**CLASS-3**

**OCI GENERATIVE AI SERVICE**

-------------------------

FULLY MANAGED

>provides a set of customizable LLM available via a single API to build generative AI application

>have the flexibility to use different foundational models with minimal code changes.

>you don't have to manage any infrastructure

CHOICE OF MODELS

>We have high performing, pre-trained foundational models available from meta and cohere

FLEXIBLE FINE TUNING

>create custom models by fine-tuning foundational models with your own data-set

>Doing so you can improve model performance on specific tasks and efficiency

DEDICATED AI CLUSTER

>enables dedicated AI clusters

>GPU based compute resources that can host the fine-tuning and inference workloads

HOW DOES OCI GENERATIVE AI SERVICE WORK

---------------------------------------

TEXT INPUT ---> OCI GENERATIVE --> TEXT OUTPUT

AI SERVICE

>provide text input in natural language to the model

>Gen AI service analyzes the user input and generate, summarize, transform, extract information or classify text based on the request from the user

>Then a response is sent back to the user

>The service is build to understand, generate and process human language at a massive scale

use cases

---------

>ability to generate text

>ability to summarize text

>extract data from text

>classify text

>enable conversational AI with virtual chatbot assistant

-----------------------------------------------

PRETRAINED FOUNDATIONAL MODELS

------------------------------

FIRST CATEGORY - GENERATION MODELS

we have three models

1.Command model from cohere

-------------------------

>highly performant instruction following conversational model.

>has 52 billion parameters.

2.command light model from cohere

-------------------------------

>smaller, faster version of command but almost as capable.

>has 6 billion parameters

3.LLama-2 from META

--------------------

>70 billion chat model

>idea behind these models is to generate text

>these models are instruction following models or instruction tuned models

>we take the base models and run them through additional training called instruction tuning.

>These allows the model to follow human language instructions, such as generate a product pitch for a particular product.

----------------------------------------------

SECOND CATEGORY - SUMMARIZATION MODEL

command model available from cohere

-----------------------------------

>idea with summarization model is that you can summarize text with your instructed format, length, and tone.

-----------------------------------------------

THIRD CATEGORY - EMBEDDING MODELS

>pre-trained foundational models are called embedding models.

>we have embedding models from cohere.

>You see embed English model,embed multilingual models.

>idea behind these models are to create embeddings.

>embeddings are nothing, but you take text and you convert them into vector of numbers.

>Embeddings are mostly used for semantic searches, where the search function focuses on the meaning of the text that it is searching through rather than finding results based on keyword.

multilingual models from cohere.

>These multilingual models support 100 plus languages and can be used to search within a language.

>For example, search with a French query on French documents and across languages. For example, search with a French query on English documents

######################

FINE-TUNING

##############

>Key capability for the Generative AI service is the ability to fine-tune these models

>optimize a pre-trained foundational model on a smaller domain specific dataset

>so you have your own custom data, own domain-specific data

>and you have the pre-trained foundational model

>you train that model with this custom data

>end up with a custom model through this fine-tuning process

TWO BENEFITS TO DOING FINE-TUNING

1. you improve model performance on specific tasks by tailoring the model to domain-specific data, better understand and generate

2. you can improve the model efficiency by reducing the number of tokens

-------------------------

you use fine-tuning when a pre-trained model doesnt perform your task well or you want to teach it something new

>OCI Generative AI service uses T-Few fine tuning mechanism to enable fast and efficient customizations.

>using T-few we update only a portion of models weight

>doing so will give you a better accuracy at lower cost as a result

#######################

DEDICATED AI CLUSTER

######################

>AI clusters are GPU based computer resources that host the customers fine-tuning and inference workloads

>includes dedicated GPUs and an exclusive RDMA cluster network for connecting GPUs

>OCI has an industry level leading RDMA supercluster

**CLASS - 4**

**GENERATION MODELS**

--------------------

TOKENS

------

>one token can be part of a word

>could be entire word or a punctuation symbol

> "apple" is a token

> "friendship" is made of two tokens "friend" "ship"

