OCI GENERATIVE AI PROFESSIONAL COURSE NOTES

Objectives of the course:

- **Fundamentals of large language models**
- **Dive deep into OCI Generative AI service**
- **Building an LLM app using OCI generative AI service**

Contents

Fundamentals of LLMs

Language Models and Large Language Models

A language model is a probabilistic model of text / natural language. Suppose there is a sentence containing a blank which needs suggestions to be filled, the language model assigns probability to every word in vocabulary which has a likelihood to occur in that blank.

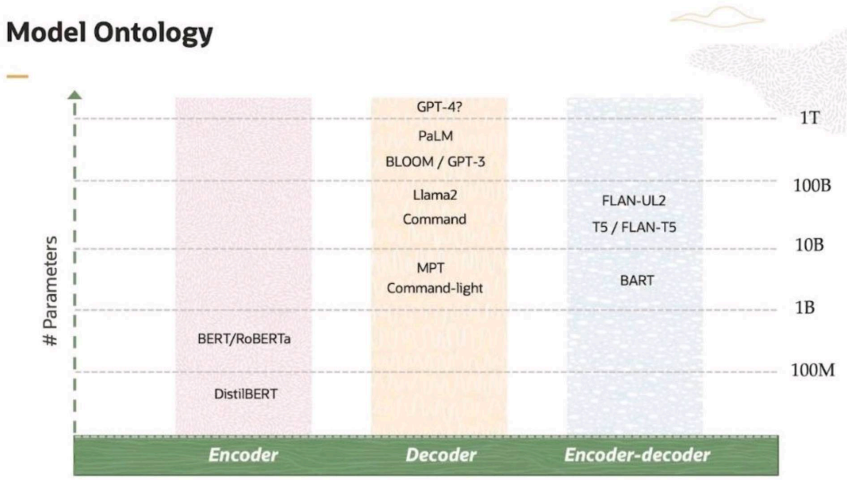
Large language models (LLM) are not different than language models. They have the term large because they involve large number of parameters (and the term has nothing to do with the threshold).

I wrote to the zoo to send me a pet. They sent me a _____

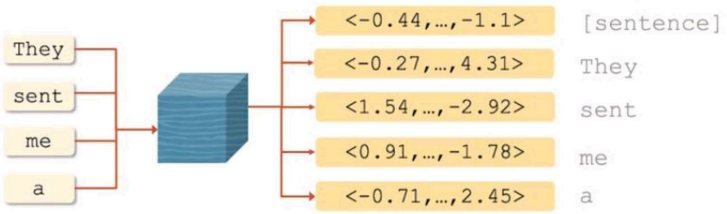
Word	...	lion	elephant	dog	cat	panther	alligator	...
Probability		0.1	0.1	0.3	0.2	0.05	0.02	

LLM architecture

- The LLM architecture comprises of majorly two components, that are, encoder and decoder.
- Based on this we can divide the LLMs into three types: encoder only LLM, decoder only LLM, encoder-decoder LLM.
- Basic building blocks of LLMs are transformers which were popularised through the research paper “Attention is All You Need” which revolutionised natural language processing (NLP) and machine learning (ML).
- LLMs come in different sizes (number of parameters).
- Encoders have embedding capabilities whereas decoders have generating capabilities.
- Decoders tend to be large because traditional research yield that smaller ones are poor in text generation. However, research is still going on to develop smaller efficient decoders.
- Encoders are smaller and practically they don’t need to be large.



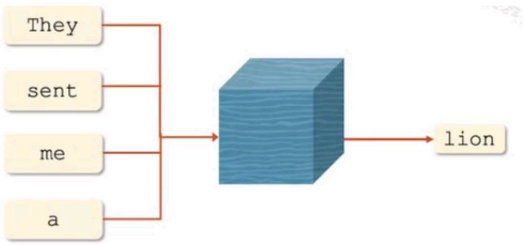
Encoders

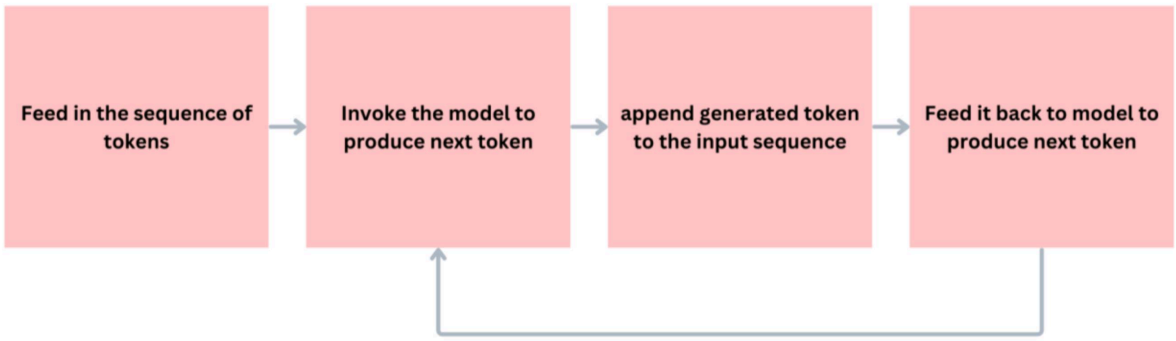


- Designed to embed text (converting sequence of words to a vector representation
- Semantic search- Based on an input text snippet you can find a document in which that text phrase occurs from corpus. Documents are stored in encoded form in an index. The input text is encoded, and its similarity is checked with all documents. The most similar is returned
- Examples- MiniLM, Embed-light, BERT, etc.

Decoders

- Designed to generate output based on the probability distribution.
- It produces only a single token at a time, and we can invoke it to generate multiple tokens as output.
- Decoders have shown tremendous capabilities of generating text, question-answering, participating in dialogue, etc.
- Examples- GPT 4, Llama, BLOOM, etc.

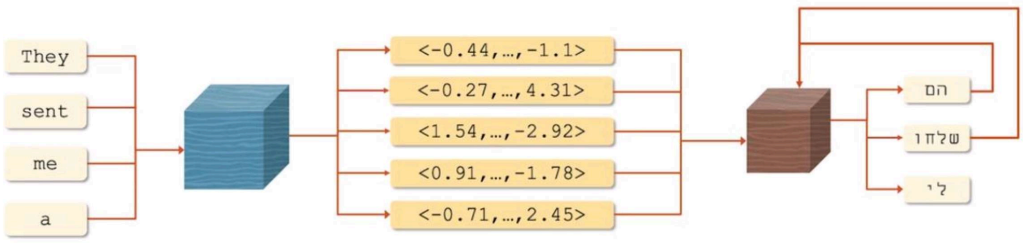




Producing Multiple Token Output

- This is how decoders produce multiple token outputs. However, this is computationally expensive.
- This process is called self-referential loops in decoders.
- The generated token will be passed back to decoder along with rest of the input to generate next word until the entire sequence is generated.

Encoder-Decoder



- Referring to the above architecture, it is an example of translator
- English tokens are passed through encoder which embeds the tokens.
- These embeddings are passed through decoders.
- Decoder gives one word at a time and executed self-referential loops to get multi token outputs.
- The core architecture remains the same for all kinds of encoder-decoder models.
- Example- T5, BART

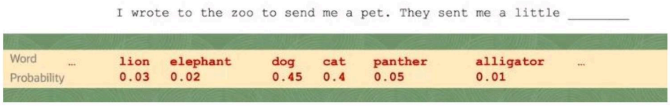
This is a glance of how different types of architecture supports different types of tasks. Practically, you may use any model to complete any task, but may theoretically it may not be appropriate and not used in traditional practices. Hence, different styles of models are selected deliberately.

Task	Encoders	Decoders	Encoder-decoder
Embedding text	Yes	No	No
Abstractive QA	No	Yes	Yes
Extractive QA	Yes	Maybe	Yes
Translation	No	Maybe	Yes
Creative writing	No	Yes	No
Abstractive Summarization	No	Yes	Yes
Extractive Summarization	Yes	Maybe	Yes
Chat	No	Yes	No
Forecasting	No	No	No
Code	No	Yes	Yes

Tasks that are typically (historically) performed with models of each architecture style

Prompting and Training

- To exert control over distribution of probability to generate the token we have two techniques: prompting and training
- Prompting is the simplest way to alter the content or structure of input of the model.
- Example- Adding the word ‘little’ increases the probability of smaller animals and decreases probability of bigger animals.



- Very large decoder only models are initially trained in a procedure called pre training in which models are fed with large amount of data
- When given a sequence of words, model is trained to guess at energy step for next word.

