v_linear_layers):
            new_q = q_linear(q) # size: (batch_size, seq_len, d_k)
            new_k = k_linear(k) # size: (batch_size, seq_len, d_k)
            new_v = v_linear(v) # size: (batch_size, seq_len, d_v)

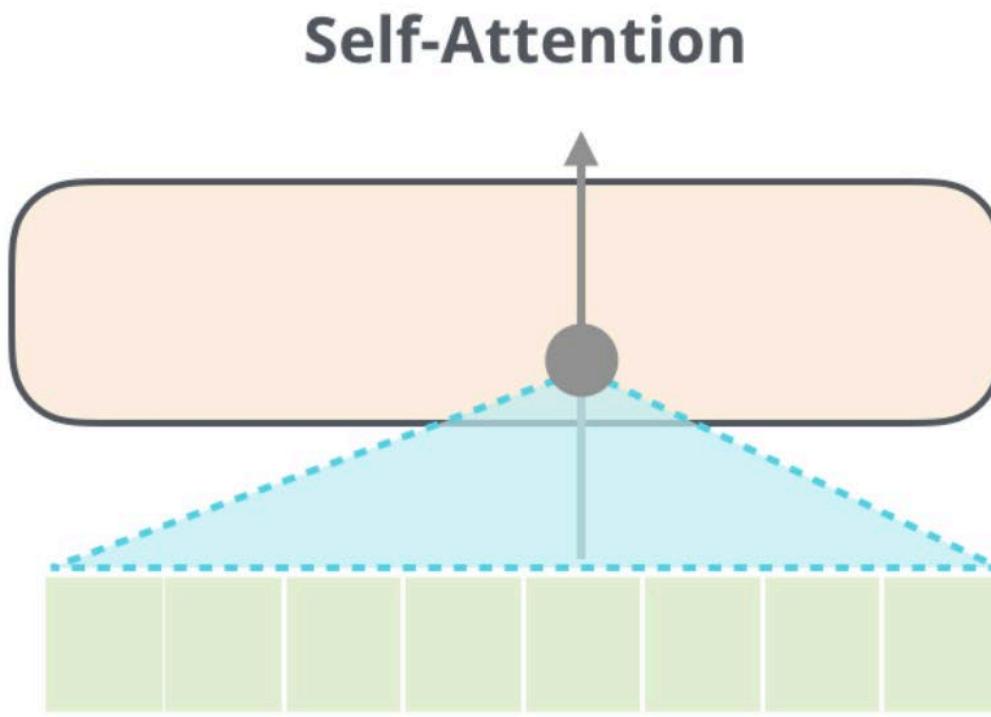
            # Scaled Dot-Product attention
            head_v = scaled_dot_product_attention(new_q, new_k, new_v, mask, self.dropout) # (batch_size, seq_len, d_v)
            multi_head_attention_outputs.append(head_v)

        # Concat
        #import pdb; pdb.set_trace()
        concat = torch.cat(multi_head_attention_outputs, -1) # (batch_size, seq_len, n_heads*d_v)

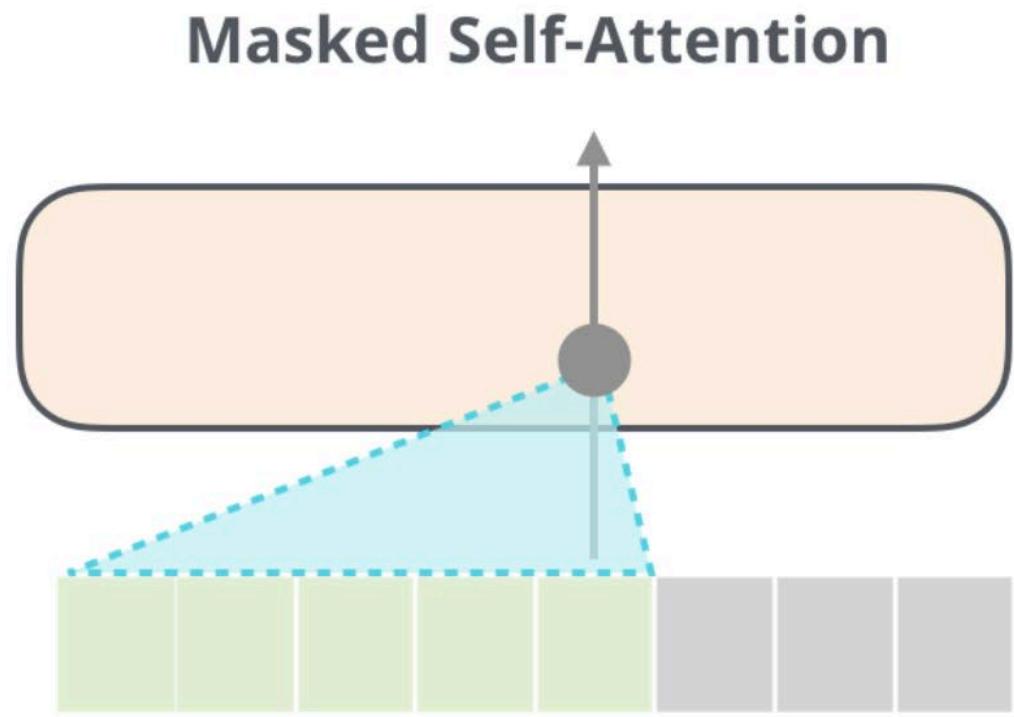
        # Linear Layer to recover to original shape
        output = self.out(concat) # (batch_size, seq_len, d_model)

    return output
```

# Masked Self Attention for Decoder



How are you today?



How are you today?  
How are you today?  
How are you today?

# Feed Forward Network

## Feed Forward layer

```
class FeedForward(torch.nn.Module):
    def __init__(self, d_model, d_ff=2048, dropout=0.1):
        super().__init__()

        self.linear_1 = torch.nn.Linear(d_model, d_ff)
        self.dropout = torch.nn.Dropout(dropout)
        self.linear_2 = torch.nn.Linear(d_ff, d_model)

    def forward(self, x):
        x = self.dropout(F.relu(self.linear_1(x)))
        x = self.linear_2(x)
        return x
```

## Layer Normalization layer

Normalization

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$$

$$\hat{Z}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Add two learnable parameters

$$\tilde{Z}_i = \alpha_i * \hat{Z}_i + \beta_i$$

```
class LayerNorm(torch.nn.Module):
    def __init__(self, d_model, eps=1e-6):
        super().__init__()
        self.d_model = d_model
        self.alpha = torch.nn.Parameter(torch.ones(self.d_model))
        self.beta = torch.nn.Parameter(torch.zeros(self.d_model))
        self.eps = eps

    def forward(self, x):
        # x size: (batch_size, seq_len, d_model)
        x_hat = (x - x.mean(dim=-1, keepdim=True))/(x.std(dim=-1, keepdim=True) + self.eps)
        x_tilde = self.alpha*x_hat + self.beta
        return x_tilde
```

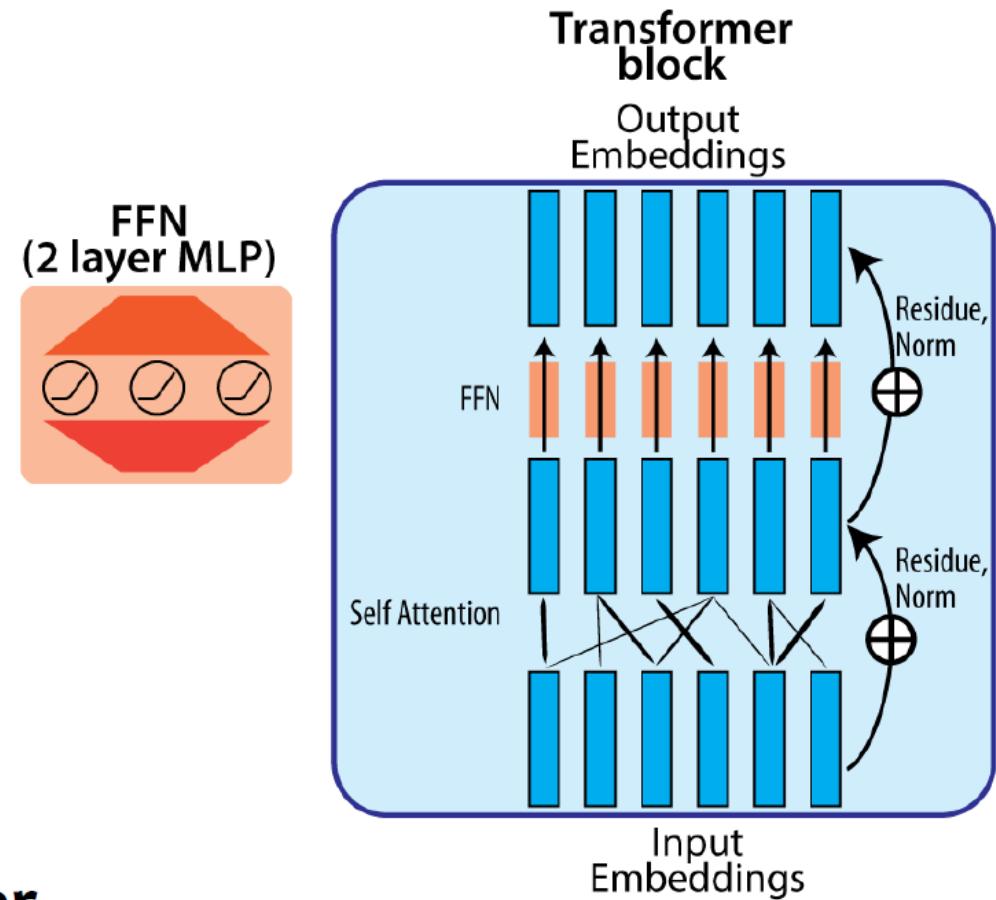
# From Attention to TransformerBlock

- A **transformer block** has

- Self Attention
  - information exchange **between tokens**
- Feed forward network
  - Information transform **within tokens**
  - E.g. a multi-layer perceptron with 1 hidden layer
  - GeLU activation is commonly used.

- Normalization (Layer normalization)

- A transformer model contains  $N \times$  **transformer block**



GELU is an activation function: Gaussian Error Linear Unit

# Rationale

- Why multiple encoder/decoder blocks?
- Why many attention heads?
- Why self attention?
- Why cross attention?

# Variations from base architecture and performance

	$N$	$d_{\text{model}}$	$d_{\text{ff}}$	$h$	$d_k$	$d_v$	$P_{\text{drop}}$	$\epsilon_{ls}$	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)						16				5.16	25.1	58
						32				5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
							0.0			5.77	24.6	
(D)							0.2			4.95	25.5	
							0.0			4.67	25.3	
							0.2			5.47	25.7	
(E)								positional embedding instead of sinusoids		4.92	25.7	
big	6	1024	4096	16			0.3		300K	<b>4.33</b>	<b>26.4</b>	213

# BERT, GPT Architectural Parameters

## 1. BERT-Base:

- Number of layers: 12 (transformer layers)
- Hidden size (dimension of the encoder layers): 768
- Number of attention heads: 12
- Total parameters: Approximately 110 million parameters

## 2. BERT-Large:

- Number of layers: 24 (transformer layers)
- Hidden size (dimension of the encoder layers): 1024
- Number of attention heads: 16
- Total parameters: Approximately 340 million parameters

	GPT-1	GPT-2	GPT-3
Parameters	117 Million	1.5 Billion	175 Billion
Decoder Layers	12	48	96
Context Token Size	512	1024	2048

**Model specifications** Our model largely follows the original transformer work [62]. We trained a 12-layer decoder-only transformer with masked self-attention heads (768 dimensional states and 12 attention heads). For the position-wise feed-forward networks, we used 3072 dimensional inner states. We used the Adam optimization scheme [27] with a max learning rate of 2.5e-4. The learning rate was increased linearly from zero over the first 2000 updates and annealed to 0 using a cosine schedule. We train for 100 epochs on minibatches of 64 randomly sampled, contiguous sequences of 512 tokens. Since layernorm [2] is used extensively throughout the model, a simple weight initialization of  $N(0, 0.02)$  was sufficient. We used a bytewise encoding (BPE) vocabulary with 40,000 merges [53] and residual, embedding, and attention dropouts with a rate of 0.1 for regularization. We also employed a modified version of L2 regularization proposed in [37], with  $w = 0.01$  on all non bias or gain weights. For the activation function, we used the Gaussian Error Linear Unit (GELU) [18]. We used learned position embeddings instead of the sinusoidal version proposed in the original work. We use the `ftfy` library<sup>2</sup> to clean the raw text in BooksCorpus, standardize some punctuation and whitespace, and use the `spaCy` tokenizer.<sup>3</sup>

# Part 3: Transformers for Generative AI

# BERT

- Bidirectional Encoder Representation from Transformers
- Two strategies to apply pretrained language representations to downstream tasks:
  - Feature-based
  - Finetuning
- BERT improves upon finetuning approaches using 2 objectives:
  - Masked Language Model (MLM)
  - Next Sentence Prediction (NSP)

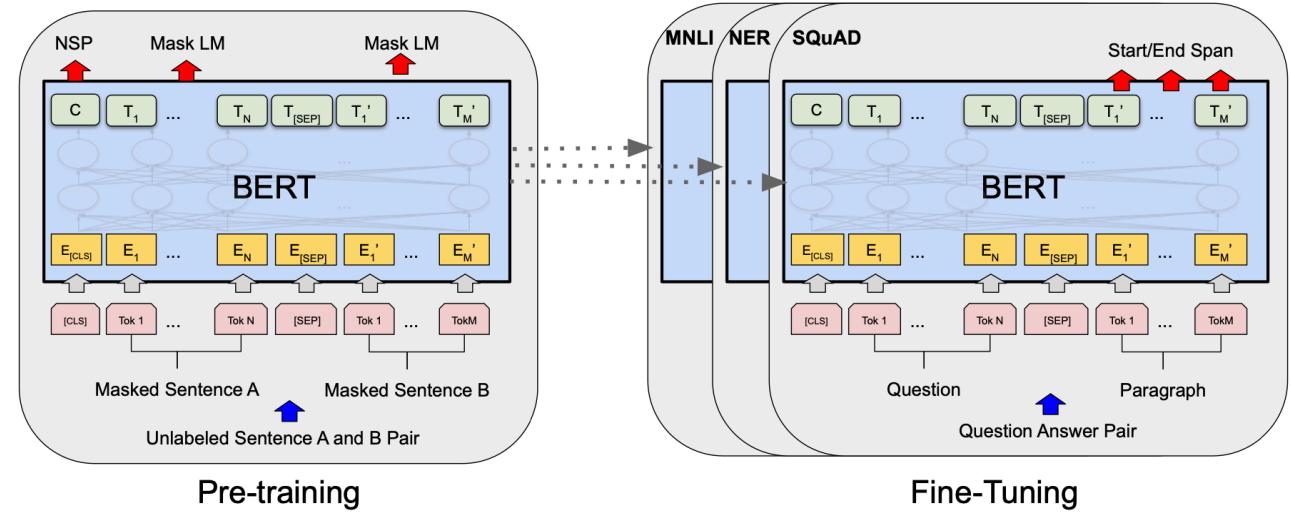


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

# BERT: Vocabulary and Representation

- BERT uses WordPiece embeddings (Wu et al., 2016) with a 30,000 token vocabulary.
- The first token of every sequence is always a special classification token ([CLS]).
- The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.
- Sentence pairs are packed together into a single sequence.
- Sentences are separated in two ways. First, with a special token ([SEP]). Second, using a learned embedding to every token indicating whether it belongs to sentence A or sentence B.

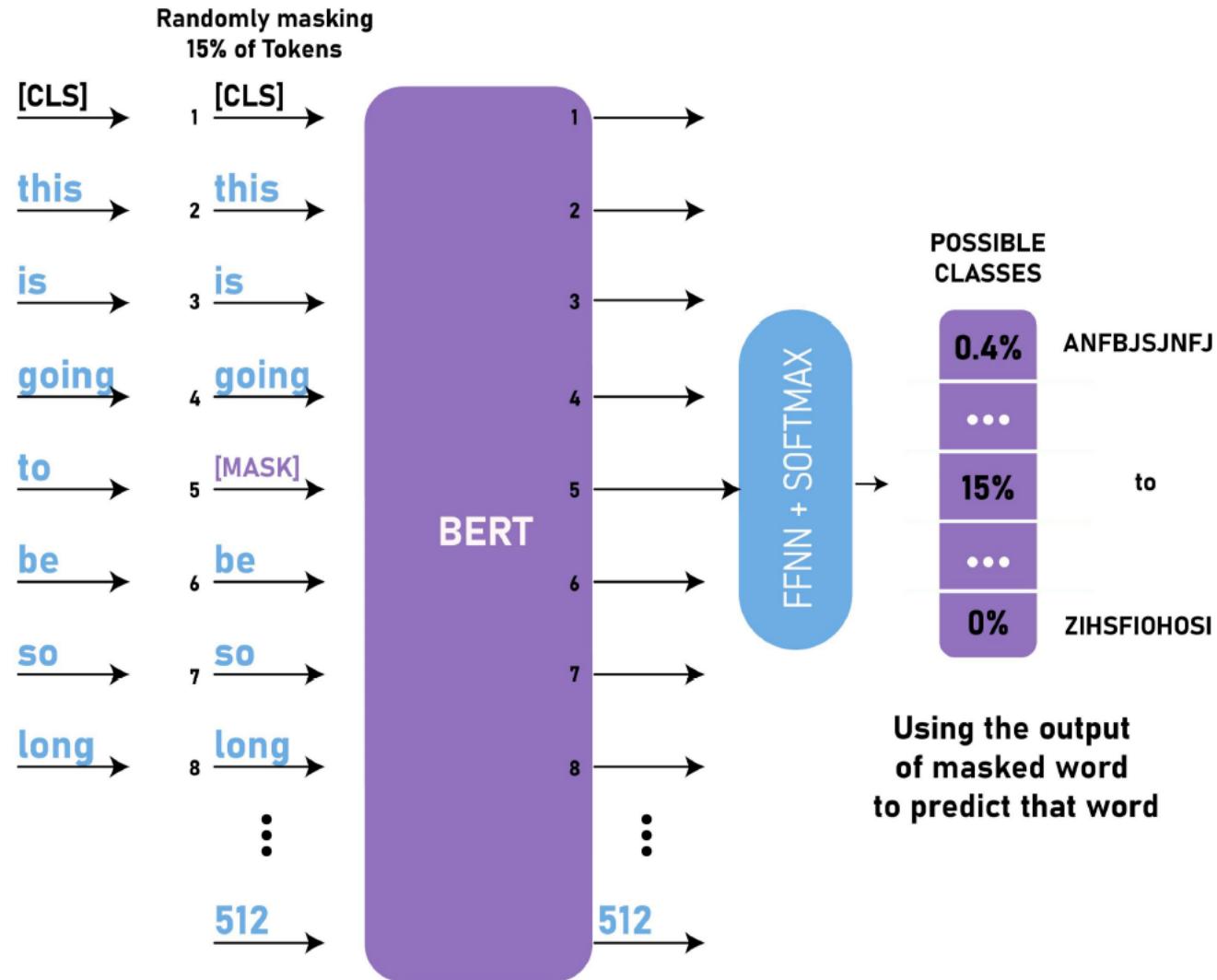
# Masked word prediction / Cloze task:

Fig Credits: Binxu Wang,  
Harvard

BERT = Bidirectional Encoder Representations from Transformers

- Mask out 15% tokens and predict those [MASK].
- **Bidirectional:** Predict the mask token from context on both sides .
- Transformer **Encoder**, all-to-all self-attention
- Suitable for text **representation**

INPUT



# BERT Masked Language Model Training

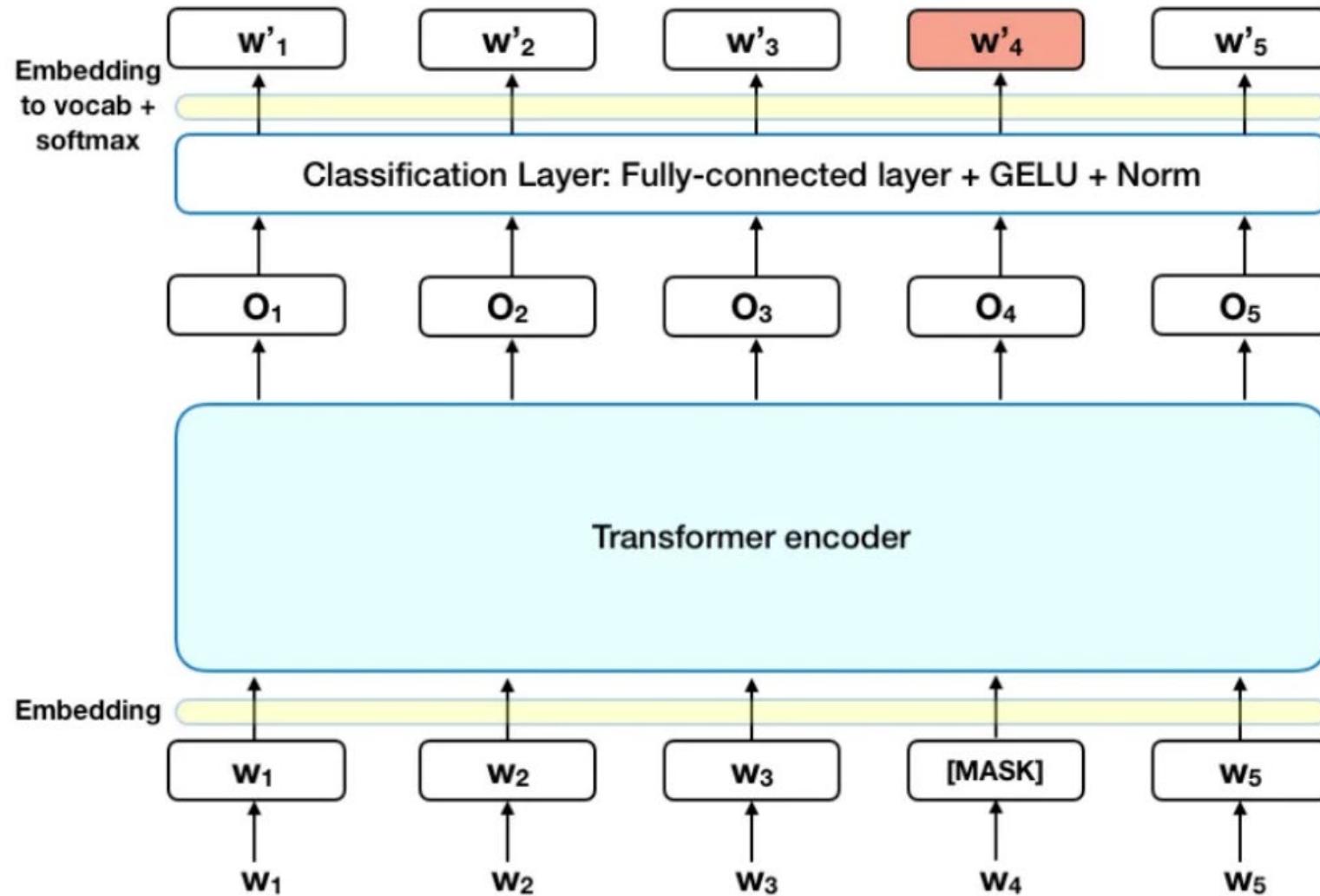


Fig Credits: Bin Xu Wang,  
Harvard

# Next Sentence Prediction

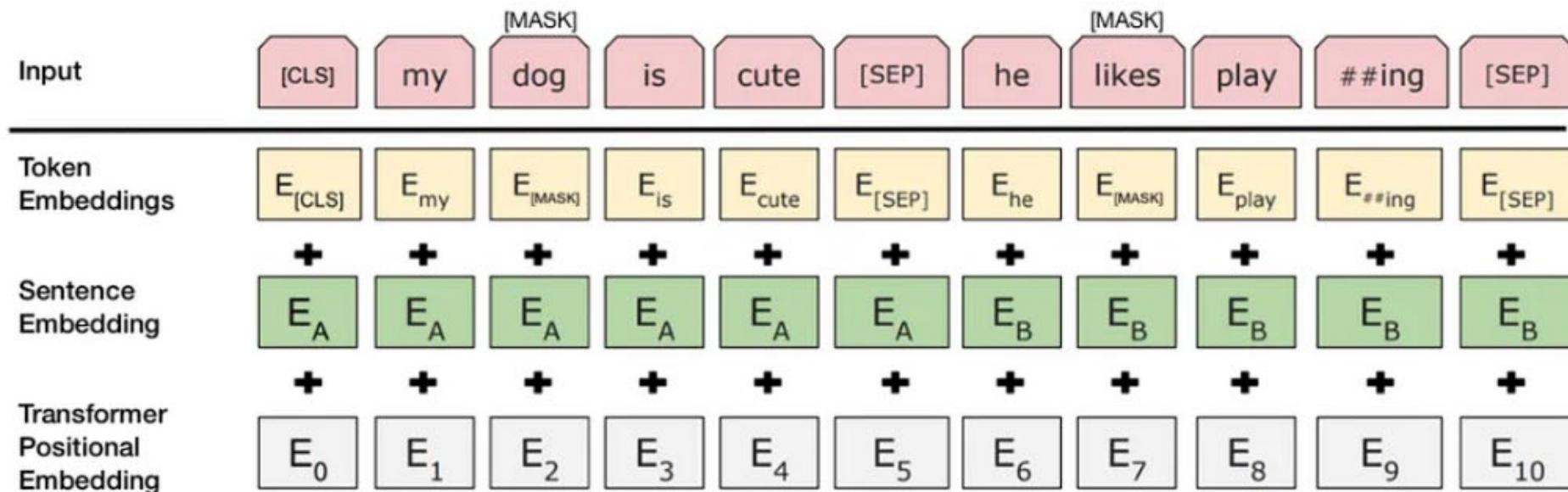
- In the BERT training process, the model receives pairs of sentences as input
- Model learns to predict if the second sentence in the pair is the subsequent sentence in the original document.
- During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. We assume that the random sentence is not subsequent to the first sentence.

# Next sentence Prediction

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence.

# Next Sentence Prediction Architecture



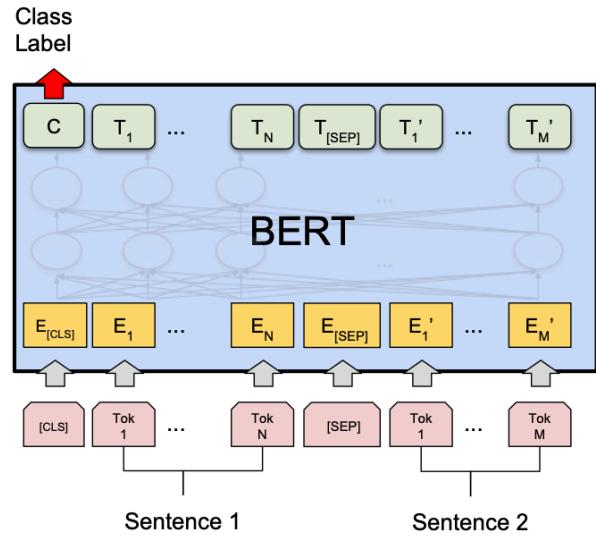
Source: [BERT](#) [Devlin et al., 2018], with modifications

# BERT Training

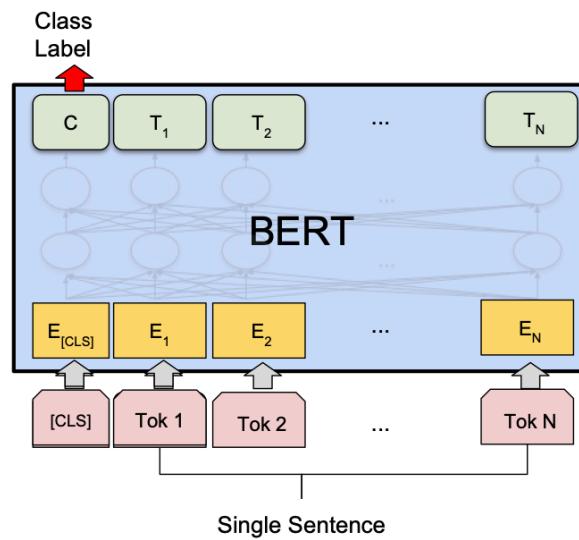
To predict if the second sentence is indeed connected to the first, the following steps are performed:

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a  $2 \times 1$  shaped vector, using a simple classification layer (learned matrices of weights and biases).
- Calculating the probability of IsNextSequence with softmax.
- When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

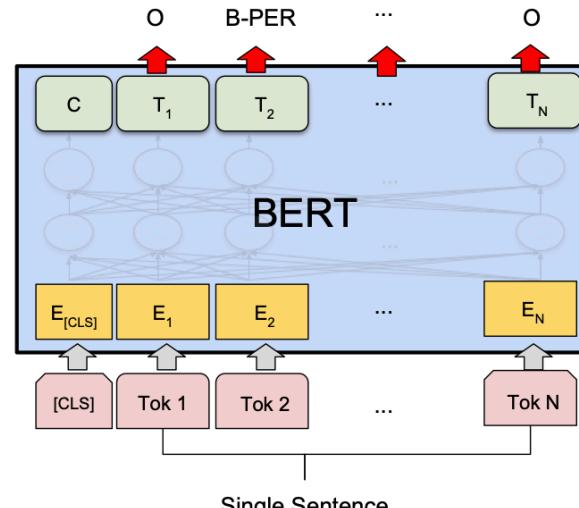
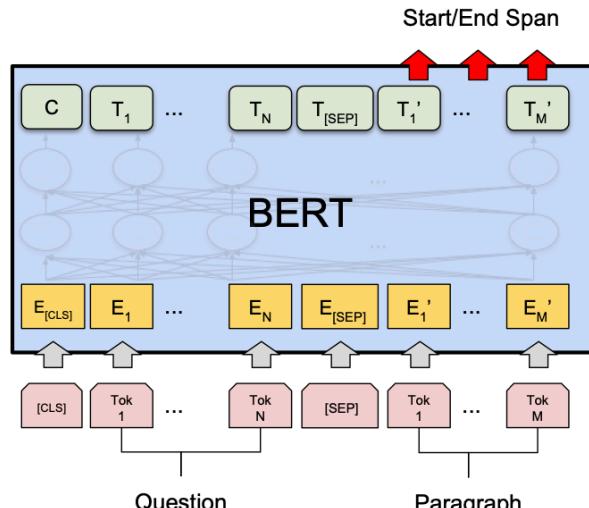
# BERT Fine-tuning on different tasks



(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



MNLI: Multi Genre NL Inference

QQP: Quora Question Pair

QNLI: Question Natural Language Inference

SST-2: Stanford Sentiment Treebank

CoLA: Corpus of Linguistic Acceptability

STS-B: Semantic Textual Similarity Benchmark

MRPC: Microsoft Research Paraphrase Corpus

RTE: Recognizing Textual Entailment

WNLI: Winograd NLI

SQuAD: Stanford Question Answering dataset

CoNLL 2003 NER: Cornell University Named Entity

Recognition Dataset

# BERT Key Insights

- Unified architecture across multiple tasks
- Compact representation of vocabulary using Wordpiece representation
- Results have shown that finetuning is relatively inexpensive, transfer learning is shown to be very effective. (A few hours of GPU time)
- Larger models lead to a strict accuracy improvement across all the datasets used during training

It has long been known that increasing the model size will lead to continual improvements on large-scale tasks such as machine translation and language modeling, which is demonstrated by the LM perplexity of held-out training data shown in Table 6. However, we believe that this is the first work to demonstrate convincingly that scaling to extreme model sizes also leads to large improvements on very small scale tasks, provided that the model has been sufficiently pre-trained. Peters et al. (2018b) presented

# Usage Examples

```
>>> from transformers import pipeline
>>> unmasker = pipeline('fill-mask', model='bert-base-uncased')
>>> unmasker("Hello I'm a [MASK] model.")