>number of tokens per word depends on the complexity of text

>can assume one token per word on average

>text with less common words - can assume two or three tokens on average

eg THERE ARE ONLY TWO I DON'T KNOW ABOUT THE INDULGENCE

1 1 1 1 1 2 1 1 1 2

>large language models take text as input

>tokenize that input

>understand token rather than characters

####################################

PRETRAINED GENERATION MODELS IN GENERATIVE AI

####################################

1.COMMAND MODEL FROM COHERE

>highly performing instruction following conversational model

>52B parameters

>used for text generations, text summarizations or chat use cases

>has context window of 4096 tokens

"context window means number of tokens its capable of processing at one time"

sum of input tokens and output tokens for particular model

2.COMMAND LIGHT FROM COHERE

>smaller, faster version of command model

>6B parameters far fewer

>the context window is the same

>used when speed and cost are important

>more specific your prompt, the better this model performs

3.LLAMA-2 70B CHAT MODEL

>comes in three sized 13B or 70B, 13B and 70B

>OCI supports 70B parameter model

>highly performance model

>open source model

>This model is optimized for dialogue use cases "to be noted"

>used for chat and text generation as well

------------------------------------------------------

PARAMETERS IN GENERATION MODELS

------------------------------------------------------

MAXIMUM OUTPUT TOKEN

>max number of tokens model generates per response

TEMPARATURE

>determines how creative the model should be

>close second to prompt engineering in controlling the output of generation models

>parameter that controls the randomness of the LLM output

>Temperature of 0 makes the model deterministic (limits the model to use the highest probability)

>when temperature is increased, the distribution is flattened over all words

>with increase in temperature model used words with lower probabilities.

The sky is \_\_\_\_\_\_\_\_

blue the limit red high

0.4 0.25 0.02 0.12

dec inc

TOP p, TOP k

>two additional ways to pick the output tokens besides temparture

Top k tells the model to pick the next token from the top 'k' tokens in its list, sorted by probability

The name of the country is the \_\_\_\_\_\_\_\_\_

united netherland Czech Kingdom

0.12 0.027 0.016 0.01

if Top k is set to 3, model will only pick from the top 3 options and ignore all other

then mostly pick "united" but will pick

"netherlands" and "Czech" some times

Top p is similar to Top k but picks from the top tokens based on the sum of their probabilities

if p is set as .15 then it will only pick from united and netherland as their probabilities add up to 14.7%

if p is set to 0.75, the bottom 25% of probable outputs are excluded

STOP SEQUENCE

>stop sequence is a string that tells the model to stop generating more content

>it is a way to control your model output

>if a period (.) is used as a stop sequence, the model stops generating text once it reaches the end of the first sentence, even if the number of tokens limit is much higher

PRESENCE/FREQUENCY PENALTY

>assign a penalty when a token appears frequently and produces less repetitive text

>useful if you want to get rid of repetition in your outputs

>frequency penalty penalizes tokens that have already appeared in the preceding text, and scales based on how many times that token has appeared

>token that has already appeared 10 times gets a higher penalty than a token that has appeared only once

>presence penalty applies the penalty regardless of frequency. as long as the token has appeared once before, it will get penalized

SHOW LIKELIHOODS

>determines how likely it would be for a token to follow the current generated token

>Every time a new token is to be generated, a number between -15 and 0 is assigned to all tokens

>tokens with higher numbers are more likely to follow the current token

this is my favorite\_\_\_\_\_\_\_\_\_

Book Food Zebra

-4.5 -5.0 -14

>Book, Food are high likelihood

>zebra is low likelihood

################################

Demo on

Text Generation

>generated text sentence

Text Extraction

>given a paragraph and asked for extraction like company, topic and summary

Text Classification

>trained with some sentences with classification and asked to classify the last five sentences

#################################

Demo on OCI Generative AI service API interface

#################################

Demo on setting up OCI config for Generative AI API

##################################

SUMMARIZATION MODEL

-------------------

>generates a succinct version of the original text that relays the most important information

>same as one of the pretrained text generation model but with parameters that you can specify for text summarization

>use cases include, but not limited to:

news articles, blogs, chat trnascripts,

scientific articles, meeting notes, and any text that you should like to see a summary of

SUMMARIZATION MODEL PARAMETERS

-------------------------------

TEMPERATURE

>determines how creative the model should be

default temp = 1 and max temp = 5

LENGTH

>approx length of summary. choose from short, medium and long

FORMAT

>whether to display the summary in a free-form paragraph or in bullet points

EXTRACTIVENESS

>how much to reuse the input in the summary

>summaries with high extractiveness lean toward reusing sentences verbatim(word to word)