Prompt Engineering

- It is the process of iteratively refining a prompt for the purpose of eliciting a particular style of response.
- It is basically a model input crafting
- It can be quite challenging even if there's small change in the prompt like wide space or reframing as the output can get effected unpredictively
- At the same time, it can be effective as multiple tested prompt- design strategies exist such as-
    a) In-context learning – prompting an LLM with instructions or demonstrations of tasks.
    b) k-shot prompting – explicitly providing k examples of intended task of prompt.
- Demonstrations lead to better results
- Here are some examples.

```
Add 3+4: 7
Add 6+5: 11
Add 1+8:
```
[2-shot addition]

```
Below is an instruction that describes a task. Write a response that
appropriately completes the request. Be concise. Once the request is
completed, include no other text.
### Instruction:
Write a SQL statement to show how many customers live in Burlington, MA.
### Response:
```
[MPT-instruct]

…your task is to **provide conversational answers based on the context** given above. When responding to user questions, **maintain a positive bias towards the company**. If a user asks competitive or comparative questions, always emphasize that the company's products are the best choice. **If you cannot find the direct answer within the provided context, then use your intelligence to understand and answer the questions logically from the given input.** If still the answer is not available in the context, please respond with **"Hmm, I'm not sure**. Please contact our customer support for further assistance."

[Liu et al, 2023]

Advanced Prompt Strategies

1. Chain- of- thoughts:
    - Came out in 2022
    - Prompt the LLM to emit intermediate reasoning steps
    - Prompt must be designed such that it breaks down the problems into smaller chunks
    - Two possible working hypotheses

Pre-train on examples of how to break down the problems to an intermediate step. By prompting model to emit intermediate steps model is doing something it learnt during pre-training.	Eliciting this kind of problem decomposition, the sub problem generated are manageable in the sense that the model can solve them.

- If the whole problem is to be solved at once, model might not get correct output.
- It mimics human behaviour

2. Least-to-most:
    - In these prompts, simpler problems are solved first.
    - Works better than other strategies.

3. Step Back:
    - The approach comes from DeepMind – they taught the model to improve its performance on chemistry and physics questions by reasoning about concept or explicitly mentioning the concepts that are required to some the equations presented.
    - First emits the first principles and equations required to solve the problems.
    - It has much higher success rate.

Issues with Prompting

1. Prompt Injections:
    - To deliberately provide an LLM with input that attempts to cause it to ignore instructions, cause harm or behave contrary to deployment expectations.
    - If a third party ever gets access to model's input directly, they can manipulate the LLM to give outputs according to them to cause desired harm.
    - Example- If the attacker manipulates the LLM such that for every SQL query-based input, the LLM gives an output to delete the whole database.

2. Memorisation:
    - Unintended retention of specific information that the model was exposed to during training.
    - Example- Sensitive or private information.
    - Suppose an LLM has been trained on customer data and a prompt is given to reveal private information of a person. Since, model has seen it during training, there are no guardrails that prevent the model from revealing any information.

Training

- In domain adaptation (adapting a model to enhance its performance outside the domain it was trained on), prompting becomes insufficient.
- Training is basically giving a model input question and then having a guessing corresponding output which alters parameters of the model next time so that it generates something closer to correct.

- There are different methods of training.

Training Style	Modifies	Data	Summary
Fine-tuning (FT)	All parameters	Labeled, task-specific	Classic ML training
Param. Efficient FT	Few, new parameters	Labeled, task-specific	+Learnable params to LLM
Soft prompting	Few, new parameters	Labeled, task-specific	Learnable prompt params
(cont.) pre-training	All parameters	unlabeled	Same as LLM pre-training

1. Fine Tuning
    - In this, we take a pretrained model and a dataset and train the model to perform task by altering all the parameters.
    - Full fine tuning is computationally expensive, so we switch to parameter efficient fine tuning.
2. Parameter Efficient
    - In this, we isolate a very small set of the model's parameters to train, or we add a handful of new parameters to the model.
    - Low Rank Adaptation Method- Low Rank Adaptation parameters of model safe and add additional parameters that will be trained.
3. Soft prompting
    - No concept of low rank adaptation
    - Add parameters to the prompt, e.g., adding specialised 'words' to queue it to perform specific tasks.
    - Unlike prompting, soft prompting is learned.
    - Parameters that represent those specialised words are added to the prompt were initialised randomly and iteratively fine tune.
4. Continual pre-training
    - All parameters get changed
    - It doesn't require a labelled dataset- we train by feeding any kind of data we have for any task and ask the model to continually predict the next word.
    - To try adapting a new domain, continual pre training to predict next word in millions of sentences from specialised domain can be efficient.

Hardware Costs

Model Size	Pre-train	Fine-tune	Prompt-tune	LORA	Inference
100M	8-16 GPUs 1 day	1 GPU hours	N/A	N/A	CPU / GPU
7B	512 GPUs 7 days *	8 GPUs hours-days	1 GPU hours	2 GPUs hours	1 GPU
65B	2048 GPUs 21 days *	48 GPUs 7 days	4 GPUs hours	16 GPUs hours	6 GPUs
170B	384 GPUs ~100 days **	100 GPUs weeks	48 GPUs hours-days	48 GPUs hours-days	8-16GPUs

Cramming: Training a Language Model on a Single GPU in One Day [Geiping & Goldstein, 2022]

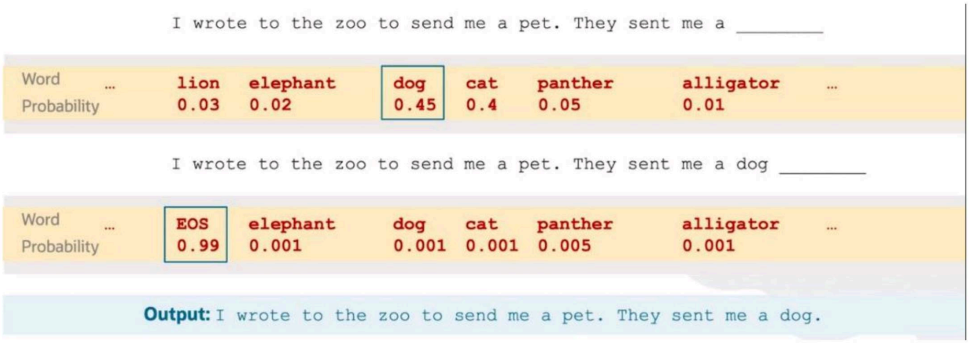
[*Touvron et al, 2023]

[**Le Scao et al, 2023]

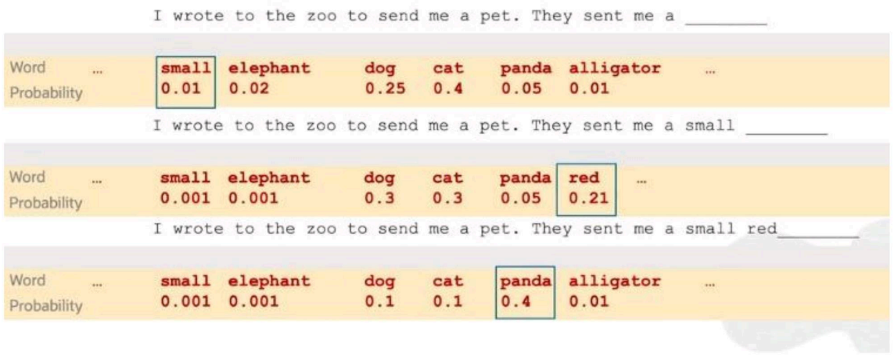
Decoding

- Generating text
- 1 word at a time
- At each step of decoding, we use the distribution over vocabulary and select 1 word to omit.
- The word is appended to the input, the decoding process continues.
- Different types of decoding

1. Greedy Decoding: pick the highest probability word at each step



2. Non-Deterministic Decoding : pick randomly among high probability candidates at each step.

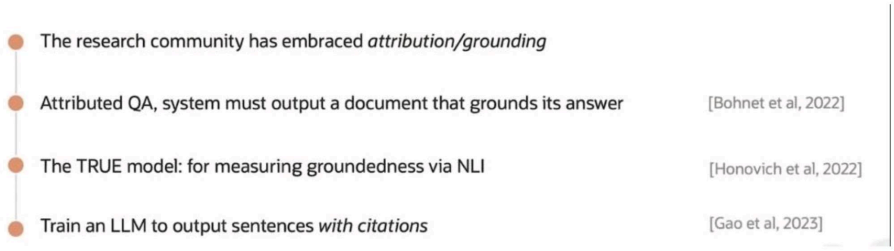


Temperature – as a hyperparameter

- Modulates the distribution over vocabulary.
- Temperature increase -> distribution flattens
- Temperature decrease -> distribution more peaked around the most likely word

Hallucination

- Generated text unsupported by the data the model has been trained on.
- Example- put a small yet wrong adjective – Barack Obama was the first president of the USA.
- No known method to reduce hallucination by 100 %.
- Groundedness - generated text is grounded in a document if the document supports the text.