[{'sequence': '[CLS] hello i'm a fashion model. [SEP]",
 'score': 0.1073106899857521,
 'token': 4827,
 'token_str': 'fashion'},
 {'sequence': '[CLS] hello i'm a role model. [SEP]",
 'score': 0.08774490654468536,
 'token': 2535,
 'token_str': 'role'},
 {'sequence': "[CLS] hello i'm a new model. [SEP]",
 'score': 0.05338378623127937,
 'token': 2047,
 'token_str': 'new'},
 {'sequence': "[CLS] hello i'm a super model. [SEP]",
 'score': 0.04667217284440994,
 'token': 3565,
 'token_str': 'super'},
 {'sequence': "[CLS] hello i'm a fine model. [SEP]",
 'score': 0.027095865458250046,
 'token': 2986,
 'token_str': 'fine'}]
```

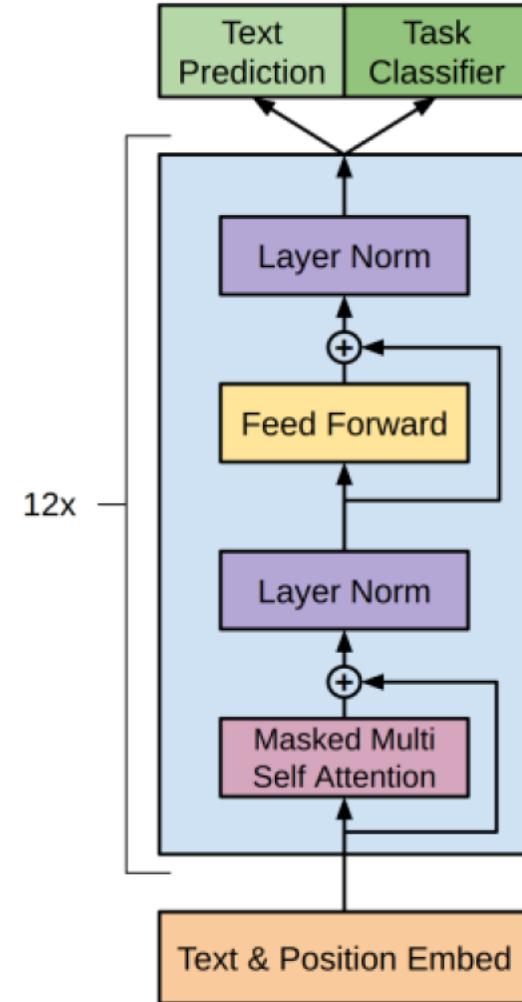
Here is how to use this model to get the features of a given text in PyTorch:

```
from transformers import BertTokenizer, BertModel
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained("bert-base-uncased")
text = "Replace me by any text you'd like."
encoded_input = tokenizer(text, return_tensors='pt')
output = model(**encoded_input)
```

By passing labels for different tasks, Hugging Face Transformer based models allow inferencing for various tasks

# GPT architecture

- Learned text & position embedding
- Transformer block with causal masked self-attention
- *Optional cross attention module for adding conditional info.*



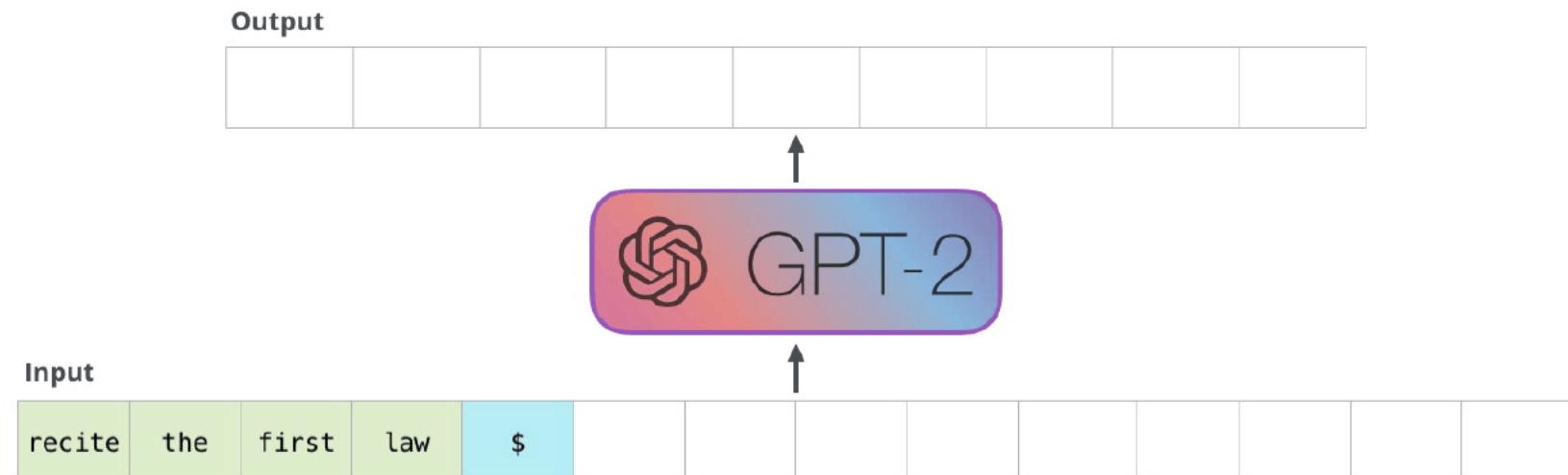
## GPT1

Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I., 2018. [Improving language understanding by generative pre-training](#).

# Next word prediction: GPT = Generative Pre-trained Transformer

- **Pre-trained:** Each word predict the next word considering all previous words
  - Enforcing causal attention.
  - Enabling text generation

- **Generative:** Autoregressive model, suitable for text generation.



# How does the decoding happen in a GPT?

- It is essential to understand how the instruction following GPT works.
- Input sequences (prompt) is processed, self attentions computed and abstract representations generated
- Till the prompt is processed and understood, the decoder doesn't emit the response
- Once prompt is processed, decoder emits the response in an auto regressive way
- Some models use explicit special tokens to delimit prompt and the response, while ChatGPT or GPT 4 do this demarcation implicitly.

# GPT Finetuning

- GPT training procedure consists of two stages:
  - The first stage is learning a high-capacity language model on a large corpus of text.
  - In the second fine-tuning stage, GPT is adapted to a discriminative task with labeled data.

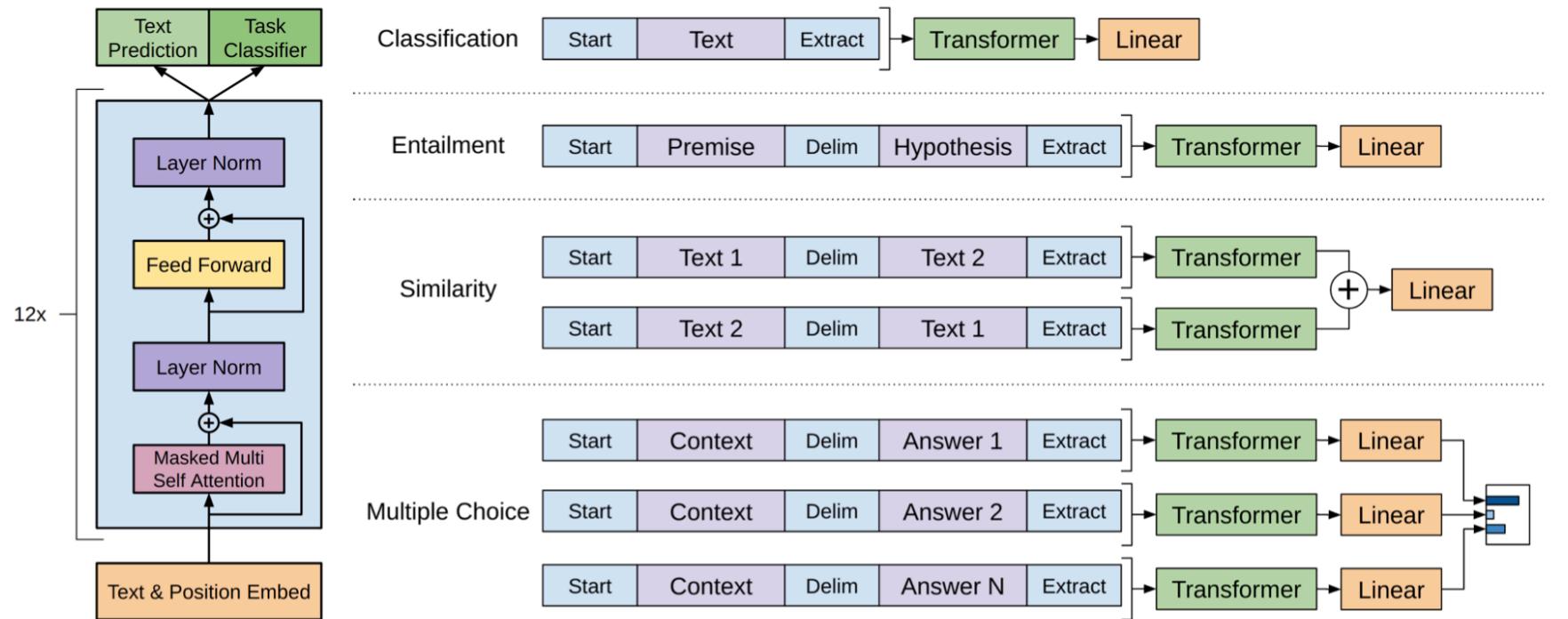
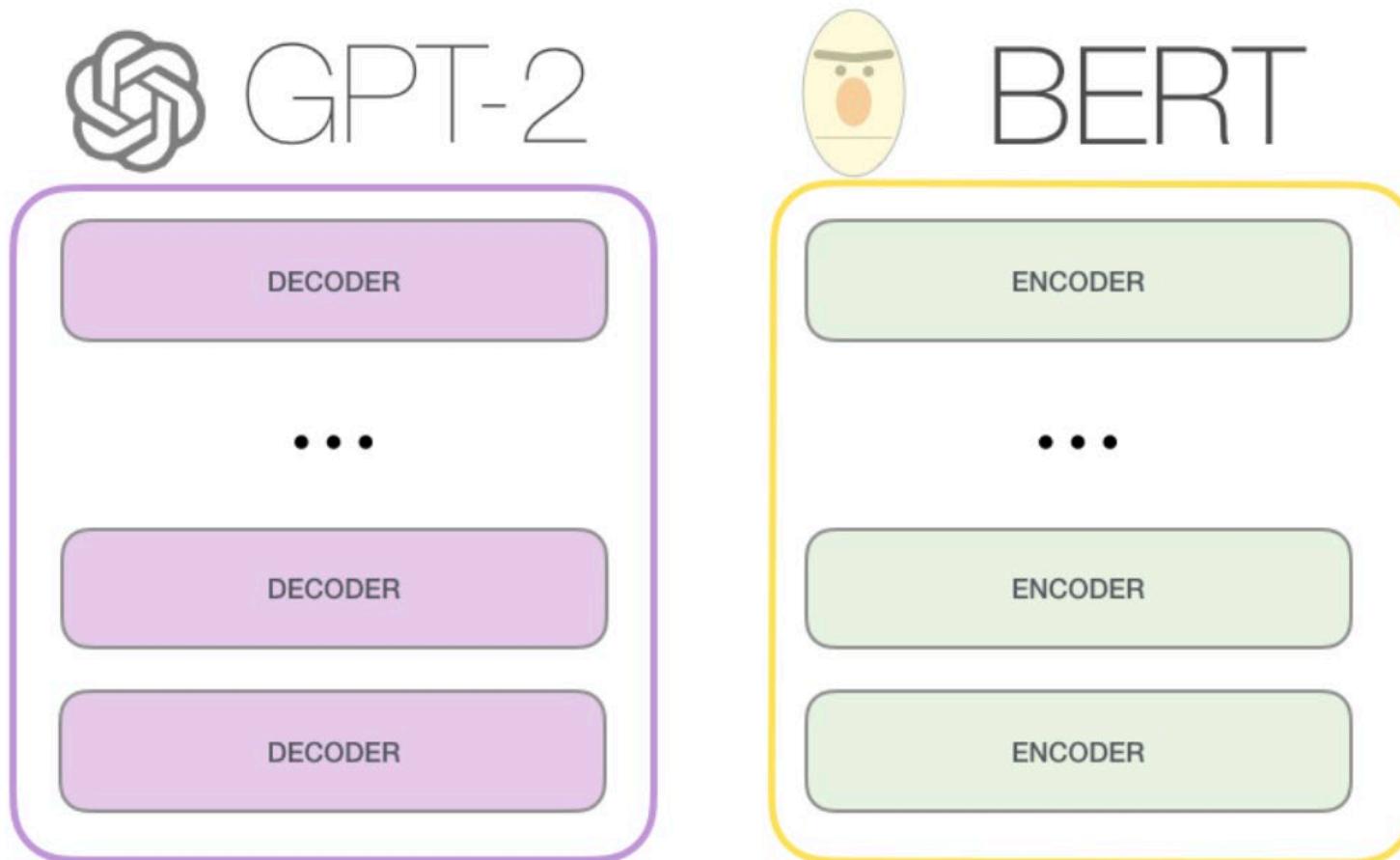


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

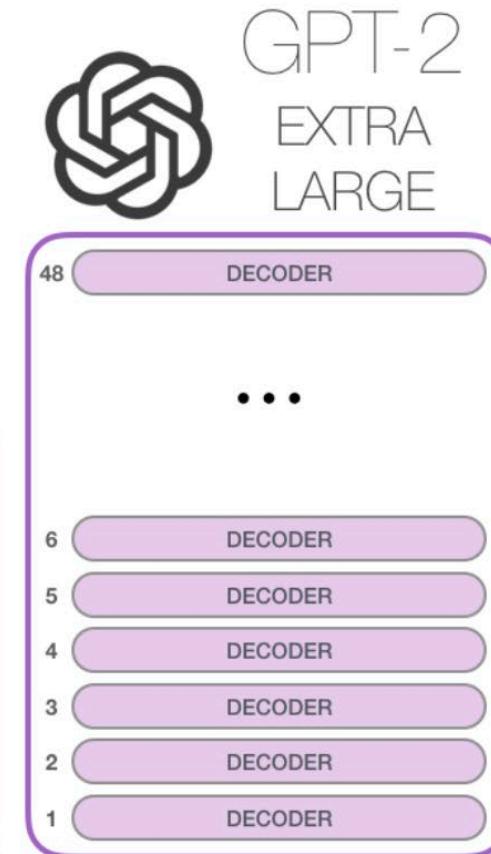
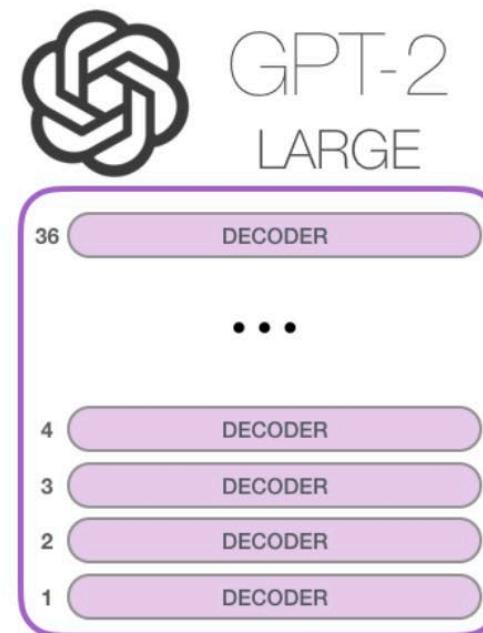
# GPT versus BERT Approaches

Fig Credits: Jay Alammar

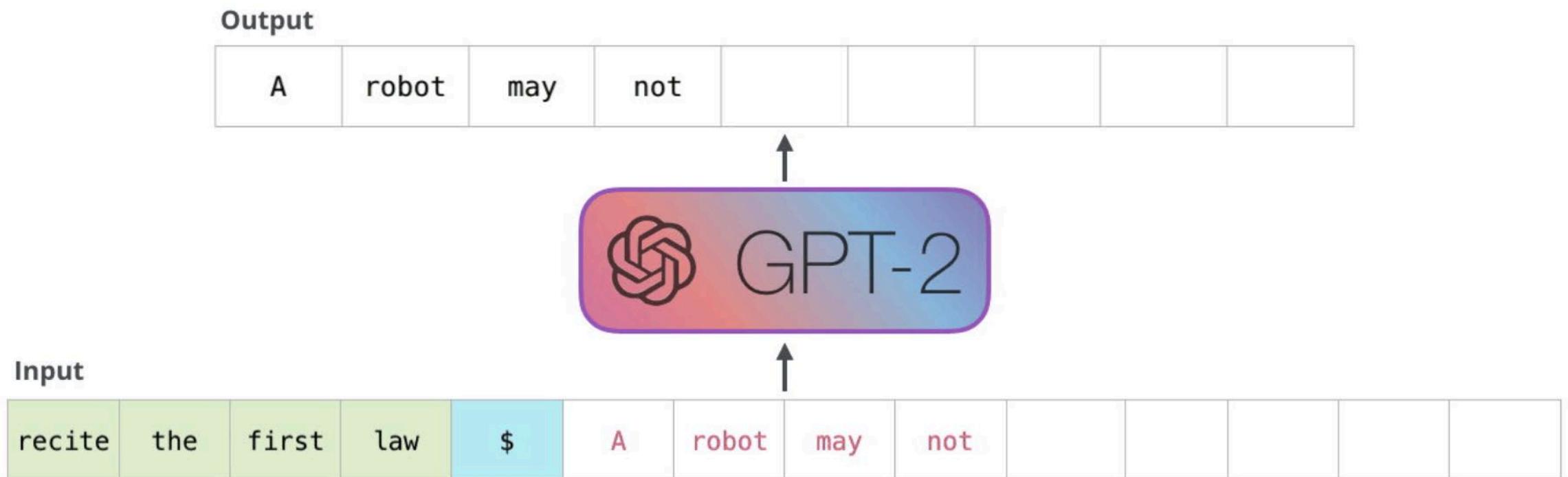


# GPT-2 Variants

Fig Credits: Jay Alammar

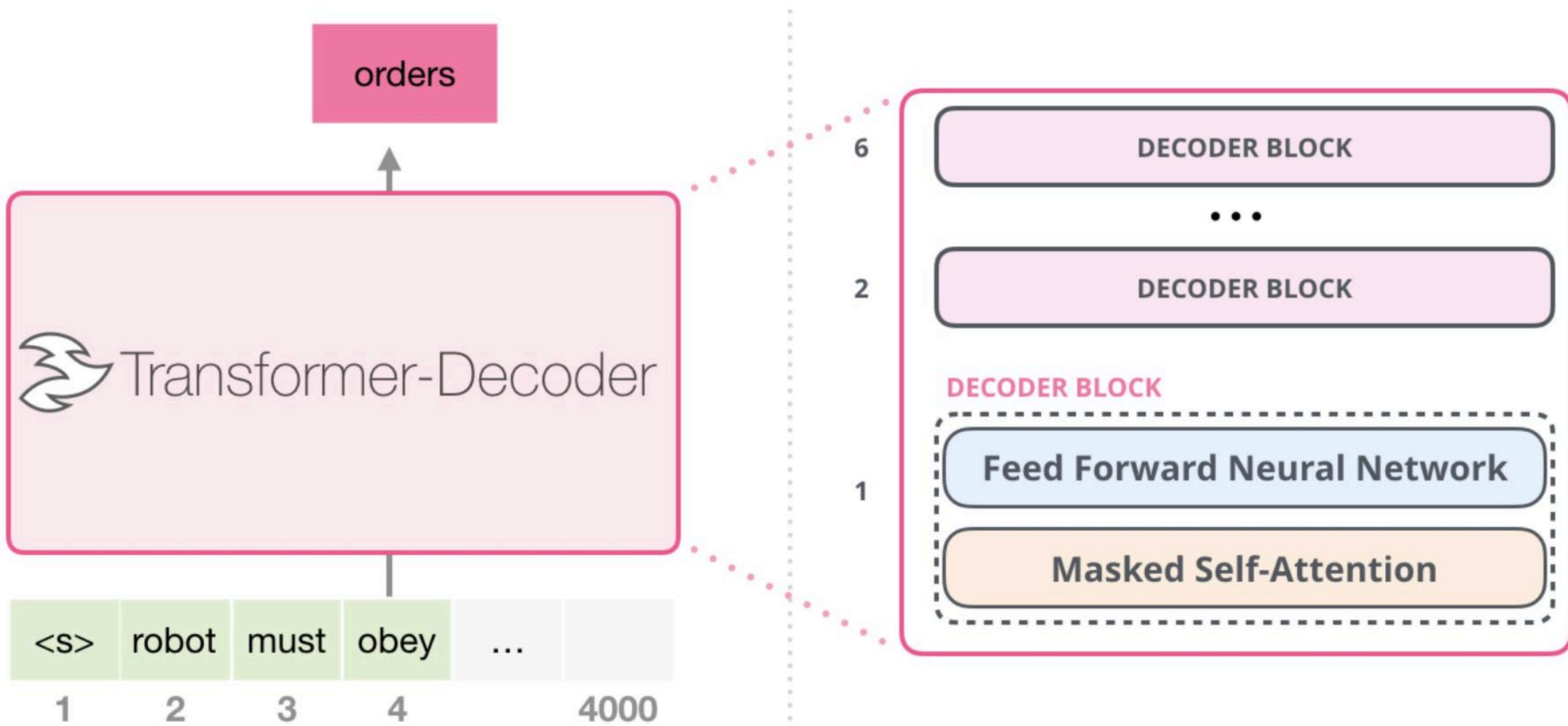


# Decoding in GPT



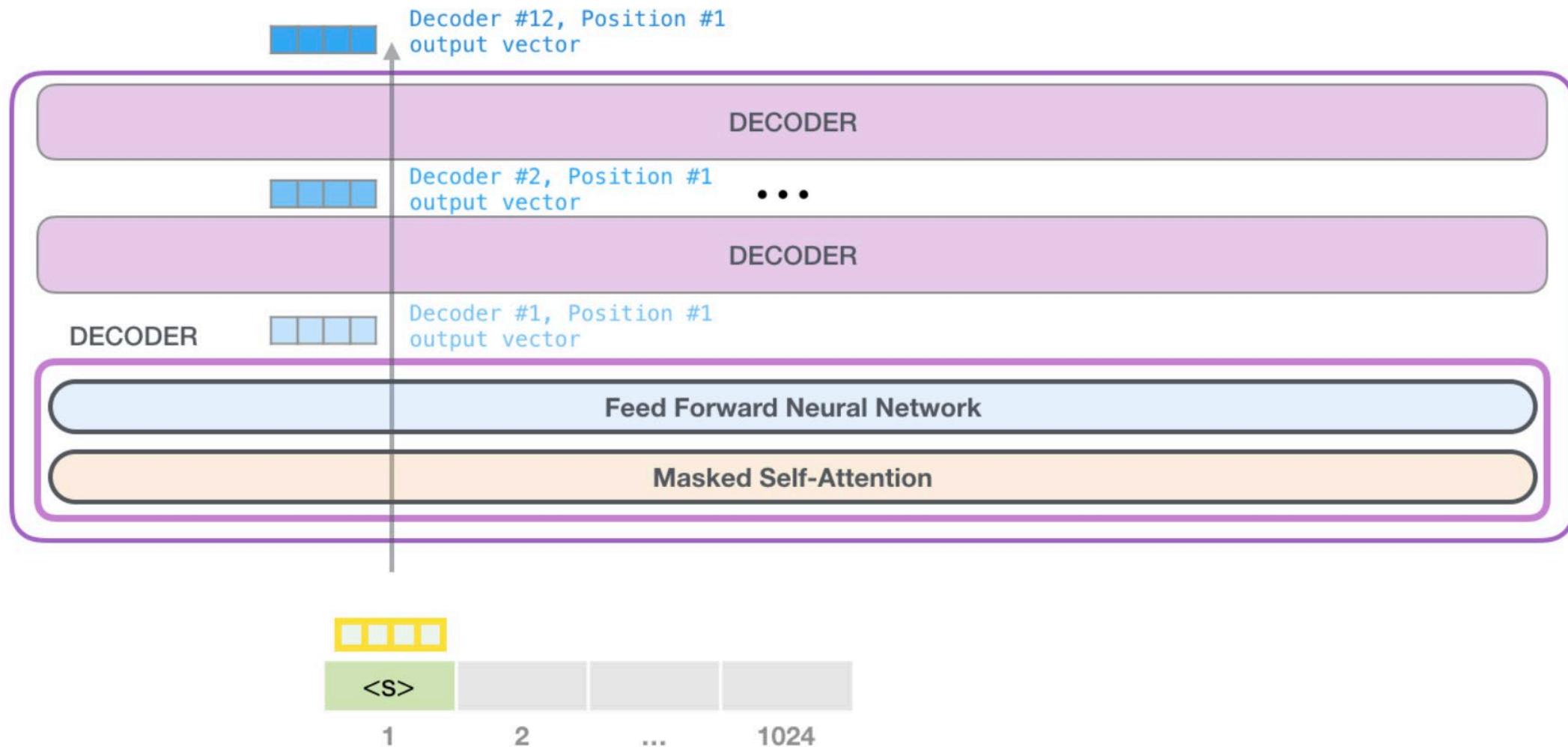
# Decoder Only Model

Fig Credits: Jay Alammar



# Processing through the decoder stack

Fig Credits: Jay Alammar



# Self Attention Example

Fig Credits: Jay Alammar

Language heavily relies on context. For example, look at the second law:

*Second Law of Robotics*

*A robot must obey the orders given **it** by human beings except where **such orders** would conflict with the **First Law**.*

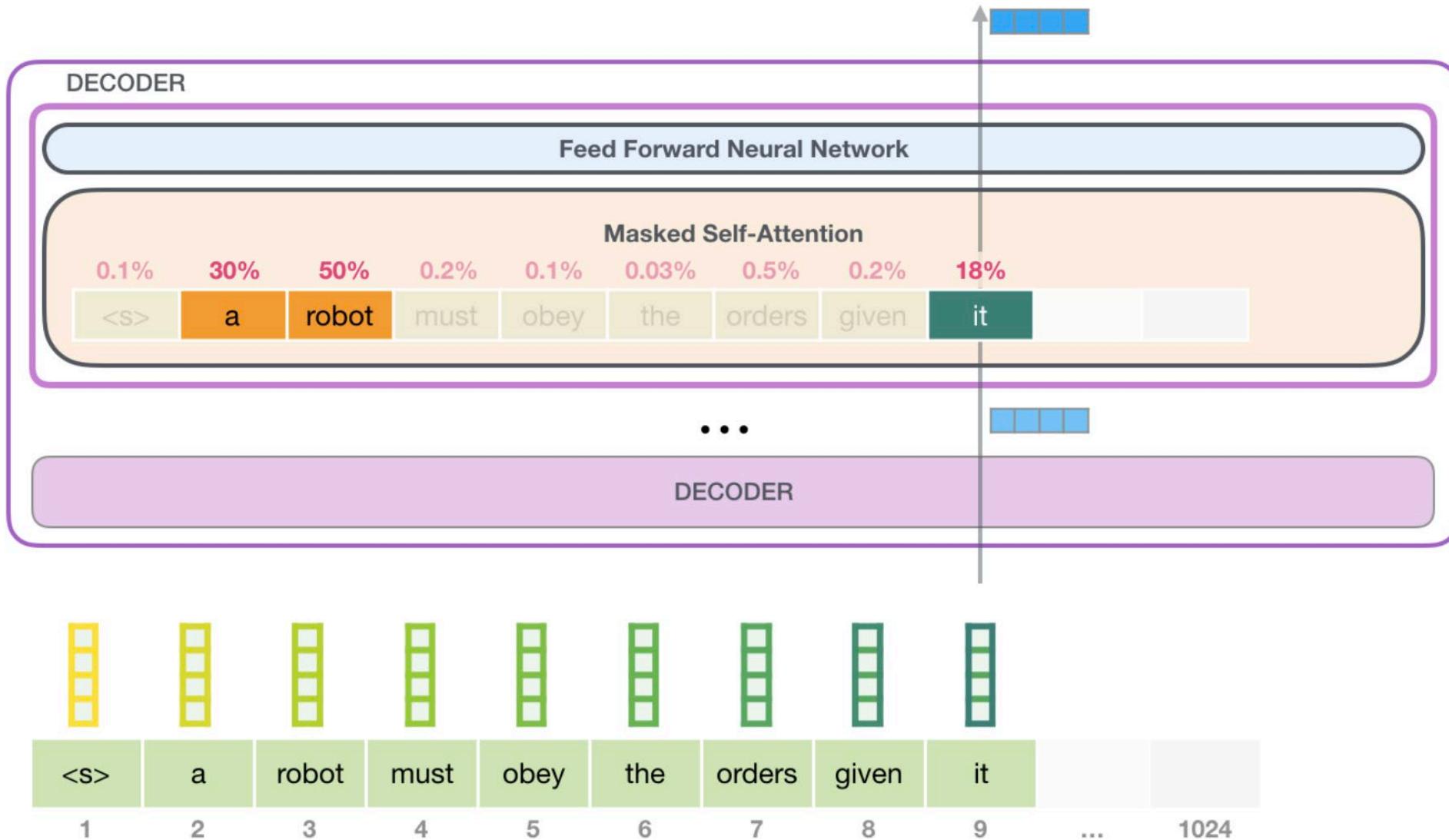
I have highlighted three places in the sentence where the words are referring to other words. There is no way to understand or process these words without incorporating the context they are referring to. When a model processes this sentence, it has to be able to know that:

- **it** refers to the robot
- **such orders** refers to the earlier part of the law, namely “the orders given it by human beings”
- **The First Law** refers to the entire First Law

This is what self-attention does. It bakes in the model’s understanding of relevant and associated words that explain the context of a certain word before processing that word (passing it through a neural network). It does that by assigning scores to how relevant each word in the segment is, and adding up their vector representation.

As an example, this self-attention layer in the top block is paying attention to “a robot” when it processes the word “it”. The vector it will pass to its neural network is a sum of the vectors for each of the three words multiplied by their scores.

# Self Attention in Decoder



# Summarization

**Positronic brain**

This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see [Positronic \(company\)](#).

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsubstantiated material may be challenged and removed.

[Find sources: "Positronic brain" – news · newspapers · books · scholar · JSTOR](#) (July 2008) ([Learn how and when to remove this template message](#))

A positronic brain is a fictional technological device, originally conceived by science fiction writer Isaac Asimov.<sup>[1][2]</sup> It functions as a central processing unit (CPU) for robots, and, in some unexpected way, provides them with a form of consciousness recognizable to humans. When Asimov wrote his first robot stories in 1939 and 1940, the positron was a newly discovered particle, and so the *Isaac Asimov's Positronic Brain* added a contemporary gloss of popular science to the concept. The short story "Runaround", by Asimov, elaborates on the concept, in the context of his fictional Three Laws of Robotics.

**Conceptual overview** [edit]

Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of platinum and indium. They were said to be vulnerable to radiation and apparently involve a type of volatile memory (since robots in storage required a power source keeping their brains "alive"). The focus of Asimov's stories was directed more towards the software of robots—such as the Three Laws of Robotics—than the hardware in which it was implemented, although it is stated in his stories that to create a positronic brain without the Three Laws, it would have been necessary to spend years redesigning the fundamental approach towards the brain itself.

Within his stories of robotics on Earth and their development by U.S. Robots, Asimov's positronic brain is less of a *plot device* and more of a technological item worthy of study.

A positronic brain cannot ordinarily be built without incorporating the Three Laws; any modification thereof would drastically modify robot behavior. Behavioral dilemmas resulting from conflicting potentials set by inexperienced and/or malicious users of the robot for the Three Laws make up the bulk of Asimov's stories concerning robots. They are resolved by applying the science of logic and psychology together with mathematics, the supreme solution finder being Dr. Susan Calvin, Chief Robopsychologist of U.S. Robots:

The Three Laws are also a *bulletin* in brain sophistication. Very complex brains designed to handle world economy interpret the First Law in expanded sense to include humanity as opposed to a single human; in Asimov's later works like *Robots and Empire* this is referred to as the "Zeroth Law". At least one brain constructed as a calculating machine, as opposed to being a robot control circuit, was designed to have a flexible, childlike personality so that it was able to pursue difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economies were stated to have no personality at all.

Under specific conditions, the Three Laws can be obviated, with the modification of the actual robotic design:

- Robots that are of low enough value can have the *Third Law* deleted; they do not have to protect themselves from harm, and the brain size can be reduced by half.
- Robots that do not require orders from a human being may have the *Second Law* deleted, and therefore require smaller brains again, providing they do not require the *Third Law*.
- Robots that are disposable, cannot receive orders from a human being and are not able to harm a human, will not require even the *First Law*. The sophistication of positronic circuitry renders a brain so small that it could comfortably fit within the skull of an insect.

Robots of the latter type directly parallel contemporary industrial robotics practice, though real-life robots do contain safety sensors and systems, in a concern for human safety (a weak form of the First Law; the robot is a safe tool to use, but has no "judgment", which is implicit in Asimov's own stories).

**In Allen's trilogy** [edit]

Several robot stories have been written by other authors following Asimov's death. For example, in Roger MacBride Allen's Caliban trilogy, a Seaver robot called Gutter Anshar invents the *gravitronic brain*. It offers speed and capacity improvements over traditional positronic designs, but the strong influence of tradition make robotics labs reject Anshar's work. Only one robotologist, Freddi Laving, chooses to adopt gravitronics, because it offers her a blank slate on which she could explore alternatives to the Three Laws. Because they are not dependent upon centuries of earlier research, gravitronic brains can be programmed with the standard Laws, variations of the Laws, or even empty pathways which specify no Laws at all.

**Positronic brain**

This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see [Positronic \(company\)](#).

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsubstantiated material may be challenged and removed.

[Find sources: "Positronic brain" – news · newspapers · books · scholar · JSTOR](#) (July 2008) ([Learn how and when to remove this template message](#))

A positronic brain is a fictional technological device, originally conceived by science fiction writer Isaac Asimov.<sup>[1][2]</sup> It functions as a central processing unit (CPU) for robots, and, in some unexpected way, provides them with a form of consciousness recognizable to humans. When Asimov wrote his first robot stories in 1939 and 1940, the positron was a newly discovered particle, and so the *Isaac Asimov's Positronic Brain* added a contemporary gloss of popular science to the concept. The short story "Runaround", by Asimov, elaborates on the concept, in the context of his fictional Three Laws of Robotics.

**SUMMARY**

**Conceptual overview** [edit]

Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of platinum and indium. They were said to be vulnerable to radiation and apparently involve a type of volatile memory (since robots in storage required a power source keeping their brains "alive"). The focus of Asimov's stories was directed more towards the software of robots—such as the Three Laws of Robotics—than the hardware in which it was implemented, although it is stated in his stories that to create a positronic brain without the Three Laws, it would have been necessary to spend years redesigning the fundamental approach towards the brain itself.

Within his stories of robotics on Earth and their development by U.S. Robots, Asimov's positronic brain is less of a *plot device* and more of a technological item worthy of study.

A positronic brain cannot ordinarily be built without incorporating the Three Laws; any modification thereof would drastically modify robot behavior. Behavioral dilemmas resulting from conflicting potentials set by inexperienced and/or malicious users of the robot for the Three Laws make up the bulk of Asimov's stories concerning robots. They are resolved by applying the science of logic and psychology together with mathematics, the supreme solution finder being Dr. Susan Calvin, Chief Robopsychologist of U.S. Robots:

The Three Laws are also a *bulletin* in brain sophistication. Very complex brains designed to handle world economy interpret the First Law in expanded sense to include humanity as opposed to a single human; in Asimov's later works like *Robots and Empire* this is referred to as the "Zeroth Law". At least one brain constructed as a calculating machine, as opposed to being a robot control circuit, was designed to have a flexible, childlike personality so that it was able to pursue difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economies were stated to have no personality at all.

**ARTICLE**

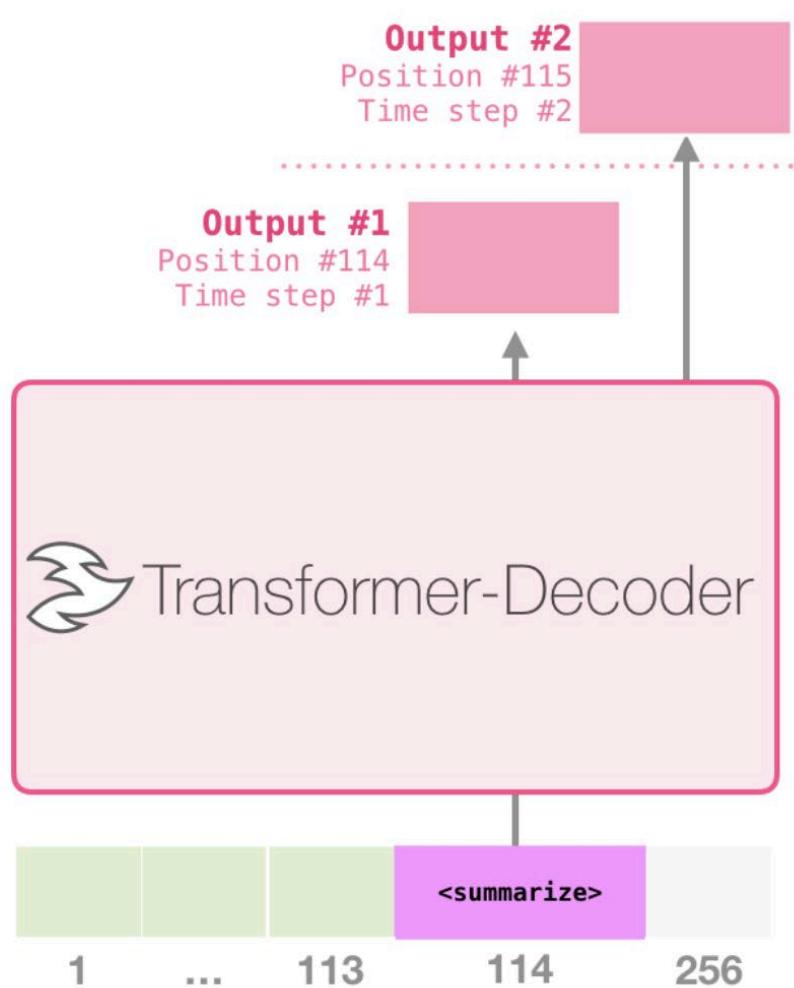
- Robots that are of low enough value can have the *Third Law* deleted; they do not have to protect themselves from harm, and the brain size can be reduced by half.
- Robots that do not require orders from a human being may have the *Second Law* deleted, and therefore require smaller brains again, providing they do not require the *Third Law*.
- Robots that are disposable, cannot receive orders from a human being and are not able to harm a human, will not require even the *First Law*. The sophistication of positronic circuitry renders a brain so small that it could comfortably fit within the skull of an insect.

Robots of the latter type directly parallel contemporary industrial robotics practice, though real-life robots do contain safety sensors and systems, in a concern for human safety (a weak form of the First Law; the robot is a safe tool to use, but has no "judgment", which is implicit in Asimov's own stories).

# Training for summarization

## Training Dataset

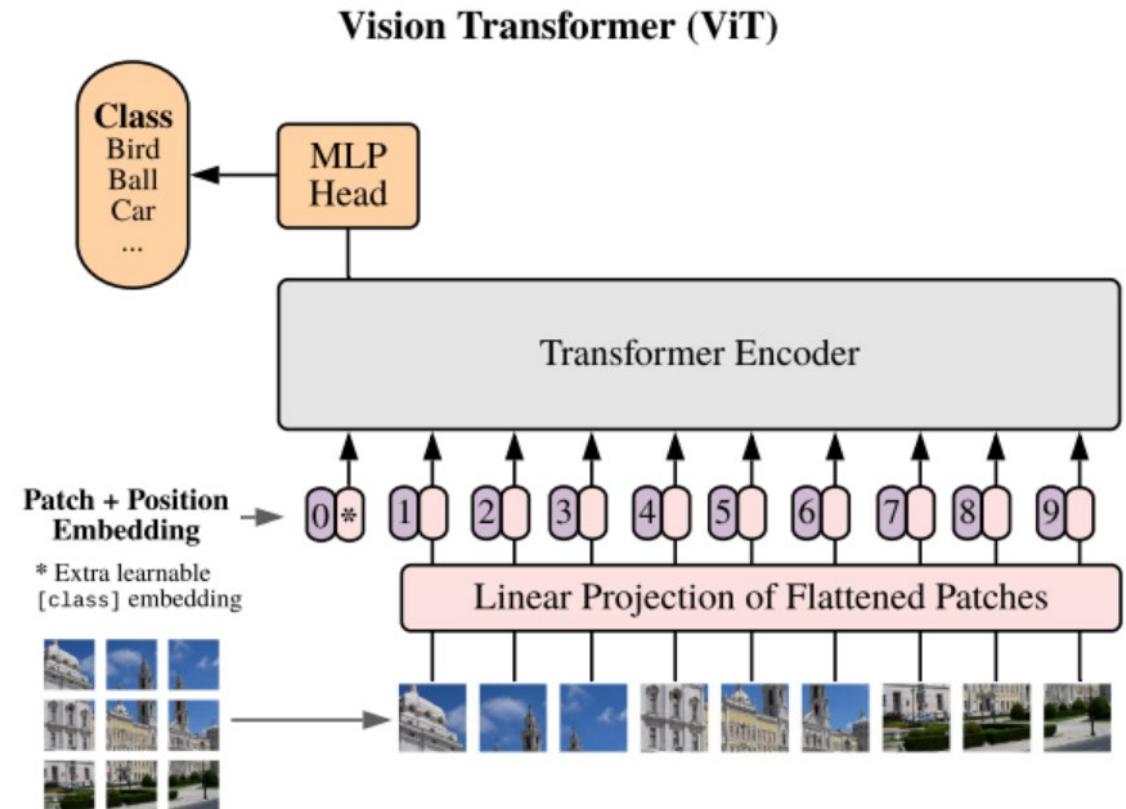
Article #1 tokens	<summarize>	Article #1 Summary
Article #2 tokens	<summarize>	Article #2 Summary padding
Article #3 tokens		<summarize> Article #3 Summary



# Vision Transformers

Fig Credits: Binxu Wang, Harvard

- Transformers can be used in multimodal applications
- When the input is an image, patches of an image can be used as tokens
- For classification extra [CLS] token to be added.
- Vision transformers need a higher level of data and training effort.
- CNNs may provide better performance on models trained with less data, vision transformers are more scalable and a preferred technique for contemporary applications, particularly due to availability of pre-trained models



# Summary

- Transformers brought in the era of transfer learning for NLP – the ImageNet moment
- Transformer based models like GPT are the basis for modern LLMs
- Transformers are versatile, they are getting to be the “go to” model for other data types like images.
- Transformers can be trained on multiple tasks at the same time and thus are foundation for LLMs. This also enables novel programming metaphors like prompt engineering

Thank You

# Hands On: Prompt Engineering

Palacode Narayana Iyer Anantharaman

# Disclaimer

- We will be using datasets that pertain to Indian General Elections during our hands on sessions.
- These datasets are chosen for our projects as they allow development of excellent data analytics and constitute current topics of interest.
- Any discussions and analysis of this data are purely meant as illustrations of data analytics only. They don't imply any political opinions or affiliations.
- All the data used here are public, permitted by Government of India (ECI website).

# Prompt Engineering and Prompt Tuning

- Hard Prompts



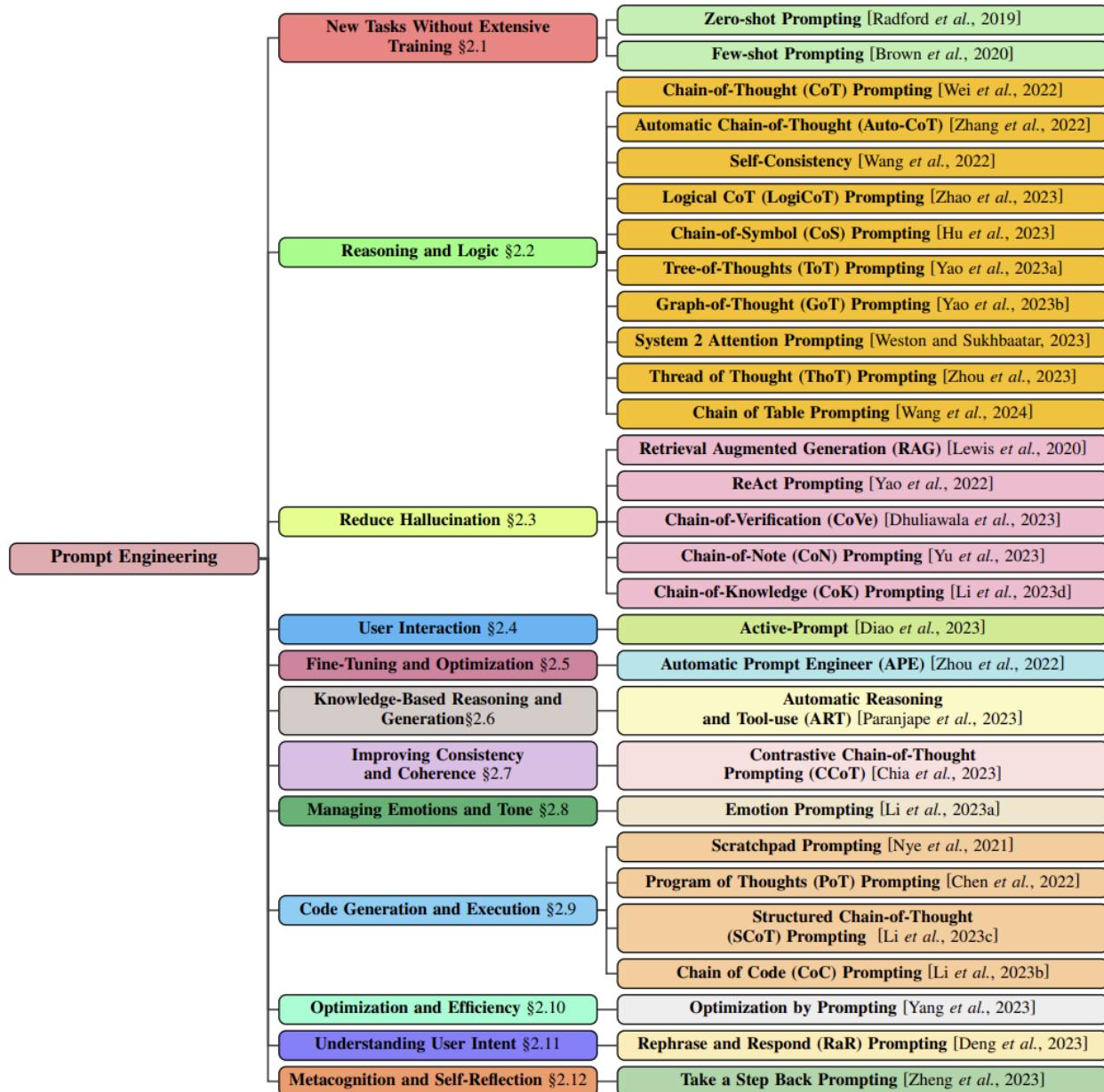
- Soft Prompts

## Using Soft-Prompt Tuning to Evaluate Bias in Large Language Models

**Jacob-Junqi Tian<sup>1,2</sup> David Emerson<sup>1</sup> Sevil Zanjani Miyandoab<sup>3</sup>**  
**Deval Pandya<sup>1</sup> Laleh Seyed-Kalantari<sup>1,4</sup> Faiza Khan Khattak<sup>1</sup>**

<sup>1</sup>Vector Institute for AI   <sup>2</sup>McGill University   <sup>3</sup> Amirkabir University   <sup>4</sup> York University  
jacob.tian@mail.mcgill.ca, sev.zanjani@gmail.com, lsk@yorku.ca,  
{david.emerson, deval.pandya, faiza.khankhattak}@vectorinstitute.ai

# Prompt engineering Taxonomy



Ref: A Systematic Survey of Prompt Engineering in Large Language Models  
Pranab Sahoo et al.

Application	Prompting Technique	Comparison Scope					
		Prompt Acquisition	Prompt Turn	Language Model(s)	Dataset	Metric(s)	
Reasoning and Logic	New Tasks Without Training Data	Zero-shot	Manual	Single	GPT-2	Arithmetic, Symbolic	Accuracy, ROUGE Score
		Few-shot	Manual	Single	GPT-3	NaturalQS, WebQS, TriviaQA	Accuracy
	Self-Consistency	CoT	Manual	Multi	PaLM 540B	GSM8K	Accuracy
		LogiCoT	Manual	Multi	Vicuna-33b, GPT-4	GSM8K, AQuA, SocialQA	Accuracy
		CoS	Manual	Multi	gpt-3.5-turbo, GPT-4	SPARTUN	Accuracy, Precision, Recall
		Auto-CoT	LM Generated	Multi	GPT-3	Arithmetic, Symbolic	Accuracy
		Self-Consistency	Manual	Single	PaLM 540B	Arithmetic, Commonsense	Accuracy
		ToT	Retrieval Based	Multi	GPT-4	Game of 24, Creative Writing	Success Rate
		GoT	Retrieval Based	Multi	T5-large	GSM8K, ScienceQA	ROUGE Score
		S2A	Manual	Single	Llama 2-70B	QA, GSM8K	Accuracy
	Chain of Table	ThoT	Hybrid	Multi	gpt-3.5-turbo, Llama 2-70b-chat	PopQA, EntityQ, MTCR	Exact Match (EM) Score
		Chain of Table	Manual	Multi	GPT 3.5, LLaMA 2	TabFact, WikiTQ	BLEU, ROUGE Score
Reduce Hallucination	CoVe	Retrieval Based	Multi	Llama 65B	Wikidata, QUEST, MultiSpanQA	Precision, F1	
	ReAct	Retrieval Based	Multi	PaLM-540B, GPT-3	HotpotQA, FEVER	Exact Match (EM), Accuracy	
	RAG	Retrieval Based	Single	RAG-Token, RAG-Seq.	MSMARCO, SearchQA	ROUGE, BLEU score	
	CoN	LM Generated	Multi	Llama 2, DPR	NQ, TriviaQA, WebQ	Exact Match (EM), F1 Score	
	CoK	LM Generated	Multi	gpt-3.5-turbo-0613	HotpotQA, FEVER, MedMCQA, MMLU Physics and Biology	Exact Match (EM), Accuracy	
User Interaction	Active-Prompt	Manual	Single	code-davinci-002, text-davinci-003	Arithmetic, Commonsense, Symbolic	Disagreement, Entropy Variance, Self-confidence Score	
Fine-Tuning and Optimization	APE	LM Generated	Single	text-curie-001, text-davinci-002	BBII, TruthfulQA	Execution accuracy, Log probability, Efficient score estimation	
Knowledge-Based Reasoning and Generation	ART	Hybrid	Multi	GPT-3 (175B)	BigBench, MMLU	Accuracy	
Improving Consistency and Coherence	CCoT	LM Generated	Multi	gpt-3.5-turbo-0301	Arithmetic, Factual QA	Accuracy	
Managing Emotions and Tone	Emotion Prompting	Manual	Single	GPT-4	BIG-Bench, Instruction Induction	Accuracy	
Code Generation and Execution	SCoT	Hybrid	Multi	ChatGPT, Codex	HumanEval, MBPP, MBCPP	pass@k	
	PoT	Manual	Single	gpt-3.5-turbo	GSM8K, SVAMP, FinQA	Exact Match(EM) Score	
	Scratchpad Prompting	Manual	Single	text-davinci-003, gpt-3.5-turbo	BIG-Bench Hard	Accuracy	
		Manual	Single	GPT-3	MBPP, MBPP-aug	Accuracy	
Optimization and Efficiency	OPRO	Manual	Single	PaLM 2-L-IT, text-bison	GSM8K, BIG-Bench Hard	Accuracy	
Understanding User Intent	RaR	Manual	Single	GPT-4-0613	Knowledge, Symbolic	Accuray, Fair Score, Language Modeling Score	
Metacognition and Self-Reflection	Take a Step Back	Manual	Single	PaLM2-L, GPT-4	MMLU-Physics, MMLU-Chemistry TimeQA, SituatedQA, StrategyQA	Accuracy	

# Takeaways and insights

- Prompt engineering is the “first line of defense” to adopt when building an LLM application
- There are a plethora of prompt engineering techniques that are evolved.
- The categorization of them in to buckets help us to choose the right technique for our application.
  - E.g. Your application may pertain to coding LLMs. So, consider those techniques whose benchmarks show a good number on coding datasets.
- The techniques, even proposed a couple of years back, may not be needed for present day LLMs. So, evaluate them before applying.
  - E.g. The famous example of “Roger has 5 tennis balls...” used to justify CoT technique is no longer a problem for many modern open source LLMs, though the concept is relevant for less powerful models.

# Illustrations

You are given a list of party names and the votes polled.

Tell me which party won this election, who is the runner up and what is the victory margin against the runner up.

TDP 54056

BJP 1753

YSRCP 69588

INC 1327

JnP 2987

JAJGP 203

IND 568

IND 418

IND 743

IND 636

TDP 51399

BJP 3273

YSRCP 71694

INC 1954

JnP 5391

JAJGP 469

IND 1082

IND 886

IND 1127

IND 793

TDP 57159

BJP 1474

YSRCP 68836

INC 1950

## Response of Llama 3 8B K\_M GGUF

After analyzing the votes polled, I can tell you that:

\*\*Winner:\*\* YSRCP (YSR Congress Party) with a total of 105037 votes.

\*\*Runner-up:\*\* TDP (Telugu Desam Party) with a total of 63471 votes.

\*\*Victory margin:\*\* The victory margin against the runner-up is 4166 votes ( $105037 - 63471 = 4166$ ).

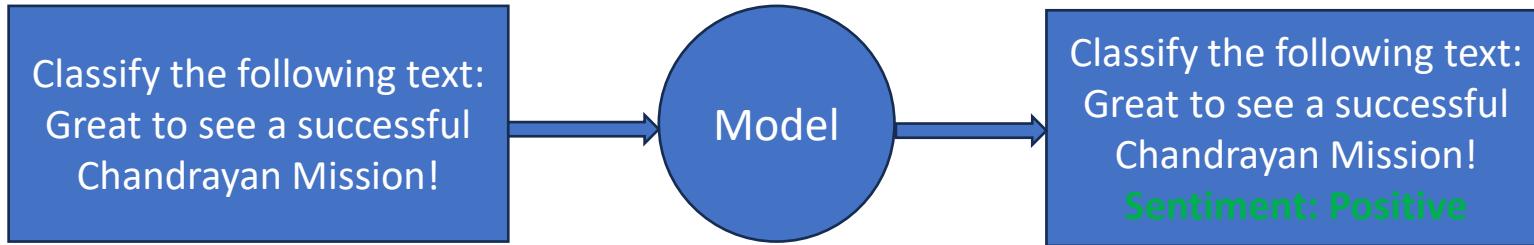
Note that I've added up all the votes for each party across all the polling stations.

## Response of Microsoft Phi-3 medium 128k K\_M

GGUF

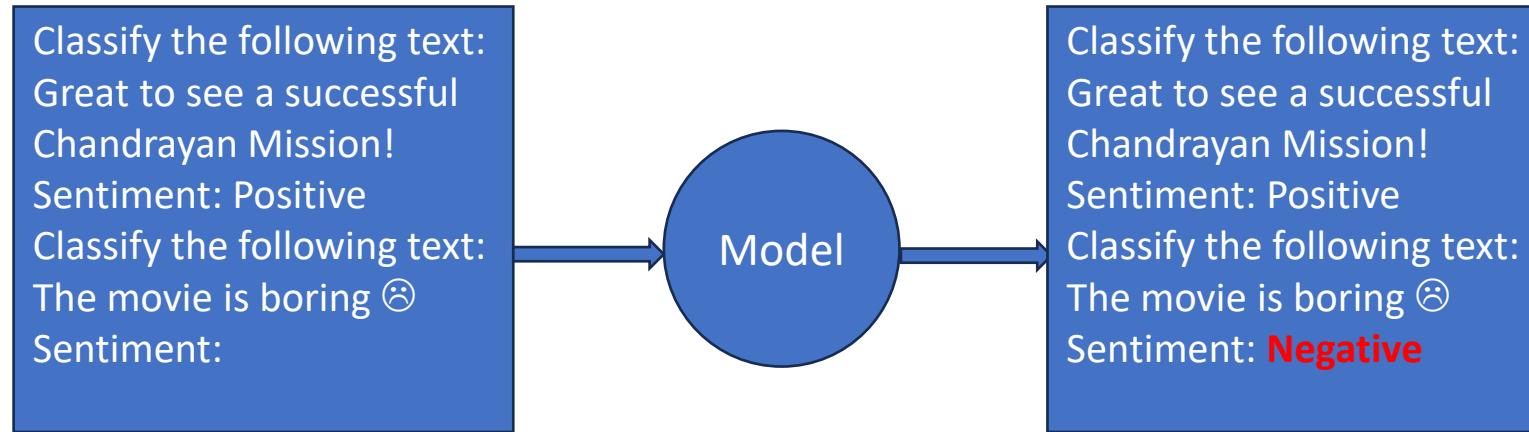
The party that won this election is YSRCP with a total of 459,386 votes. The runner up is TDP with a total of 276,475 votes. The victory margin against the runner up (TDP) is 182,911 votes.

# Prompt engineering: Zero shot inference



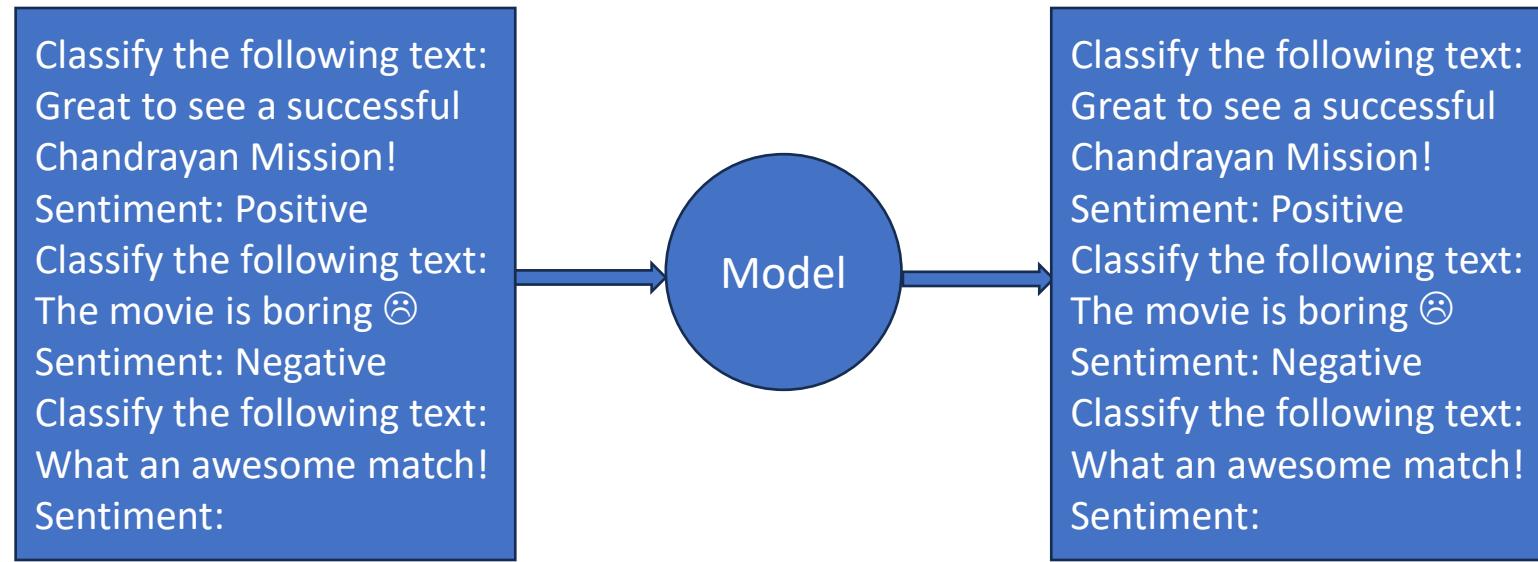
- Prompt engineering: Providing a prompt that can get the best response
- Zero shot inference: Provide a well crafted prompt to the foundational LLM, use the response from the model as the output. No additional training of the model is required

# Prompt engineering: One shot inference



- Prompt engineering: Providing a prompt that can get the best response
- One shot inference: Provide a well crafted prompt to the foundational LLM along with one example, use the response from the model as the output. No additional training of the model is required

# Prompt engineering: Few shot inference



- Prompt engineering: Providing a prompt that can get the best response
- Few shot inference: Provide a well crafted prompt to the foundational LLM along with a few examples, use the response from the model as the output. No additional training of the model is required

# Elements of Prompts

A prompt contains any of the following elements:

**Instruction** - a specific task or instruction you want the model to perform

**Context** - external information or additional context that can steer the model to better responses

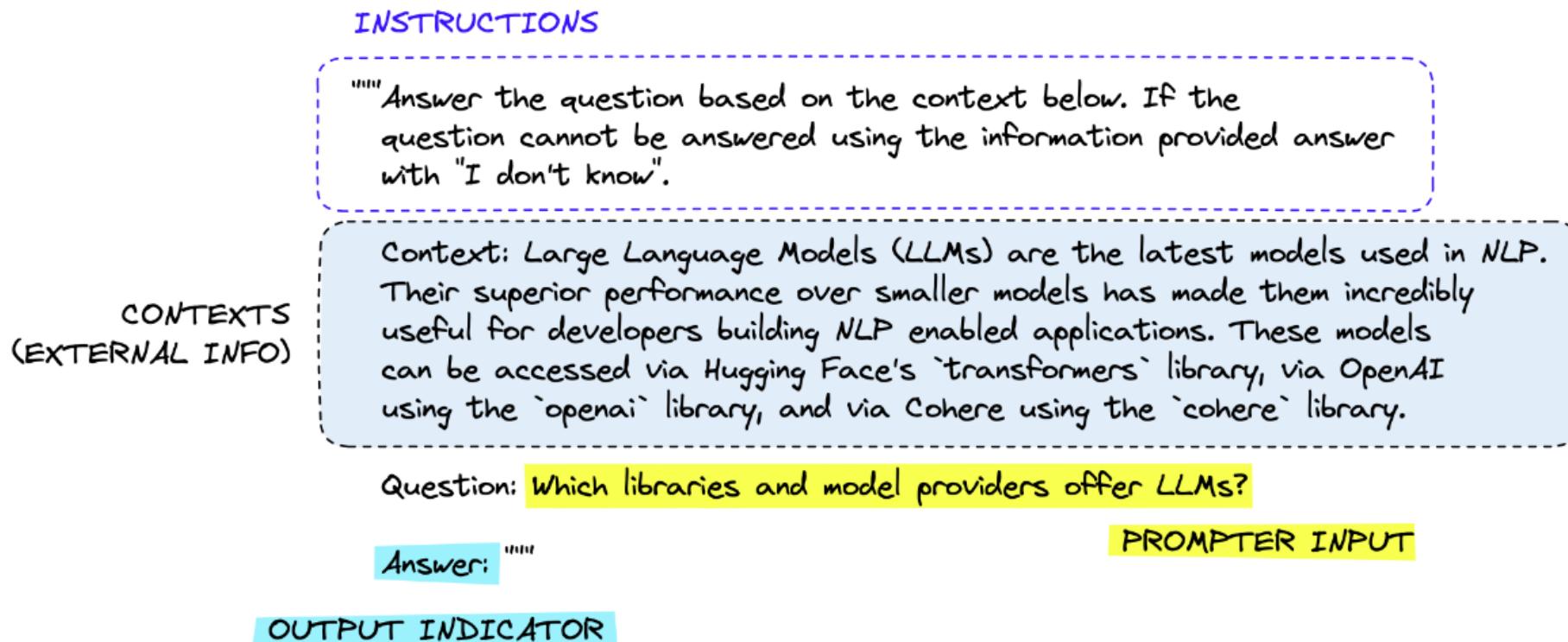
**Input Data** - the input or question that we are interested to find a response for

**Output Indicator** - the type or format of the output.

You do not need all the four elements for a prompt and the format depends on the task at hand.

Each LLM may specify a prompt template, refer to the model card in the Hugging Face

# Example Prompt Structure



A typical prompt structure.

# Langchain Prompt Templates Examples

# Prompt templates

Prompt templates are pre-defined recipes for generating prompts for language models.

A template may include instructions, few-shot examples, and specific context and questions appropriate for a given task.

LangChain provides tooling to create and work with prompt templates.

LangChain strives to create model agnostic templates to make it easy to reuse existing templates across different language models.

Typically, language models expect the prompt to either be a string or else a list of chat messages.

# Prompt template

---

Use `PromptTemplate` to create a template for a string prompt.

By default, `PromptTemplate` uses Python's `str.format` syntax for templating; however other templating syntax is available (e.g., `jinja2`).