>summaries with low extractiveness tend to paraphrase

##################################

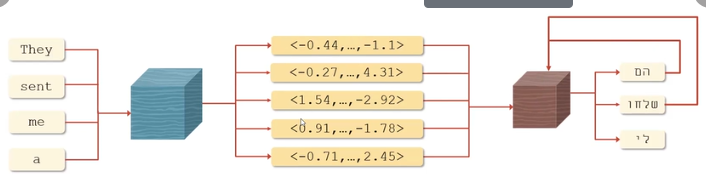
EMBEDDING MODELS

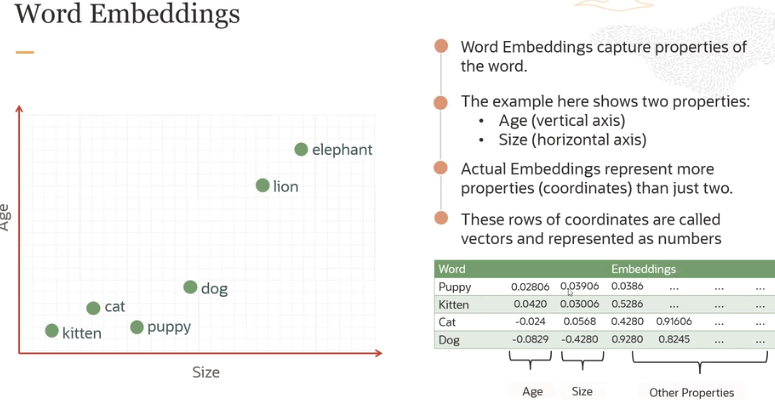
-----------------

>embeddings are numerical representations of a piece of text converted to number sequence

>piece of text could be a word, phrase, sentence, paragraph or one or more paragraph

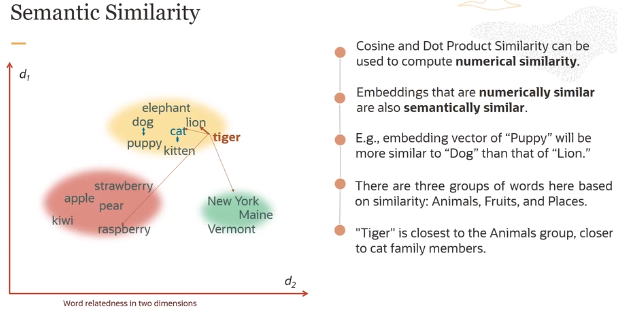
>embeddings make it easy for computers to understand the relationships b/w pieces of text





Semantic similarity

-------------------



Sentence Embeddings

-------------------

>A sentence embedding associates every sentence with a vector of numbers

>Similar sentences are assigned to similar vectors, different sentences are assigned to different vectors

>The embedding vector of “canine companions say” will be more similar to the embedding vector of “woof” than that of “weow”

>Imp concept to remember or understand

comparing two phrases

> collection of words, to a single word

> words against sentence against paragraph or group of paragraph

Embeddings use case

-------------------

>saw earlier use case of sequence to sequence task like translation where embeddings are used behind the scenes

>There is another very imp use case around retrieval-augemented generation

>One of the main challenges faced by todays generative or embedding models is their inablity to connect with your company data

>A promising approach to overcoming this limitation is retrieval-augmented Generation, RAG.

How it works

>you can take large corpus of documents break it into chunks or paragraphs

>and generate the embedding for each paragraph

>store all the embeddings into a vector database

>now vector database are capable of automating the cosine similarity and

doing nearest-match searches through that database for some search embedding you want to search for

>this basically powers the whole RAG system

>The way RAG works

>lets say you have a user question which cannot be answered by LLM

>the question is encoded as a vector and sent to the vector database

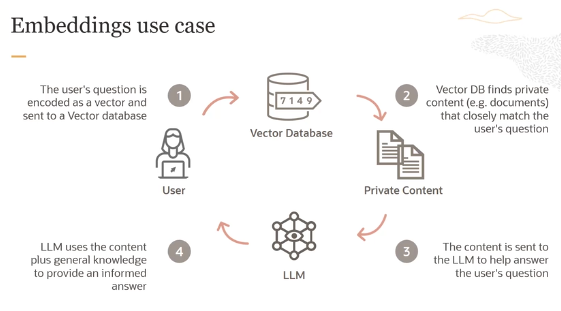
>now vector batabase can run a nearest match to identify the most closely associated document or paragraph