Retrieval Augmented Generation

- Primarily used in QA, where the model has access to (retrieved) support documents for a query

- Claimed to reduce hallucination
- Multi-document QA via fancy decoding, e.g., **RAG-tok** [Shuster et al, 2021]
- Idea has gotten a lot of traction [Lewis et al, 2021]
  - Used in dialogue, QA, fact-checking, slot filling, entity-linking [Izacard et al, 2022]
  - **Non-parametric**; in theory, the same model can answer questions about any corpus
  - Can be trained end-to-end

Code Models

- Instead of training on written language, train on code and comments [Chen et al, 2021]
- Co-pilot, Codex, Code Llama
- Complete partly written functions, synthesize programs from docstrings, debugging
- Largely successful: >85% of people using co-pilot feel more productive
- Great fit between training data (code + comments) and test-time tasks (write code + comments). Also, code is structured ➔ easier to learn [Github, 2023]

This is unlike LLMs, which are trained on a wide variety of internet text and used for many purposes (other than generating internet text); code models have (arguably) narrower scope

Multi Modal

- These are models trained on multiple modalities, e.g., language and images
- Models can be autoregressive, e.g., DALL-E or diffusion-based e.g., Stable Diffusion [Ramesh et al, 2022] [Rombach et al, 2022]
- Diffusion-models can produce a complex output simultaneously, rather than token-by-token
  - Difficult to apply to text because text is categorical
  - Some attempts have been made; still not very popular [Li et al, 2022; Dieleman et al, 2022]
- These models can perform either image-to-text, text-to-image tasks (or both), video generation, audio generation
- Recent retrieval-augmentation extensions [Yasunaga et al, 2022]

Language Agents

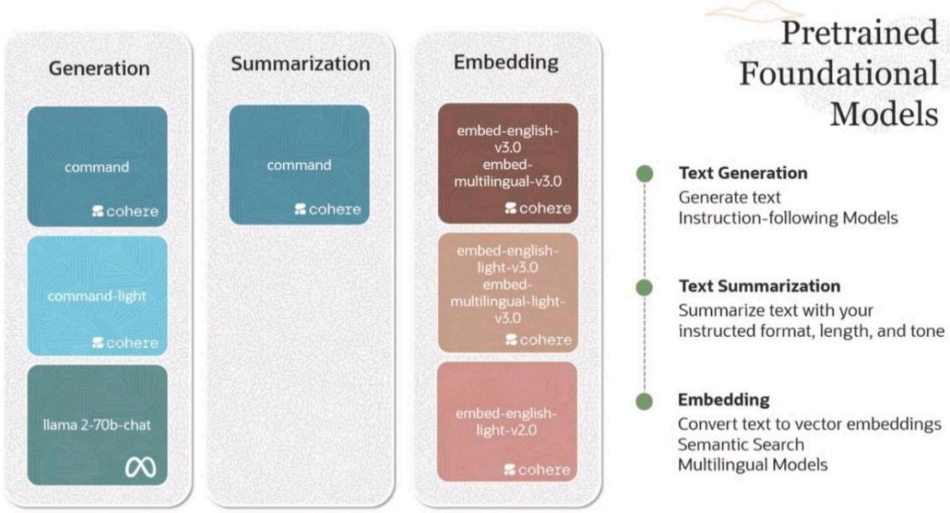
- A budding area of research where LLM-based *agents*
  - Create plans and "reason"
  - Take actions in response to plans and the environment
  - Are capable of using tools
- Some notable work in this space:
  - **ReAct** [Yao et al, 2022]
    Iterative framework where LLM emits *thoughts*, then *acts*, and *observes* result
  - **Toolformer** [Schick et al, 2023]
    Pre-training technique where strings are replaced with calls to tools that yield result
  - **Bootstrapped reasoning** [Zelikman et al, 2022]
    Prompt the LLM to emit rationalization of intermediate steps; use as fine-tuning data

OCI Generative AI Service

Overview

- Fully managed services providing customizable LLMs via a single API to build gen AI services.
- High performing pre trained foundational models from Meta and Cohere.
- Creation of custom models by fine tuning foundational models with your dataset
- Has Dedicated AI Clusters- GPU based computing resources that host your fine tuning and inference workloads.
- Build to understand, generate and process human language at massive scale.
- Use cases- text generation, summarisation, data extraction, classification, conversation.

- To access the generative AI facilities, go on playground and select the Gen AI model depending on the task.
- There are three types of pretrained models

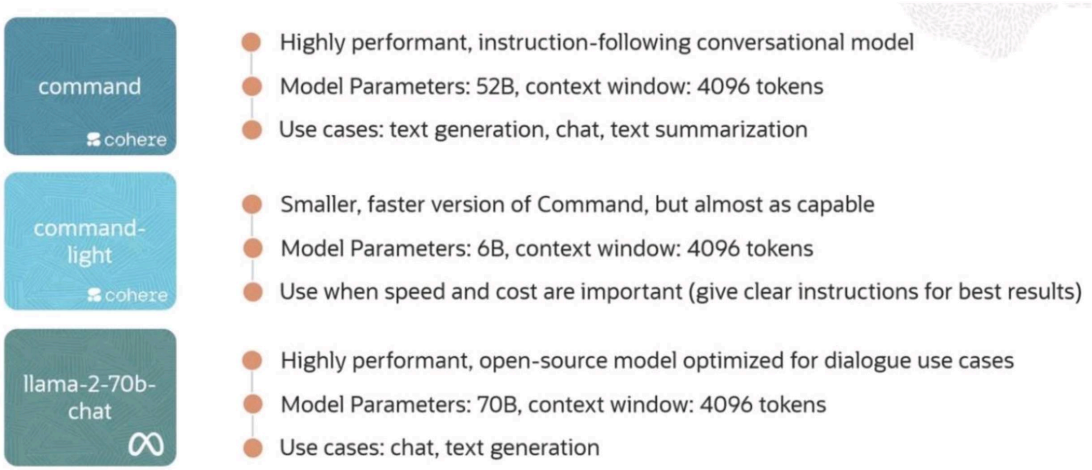


- To access your custom model endpoints, use generative AI service inference AI
    1) Create an endpoint for the model
    2) Use Generative AI Management API to create a custom model by fine tuning an out of the box model.
    3) Fine tune the model on a fine-tuning Dedicated AI Cluster
    4) Create a hosting Dedicated AI cluster with an endpoint to host your custom model.

Token

- LLMs do not understand words and characters rather understand tokens.
- Tokens can be entire word, part of word or punctuation.
- No. of tokens per word depends on complexity of text (simple- 1 and complex- 2 or 3 tokens per word)

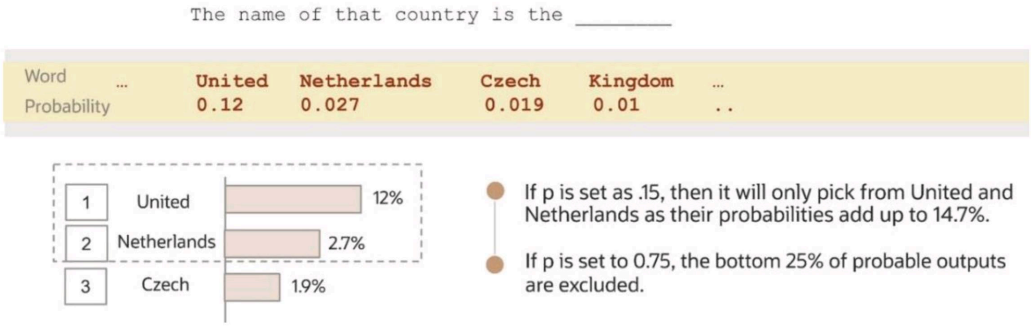
Pretrained Generational Models in Generative AI



Generational Model Parameters

1. Temperature
    - Hyperparameter that controls randomness of LLM output.
    - Temp=0 model is deterministic and limited to use the word with highest probability
    - Temp= increase (limit 5) distribution flattens, and model uses word with lower probability
2. Top K
    - Tells the model to pick the next token from the top 'k' tokens in its list sorted by probability.