```
from langchain import PromptTemplate

prompt_template = PromptTemplate.from_template(
    "Tell me a {adjective} joke about {content}."
)
prompt_template.format(adjective="funny", content="chickens")
```



"Tell me a funny joke about chickens."

# Few shot learning

- The success of LLMs comes from their large size and ability to store “knowledge” within the model parameter, which is *learned* during model training.
- However, there are more ways to pass knowledge to an LLM.
- The two primary methods are:
  - **Parametric knowledge** — the knowledge mentioned above is anything that has been learned by the model during training time and is stored within the model weights (or *parameters*).
  - **Source knowledge** — any knowledge provided to the model at inference time via the input prompt.
- Langchain’s **FewShotPromptTemplate** caters to source knowledge input. The idea is to “train” the model on a few examples — we call this few-shot learning — and these examples are given to the model within the prompt

# Demo – Few shot

# Googling versus ChatGPT

How many elections has PM of India win? Just give me a number.

As of my last update, Narendra Modi had won two general elections as the Prime Minister of India.

No, PM has contested assembly elections as well.

Narendra Modi has won seven elections as the Prime Minister of India.

Election	House	Constituency	Result	Party
2002 <sup>[a]</sup>	Gujarat Legislative Assembly	Rajkot II	Won	BJP
2002		Maninagar	Won	
2007			Won	
2012			Won	
2014	Lok Sabha	Vadodara <sup>[b]</sup>	Won	BJP
2019		Varanasi	Won	
2024			Pending	

## Chain of Thoughts

- The term PM of India must be referring to the incumbent Prime Minister of India
- “PM of India” should be “Narendra Modi”
- I need to find out the count of all elections he contested and pick those that he won.
- That would give me the required answer.

**Using Few Shot Examples, it possible to get a more accurate answer.**

# Advanced Prompt Engineering

- Chain of Thoughts

---

## Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

---

Jason Wei   Xuezhi Wang   Dale Schuurmans   Maarten Bosma  
Brian Ichter   Fei Xia   Ed H. Chi   Quoc V. Le   Denny Zhou

Google Research, Brain Team  
[{jasonwei,dennyyzhou}@google.com](mailto:{jasonwei,dennyyzhou}@google.com)

---

## Tree of Thoughts: Deliberate Problem Solving with Large Language Models

---

Shunyu Yao   Dian Yu   Jeffrey Zhao   Izhak Shafran  
Princeton University   Google DeepMind   Google DeepMind   Google DeepMind  
Thomas L. Griffiths   Yuan Cao   Karthik Narasimhan  
Princeton University   Google DeepMind   Princeton University

### Abstract

Language models are increasingly being deployed for general problem solving across a wide range of tasks, but are still confined to token-level, left-to-right decision-making processes during inference. This means they can fall short in tasks that require exploration, strategic lookahead, or where initial decisions play a pivotal role. To surmount these challenges, we introduce a new framework for language model inference, “Tree of Thoughts” (ToT), which generalizes over the popular “Chain of Thought” approach to prompting language models, and enables exploration over coherent units of text (“thoughts”) that serve as intermediate steps toward problem solving. ToT allows LMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices. Our experiments show that ToT significantly enhances language models’ problem-solving abilities on three novel tasks requiring non-trivial planning or search: Game of 24, Creative Writing, and Mini Crosswords. For instance, in Game of 24, while GPT-4 with chain-of-thought prompting only solved 4% of tasks, our method achieved a success rate of 74%. Code repo with all prompts: <https://github.com/princeton-nlp/tree-of-thought-llm>.

- Tree of Thoughts

- Graph of Thoughts

- ReAct Agents

---

Published as a conference paper at ICLR 2023

## REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao<sup>\*,1</sup>, Jeffrey Zhao<sup>2</sup>, Dian Yu<sup>2</sup>, Nan Du<sup>2</sup>, Izhak Shafran<sup>2</sup>, Karthik Narasimhan<sup>1</sup>, Yuan Cao<sup>2</sup>

<sup>1</sup>Department of Computer Science, Princeton University

<sup>2</sup>Google Research, Brain team

<sup>\*</sup>[{shunyuy,karthikn}@princeton.edu](mailto:{shunyuy,karthikn}@princeton.edu)

<sup>2</sup>[{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com](mailto:{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com)

---

Published as a conference paper at ICLR 2023

## SELF-CONSISTENCY IMPROVES CHAIN OF THOUGHT REASONING IN LANGUAGE MODELS

Xuezhi Wang<sup>†‡</sup>   Jason Wei<sup>†</sup>   Dale Schuurmans<sup>†</sup>   Quoc Le<sup>†</sup>   Ed H. Chi<sup>†</sup>  
Sharan Narang<sup>†</sup>   Aakanksha Chowdhery<sup>†</sup>   Denny Zhou<sup>†§</sup>

<sup>†</sup>Google Research, Brain Team

<sup>‡</sup>[xuezhiw@google.com](mailto:xuezhiw@google.com), <sup>§</sup>[dennyyzhou@google.com](mailto:dennyyzhou@google.com)

# Chain-of-Thought (CoT) Prompting

## Standard Prompting

Model Input

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

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27.

## Chain-of-Thought Prompting

Model Input

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

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9.

# Example

A