>it finds this is the private content which closely matches the user query

>it can take those documents of paragraphs and insert those into a prompt to be sent to the Llm

>Basic idea is to help answer the user question by changing the prompt

>then the LLM uses the content which has been given by the vector database plus its general knowledge to provide an informed answer



EMBEDDING MODELS IN GENERATIVE AI

---------------------------------

>like you already saw

>take a text > convert this text into numbers > text as vectors and these are embeddings

>The embedding models supported by OCI AI service

> Cohere.embed-english converts english text into vector embeddings

> Cohere.embed-english-light is the smaller and faster version of embed- english

> Cohere.embed-multilingual is the state of the art multilingual embedding

model that can convert text in over 100 languages into vector embedding

you can search with a french query on english document

>Use cases: Semantic search, Text classification, Text clustering

Dive Deeper

-----------

>English model and multiligual model they support English and multilingual languages

>creates 1024 dimensional vector for each embedding

>every embedding whether a sentence or paragraph gets converted into

1024 dimensional vector

>The model takes a max of 512 tokens per embeddings

>v3 embed model is the recently launched by Cohere

>v3 model has the ability to query match a document topic and asses the

overall quality of the content especially helpful when dealing with noisy

data sets

>And it can drastically, significantly improve retrievals for RAG systems

>We also have the light version of these v3 models – smaller, faster version

>creates 384 dimentional vector

>model takes a max of 512 tokens per embedding

>We also have the previous generation model available v2

>1024 dimensional vector

>512 tokens

>Now you can input sentence, phrases or paragraphs for embeddings either one phrase at a time in these models or by uploading a file

>Keep in mind that a max of 96 inputs are allowed for each run

>And each input must be less that 512 tokens

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DEMO – SUMMARIZATION AND EMBEDDING MODELS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**PROMPT ENGINEERING**

------------------

**Prompt** – The input or intial text provided to the model

**prompt Engineering** – The process of iteratively refining a prompt for the purpose of eliciting a particular sytle or a particular type of response from the LLM

>We have seen LLMs basically our next word predictors

>They basically predict next set of words or next set of token

>Text prompts are basically how users interact with LLMs

>LLMs do is, they attempt to produce the next series of words that ate most likely to follow from the previous text

>The way completion LLMs are trained is to predict the next word on a large dataset of internet text rather than to safely perform the language tasks that the user wants

>in this case you cannot give instructions or ask questions to a completion LLM

>Instead for every prompt you need to formulate your input as a prompt whose natural continuation or completion is your desired output.

>And this is not how LLM work

>They have fine-tuned their models putting several research papers

eg:- An article on Llama2 came out in 2023 about how reinforcement

learning from human feedback is used to fine-tune LLMs to follow a broad

class of written instructins.

>Llama2 foundational models were trained on a data set with 2 trillion tokens

>Now Llama2 chat model is different than the base model

>That was additionally fine-tuned like 28000 prompt response pairs created for this particular project

>And to align Llama2 to follow instructions

>RLHF- Reinforcement Learning with Human Feedback was used with a combination of morethan 1.4 million meta examples and 7 smaller data sets

>RLHF is a model training procedure that is applied to a fine-tuned language model to further align model behavior with human preferences and instruction following

>In this case human annotators write promts and they compare the model outputs

>RLHF is also used to train reward models which learns patterns in the preferences of the human annotators and can then automate preference decisions

This is how RLHF works

>In todays context most of the LLMs can follow instructions because those models are fine-tuned and RLHF is used to fine-tune those LLMs to follow a broad class of written instructions

IN-CONTEXT LEARNING AND FEW-SHOT PROMPTING

------------------------------------------

>Prompt Engineering is Challenging

>There are many successful strategies for generating prompts that are useful and successful for specific tasks ans particular models

1. in-context learning

> prompting an LLM with instructions and or demonstrations of the task it is meant to complete