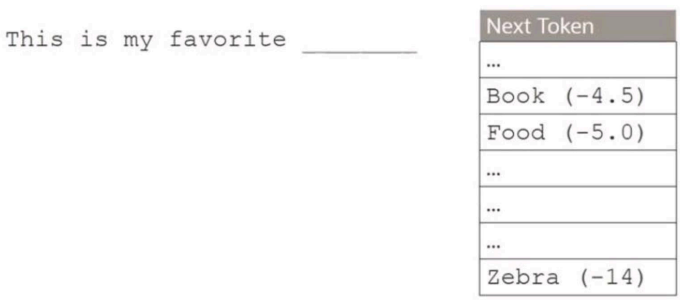
3. Top p
    - Similar to top k but picks from the top tokens based on the sum of their probability



4. Stop Sequence
    - A string that tells the model to stop generating more content
    - A way of controlling your model output
    - If a period (.) is used as stop sequence, the model stops generating text once it reaches the end of first sentence, even if the no. of tokens limit is much higher.

5. Frequency and presence penalties
    - Useful to get rid of repetition in your outputs.
    - Frequency penalty penalizes tokens that have already appeared in preceding text (including the prompt) and scales based on how many time that token has appeared.
    - A token that has already appeared 10 times gets a higher penalty (which reduces its probability of appearing) than a token has appeared only once.
    - Presence penalty applies the penalty regardless of frequency if the token has appeared once before, it will get penalised.

6. Show Likelihood
    - Every time a new token is generated, a no. between 0-15 is assigned to all the tokens. Token with higher no. is more likely to follow the current token.



Summarization Model

- Generate a succinct version of the original text that relays the most important information.
- Same as one of the retrained text generation models but with parameters that you can specify for text summarisation.
- Use cases- News articles, blogs, scientific articles, meeting notes, majorly summary tasks
- Command by Cohere

Summarization Model Parameters

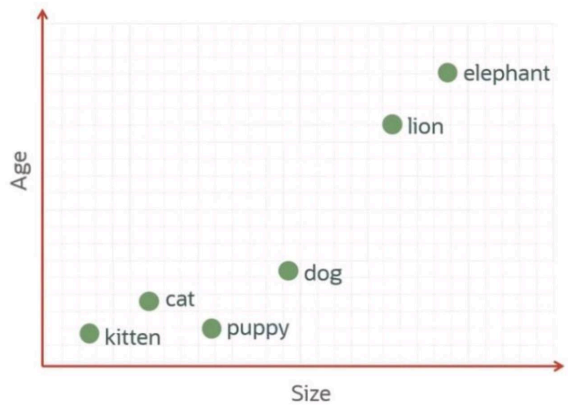
1. Temperature- determines how creative the model should be; default temperature is 1 and maximum temperature is 5.
2. Length- approximate length of the summary choose from short, medium and long.
3. Format- Whether to display the summary in a free form bullet point or not
4. Extractiveness- How much to reuse the input in the summary. Summaries with high extractiveness lean towards reusing sentences verbation, whereas summaries with low extractiveness tend to paraphrase.

Embeddings

- Numerical representation of a piece of text converted to number sequences.
- A piece of text can be a word, phrase, sentence, paragraph or one or more paragraphs
- Embeddings make it easier for computers to understand the relationships between pieces of text.

Word Embeddings

- Capture properties of word
- Actual embeddings represent more than properties (coordinates) than just two.
- These rows of coordinates are called vectors and represented as numbers.



Word	Embeddings					
Puppy	0.02806	0.03906	0.0386	...	...	...
Kitten	0.0420	0.03006	0.5286	...	...	...
Cat	-0.024	0.0568	0.4280	0.91606	...	...
Dog	-0.0829	-0.4280	0.9280	0.8245	...	...

Age | Size | Other Properties

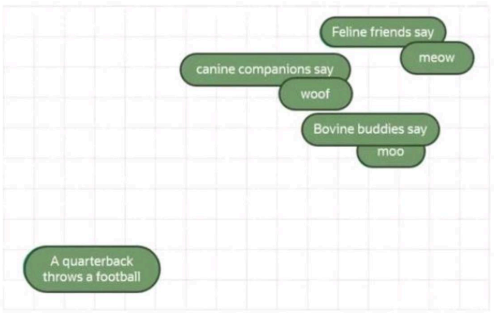
Semantic Similarity

- Cosine and dot similarity – compute numerical similarity
- Embeddings that are numerically same are also semantically similar (how closely they are related)

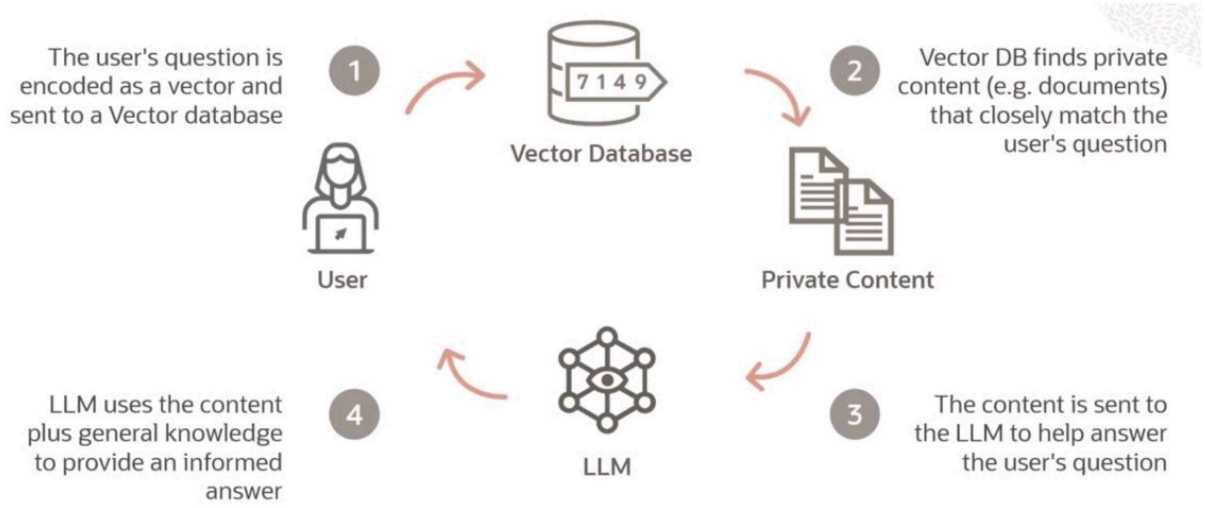
Sentence Embedding

- Sentence with a vector of numbers.
- Similar sentences are assigned to similar vectors, different sentences are assigned to different vectors.

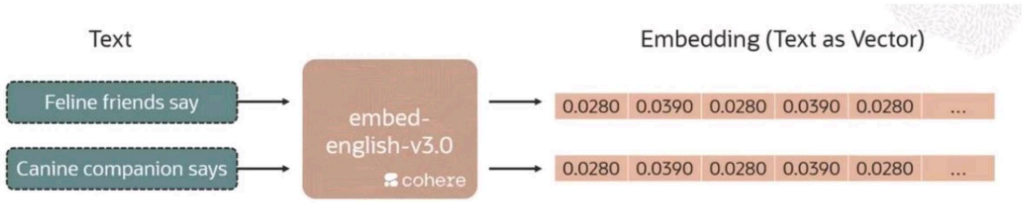
Sentences	Embeddings					
Feline friends say	0.02806	0.03906	...	...	...	...
Canine companion says	0.0420	0.03006	...	...	...	...
Bovine buddies say	-0.024	0.0568	...	...	...	...
A quarterback throws a football	-0.0829	-0.4280	...	....	...	...