```
standard_prompt = f'''Use numbers and basic arithmetic operations (+ - * /) to obtain 24  
Input: 4 4 6 8  
Answer: (4 + 8) * (6 - 4) = 24  
Input: 2 9 10 12  
Answer: 2 * 12 * (10 - 9) = 24  
Input: 4 9 10 13  
Answer: (13 - 9) * (10 - 4) = 24  
Input: 14 8 8  
Answer: (8 / 4 + 1) * 8 = 24  
Input: 5 5 5 9  
Answer: 5 + 5 + 5 + 9 = 24  
Input: 11 2 12  
'''
```



It appears that you've provided a set of input numbers and specified arithmetic operations to obtain the result of 24 for each input. The provided examples demonstrate various mathematical expressions that yield the desired result. Here are the corresponding answers for the given inputs:

1. Input: 4 4 6 8  
Answer:  $(4 + 8) * (6 - 4) = 24$
2. Input: 2 9 10 12  
Answer:  $2 * 12 * (10 - 9) = 24$
3. Input: 4 9 10 13  
Answer:  $(13 - 9) * (10 - 4) = 24$
4. Input: 14 8 8  
Answer:  $(8 / 4 + 1) * 8 = 24$
5. Input: 5 5 5 9  
Answer:  $5 + 5 + 5 + 9 = 24$
6. Input: 11 2 12  
Answer:  $(1 + 1) * (2 + 12) = 24$

These are valid solutions that use basic arithmetic operations to reach the target number of 24 for each set of input numbers.

Chain of Thoughts – see cot.py in my code

# Tree of Thoughts

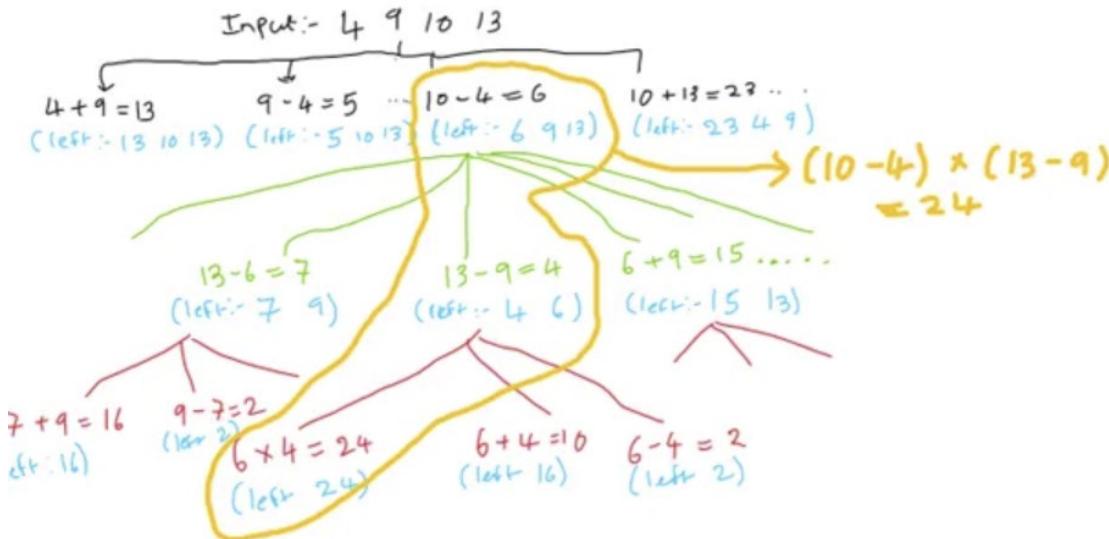


Fig 1: Solve a problem using a systematic way — Tree based approach

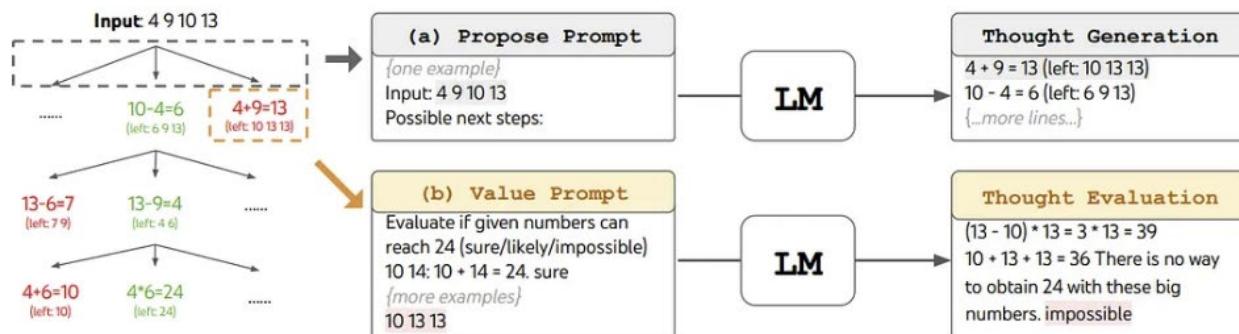


Fig 2: Source: Tree of Thoughts <https://arxiv.org/pdf/2305.10601.pdf>

We can solve this in 3 steps:

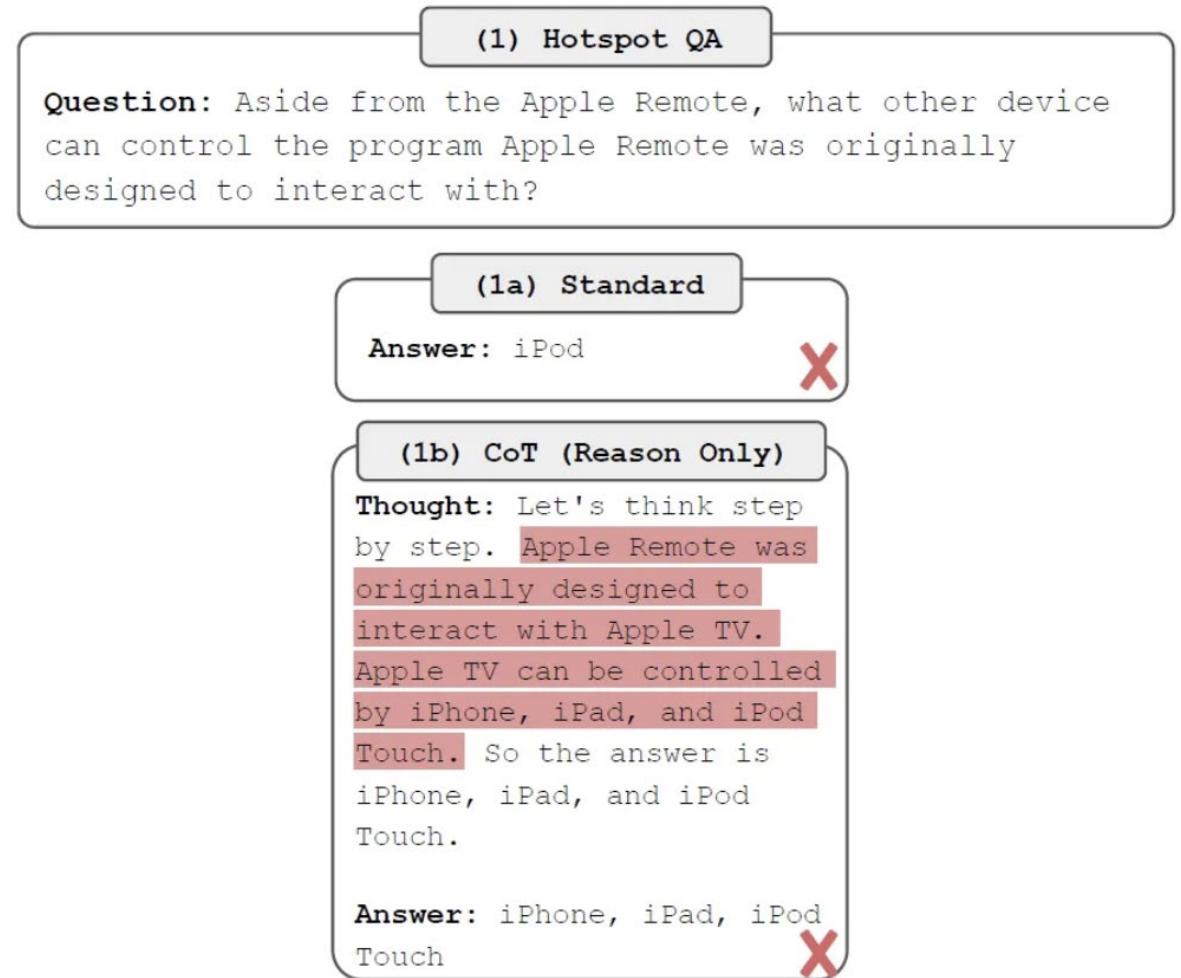
**Step 1 :** Take 2 numbers and apply all the arithmetic operation at the first level for e.g.  $4 + 9 = 13$  — sum of first 2 numbers

**Step 2 :** Identify left numbers as we need to use each number once. Once first 2 numbers are used we left with (left 13 10 13)

**Step 3 :** Continue to build the tree till we receive final sum.

# Limitations of CoT Reasoning

- Chain-of-thought reasoning is intended to combat reasoning errors.
- Giving the LLM one or more examples (few-shot learning) and illustrating how to reason through examples, helps to solve a different problem in a more accurate way.
- But it still suffers from hallucination, and hallucinated “facts” can propagate through the reasoning, causing the model to come to the wrong conclusion regardless.
- ReAct aims to solve this issue by allowing the LLM to take actions such as searching Wikipedia so that it can find facts and reason from those.



# Self Consistency

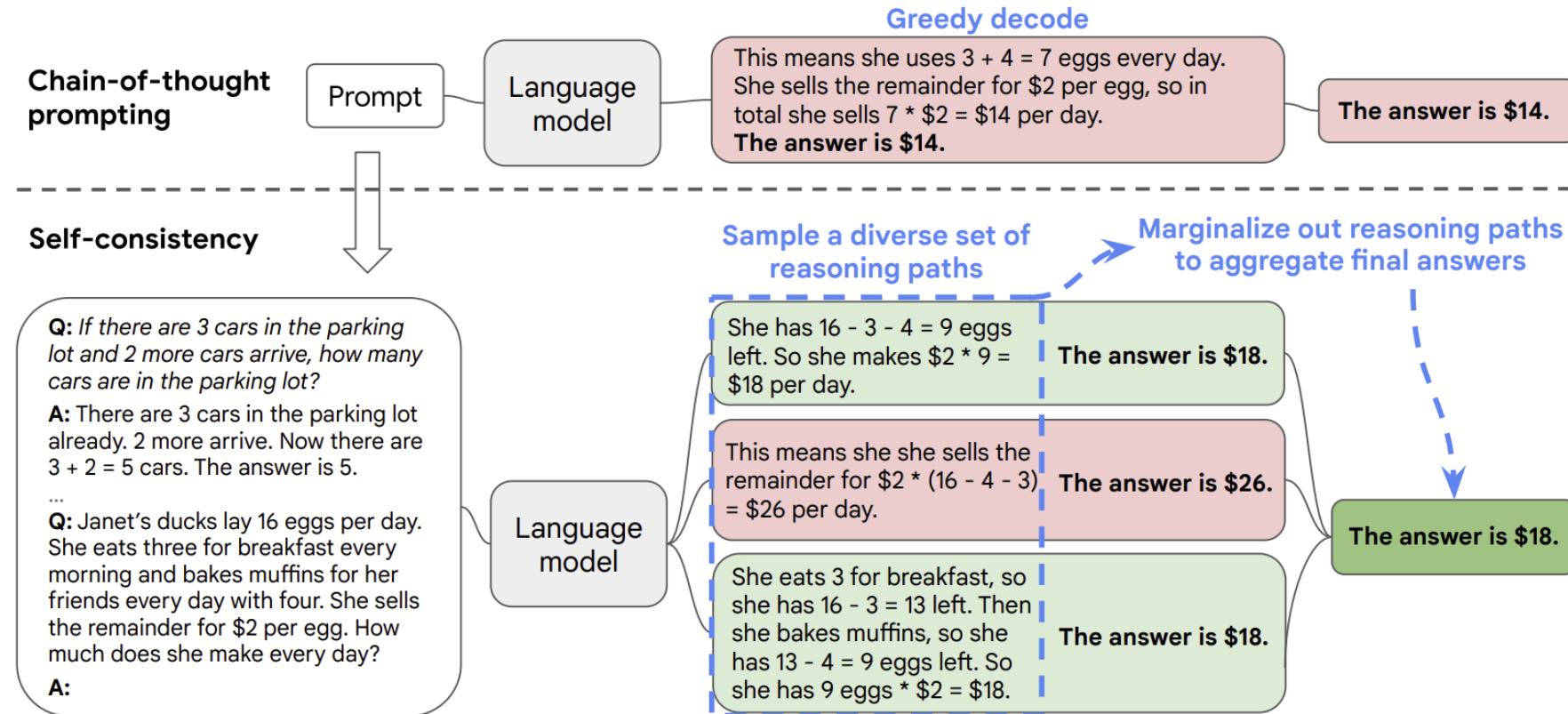
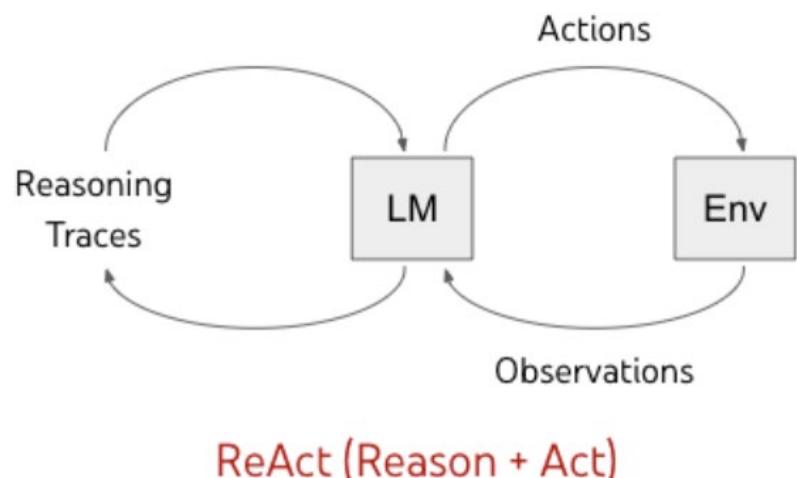
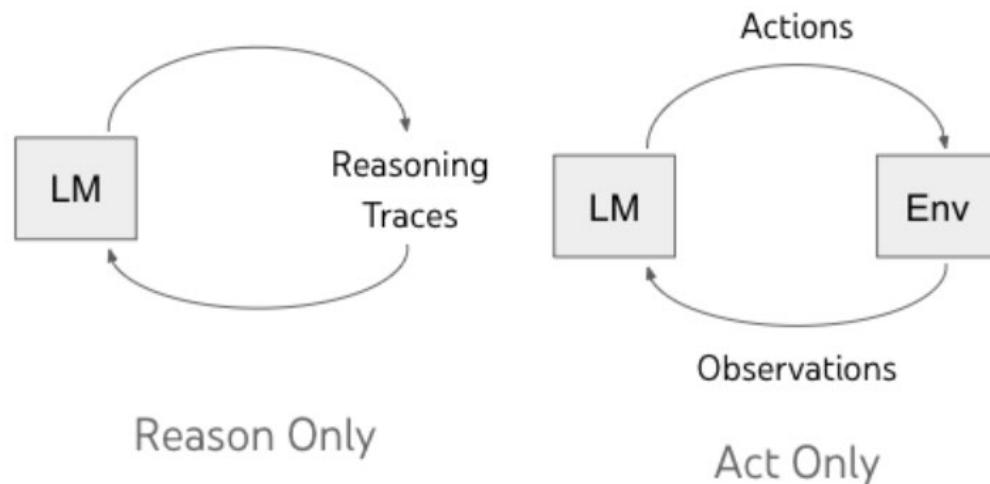


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the “greedy decode” in CoT prompting by sampling from the language model’s decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set.

# ReAct Approach: Key Idea



# Approach

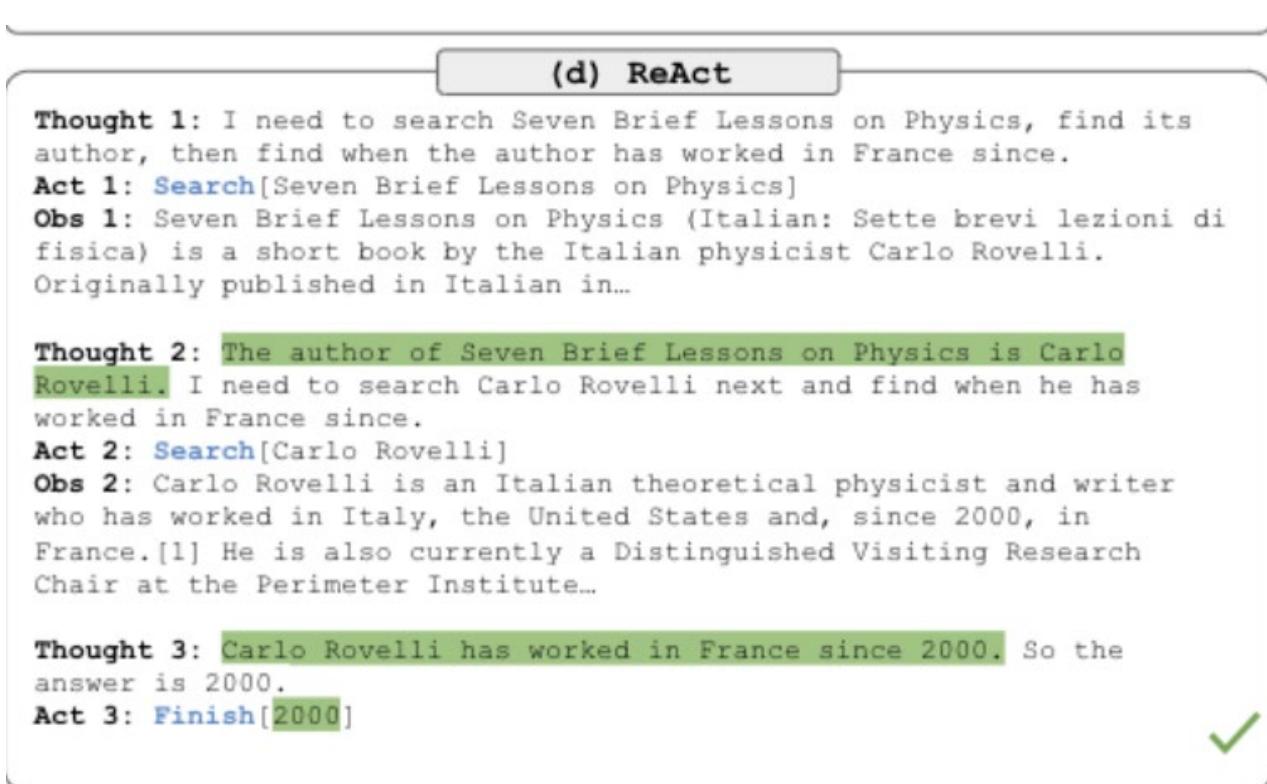
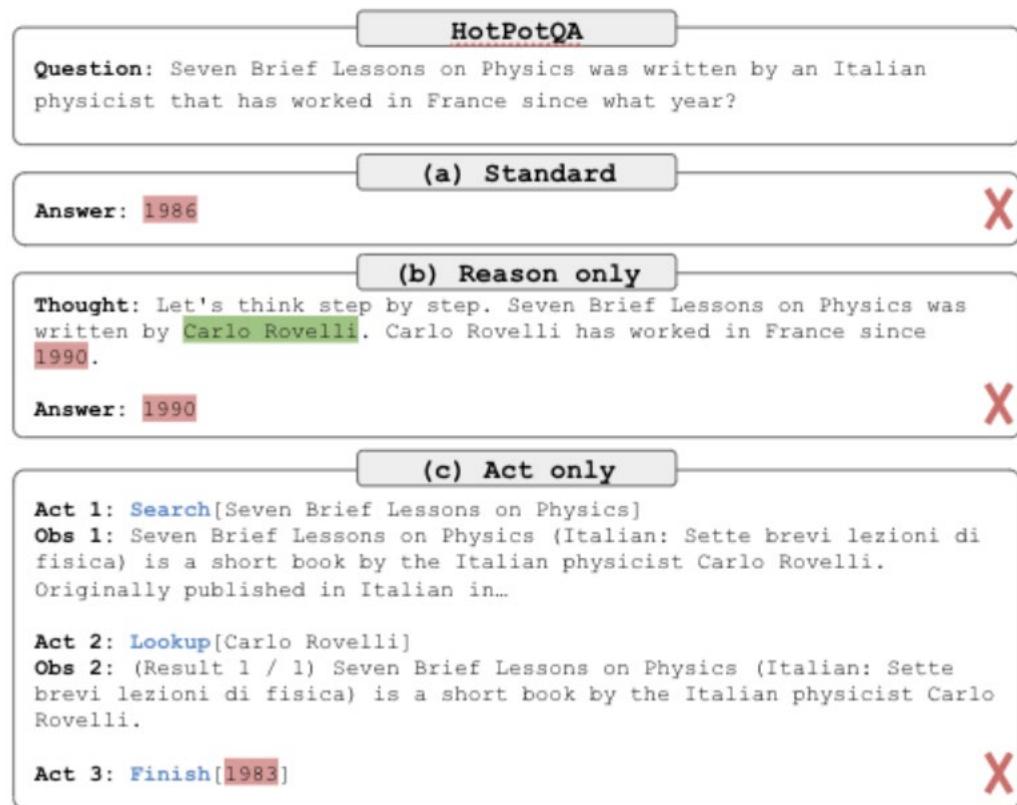
1. An environment that takes a text action (out of a set of potential actions which can change based on the environment's internal state) and returns a text observation.
2. An output parser framework that stops the agent from generating text once it has written a valid action, executes that action in the environment, and returns the observation (appends it to the text generated so far and prompts the LLM with that).
3. Human-generated examples of intermixed thoughts, actions, and observations in the environment to use for few-shot learning.

# Illustration

Question	What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?
Thought 1	I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.
Action 1	Search[Colorado orogeny]
Observation 1	The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.
Thought 2	It does not mention the eastern sector. So I need to look up eastern sector.
Action 2	Lookup[eastern sector]
Observation 2	(Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.
Thought 3	The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.
Action 3	Search[High Plains]

Yao et al. (2023)

# Illustration#2



# Illustration#3

## ReAct

Claim: Reign Over Me is an American film made in 2010.  
Thought 1: I need to search for Reign Over Me and find if it is an American film made in 2010.  
Action 1: Search[Reign Over Me]  
Observation 1: Reign Over Me ...  
Thought 2: The observation says that it is an American film made in 2007, so it is not made in 2010.  
Action 2: Finish[REFUTES]  
Observation 2: Episode finished

---

## CoT

Claim: Reign Over Me is an American film made in 2010.  
Thought: Let's think step by step. First, is Reign Over Me an American film? Yes, it is. Second, was it made in 2010? Yes, it was.  
Answer: SUPPORTS

---

*A comparison of the ReAct (top) and CoT (bottom) reasoning trajectories on an example from Fever (observation for ReAct is omitted to reduce space). In this case ReAct provided the right answer, and it can be seen that the reasoning trajectory of ReAct is more grounded on facts and knowledge, in contrast to CoT's hallucination behavior.*

# Demo: ReAct with LangChain, Tavley

# RAG Techniques

Palacode Narayana Iyer Anantharaman

6<sup>th</sup> Sep 2024

# Why Retrieval Augmented Generation (RAG)?

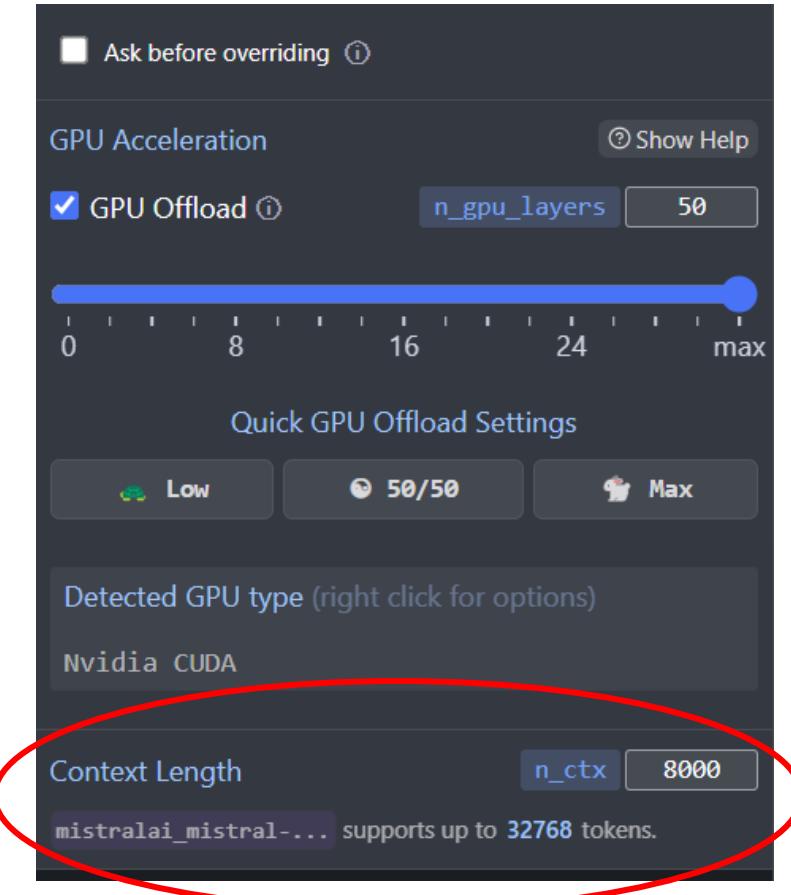
- The datasets that were used for LLM training are not live and real time.



- Training datasets are usually public information. Private data are not included in foundational LLMs. Supporting custom data is a major requirement for the LLMs. For example:
  - Proprietary code, design documents
  - Company confidential financial documents
- One way to provide live data to LLM is by using the context window. But this has limitations.

# Context Limitations

- LLM's are unaware of concepts outside of their training set
- Filling gaps in knowledge with assumptions
- Very hard to teach LLM's about new concepts



# Naïve RAG: Illustration

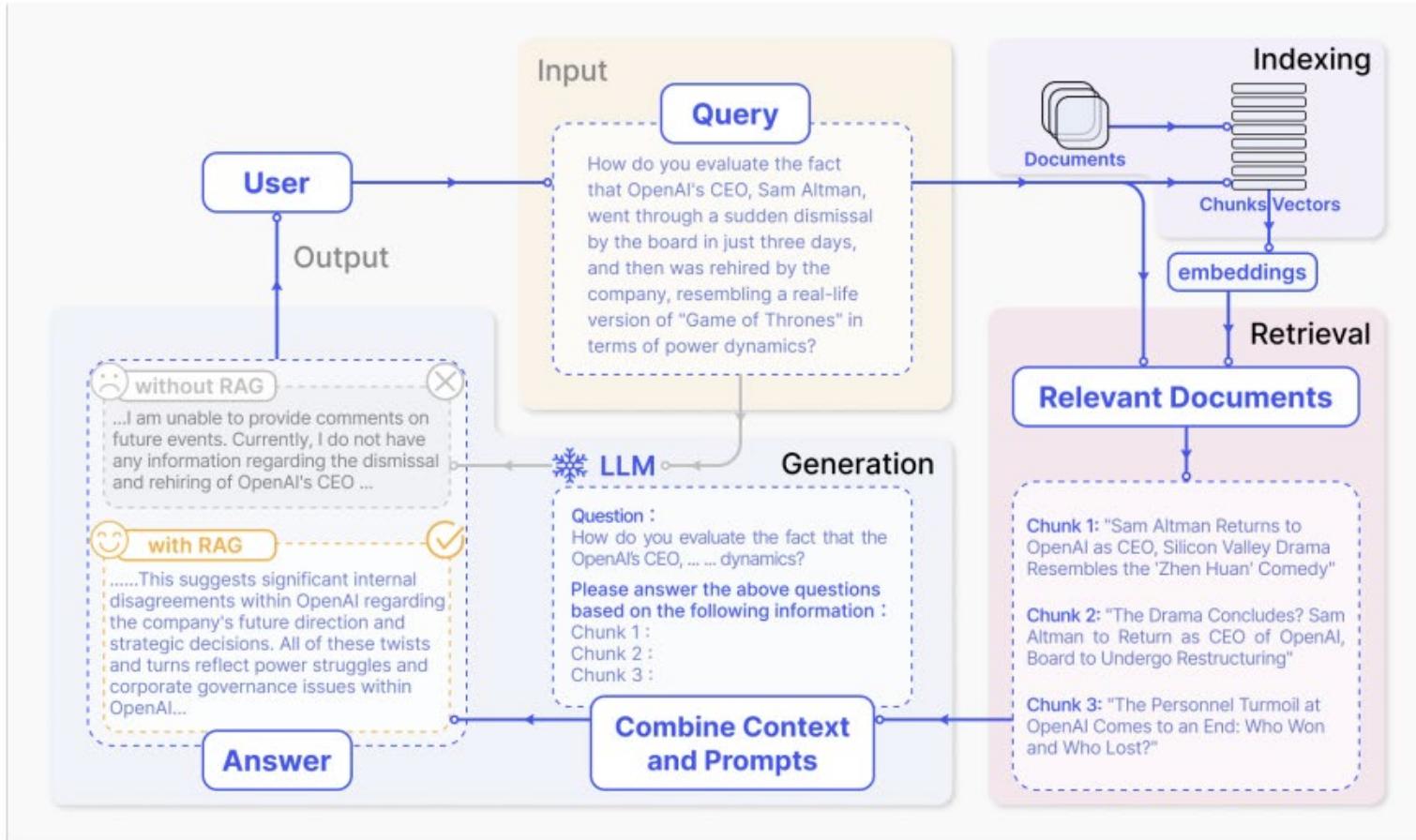


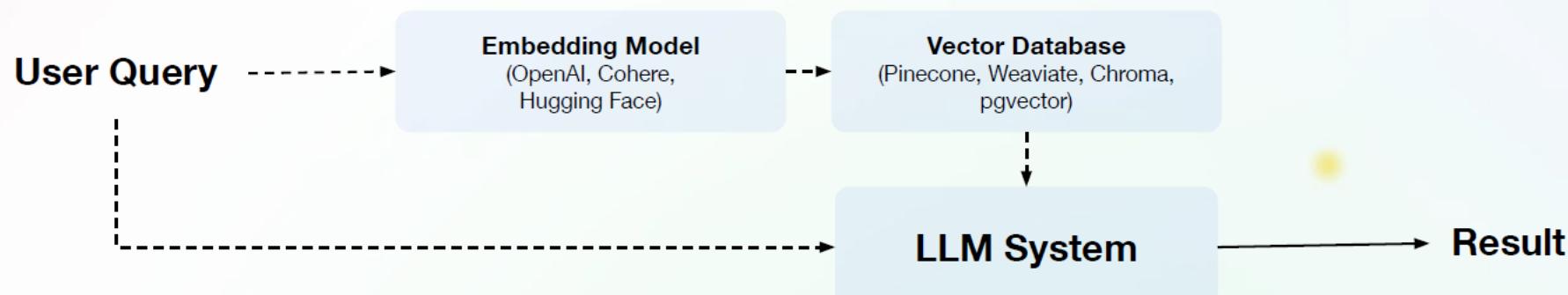
Fig. 2. A representative instance of the RAG process applied to question answering. It mainly consists of 3 steps. 1) Indexing. Documents are split into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the Top k chunks most relevant to the question based on semantic similarity. 3) Generation. Input the original question and the retrieved chunks together into LLM to generate the final answer.

Fig Credits: RAG for LLM Models – A survey, Gao et al.

# Retrieval Augmented Generation

## Use-cases:

- **Knowledge Base question answering**
  - Library documentation
  - Technical documents
  - Code
- **Technical Summarization**



Ref: "Emerging architectures for LLM applications", <https://www.youtube.com/watch?v=Pft04KLw5Lk&t=670s>

# Euclidean Distance and Cosine Similarity

- Euclidean Distance is defined as:

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Where  $x$  and  $y$  are two vectors. Or:

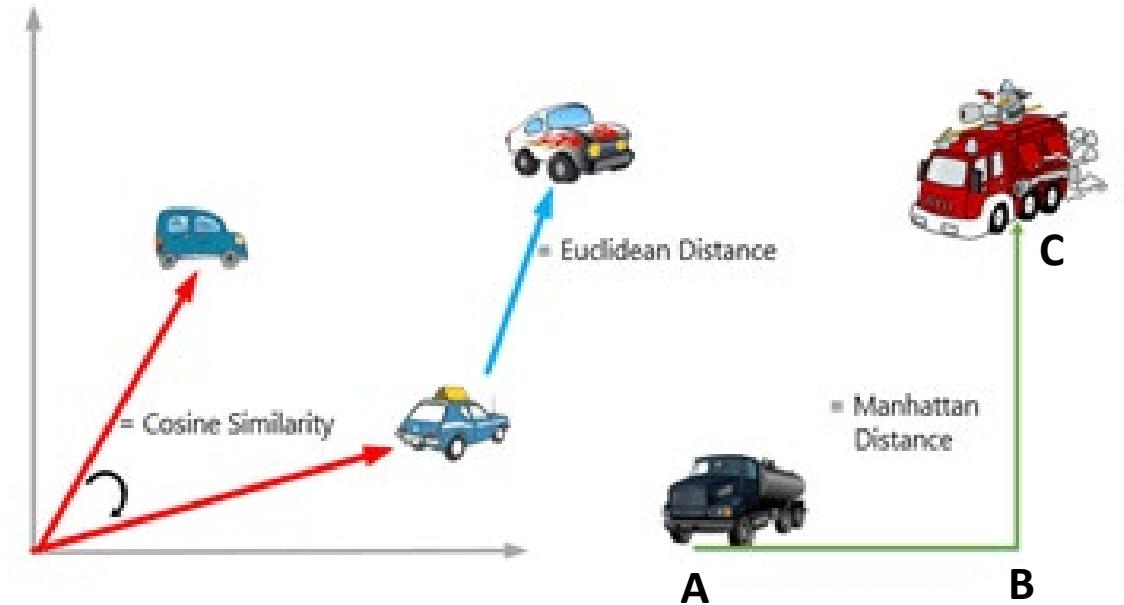
```
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y) ** 2))
```

- Cosine Similarity:

$$\frac{x \cdot y}{\sqrt{x \cdot x} \sqrt{y \cdot y}}$$

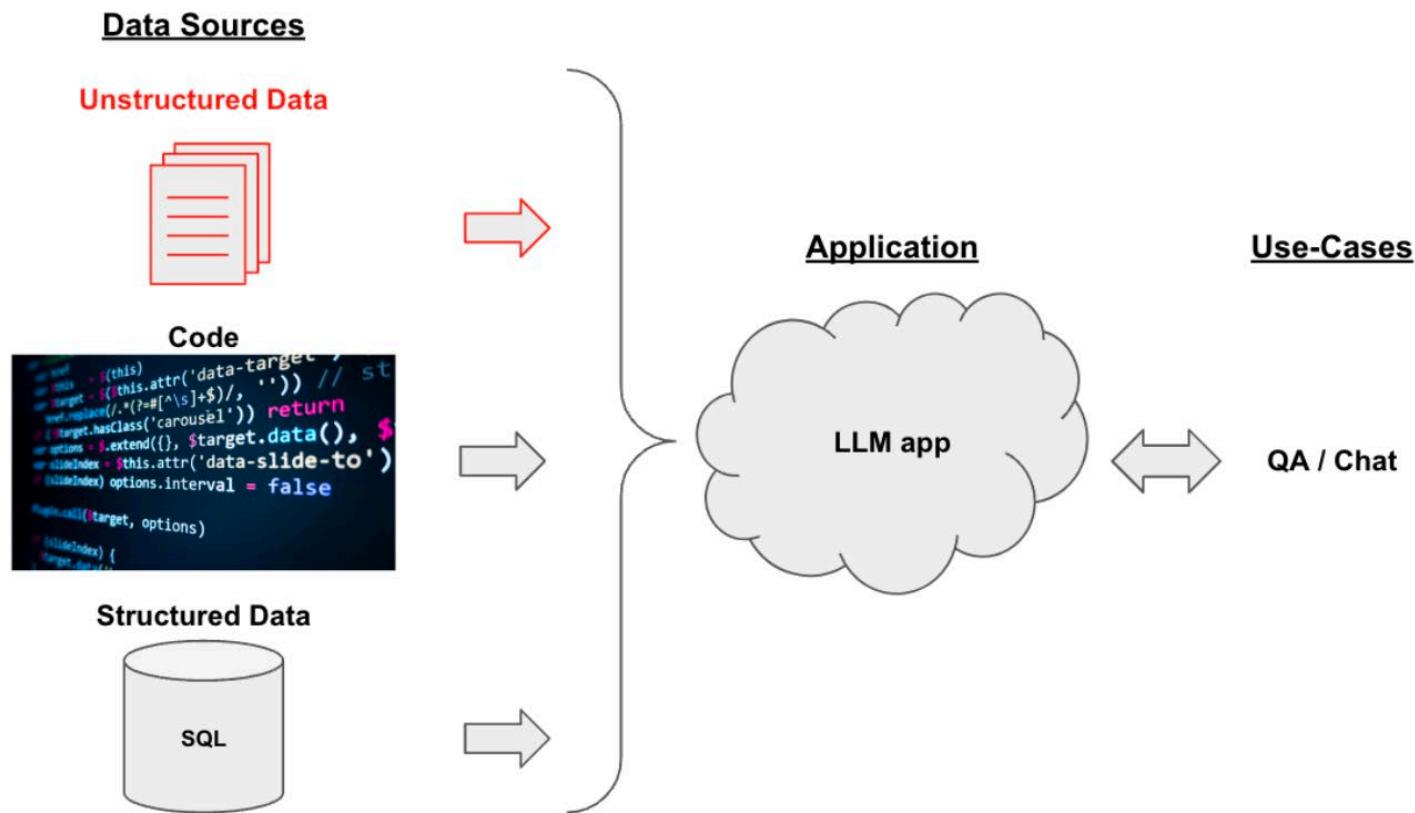
Where  $x$  and  $y$  are two vectors. Or:

```
def cosine_similarity(x, y):
    return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))
```

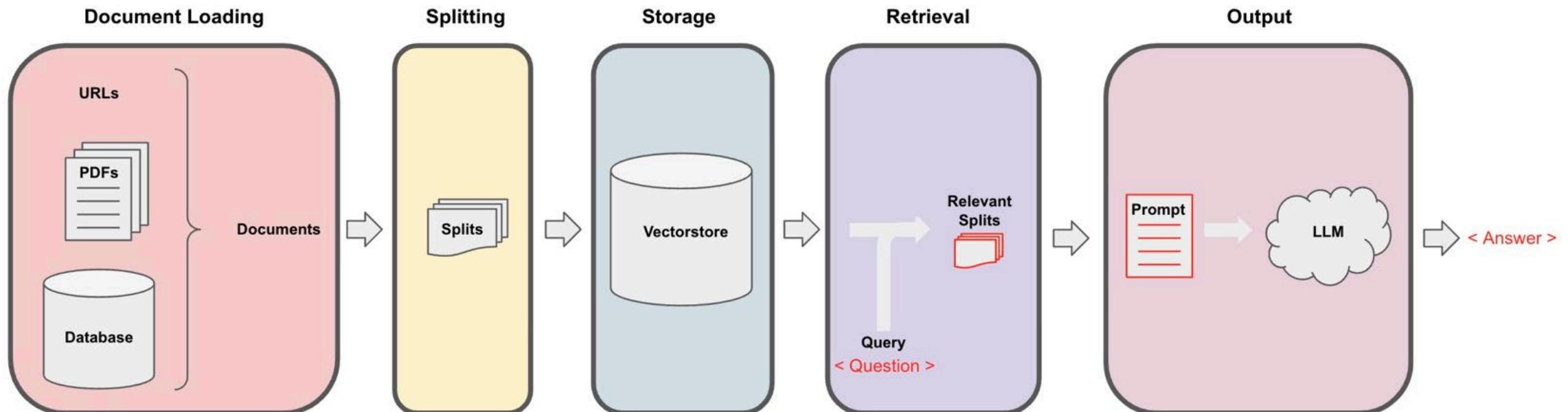


# Question Answering on custom documents

- QA over structured data (e.g., SQL)
- QA over code (e.g., Python)



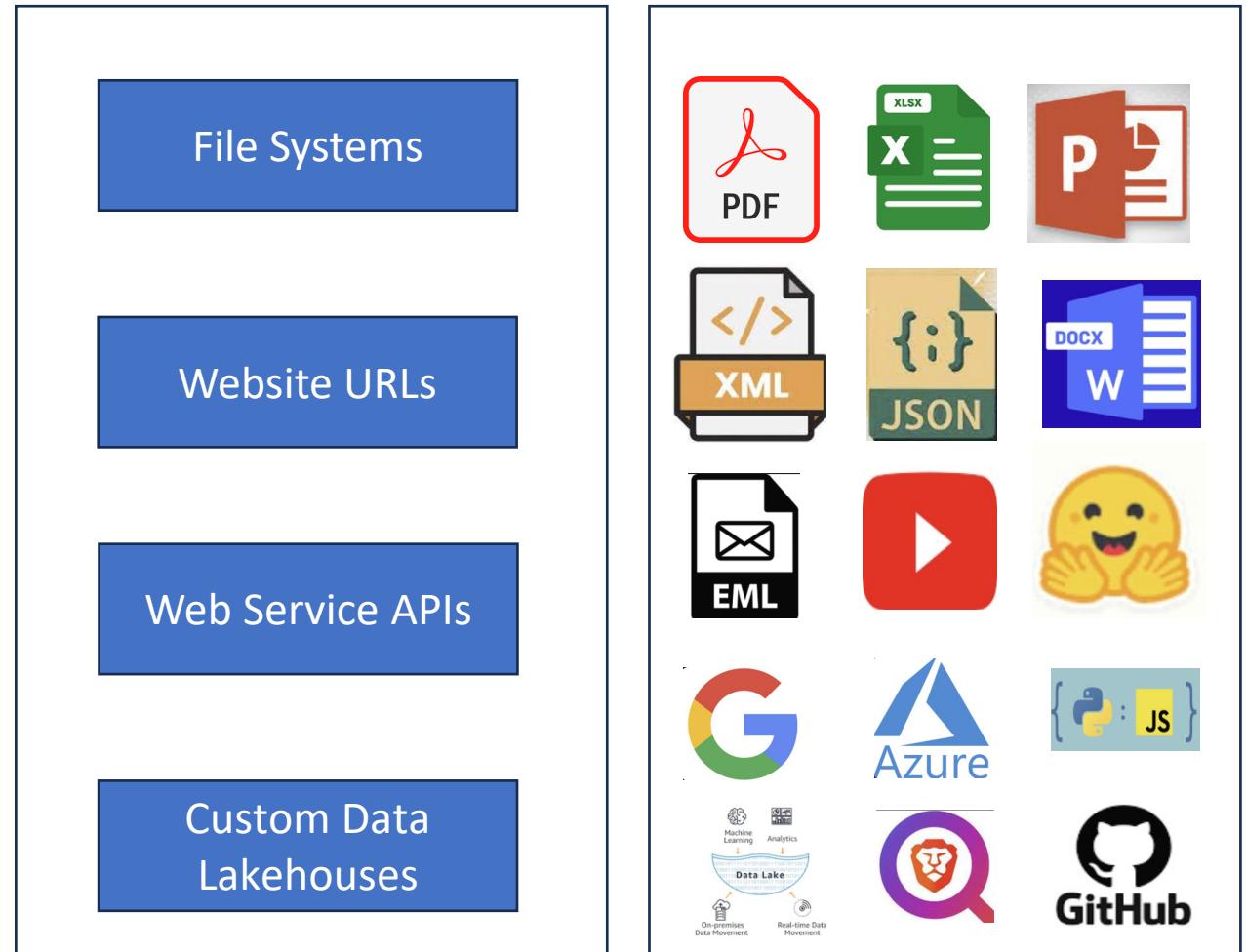
# Langchain Pipeline for Question Answering



- The above subsystems can be viewed as independent modules that obey well defined interfaces
- This means that one can replace a module with another technique, so long as interfaces are respected

# RAG Big Picture: Data Sources and Data formats

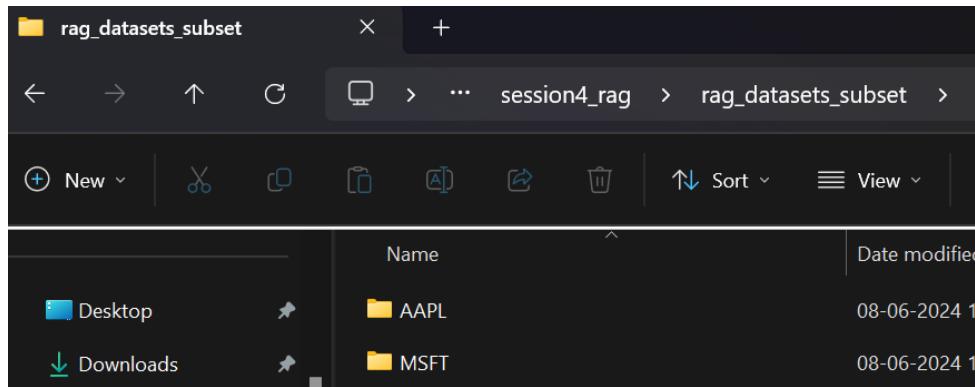
- We work with different data sources that arise from local file systems to web/cloud based resources.
- SaaS products, e-mails, Web Service APIs, Cloud databases etc are massive sources of data.
- Data can be unstructured (e.g. natural language texts in PDF documents) or structured (json data, excel tables storing numerical values), images, videos, audio, tables, source code, etc
- Data is represented in a variety of formats – pdfs, xlsx, pptx, py, json....
- Langchain doc loaders allow access to data using a common interface: Document class.



# RAG Case Study #1 – Financial Analysis

- We are given a dataset that has annual reports of Apple and Microsoft for 2018. Each annual report is split in to a number of PDF documents.
- We are required to build a Copilot Assistant, that takes a question from the user and provides the best possible answer.
- Extend this product to provide summarization.
- You can build an end to end product by including suitable UI and also graphics.
- With multimodal LLMs (e.g. Microsoft Phi-3V) it is possible to extend the problem to chat with charts and other visual components.

# Dataset Organization



Name	Date modified
Desktop	08-06-2024 12:20
Downloads	08-06-2024 12:20
Documents	08-06-2024 12:20
2016	08-06-2024 12:20
2017	08-06-2024 12:20
2018	08-06-2024 12:20

Name	Date modified	Type
Desktop	08-06-2024 11:56	Ado
Downloads	08-06-2024 11:56	Ado
Documents	08-06-2024 11:56	Ado
Pictures	08-06-2024 11:56	Ado
Music	08-06-2024 11:56	Ado
page_3	08-06-2024 11:56	Ado
page_23	08-06-2024 11:56	Ado
page_24	08-06-2024 11:56	Ado
page_26	08-06-2024 11:56	Ado
page_27	08-06-2024 11:56	Ado

# Sample page in PDF

## Greater China

The following table presents Greater China net sales information for 2016, 2015 and 2014 (dollars in millions):

	2016	Change	2015	Change	2014
Net sales	\$ 48,492	(17)%	\$ 58,715	84%	\$ 31,853
Percentage of total net sales	22%		25%		17%

Greater China net sales decreased during 2016 compared to 2015 due primarily to lower net sales and unit sales of iPhone and the effect of weakness in foreign currencies relative to the U.S. dollar.

Greater China experienced strong year-over-year increases in net sales during 2015 driven primarily by iPhone sales.

## Japan

The following table presents Japan net sales information for 2016, 2015 and 2014 (dollars in millions):

	2016	Change	2015	Change	2014
Net sales	\$ 16,928	8%	\$ 15,706	3%	\$ 15,314
Percentage of total net sales	8%		7%		8%

Japan net sales increased during 2016 compared to 2015 due primarily to higher net sales of Services and the strength in the Japanese yen relative to the U.S. dollar.

The year-over-year increase in Japan net sales during 2015 was driven primarily by growth in Services largely associated with strong App Store sales, partially offset by the effect of weakness in the Japanese yen relative to the U.S. dollar.

## Rest of Asia Pacific

The following table presents Rest of Asia Pacific net sales information for 2016, 2015 and 2014 (dollars in millions):

	2016	Change	2015	Change	2014
Net sales	\$ 13,654	(10)%	\$ 15,093	34%	\$ 11,248
Percentage of total net sales	6%		6%		6%

## Gross Margin

Gross margin for 2016, 2015 and 2014 is as follows (dollars in millions):

	2016	2015	2014
Net sales	\$ 215,639	\$ 233,715	\$ 182,795
Cost of sales	131,376	140,089	112,258
Gross margin	\$ 84,263	\$ 93,626	\$ 70,537
Gross margin percentage	39.1%	40.1%	38.6%

Gross margin decreased in 2016 compared to 2015 due primarily to the effect of weakness in most foreign currencies relative to the U.S. dollar and, to a lesser extent, unfavorable leverage on fixed costs from lower net sales, partially offset by a favorable shift in mix to Services.

The year-over-year increase in the gross margin percentage in 2015 was driven primarily by a favorable shift in mix to products with higher margins and, to a lesser extent, by improved leverage on fixed costs from higher net sales. These positive factors were partially offset primarily by higher product cost structures and, to a lesser extent, by the effect of weakness in most foreign currencies relative to the U.S. dollar.

## OTHER INCOME (EXPENSE), NET

The components of other income (expense), net were as follows:

(In millions)	2018	2017	2016
Year Ended June 30,			
Dividends and interest income	\$ 2,214	\$ 1,387	\$ 903
Interest expense	(2,733)	(2,222)	(1,243)
Net recognized gains on investments	2,399	2,583	668
Net losses on derivatives	(187)	(510)	(443)
Net losses on foreign currency remeasurements	(218)	(111)	(129)
Other, net	(59)	(251)	(195)
Total	\$ 1,416	\$ 876	\$ (439)

# Naïve RAG Implementation: Ingesting

- Step#1: Load the required documents (In our case all docs are PDF)