2. K-shot prompting

> explicilty providing k examples of the intended task in the prompt

k = 0,1,2 of the intended task in the prompt

-----------------------------------------------------

Translate English to Fresh ---> task description

sea otter => loutre de mer --> example

peppermint => menthe poivree --> example

plush girafe => girafe peluche --> example

cheese => --> prompt

----------------------------------------------------

>asking the model to take the next word “cheese” and translate it into french

>the whole statement is a prompt

>This is a three-shot prompt because you are giving three examples

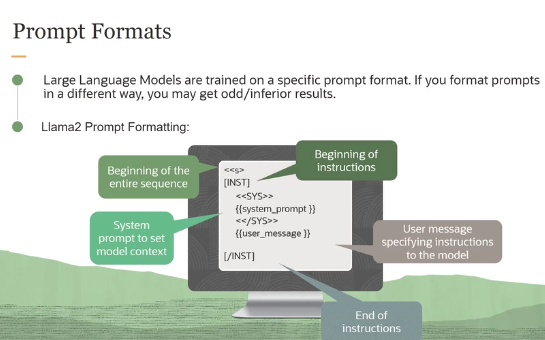
>few-shot prompting is widely believed to improve results over zero-shot prompting

>You have to care about prompt format LLM like Llama2

>Llama2 are trained on a specific prompt format

>If you format your prompt in a different way --> you may have diffrent results

inferior results, suboptimal results



ADVANCED PROMPTING STRATEGIES

------------------------------

Chain-of-Thought

----------------

>provide examples in a prompt is to show responses that include a reasoning step

>the idea is if you have a complicated task

>we are going to prompt the model to break the problem into small chunks

Zero shot Chain-of Thought

--------------------------

>Apply chain-of Thought prompting without providing examples

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CUSTOMIZE LLMs WITH YOUR DATA

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Training LLMs from Scratch with my data

> COST - $1M per 10B parameters to train

> DATA – A lot of data is needed

eg: Meta’s Llama-2 7B model was trained on 2 trillion tokens

(1T tokens ~ 20M novels ~ 1B legal briefs)

And you need a lot of annotated data

> Expertise – Pretraining models is hard: require a thorough understanding of

model performance. How to monitor for it, detect and mitigate hardware

failures, understand the limitations of the model

There are three options you could customize your LLMs

1.in-context learining

>The idea is user provides demonstrations in the prompt to teach the model how to perform certain taks

Chain-of-Thought prompting

>where you are asking the model to break a problem into smaller chunks and solve each of these intermediate steps

>Main limitation here is the model context window and the length

> many models have length around 4096 tokens or even smaller

>That is all the number of tokens a model can process at any given time

> This is the main limitation why you would not use few-shot prompting

2.fine-tuning

>The main idea is you are optimizing a model on a smaller domain-specific dataset

>Recommended when a pre-trained model doesnt perform your task well or when you want to teach it something new

>Using fine-tuning your model can adapt to specific style and tone and learn human preferences

Advantages of fine-tuning

>Improving the model performance on specific tasks

>more efficient of improving model performance than prompt engineering

>customizing the model to domain-specific data, it can better understand

and generate contextually relevant response

>Improve model efficiency

>Reduce number of tokens needed for you model to perform well on tasks

>condense the expertise of a large model into a smaller, more efficient

model

3.RAG (Retrieval Augmented Generation)

>Hooking the language model to some kind of enterprise knowledge base

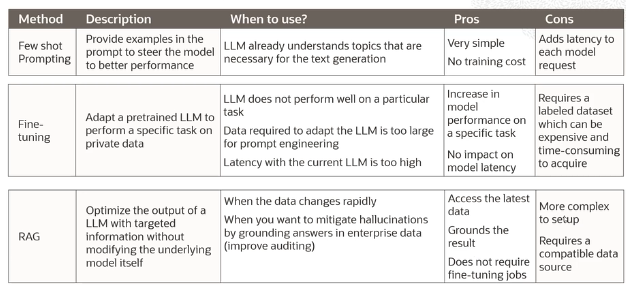
it could be database, wiki, vector database to provide grounded responses

>grounded basically means that the generated text is grounded in a document if the document support the text

>RAG can give the model access to private knowledge

>Imp thing is RAGs do not require any kind of fine-tuning or custom models

Comparing all three techniques



>The thing is which technique do I have to use

RAG vs fine-tuning vs prompt-engineering

>each of these are solving a different set of problems

>the framework you should think about is on these two dimensions