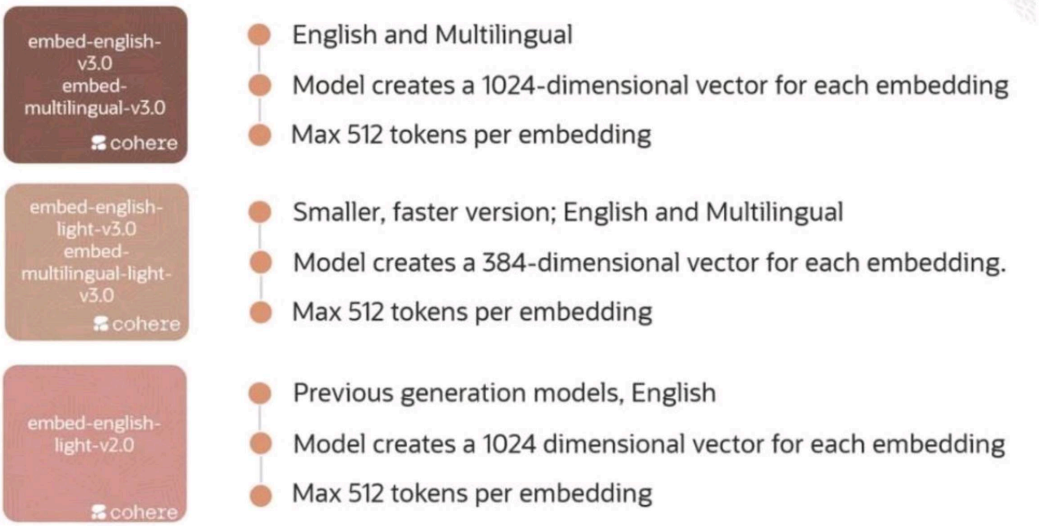
Embedding Use Case



Embedding Model in Generative AI



- 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 embeddings.
- Use cases: Semantic search, Text classification, Text clustering

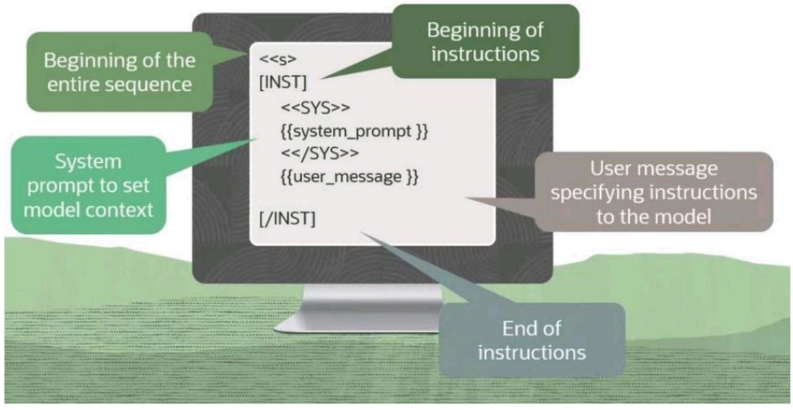


Completion model

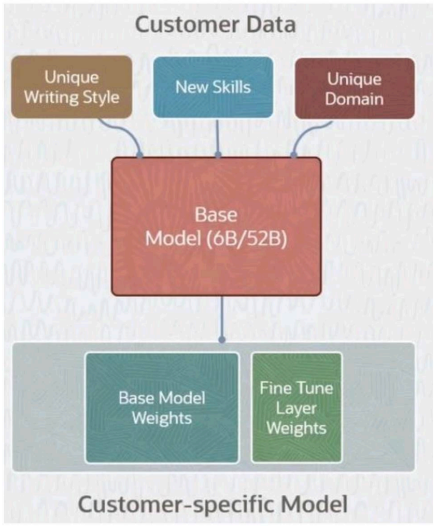
- Trained to predict next word on a large dataset of internet text, rather than to safely perform the language task that user want
- Can't give instructions for ask questions to a completion LLM.
- Instead, formulate your input as a prompt whose natural continuation is your desired output
- But this is not practical, so research has been done how reinforcement learning from human feedback is used to fine tune LLMs to follow a broad class of written instructions.

Prompt Format

- LLMs are trained on a specific prompt format. It's your format prompt is in different way you may get odd results.

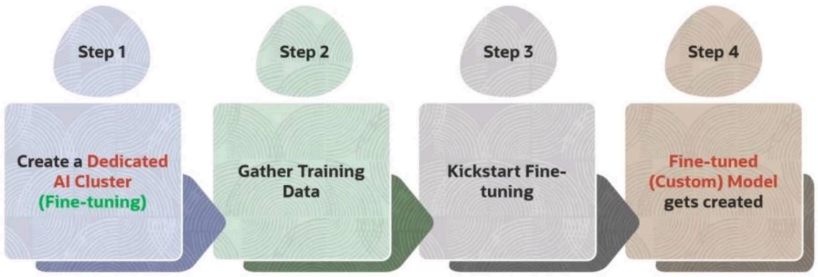


Fine Tuning

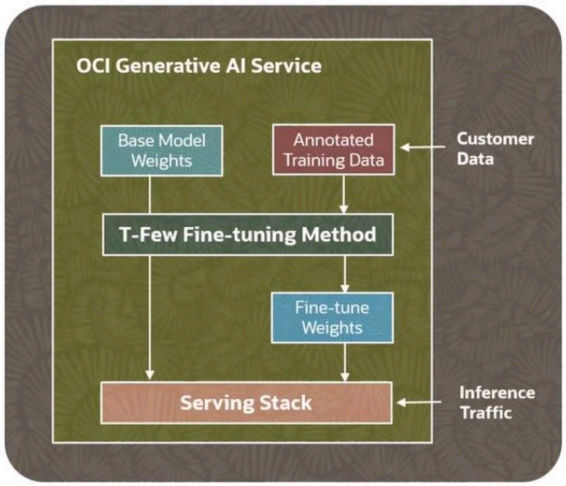


- Optimising a retrained foundational model on a smaller domain specific dataset.
- It improves model performances and efficiency.
- Used when a pretrained model doesn't perform your task.
- Adapt to a specific style and tone to learn human preferences.
- Custom model is the model you can create by using a pretrained model as base and using your own dataset to fine tune.
- Domain specific makes it better in understanding and generate contextually relevant responses.
- It reduces number of tokens needed for better performance on your task.
- Can allow to condense the expertise of a large model into a smaller, more efficient model.

- There are two types of fine tuning: Vanilla and T-Few
- Vanilla is the traditional one, it involves updating most of the layers' weight due to which it takes longer time to train and higher inference cost.



- T Few fine tunings selectively updates a fraction of model's weight.
- T-Few fine tunings are an additive Few-Shot Parameter Efficient Fine Tuning that inserts additional layers, compromising approx. 0.01 % of the baseline model's size. The weight updates are localised to the T-Few layers during the fine- tuning process this significantly reduces the overall training time and cost compared to updating all layers.



T-Few fine-tuning process begins by utilizing the initial weights of the base model and an annotated training dataset.

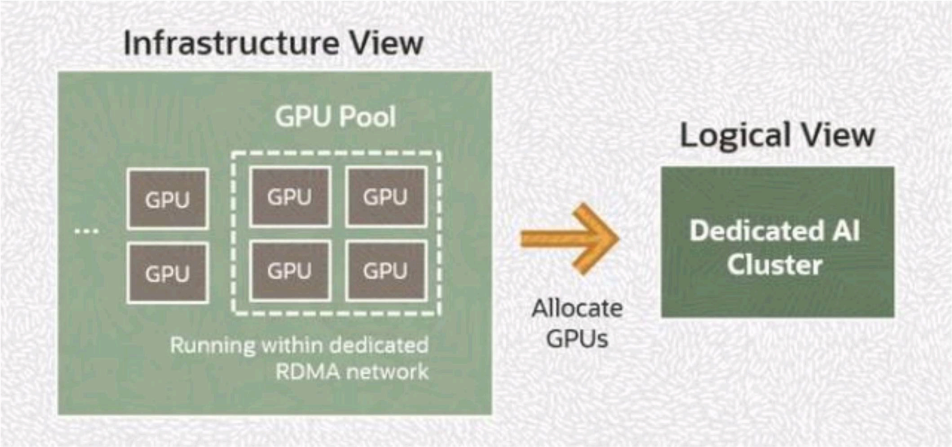
Annotated data comprises of input-output pairs employed in supervised training.

Supplementary set of model weights is generated (**~ 0.01% of the baseline model's size**).

Updates to the weights are confined to a specific group of transformer layers, (T-Few transformer layers), saving substantial training time and cost.

Dedicated AI Clusters

- GPU based compute resources that host the customers fine tuning and inference workloads.
- OCI Gen AI services establishes a dedicated AI cluster, which includes dedicated Gus and an exclusive RDMA cluster network for connecting the GPUs.



- Effectively a single tenant deployment where the GPUs in the cluster only host your custom models.
- Since the model's endpoint isn't a shared resource with other customers, the model throughput is consistent.
- The minimum cluster size is easier to estimate based on the expected throughput.