```
def get_docs(source="arxiv"):
    if source == "arxiv":
        docs = ArxivLoader(query="1706.03762", load_max_docs=2).load()
    else:
        loader = DirectoryLoader(DATA_PATH, glob="*.pdf", loader_cls=PyPDFLoader, recursive=True)
        docs = loader.load()
    return docs
```

- Step#2: Split each document to multiple chunks

```
def get_chunks(docs, chunk_size=512, chunk_overlap=50):
    """
    Given docs obtained by using LangChain Loader, split these to chunks and return them
    :param docs: documents returned by loader, that could be ArxivLoader or local directory loader
    :return: chunks
    """

    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    texts = text_splitter.split_documents(docs)
    return texts
```

# Naïve RAG Implementation: Ingesting

- Step#3: Define an embedding model and embed each chunk
- Step#4: Embed and store each chunk in the vector store (ChromaDB)

```
def get_embeddings_model(model_name=None, device="cuda"):  
    if model_name is None:  
        model_name = EMBEDDINGS_MODEL  
    embeddings_model = HuggingFaceEmbeddings(model_name=model_name,  
                                              model_kwargs={"device": device})  
    return embeddings_model  
  
def create_vector_store(texts, embeddings, db_path, use_db="chroma"):  
    """  
    Given the chunks, their embeddings and path to save the db, save and persist the data in the data store  
    :param texts: chunks for which we are constructing the data store  
    :param embeddings: vector embeddings for given chunks  
    :param db_path: storage path  
    :param use_db: type of data store to use  
    :return: None  
    """  
    flag = True  
    try:  
        if use_db == "chroma":  
            db = Chroma.from_documents(texts, embeddings, persist_directory=db_path)  
        else:  
            print("Unknown db type, exiting!")  
            db = None  
            import sys  
            sys.exit(-1)  
        db.persist()  
    except:  
        flag = False  
        print_exc()  
        print("Exception when creating data store: ", db_path, use_db)  
    return flag
```

# Naïve RAG Implementation: Model Creation

- Define required prompt

```
custom_prompt_template = """  
<s> [INST] You are an assistant for question-answering tasks.  
Use the following pieces of retrieved context to answer the question.  
If you don't know the answer, just say that you don't know.  
Keep the answer concise. [/INST] </s>  
[INST] Question: {question}  
Context: {context}  
Answer: [/INST]  
"""
```

```
def set_custom_prompt():  
    prompt = PromptTemplate(template=custom_prompt_template, input_variables=['context', 'question'])  
    return prompt
```

- Load LLM

```
def load_llm():  
    llm_config = {  
        "temperature": 0,  
        'max_new_tokens': 8000, # 2048,  
        "context_length": 8000, # 2048,  
        'gpu_layers': 50,  
    }  
  
    llm = CTransformers(  
        # model=r'C:\Users\ananth\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.1-GGUF',  
        model=r'C:\Users\ananth\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.2-GGUF',  
        model_file=r'C:\Users\ananth\.cache\lm-studio\models\TheBloke\Mistral-7B-Instruct-v0.2-GGUF\mistral-7b-instruct-v0.2.04_K_M.qquf',  
        config=llm_config)  
  
    return llm
```

# Naïve RAG Implementation: Model Creation

- Create a retriever (retrieves relevant docs from Vector Store, based on the query)

```
def get_retriever():
    """
    after texts are ingested in vectordb, get it as a retriever
    :return:
    """
    embeddings = HuggingFaceEmbeddings(model_name=EMBEDDINGS_MODEL,
                                        model_kwargs={'device': 'cuda'})
    vectordb = Chroma(persist_directory=DB_CHROMA_PATH, embedding_function=embeddings)
    return vectordb
```

- Create a document formatter (formats retrieved docs in to a form suitable for LLM to process)

```
def format_docs(docs):
    for doc in docs:
        print(doc)
    return "\n\n".join([d.page_content for d in docs])
```

# Naïve RAG Implementation: Model

- Build the qa\_bot

```
def qa_bot():
    vectordb = get_retriever()
    retriever = vectordb.as_retriever(search_kwargs={"k": 10})

    # retrieved_docs = retriever.invoke("What products Micro Labs produce?")
    # print("Num retrieved docs = ", len(retrieved_docs))
    # print(retrieved_docs[0].page_content)

    llm = load_llm()
    print("LLM Loaded: ", llm)

    chain = (
        {"context": retriever | format_docs, "question": RunnablePassthrough()}
        | set_custom_prompt()
        | llm
        | StrOutputParser()
    )
    query = ""
    while query != "quit":
        query = input("Your Query: ")
        output = chain.invoke(query)
        print(output)
```

# Demo: Financial Analysis

- We will build a product that allows a business strategist to analyze financial results of our company along with our key competitors and formulate our strategy.
  - How does our competitor grow over last 3 years?
  - What are their new investments? New products?
  - What acquisitions they have made?
  - Are they expanding their operations in different geos?
  - What are their retention levels?
  - Can we visualize these with a finance dashboard?

# Some challenges

- If the matches are a large number of chunks, say 50 and we have set  $k = 20$ , we may miss vital information.
- Suppose we set  $k=100$ , then there is a possibility of overflowing the LLM context.
- Chunk size and overlap settings can affect the accuracy
- Suppose all chunks returned has only Apple's documents and no Microsoft, we can't compare them.
- When the number of source documents are very large and if the query requires answers contained in each document, one may have to deal with a large number of chunks.

# Splitting/Chunking

- PDF Documents may contain images and tables besides text data.
- Chunking them with character or recursive character splitter is not enough.
- Similarly, splitting source code (e.g. Python, JS and so on) requires an understanding of the programming language.
- In such cases, a sophisticated semantic approach to chunking is required.
- Refer: [RetrievalTutorials/tutorials/LevelsOfTextSplitting/5\\_Levels\\_Of\\_Text\\_Splitting.ipynb](https://RetrievalTutorials/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb) at [main · FullStackRetrieval.com/RetrievalTutorials · GitHub](https://main.FullStackRetrieval.com/RetrievalTutorials)

Published as a conference paper at ICLR 2023

Prompt Method <sup>a</sup>	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT (Wei et al., 2022)	29.4	56.3
CoT-SC (Wang et al., 2022a)	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	<b>64.6</b>
ReAct → CoT-SC	<b>35.1</b>	62.0
Supervised SoTA <sup>b</sup>	67.5	89.5

Table 1: PaLM-540B prompting results on HotpotQA and Fever.

<sup>a</sup>HotpotQA EM is 27.1, 28.9, 33.8 for Standard, CoT, CoT-SC in Wang et al. (2022b).

<sup>b</sup>(Zhu et al., 2021; Lewis et al., 2020)

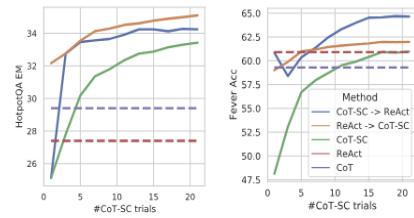


Figure 2: PaLM-540B prompting results with respect to number of CoT-SC samples used.

search reformulation (“maybe I can search/look up x instead”), and synthesize the final answer (“...so the answer is x”). See Appendix C for more details.

**Baselines** We systematically ablate ReAct trajectories to build prompts for multiple baselines (with formats as Figure 1(a-1c)): (a) **Standard prompting** (Standard), which removes all thoughts, actions, observations in ReAct trajectories. (b) **Chain-of-thought prompting** (CoT) (Wei et al., 2022), which removes actions and observations and serve as a reasoning-only baseline. We also build a self-consistency baseline (CoT-SC) (Wang et al., 2022a,b) by sampling 21 CoT trajectories with decoding temperature 0.7 during inference and adopting the majority answer, which is found to consistently boost performance over CoT. (c) **Acting-only prompt** (Act), which removes thoughts in ReAct trajectories, loosely resembling how WebGPT (Nakano et al., 2021) interacts with the Internet to answer questions, though it operates on a different task and action space, and uses imitation and reinforcement learning instead of prompting.

# Chunk Visualization Demo

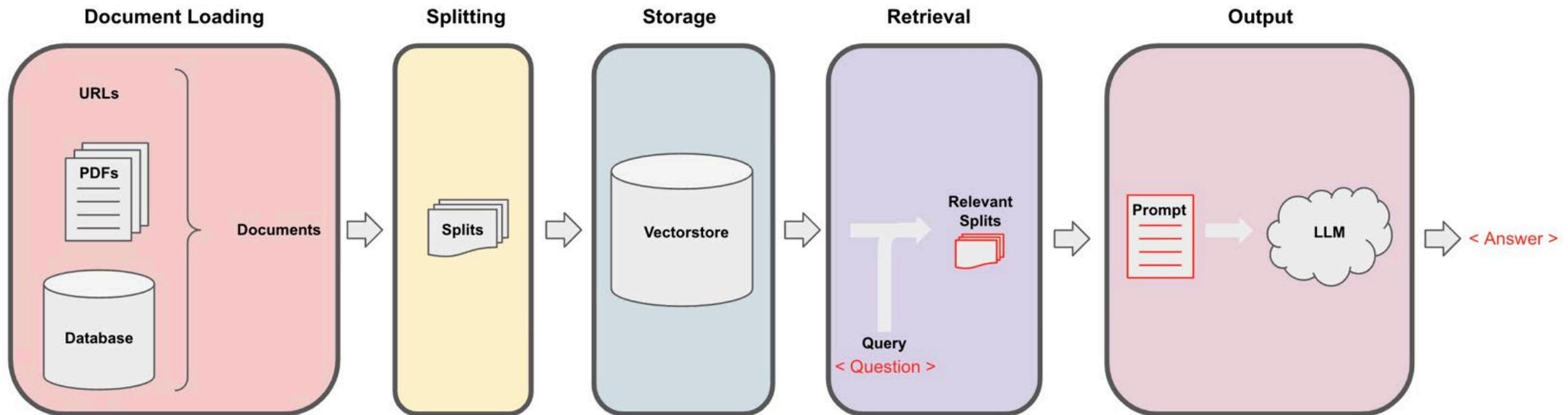
- Refer: [https://huggingface.co/spaces/m-ric/chunk visualizer](https://huggingface.co/spaces/m-ric/chunk_visualizer)

# RAG Techniques – Part 2

Palacode Narayana Iyer Anantharaman

13<sup>th</sup> Sep 2024

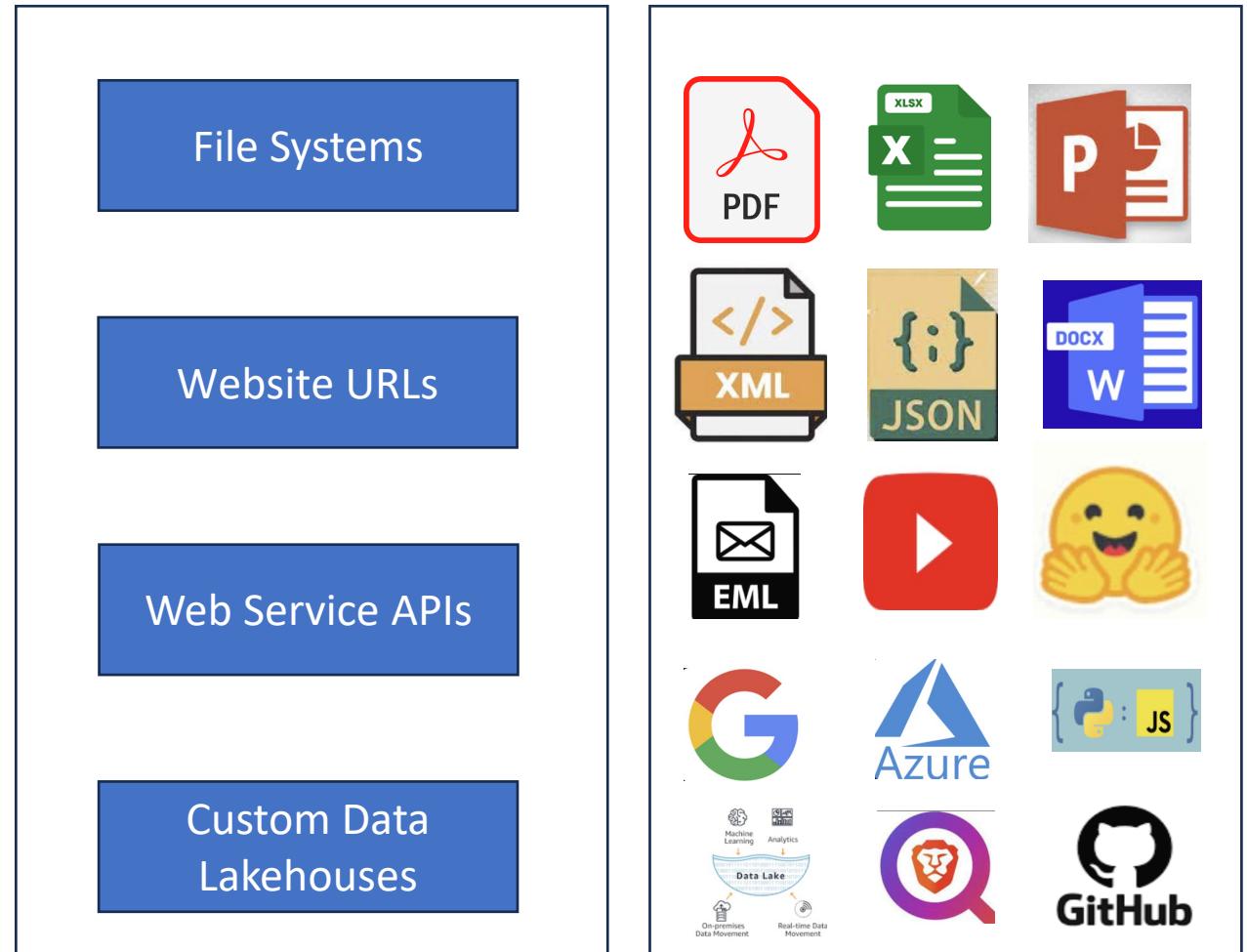
# Langchain Pipeline for Question Answering



- The above subsystems can be viewed as independent modules that obey well defined interfaces
- This means that one can replace a module with another technique, so long as interfaces are respected

# RAG Big Picture: Data Sources and Data formats

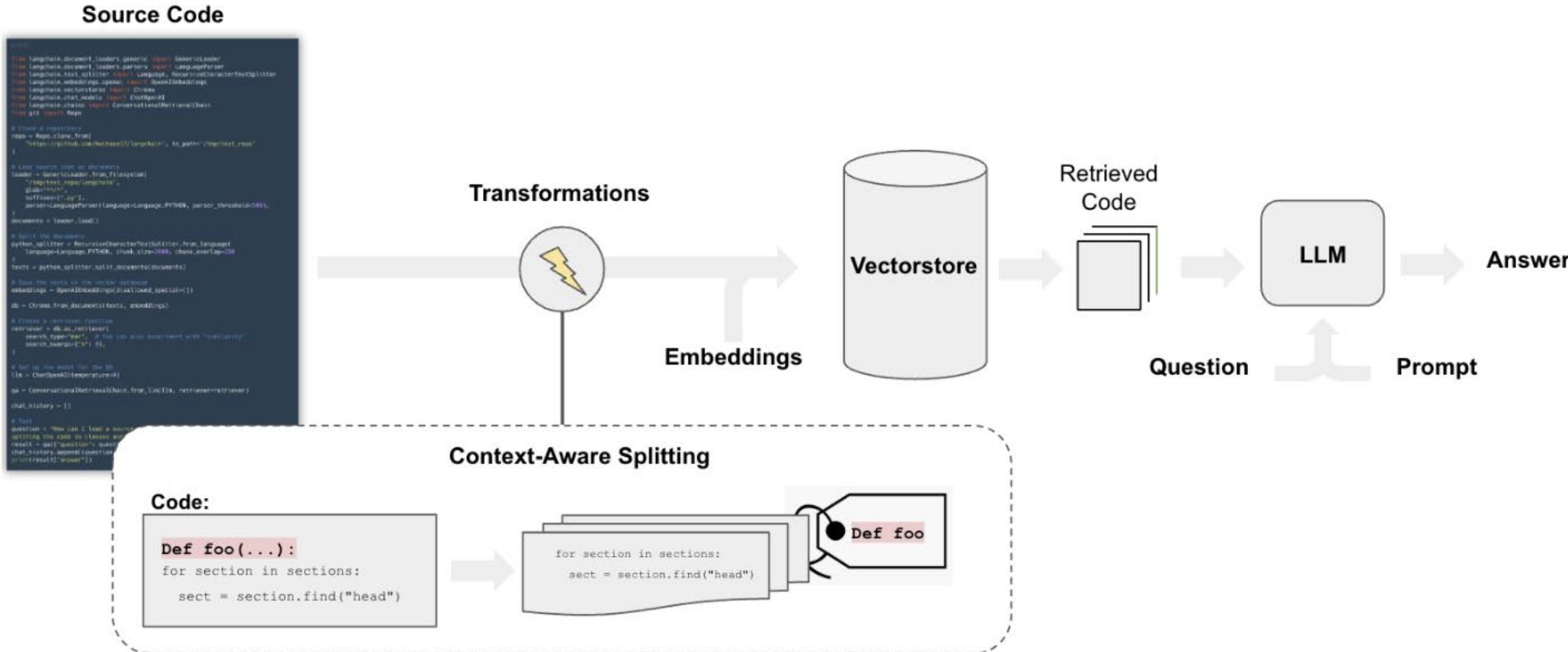
- We work with different data sources that arise from local file systems to web/cloud based resources.
- SaaS products, e-mails, Web Service APIs, Cloud databases etc are massive sources of data.
- Data can be unstructured (e.g. natural language texts in PDF documents) or structured (json data, excel tables storing numerical values), images, videos, audio, tables, source code, etc
- Data is represented in a variety of formats – pdfs, xlsx, pptx, py, json....
- Langchain doc loaders allow access to data using a common interface: Document class.



# RAG Case Study #2: Chat with source code

- Ask the RAG to list specific functions (searching for certain functions)
- Get the function and ask the LLM to code review it
- Get some test cases from LLM for the specified classes
- Ask the LLM to performance optimize the given module
- Refactor some code using the LLM

# RAG over code



# Chat with your source code

- Splitting strategy that does a few things:
  - Keeps each top-level function and class in the code is loaded into separate documents.
  - Puts remaining into a separate document.
  - Retains metadata about where each split comes from
- Demo: Code splitter with LangChain
- Refer:  
[https://python.langchain.com/docs/integrations/document\\_loaders/source\\_code](https://python.langchain.com/docs/integrations/document_loaders/source_code)

# Demo: RAG on Source code

- Develop a RAG based chatbot that chats with your source code.
- You can support only Python for this exercise.
- E.g. Try it on your own code in your product or clone any github python repository and chat with that code.
- See the sample code and the demo provided here: `ingest_code.py` and `coding_model.py`

# Hands On Exercise: Cricket Analysis

- You are given a dataset pertaining to Rohit Sharma's international centuries. Download this from:  
<https://www.kaggle.com/datasets/mayankanand2701/rohit-sharma-international-centuries?select=Rohit%20Sharma%20Centuries.xlsx>
- You are required to implement a RAG Q&A system ingesting this excel sheet.
- Use the excel document loader for your implementation:  
[https://python.langchain.com/v0.2/docs/integrations/document\\_loaders/microsoft\\_excel/](https://python.langchain.com/v0.2/docs/integrations/document_loaders/microsoft_excel/)
- Write a suitable prompt to instruct the LLM (see my other examples)
- Generate a number of questions using ChatGPT or any other LLM
- Discuss RAG's limitations in the class.

# Dataset Description

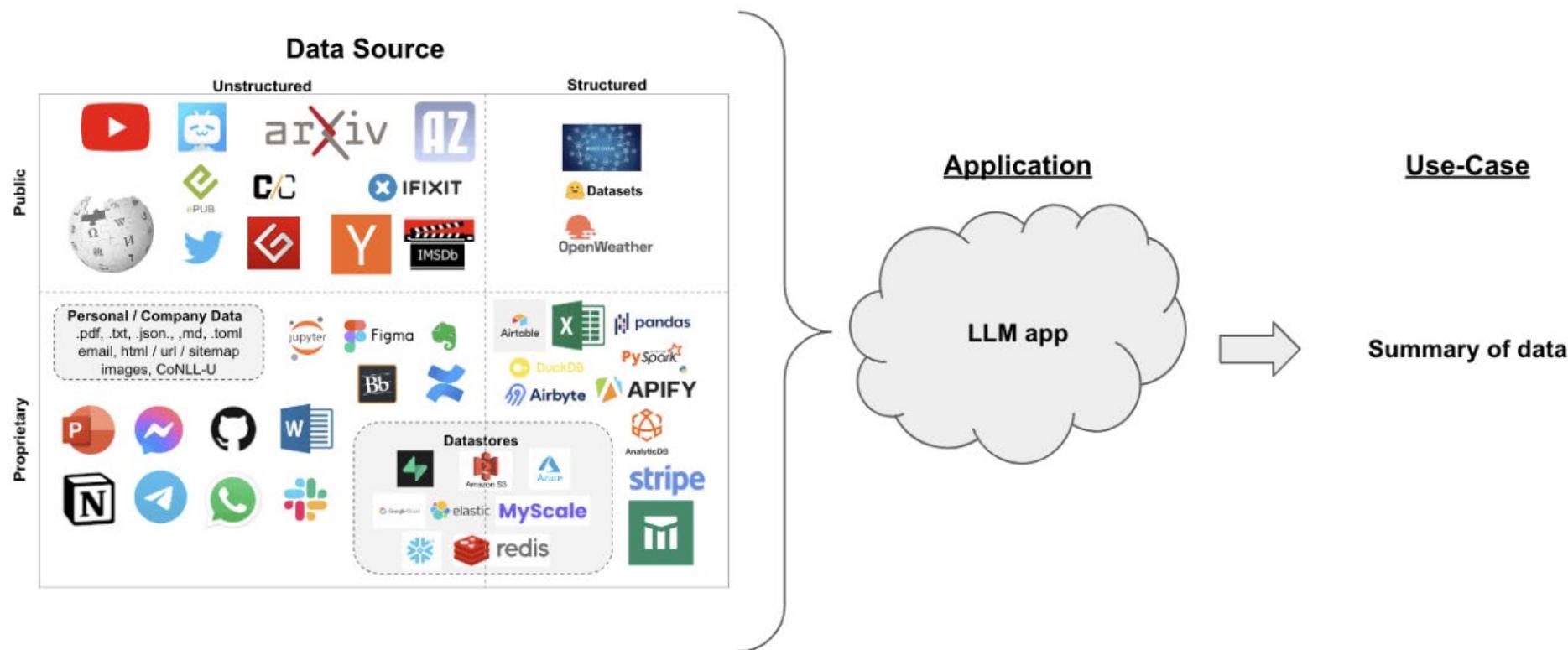
This **dataset** contains information about Rohit Sharma's centuries in international cricket. Each row represents a specific century scored by Rohit Sharma, detailing various aspects of the match and his performance. Dataset contains the following columns :

- **S.No.** : Serial number or index of the century in chronological order.
- **Date** : Date on which Rohit Sharma scored the century.
- **Score** : Runs scored by Rohit Sharma in that innings.
- **Strike Rate** : Strike rate at which Rohit Sharma scored (calculated as runs per 100 balls).
- **Type of Match** : Indicates the format of the match (Test, ODI, T20).
- **Position** : Batting position of Rohit Sharma in that innings.
- **Innings** : Innings number (1st or 2nd) played by Rohit Sharma.
- **Dismissed** : Indicates whether Rohit Sharma was dismissed (Yes) or remained not out (No).
- **Man of the Match** : Whether Rohit Sharma was awarded Man of the Match for his performance in that match (Yes or No).
- **Captain** : Indicates if Rohit Sharma was the captain of the team during that match (Yes or No).
- **Against** : Name of the opposition team against whom Rohit Sharma scored the century.
- **Venue** : Name of the cricket ground where the match was played.
- **H/A/N** : Indicates if the match was played at Home (India), Away (outside India), or Neutral venue.
- **Result** : Outcome of the match for Rohit Sharma's team (Won or Lost).

# Hands On: Election 2019 data

- Use the elections 2019 dataset that we used in the previous session
- Ingest the excel using UnstructuredExcelLoader as in the cricket exercise
- Write the model code and set the prompts appropriately
- Implement the RAG system and test
- Determine if this approach is better than SQL approach in terms of accuracy.

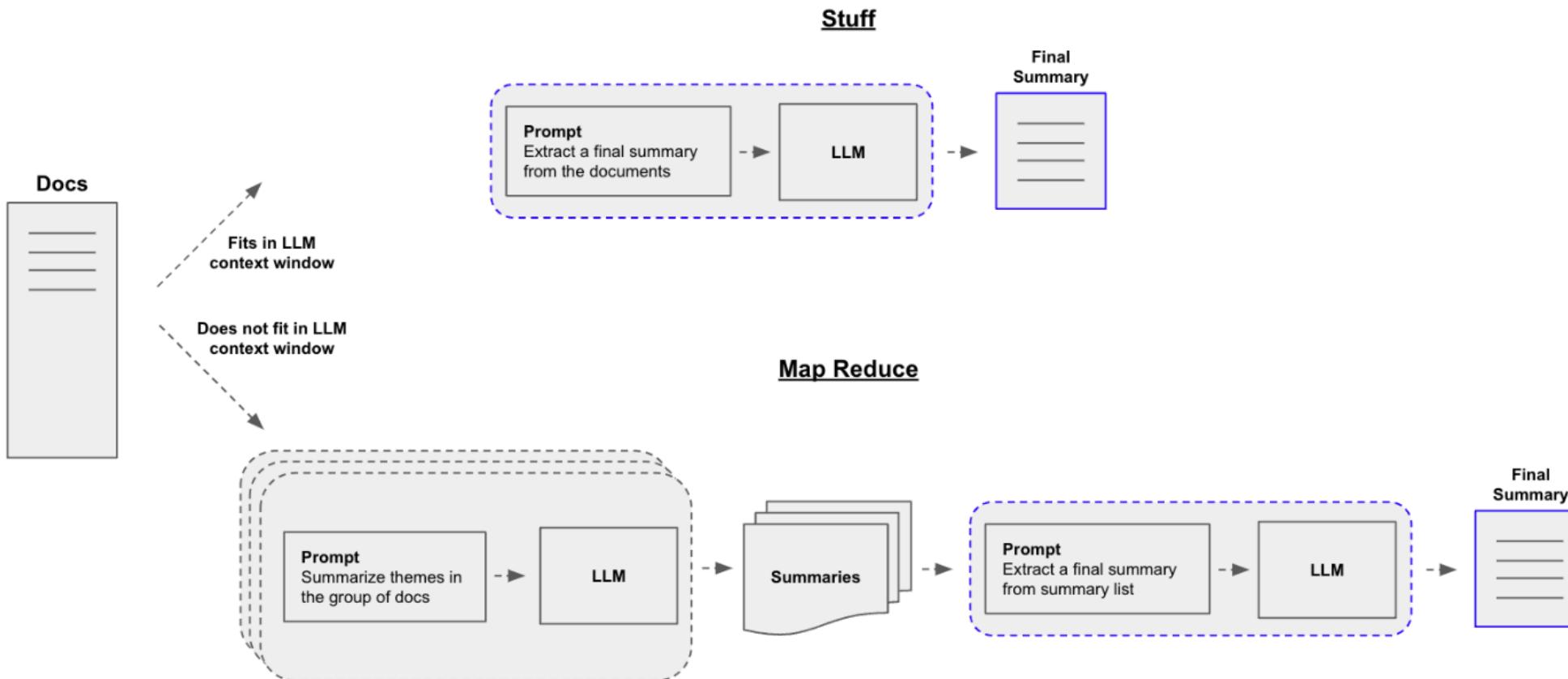
# RAG Summarization



# RAG Summarization

- RAG techniques ingest multiple documents from a variety of sources
- If the relevant retrieved chunks are too many, it may be a good idea to summarize their content before sending them to the LLM
- Three ways to summarize:
  - Stuff the content in the context window of LLM (if contents are less than context size)
  - Map-reduce: create low level summaries, summarize the summaries
  - Refine: updates a rolling summary by iterating over the documents in a sequence

# Summarizing



# Summarizer Demo

# Quiz

- How to choose the chunking technique?
- What happens if the chunk size is too large? Too small?
- How do we know which chunk size is optimal?
- What happens if the embedding is not accurate?

# RAG Limitations

- Retrieval Challenges:
  - Precision and recall issues, leading to the selection of misaligned or irrelevant chunks, and the missing of crucial information.
- Generation Difficulties:
  - The model may hallucinate, where it produces content not supported by the retrieved context. Issues of irrelevance, toxicity, or bias in the outputs, detracting from the quality and reliability of the responses.
- Augmentation Hurdles:
  - Redundancy when similar information is retrieved from multiple sources, leading to repetitive responses.
- Generation models might overly rely on augmented information, leading to outputs that simply echo retrieved content without adding insightful or synthesized information.

# RAG Paradigms

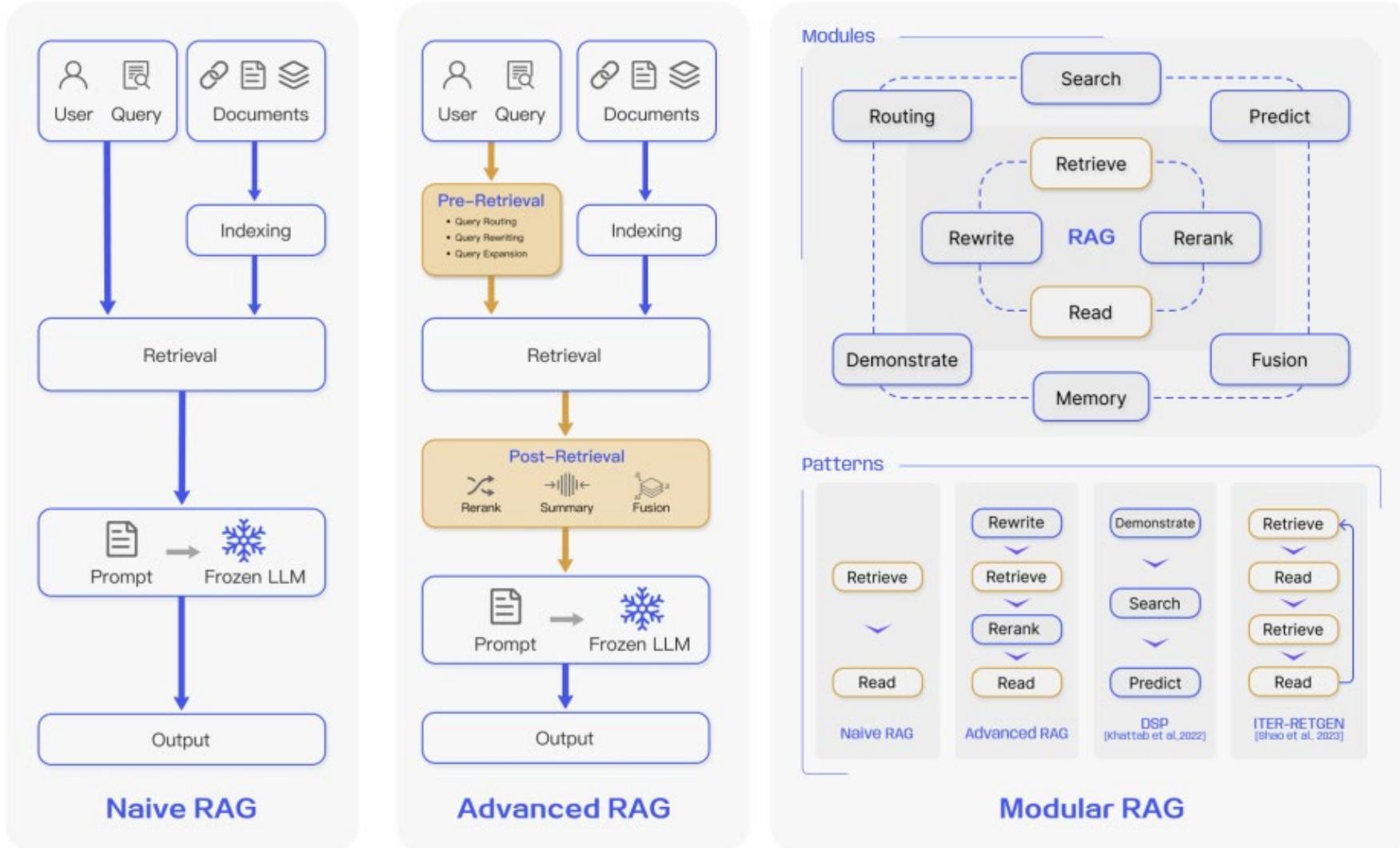


Fig Credits: RAG for LLM Models – A survey, Gao et al.

# Retrieval Augmentation Processes

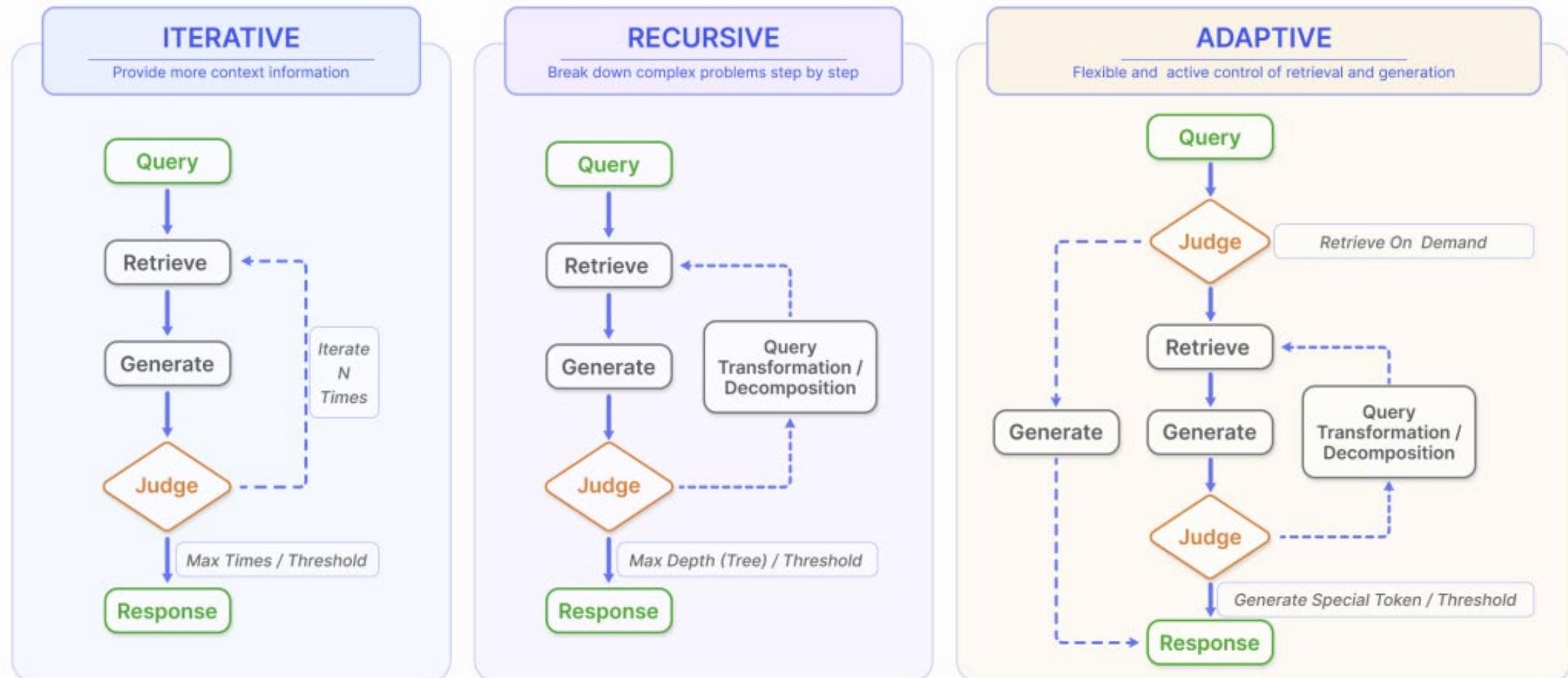
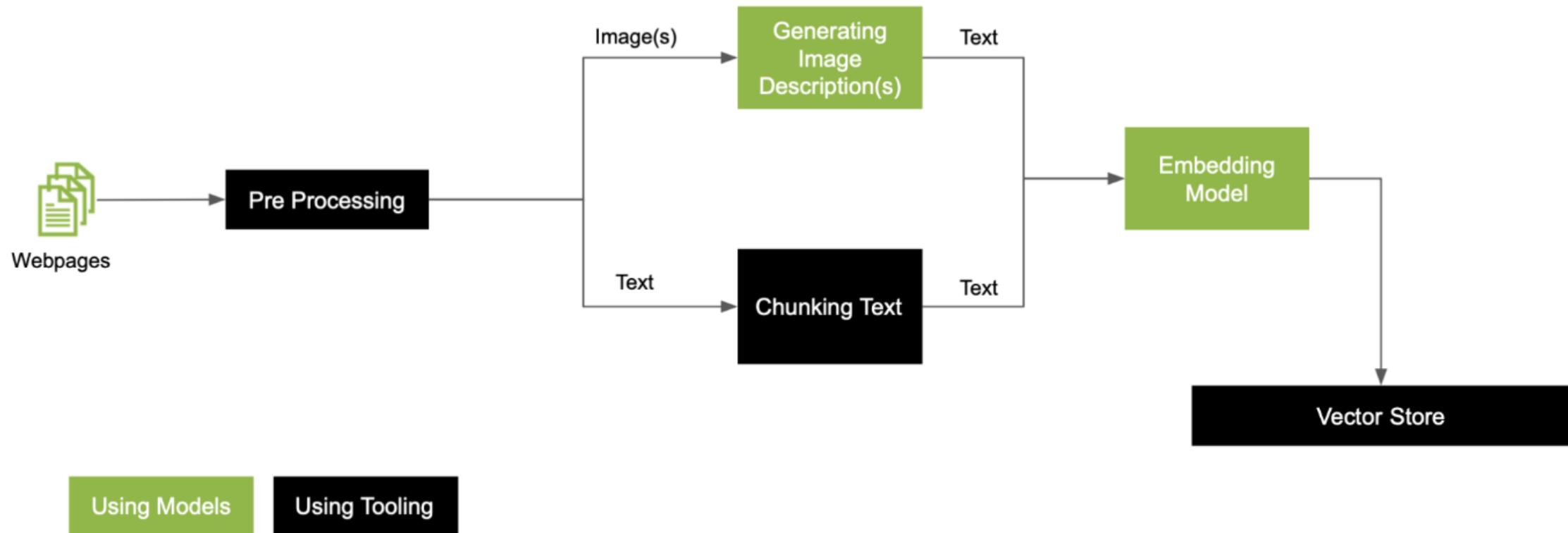


Fig Credits: RAG for LLM Models – A survey, Gao et al.

# Multimodal RAG: Preprocessing



# Part 2: Advanced Techniques

# Approach to enhancing RAG

- RAG is a multi step workflow involving several subsystems like vector store
- To improve RAG performance, each of these steps can be improved

# Enhancement Opportunities

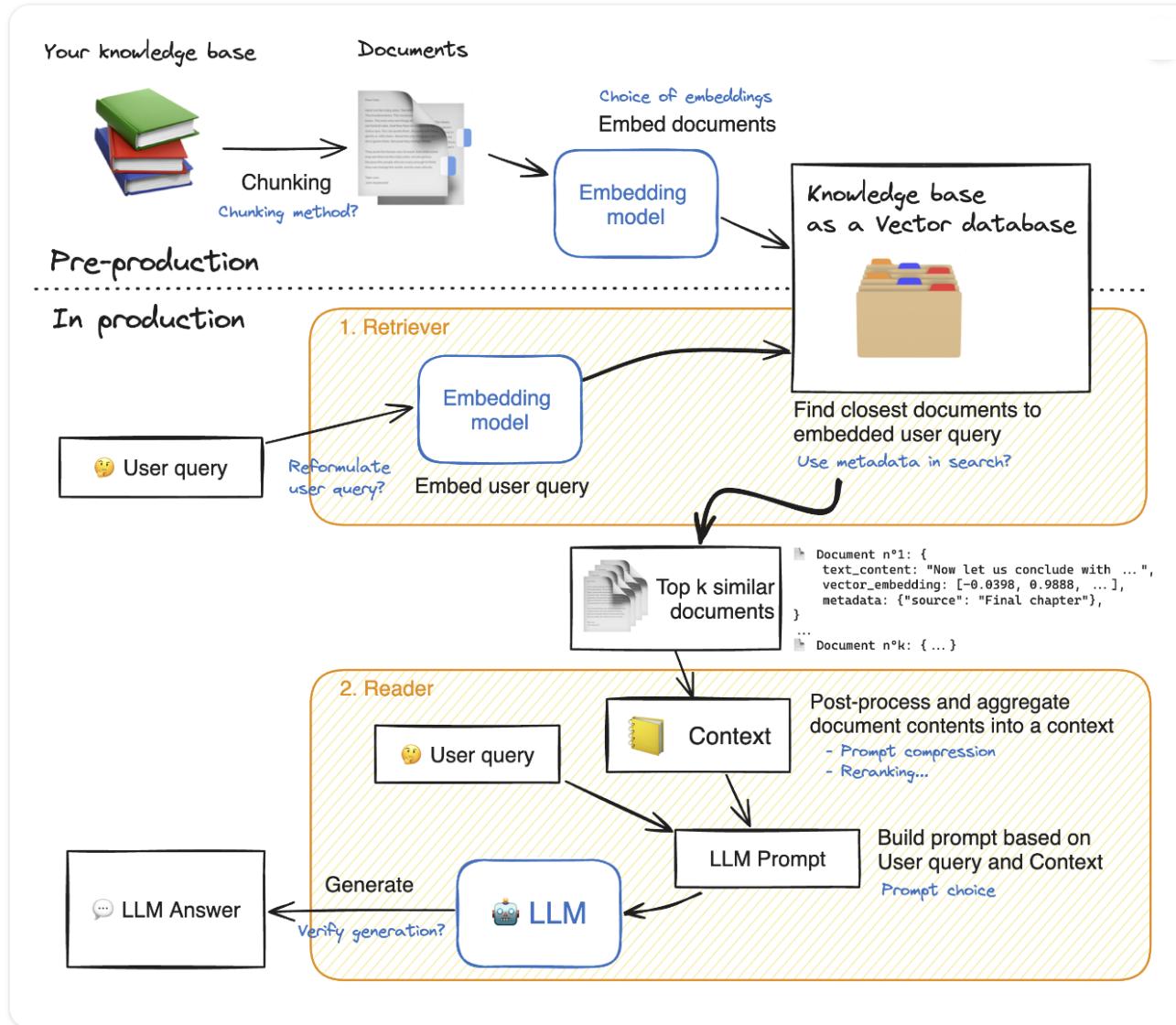


Image from: [Advanced RAG on Hugging Face documentation using LangChain - Hugging Face Open-Source AI Cookbook](#)

# RAG: Seven Failure Points

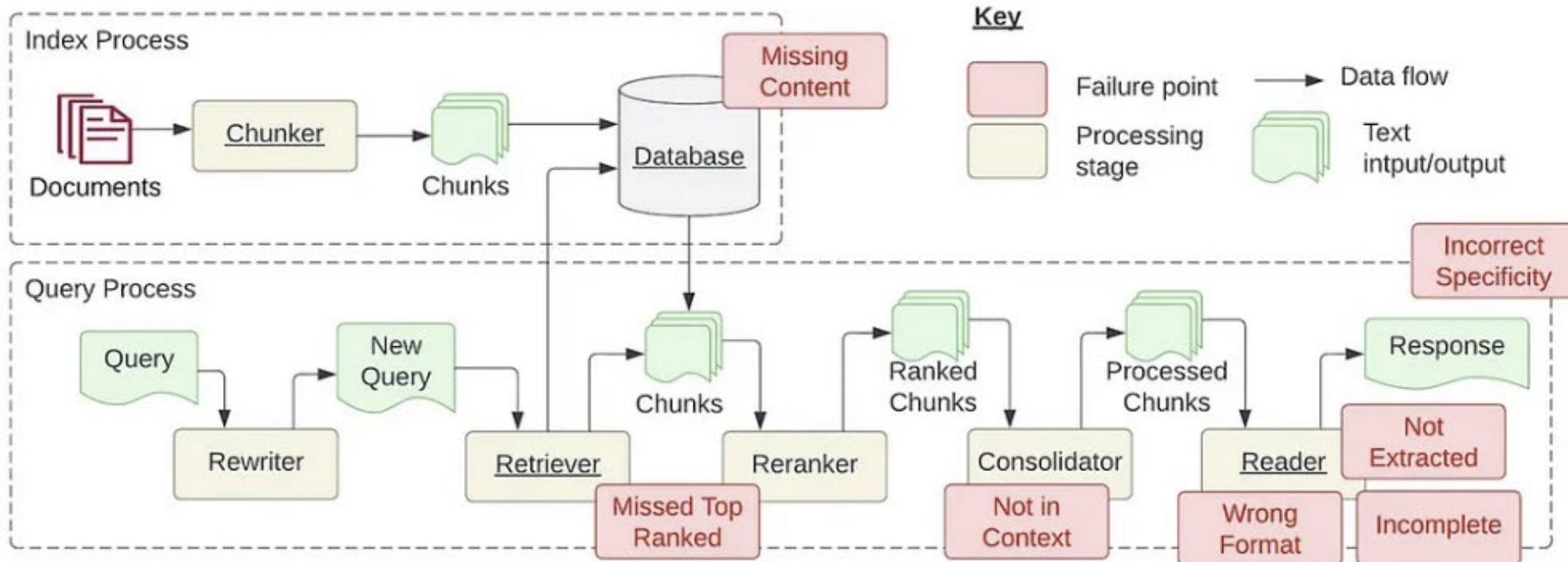


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

Image source: [Seven Failure Points When Engineering a Retrieval Augmented Generation System](#)

# Failure Points

## 5 FAILURE POINTS OF RAG SYSTEMS

From the case studies we identified a set of failure points presented below. The following section addresses the research question *What are the failure points that occur when engineering a RAG system?*

**FP1 Missing Content** The first fail case is when asking a question that cannot be answered from the available documents. In the happy case the RAG system will respond with something like "Sorry, I don't know". However, for questions that are related to the content but don't have answers the system could be fooled into giving a response.

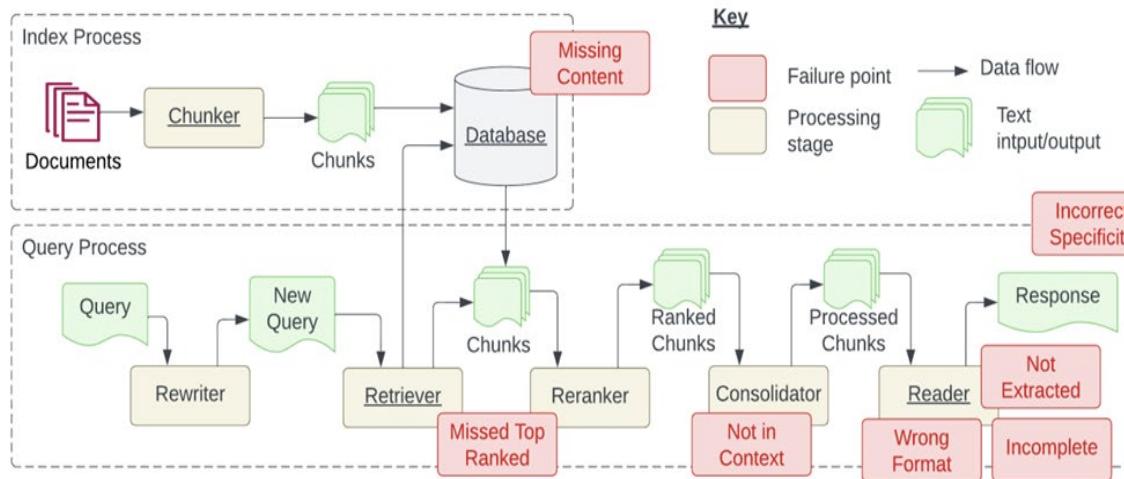
**FP2 Missed the Top Ranked Documents** The answer to the question is in the document but did not rank highly enough to be returned to the user. In theory, all documents are ranked and used in the next steps. However, in practice the top K documents are returned where K is a value selected based on performance.

**FP3 Not in Context - Consolidation strategy Limitations** Documents with the answer were retrieved from the database but did not make it into the context for generating an answer. This occurs when many documents are returned from the database and a consolidation process takes place to retrieve the answer.

**FP4 Not Extracted** Here the answer is present in the context, but the large language model failed to extract out the correct answer. Typically, this occurs when there is too much noise or contradicting information in the context.

**FP5 Wrong Format** The question involved extracting information in a certain format such as a table or list and the large language model ignored the instruction.

**FP6 Incorrect Specificity** The answer is returned in the response but is not specific enough or is too specific to address the user's need. This occurs when the RAG system designers have a desired outcome for a given question such as teachers for students. In this case, specific educational content should be provided with answers not just the answer. Incorrect specificity also occurs when users are not sure how to ask a question and are too general.



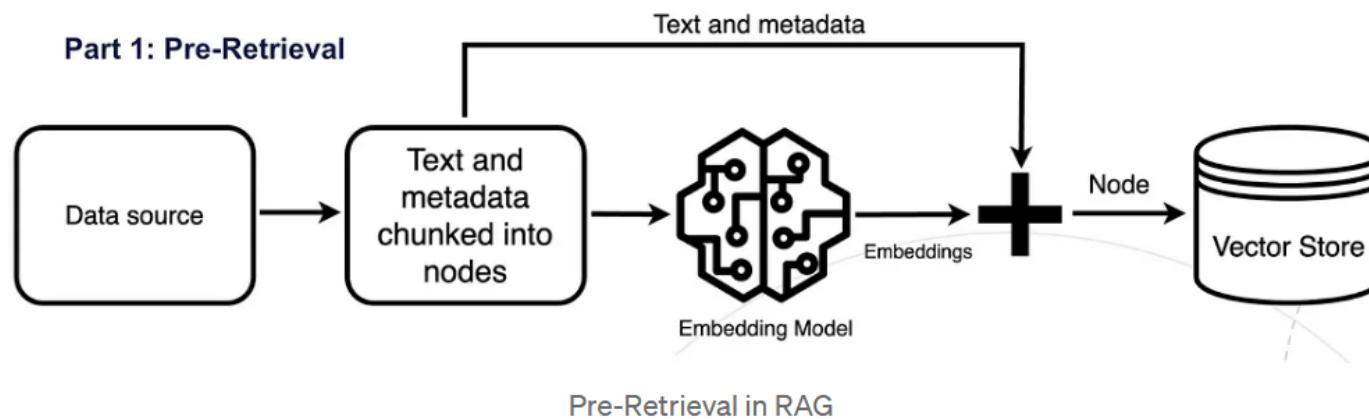
**FP7 Incomplete** Incomplete answers are not incorrect but miss some of the information even though that information was in the context and available for extraction. An example question such as "What are the key points covered in documents A, B and C?" A better approach is to ask these questions separately.

# Advanced RAG: Approach

- The failure points can be associated with 3 broad steps of the RAG workflow:
  - Pre retrieval
  - Retrieval
  - Post Retrieval
- The main approach of advanced RAG techniques is to look at each stage and find ways to improve the quality of each of the components.
- In this section, we discuss some of the key techniques.

# Quick Recap

# RAG Stages: Pre retrieval

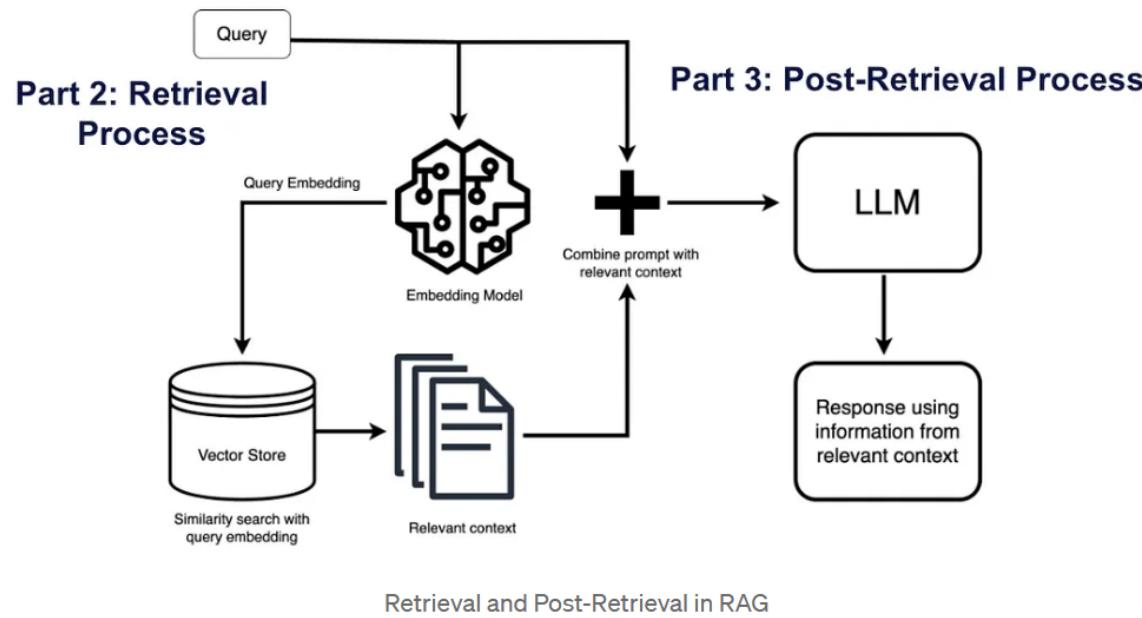


- Prepare external data, split into chunks and index the chunk data using embedding models
- Store the embeddings and chunks in the vector database
- This process creates a knowledge library that the LLM can understand.

# RAG Stages: Retrieval

- In this step, convert the user query to a vector representation called embedding
- Obtain relevant chunks using cosine similarity from the vector database.
- Find highly-relevant document chunks (top-k) from vector store.

# RAG Stages: Post retrieval



- RAG augments the user input by adding the relevant retrieved data in context (query + context).
- Use prompt engineering techniques to communicate effectively with the LLM.
- Given this augmented prompt, the LLM generates an answer to the user query using given context.

# Enhancing Data Quality

- Data cleaning on the candidate documents: Remove unnecessary or irrelevant content from the documents before ingesting
- Replace pronouns with names to improve possibility of providing required context.
- Example Text split as 2 chunks (colored Red and Black outlines):

Mayank Yadav's IPL debut has been remarkable, with the young pacer from Lucknow Super Giants securing consecutive Player of the Match awards. The 21-year-old's exceptional pace has captivated the cricketing world, as he surpassed his own record of 155.8 kph for the fastest ball of the season, unleashing a blistering 156.7 kph delivery against RCB.
- User Query: Who bowled the fastest delivery in the ongoing IPL?

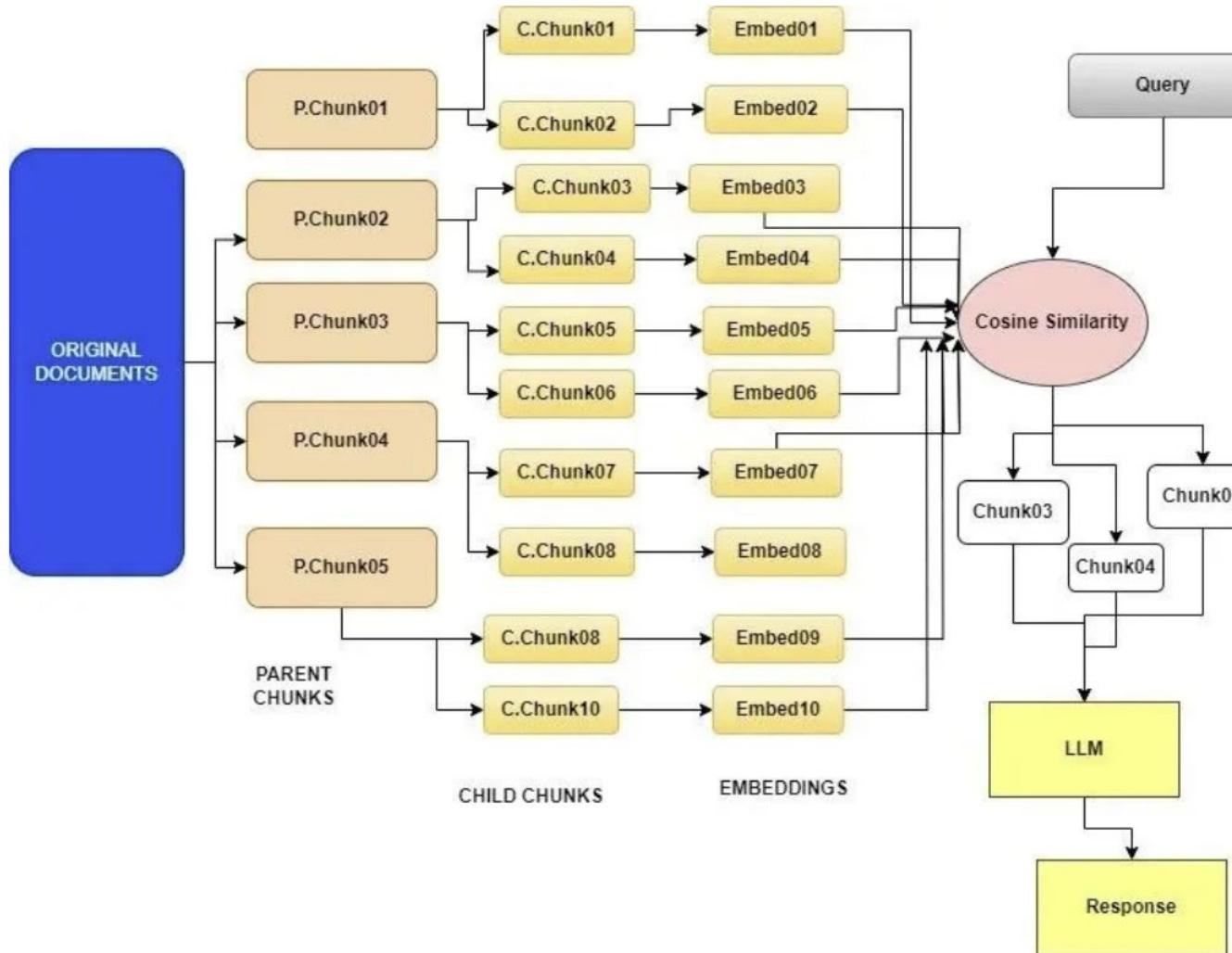
# Metadata to enhance RAG quality

- Metadata, such as concept and level tags, can improve the quality of indexed data.
  - In our example, we can tag all documents pertaining to the ongoing IPL as “IPL 2024”
  - Notion is similar to using hashtags in tweets
- If you search for items and recency is a criterion, you can sort over a date metadata
- If you search over scientific papers and you know in advance that the information you’re looking for is always located in a specific section, say the experiment section for example, you can add the article section as metadata for each chunk and filter on it to match experiments only
- Metadata is useful because it brings an additional layer of structured search on top vector search.

# Chunk Size optimization

- A small chunk gives more accurate matching when cosine similarity is used while it can miss some of the relevant context
- A large chunk can encompass a larger information but can also include irrelevant data for the query posed
- A large chunk may result in a slower response as more tokens need to be processed
- Based on downstream task optimal length of the chunk need to determine and how much overlap you want to have for each chunk.
  - High-level tasks like summarization requires bigger chunk size and low-level tasks like coding requires smaller chunks

# Parent Document Retrieval



# Parent Document Retrieval

- The ParentDocumentRetriever strikes that balance by splitting and storing small chunks of data.
- During retrieval, it first fetches the small chunks but then looks up the parent ids for those chunks and returns those larger documents.
- The term “parent document” refers to the document that a small chunk originated from.
- This can either be the whole raw document OR a larger chunk.