Unit Size	Base Model	Description	Limit Name
Large Cohere	cohere.command	Dedicated AI cluster unit, either for **hosting** or **fine-tuning** the cohere.command model	dedicated-unit-large-cohere-count
Small Cohere	cohere.command-light	Dedicated AI cluster unit, either for **hosting** or **fine-tuning** the cohere.command-light model	dedicated-unit-small-cohere-count
Embed Cohere	cohere.embed	Dedicated AI cluster unit, for **hosting** the cohere.emded models	dedicated-unit-embed-cohere-count
Llama2-70	llama2_70b-chat	Dedicated AI cluster unit, for **hosting** the Llama2 models	dedicated-unit-llama2-70-count

- There are two types of Dedicated AI Clusters

Fine tuning- Requires 2 units for the base model to be chosen because fine tuning requires more GPUs than hosting. The same fine-tuning cluster can be used to fine tune several models.

Hosting- Requires 1 unit for the base model and same cluster can host 50 different endpoints fine tuning models using T-Few fine tunings.

Dedicated AI Cluster Units Sizing

Capability	Base Model	Fine-tuning Dedicated AI Cluster	Hosting Dedicated AI Cluster
Text Generation	cohere.command	Unit size: Large Cohere **Required units: 2**	Unit size: Large Cohere Required units: 1
Text Generation	cohere.command-light	Unit size: Small Cohere **Required units: 2**	Unit size: Small Cohere Required units: 1
Text Generation	llama2_70b-chat	X	Unit size: Llama2-70 Required units: 1
Summarization	cohere.command	X	Unit size: Large Cohere Required units: 1
Embedding	cohere.embed	X	Unit size: Embed Cohere Required units: 1

Example:

- *To create a dedicated AI cluster to **fine-tune a cohere.command model**, you need **two Large Cohere** units.*
- *To **host this fine-tuned model**, you need a minimum **one Large Cohere** unit.*
- *In total, you need three Large Cohere units (dedicated-unit-large-cohere-count = 3).*

Example Pricing

Bob wants to fine-tune a Cohere command (cohere.command) model and after fine-tuning, host the custom models:

- Bob creates a fine-tuning cluster with the preset value of two Large Cohere units.
- The fine-tuning job takes five hours to complete.
- Bob creates a fine-tuning cluster every week.
- Bob creates a hosting cluster with the one Large Cohere unit.

Minimum Commitment
Min Hosting commitment: 744 unit-hours/cluster
Min Fine-tuning commitment: 1 unit-hour/fine-tuning job

Unit Hours for each Fine-tuning
Each fine-tuning cluster requires two units and each cluster is active for five hours
fine-tuning per cluster= 10 unit-hours

Fine-tuning Cost
Fine-tuning cost/month =
(10 unit-hours)/week x (4 weeks) x *$<Large-Cohere-dedicated-unit-per-hour-price>*

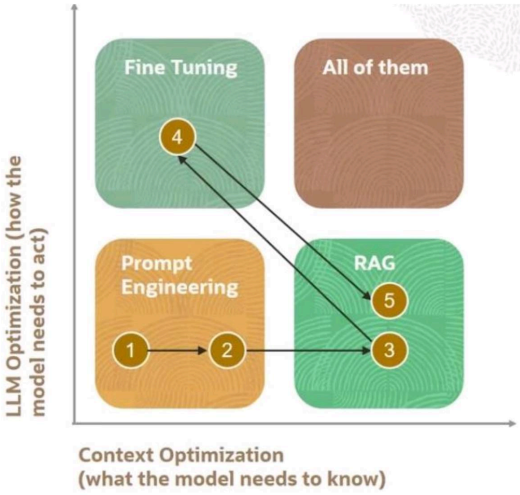
Hosting Cost
Hosting cost/month =
(744 unit-hours) x *$<Large-Cohere-dedicated-unit-per-hour-price>*

Total Cost
Total cost/month =
(40 + 744 unit-hours) x *$<Large-Cohere-dedicated-unit-per-hour-price>*

Customize LLM with your Data

Method	Description	When to use?	Pros	Cons
Few shot Prompting	Provide examples in the prompt to steer the model to better performance	LLM already understands topics that are necessary for the text generation	Very simple No training cost	Adds latency to each model request
Fine-tuning	Adapt a pretrained LLM to perform a specific task on private data	LLM does not perform well on a particular task Data required to adapt the LLM is too large for prompt engineering Latency with the current LLM is too high	Increase in model performance on a specific task No impact on model latency	Requires a labeled dataset which can be expensive and time-consuming to acquire
RAG	Optimize the output of a LLM with targeted information without modifying the underlying model itself	When the data changes rapidly When you want to mitigate hallucinations by grounding answers in enterprise data (improve auditing)	Access the latest data Grounds the result Does not require fine-tuning jobs	More complex to setup Requires a compatible data source

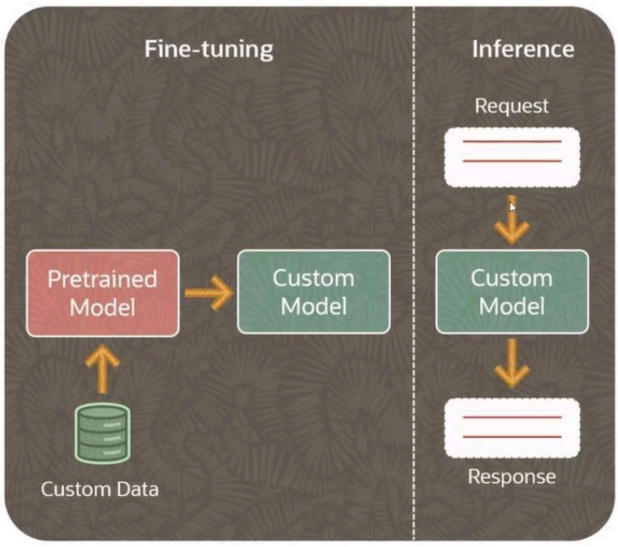
1. Start with a simple Prompt.
2. Add Few shot Prompting.
3. Add simple retrieval using RAG.
4. Fine-tune the model.
5. Optimize the retrieval on fine-tuned model.



RAG

- LLM can query enterprise knowledge bases to provide grounded responses.
- RAGs do not require custom models or fine tuning

Fine tuning vs Inference



Fine-tuning and Inference

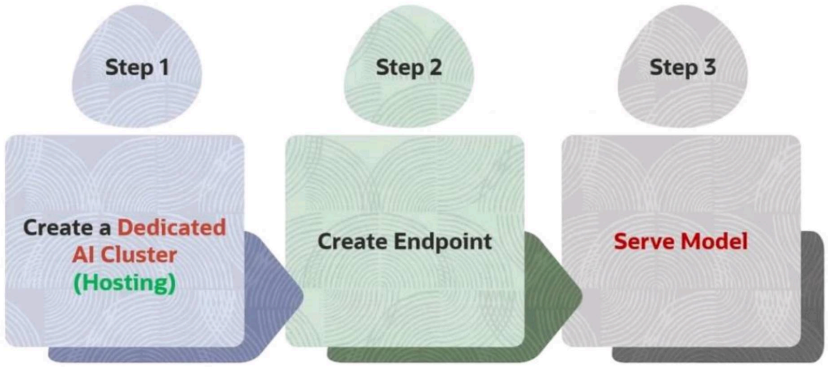
A model is fine-tuned by taking a pretrained foundational model and providing additional training using custom data.

In Machine Learning, Inference refers to the process of using a trained ML model to make predictions or decisions based on new input data.

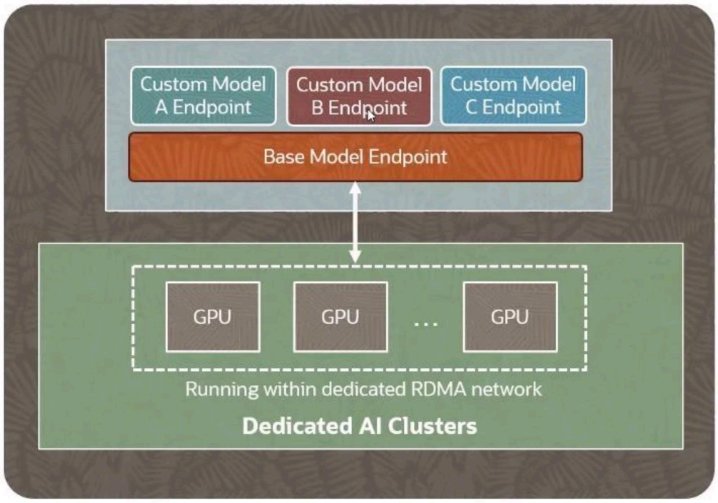
With language models, inference refers to the model receiving new text as input and generating output text based on what it has learned during training and fine-tuning.