```
loaders = [
    TextLoader("../paul_graham_essay.txt"),
    TextLoader("../state_of_the_union.txt"),
]
docs = []
for loader in loaders:
    docs.extend(loader.load())

# This text splitter is used to create the parent documents
parent_splitter = RecursiveCharacterTextSplitter(chunk_size=2000)
# This text splitter is used to create the child documents
# It should create documents smaller than the parent
child_splitter = RecursiveCharacterTextSplitter(chunk_size=400)
# The vectorstore to use to index the child chunks
vectorstore = Chroma(
    collection_name="split_parents", embedding_function=OpenAIEmbeddings()
)
# The storage layer for the parent documents
store = InMemoryStore()

retriever = ParentDocumentRetriever(
    vectorstore=vectorstore,
    docstore=store,
    child_splitter=child_splitter,
    parent_splitter=parent_splitter,
)

retriever.add_documents(docs)
sub_docs = vectorstore.similarity_search("justice breyer")
```

# Sentence Window Retrieval

[Ref: Advance RAG- Improve RAG performance | Medium](#)

## **Sentence Window Retrieval**

This chunking technique is very similar to above. The core idea behind Sentence Window Retrieval is to selectively fetch context from a custom knowledge base based on the query and then utilize a broader version of this context for more robust text generation.

This process involves embedding a limited set of sentences for retrieval, with the additional context surrounding these sentences, referred to as “window context,” stored separately and linked to them. Once the top similar sentences are identified, this context is reintegrated just before these sentences are sent to the Large Language Model (LLM) for generation, thereby enriching overall contextual comprehension.

# Sentence Window Retrieval

## Sentence-window retrieval

Query: What are the concerns surrounding the AMOC?

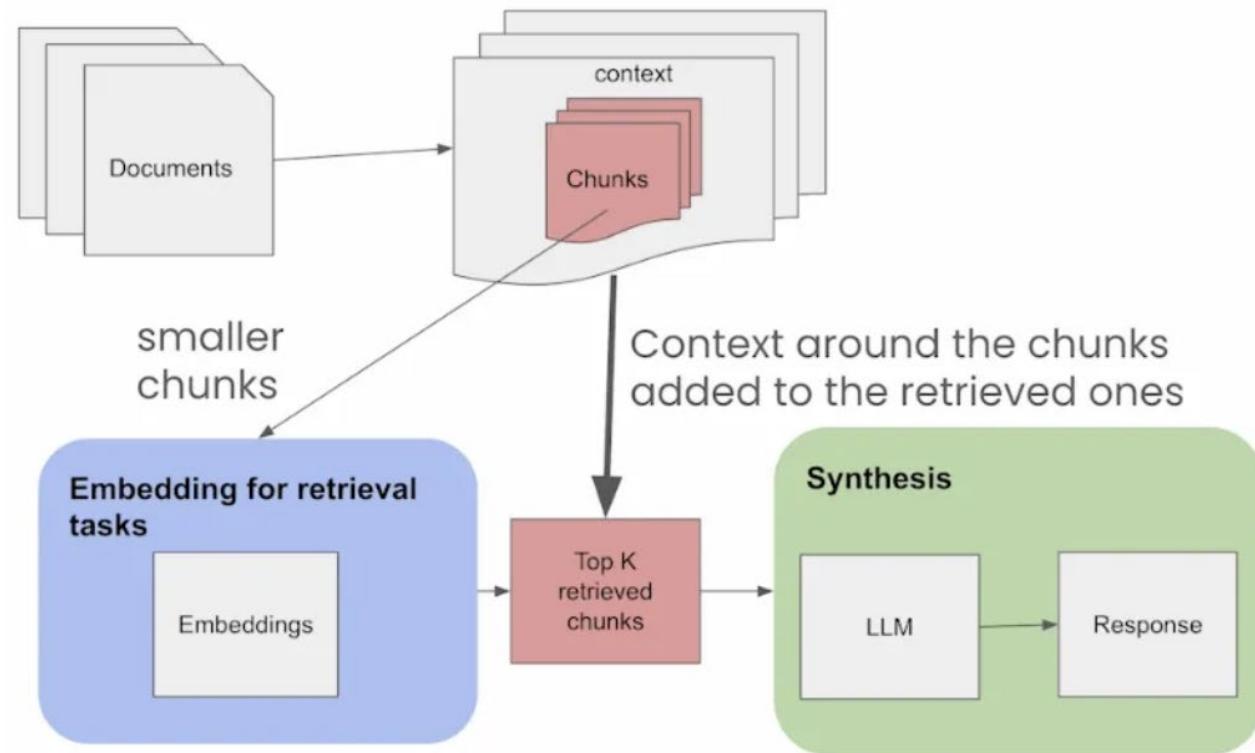
Continuous observation of the Atlantic meridional overturning circulation (AMOC) has improved the understanding of its variability (Frajka-Williams et al., 2019), but there is low confidence in the qualification of AMOC changes in the 20th century because of low agreement in quantitative reconstructed and simulated trends. Direct observational records since the mid-2000s remain too short to determine the relative contributions of internal variability, natural forcing and anthropogenic to AMOC change (high confidence). Over the 21st century, AMOC will very likely decline for all SSP scenarios but will not involve an abrupt collapse before 2100. 3.2.2.4 Sea Ice Changes Sea ice is a key driver of polar marine life, hosting unique ecosystems and affecting diverse marine organisms and food webs through its impact on light penetrations and supplies of nutrients and organic matter (Arrigo, 2014).

What the LLM sees

Embedding Lookup

What the LLM sees

## In Sentence-window retrieval pipeline



# Query Rewriting

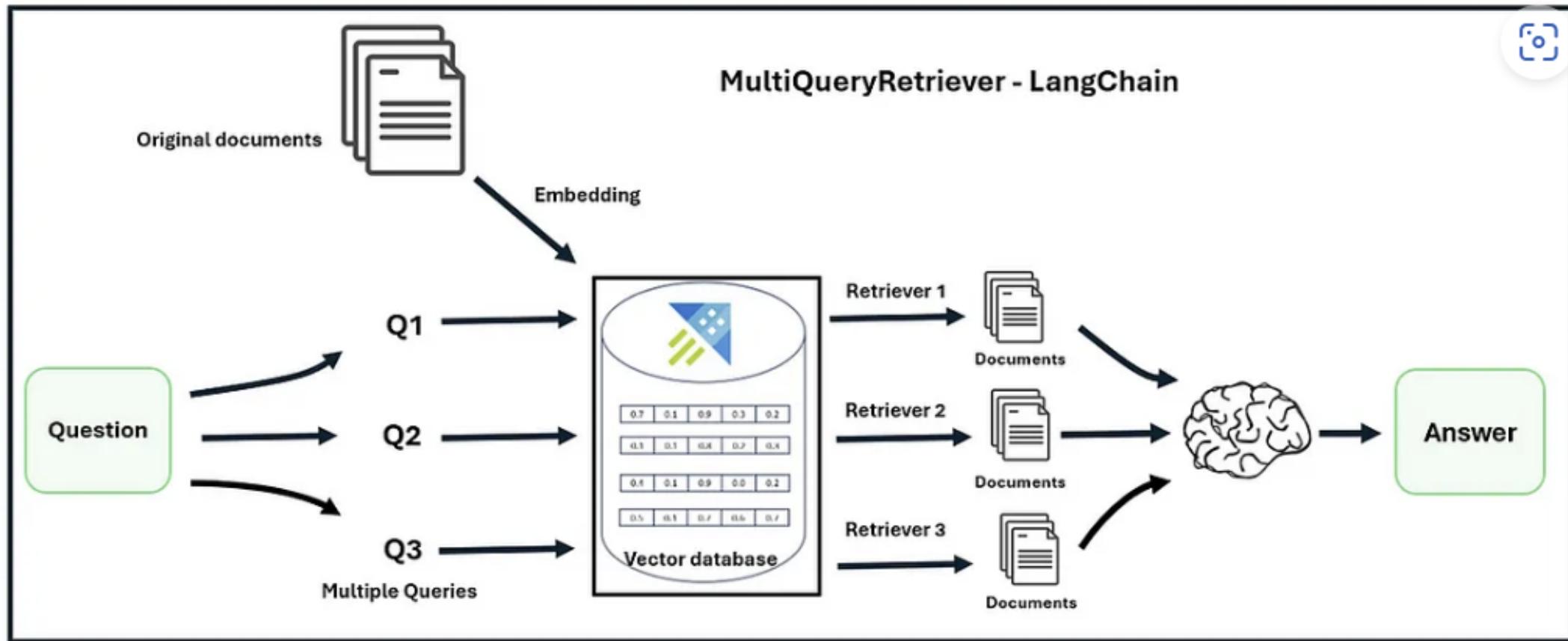
- Query rewriting aligns the semantics of a query and a document.
- An LLM is used to rephrase the user's query and give it another shot. It's important to note that two questions that might look the same to a human may not appear similar in the embedding space.
- The Multi-query Retrieval method utilizes LLMs to generate multiple queries from different perspectives for a given user input query, advantageous for addressing complex problems with multiple sub-problems.
- For each query, it retrieves a set of relevant documents and takes the unique union across all queries to get a larger set of potentially relevant documents.
- By generating multiple perspectives on the same question, the MultiQuery Retriever might be able to overcome some of the limitations of the distance-based retrieval and get a richer set of results.

# MultiQueryRetriever

Distance-based vector database retrieval embeds (represents) queries in high-dimensional space and finds similar embedded documents based on "distance". But, retrieval may produce different results with subtle changes in query wording or if the embeddings do not capture the semantics of the data well. Prompt engineering / tuning is sometimes done to manually address these problems, but can be tedious.

The `MultiQueryRetriever` automates the process of prompt tuning by using an LLM to generate multiple queries from different perspectives for a given user input query. For each query, it retrieves a set of relevant documents and takes the unique union across all queries to get a larger set of potentially relevant documents. By generating multiple perspectives on the same question, the `MultiQueryRetriever` might be able to overcome some of the limitations of the distance-based retrieval and get a richer set of results.

# Multi query Retriever



MultiQuery Retriever

# Example

USER

You are an AI language model assistant. Your task is to generate five different versions of the given user question to retrieve relevant documents from a vector database. By generating multiple perspectives on the user question, your goal is to help the user overcome some of the limitations of the distance-based similarity search.

Provide these alternative questions separated by newlines.

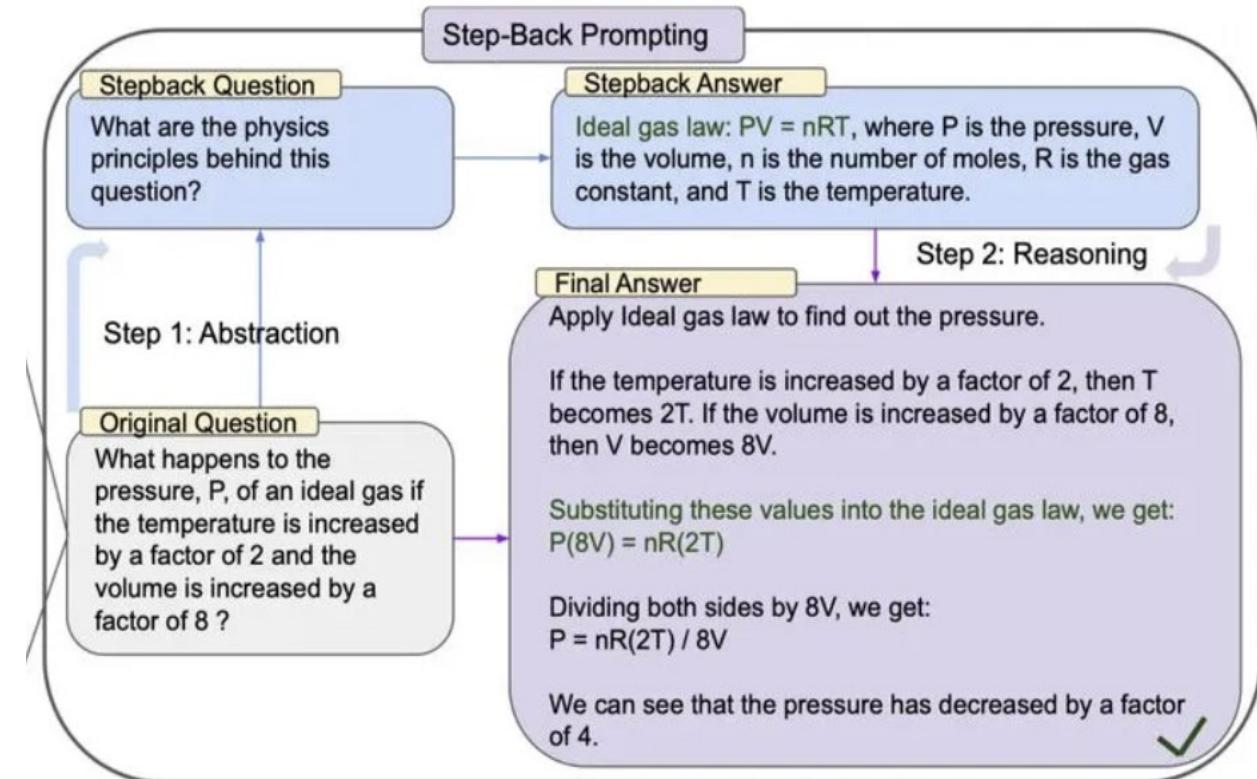
Original question: I am interested to know more about quantum mechanics, such as, is it related to quantum computing or nano computing and also will I get a job if I study it?

AI

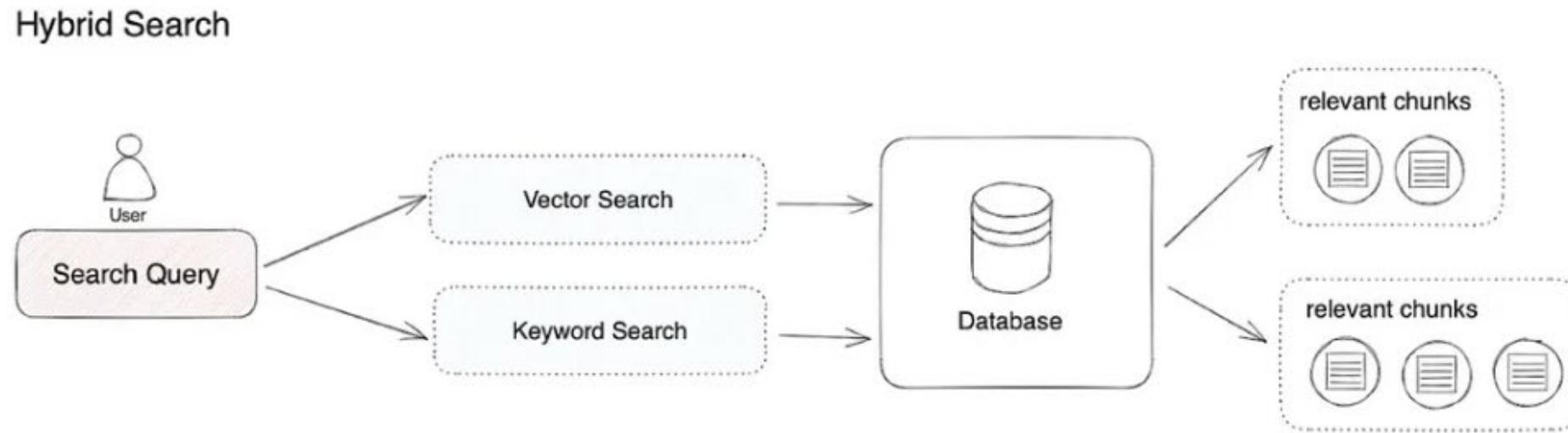
1. Could you please provide documents relating to the connections between quantum mechanics, quantum computing, and nano computing for my further exploration?
2. Can you help me find resources about the field of quantum mechanics, including its applications in quantum technology such as computing and nanotechnology?
3. I'd like to delve deeper into the concept of quantum mechanics – can you show me documents that discuss its relevance to advanced technologies like quantum computing and nano engineering, as well as potential career opportunities?
4. Could you help me locate materials on quantum mechanics and its implications for modern industries, particularly in the contexts of quantum computing and nanotechnology?
5. I'm intrigued by quantum mechanics and its role in modern technology – can you find documents that explain how it relates to both quantum computing and nano computing, as well as potential careers for those studying this field?

# Step Back Prompt

- From the given query, develop broader concepts and principles
- Retrieve Information for both original and step-back questions



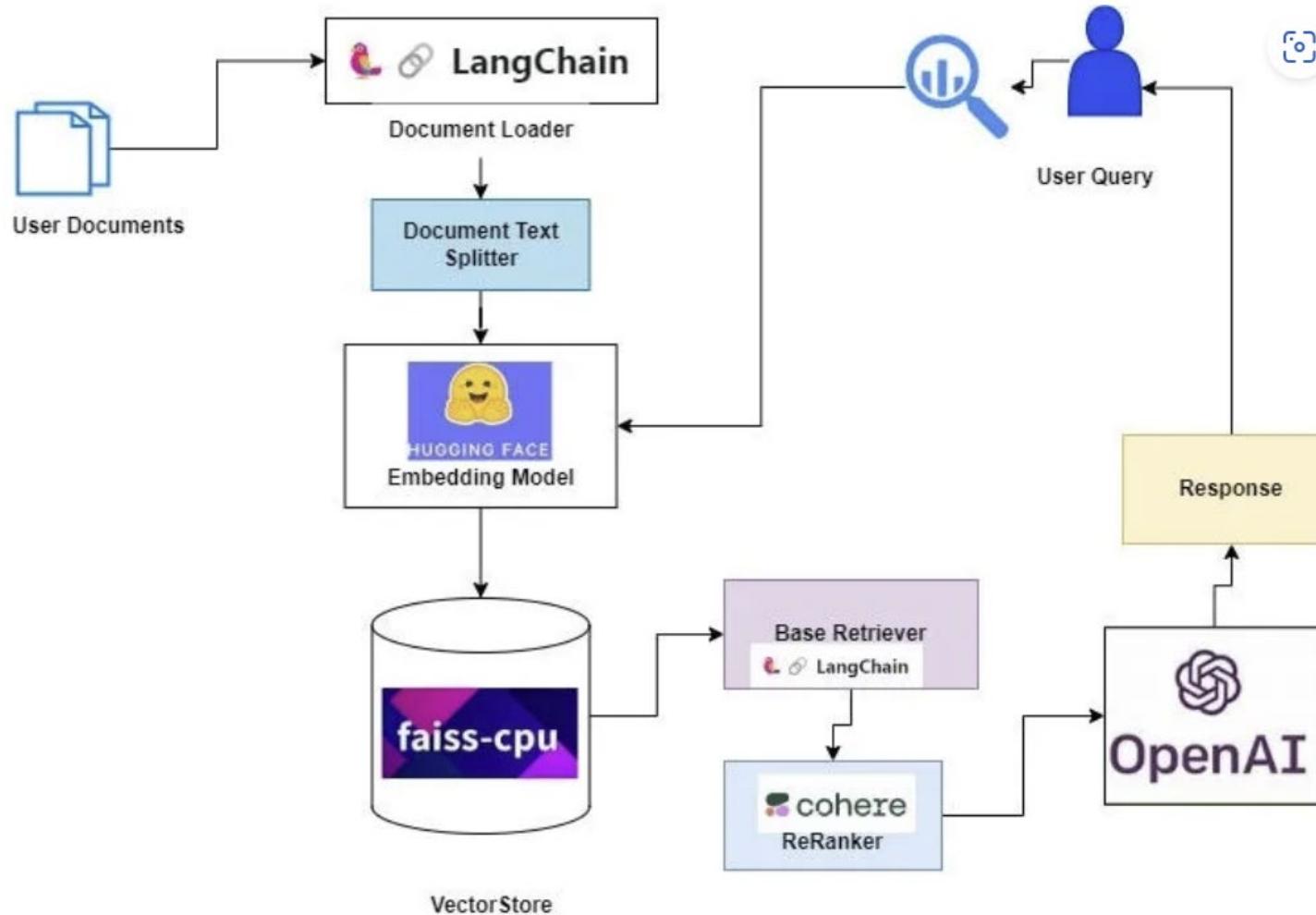
# Hybrid Search Retrievers



# Post Retrieval Optimizations: Re Ranking

- Reranking retrieval results before sending them to the LLM has significantly improved RAG performance.
- A high score in vector similarity search does not mean that it will always have the highest relevance.
- The core concept involves re-arranging document records to prioritize the most relevant items at the top, thereby limiting the total number of documents. This not only resolves the challenge of context window expansion during retrieval but also enhances retrieval efficiency and responsiveness.
- Increase the `similarity_top_k` in the query engine to retrieve more context passages, which can be reduced to `top_n` after reranking.

# Re-Ranking

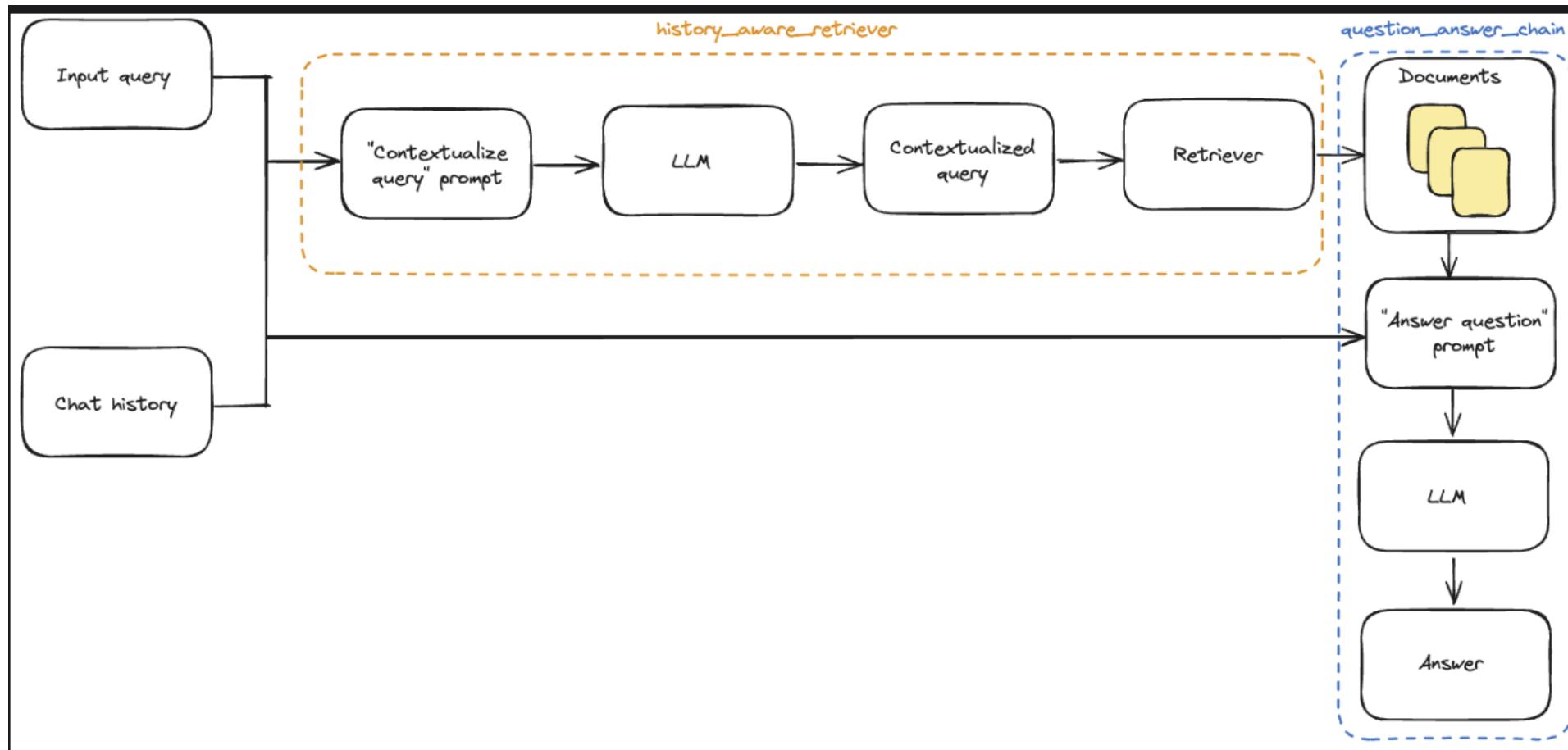


Re-Ranking the retrieved document using Cohere Reranker

# Chat History

- In a chat based RAG application, often it is useful to support chat history
- For example:
  - User: What is the highest score made by Rohit Sharma in T20I matches?
  - Assistant: 121
  - User: Against whom?
  - Assistant: ....
- One way to implement this is by concatenating the current query with past context, but this is inefficient as it takes lot of context.
- It is a better idea to use an LLM to compress the context and create a stand alone query.

# Supporting chat history



# Condensing questions with another LLM

- This chain has two steps. First, it condenses the current question and the chat history into a standalone question.
- This is necessary to create a standalone vector to use for retrieval.
- After that, it does retrieval and then answers the question using retrieval augmented generation with a separate model.
- Part of the power of the declarative nature of LangChain is that you can easily use a separate language model for each call.
- This can be useful to use a cheaper and faster model for the simpler task of condensing the question, and then a more expensive model for answering the question.

# Demo – Chat History Implementation

# Implementation without condensation

```
def chat_history_with_memory():
    memory = ConversationBufferMemory(memory_key="chat_history", return_messages=True)
    qa = ConversationalRetrievalChain.from_llm(llm, get_retriever().as_retriever(), memory=memory)
    query = "What is the total sales of Microsoft in the year 2017?"
    result = qa({"question": query})
    print(result)
    query = "How about the previous year?"
    result = qa({"question": query})
    print(result)
    return

def chat_history_explicit():
    qa = ConversationalRetrievalChain.from_llm(llm, get_retriever().as_retriever())
    chat_history = []
    query = "What does Torrent Pharma produce?"
    result = qa({"question": query, "chat_history": chat_history})
    print(result)

    chat_history = [(query, result["answer"])]
    query = "Does it produce anti diabetic drugs?"
    result = qa({"question": query, "chat_history": chat_history})
    print(result)
```

# Implementation with Condensation

```
def condense_question():
    qa = ConversationalRetrievalChain.from_llm(
        llm,
        get_retriever().as_retriever(),
        condense_question_llm=llm, # can be another LLM specialized on summarization
    )

    chat_history = []
    query = "What does Torrent Pharma produce?"
    result = qa({"question": query, "chat_history": chat_history})
    print(result)

    chat_history = [(query, result["answer"])]
    query = "Given the kind of drugs produced, tell me what is their strategy and how it differs from Micro Labs"
    result = qa({"question": query, "chat_history": chat_history})
    print(result)
```

# Using custom prompt for condensation

By default, ConversationalRetrievalQA uses CONDENSE\_QUESTION\_PROMPT to condense a question. Here is the implementation of this in the docs

```
from langchain.prompts.prompt import PromptTemplate
```

```
_template = """Given the following conversation and a follow up question, rephrase the follow up question to be a  
standalone question, in its original language.
```

Chat History:

```
{chat_history}
```

Follow Up Input: {question}

Standalone question:"""

```
CONDENSE_QUESTION_PROMPT = PromptTemplate.from_template(_template)
```

# Using custom prompt for condensation

But instead of this any custom template can be used to further augment information in the question or instruct the LLM to do something.

```
from langchain.prompts.prompt import PromptTemplate
```

```
custom_template = """Given the following conversation and a follow up question, rephrase the follow up question to be a standalone question. At the end of standalone question add this 'Answer the question in German language.' If you do not know the answer reply with 'I am sorry'.
```

Chat History:

```
{chat_history}
```

Follow Up Input: {question}

Standalone question:"""

```
CUSTOM_QUESTION_PROMPT = PromptTemplate.from_template(custom_template)
```

# RAG on Tables: 3 strategies

- Here, we consider a case where there is a document that has both text and tables
- Strategies are:
  1. Use a large context size (e.g. 128k) LLM, feed the entire document with table
  2. Extract the table as a preprocessing step and use the RAG on table
  3. Chunk the document the usual way, make some assumption like a given table may not cross page boundaries.

# Summary

- RAG system involves:
  - Chunking the text and generating embedding for the chunks
  - Retrieving the chunks by semantic similarity search
  - Generate response based on the text of the top\_k chunks
- Each of the above steps can lead to inaccuracies, implying that the data presented to the LLM is not entirely optimal
- Output generation is constrained by quality of LLM and context/sequence length restrictions
- Are the traditional search engines better than RAG?