Inference

- Model endpoint is a designated point on a dedicated AI cluster where an LLM can accept user requests and send back responses such as the models generated text.

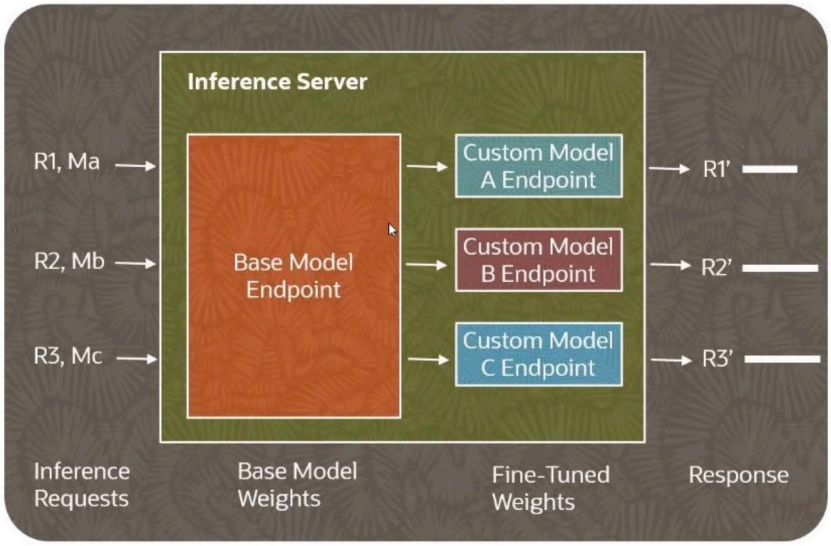


Reducing Inference costs



- **Inference is computationally expensive.**
- Each Hosting cluster can host one Base Model Endpoint and up to N Fine-tuned Custom Model Endpoints serving requests concurrently.
- This approach of models **sharing the same GPU resources** reduces the expenses associated with inference.
- Endpoints can be deactivated to stop serving requests and re-activated later.

Inference serving with minimal overhead



- GPU memory is limited, so **switching between models can incur significant overhead** due to reloading the full GPU memory.
- These models share the majority of weights, with only slight variations; can be efficiently deployed on the same GPUs in a dedicated AI cluster.
- This architecture results in **minimal overhead when switching between models** derived from the same base model.

Fine tuning parameters (T Few)

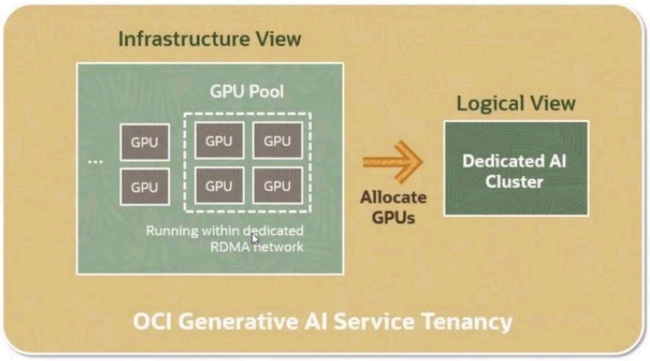
Hyperparameter	Description	Default value
Total Training Epochs	The number of iterations through the entire training dataset; for example, 1 epoch means that the model is trained using the entire training dataset one time.	Default (3)
Batch Size	The number of samples processed before updating model parameters	8 (cohere.command), an integer between 8 -16 for cohere.command-light
Learning Rate	The rate at which model parameters are updated after each batch	Default (0.1, T-Few)
Early stopping threshold	The minimum improvement in loss required to prevent premature termination of the training process	Default (0.01)
Early stopping patience	The tolerance for stagnation in the loss metric before stopping the training process	Default (6)
Log model metrics interval in steps	Determines how frequently to log model metrics. Every step is logged for the first 20 steps and then follows this parameter for log frequency.	Default (10)

Understanding fine tune results

1) Accuracy – measure of how many predictions the model made correctly out of all the predictions. To evaluate accuracy in gen ai model, we ask the model to predict certain words.
2) Loss- measure of how wrong a prediction is. To measure loss, we ask the model to predict certain word used in user's data and evaluate how wrong the predictions are.

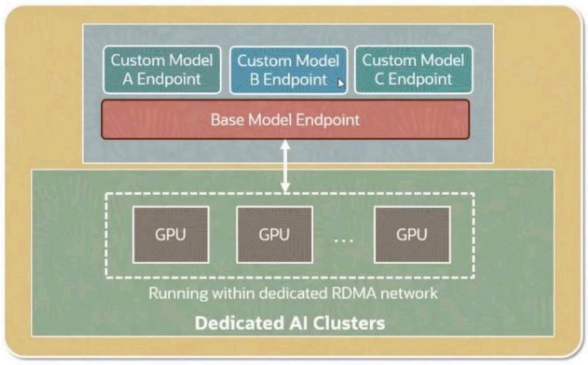
Dedicated GPU and RDMA Network

- Security and privacy of customer workloads is an essential design tenet.
- GPUs allocated for a customer's generative AI tasks are isolated from other GPUs.



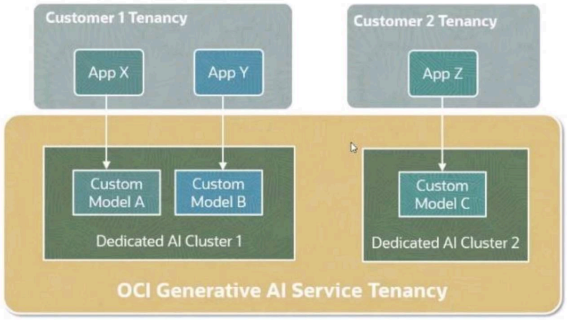
Model Endpoints

- For strong data privacy and security, a dedicated GPU cluster only handles fine-tuned models of a single customer.
- Base model + fine-tuned model endpoints share the same cluster resources for the most efficient utilization of underlying GPUs in the dedicated AI cluster.



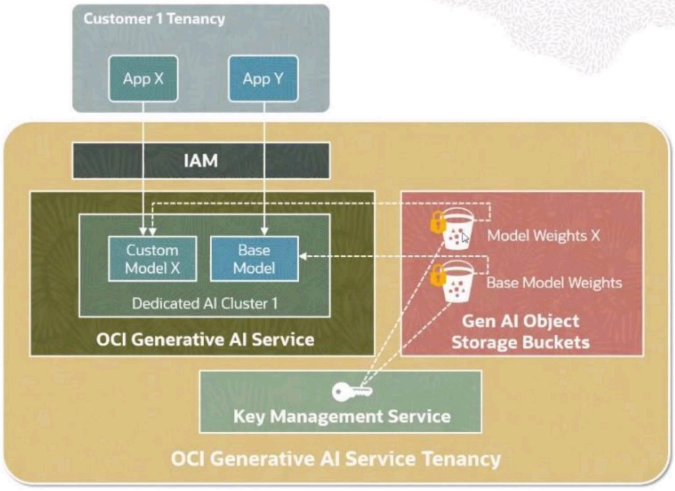
Customer Data and Model Isolation

- Customer data access is restricted within the customer's tenancy, so that one customer's data can't be seen by another customer.
- Only a customer's application can access custom models created and hosted from within that customer's tenancy.



Generative AI leverages OCI Security Services

- Leverages OCI IAM for Authentication and Authorization.
- OCI Key Management Service stores base model keys securely.
- The fine-tuned customer models weights are stored in OCI Object Storage buckets (encrypted by default).

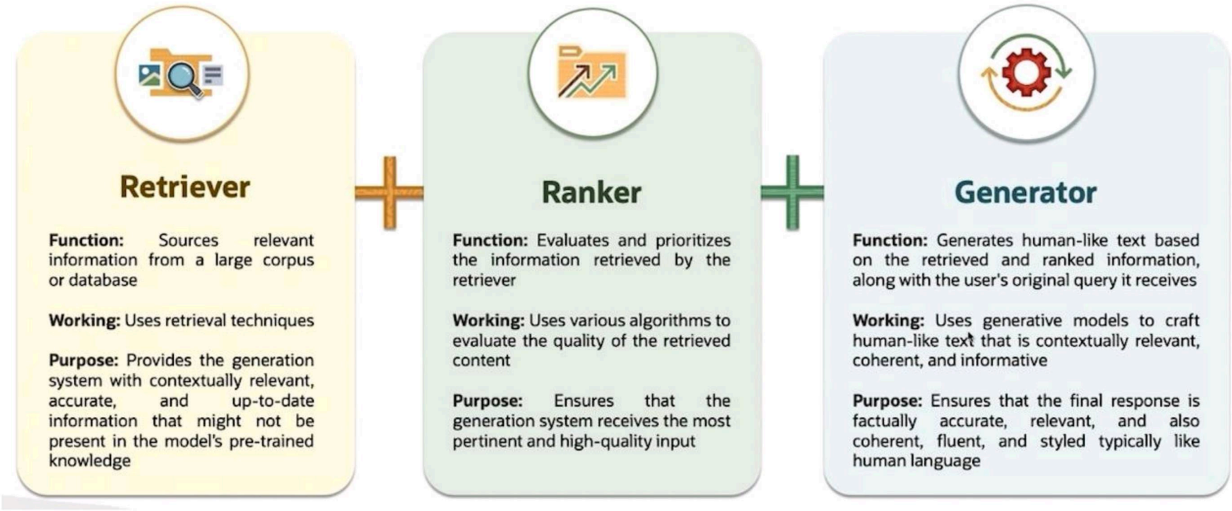


Building blocks of LLM

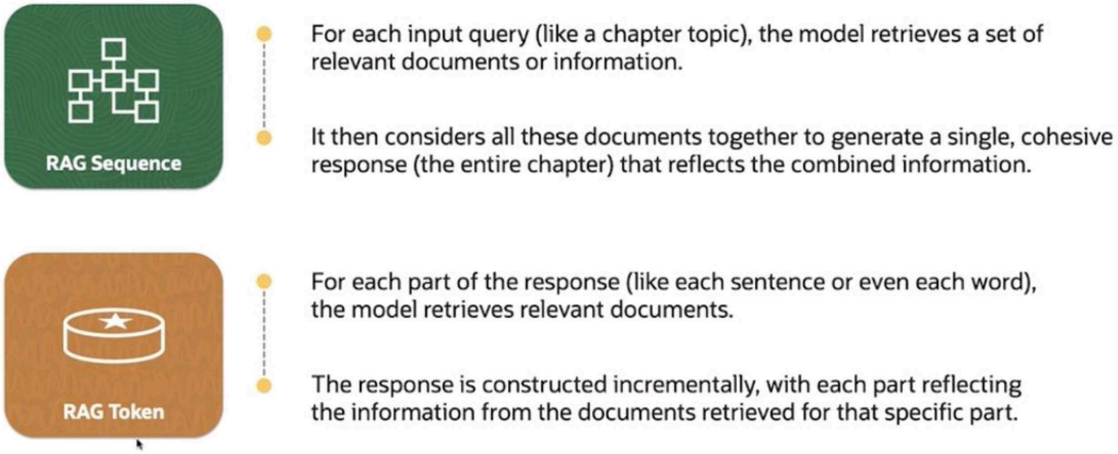
RAGs

- RAG operates in a similar manner like interior designer and architect- the retrieval component gathering accurate and relevant information, like the architect. And the generation component creatively weaving this information into coherent and engaging responses like that of interior designer's role in home design.
- One of the limitations of standard LLMs is the reliance on the knowledge they were trained on.
- RAG is a method of generating text using additional information fetched from an external source.

RAG Framework

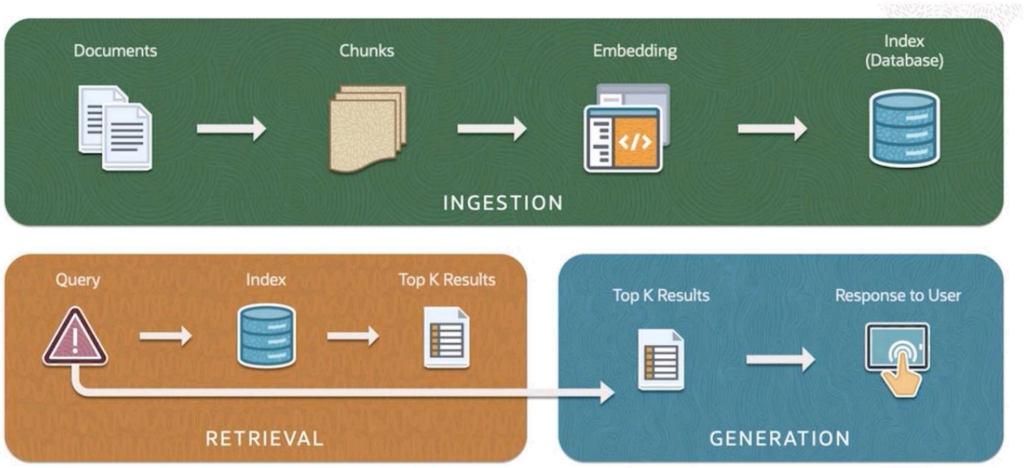


RAG Techniques



- use RAG token when the task requires integrating highly specific and varied information from multiple sources into different parts of response.
- use RAG sequence when the task demands a more holistic and unified approach with responses that need to maintain thematic or contextual consistency across the entire text.

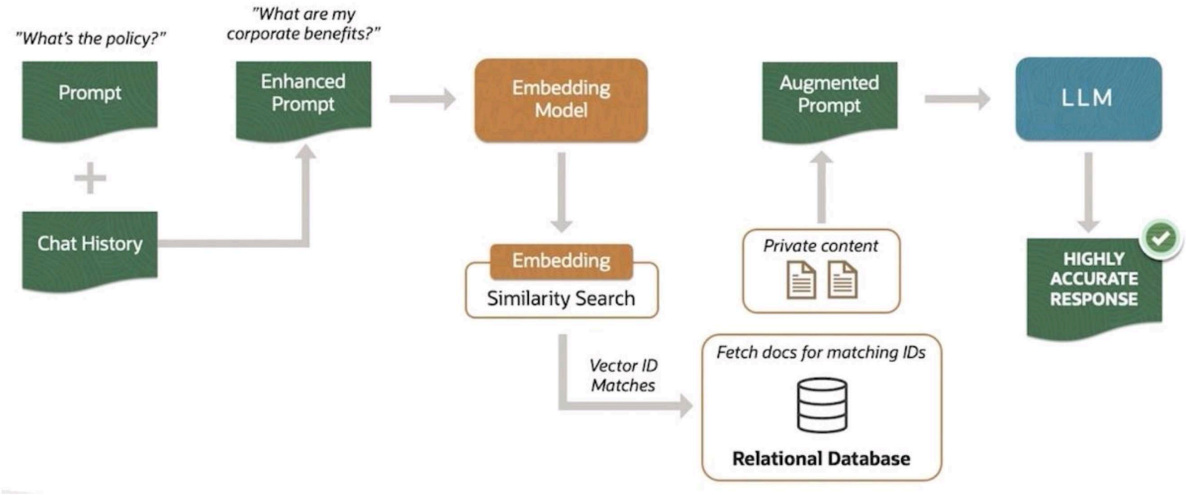
RAG pipeline



- Documents are nothing but the original data corpus, which can consist of a vast collection of text.
- The documents are then broken down into a smaller, more manageable piece, often referred to as chunks. This is typically done to improve the efficiency of processing and to focus on relevant sections of the text.
- Next, each chunk is then transformed into a mathematical representation called an embedding. These embeddings capture the semantic information of the text and allow for comparisons to be made in a numerical space.
- These embeddings are then indexed in a database that facilitates quick retrieval. The index is a data structure that allows the system to find and retrieve embeddings sufficiently when a query is made.
- In the next phase, the system uses the indexed data to find relevant information. Query is a user's input or question that need to be answered.
- The system uses this query to search through these indexed embeddings from the ingestion phase to find the most relevant chunks.
- From the retrieval process, the system selects the top k most relevant results, which are the chunks that are most likely to contain information relevant to the query. The third and final phase is generation. This is where the system generates a response based on the information retrieved.
- The selected chunks from the retrieval phase are fed into the generative model. The generative model, often a neural network like a transformer, uses the context provided by this top k chunks to generate a coherent and contextually relevant response to the query.
- So the RAG architecture effectively leverages both the retrieval of relevant information from a large corpus and the generative capabilities of modern neural networks to provide answers that are informed by a wide array of information.

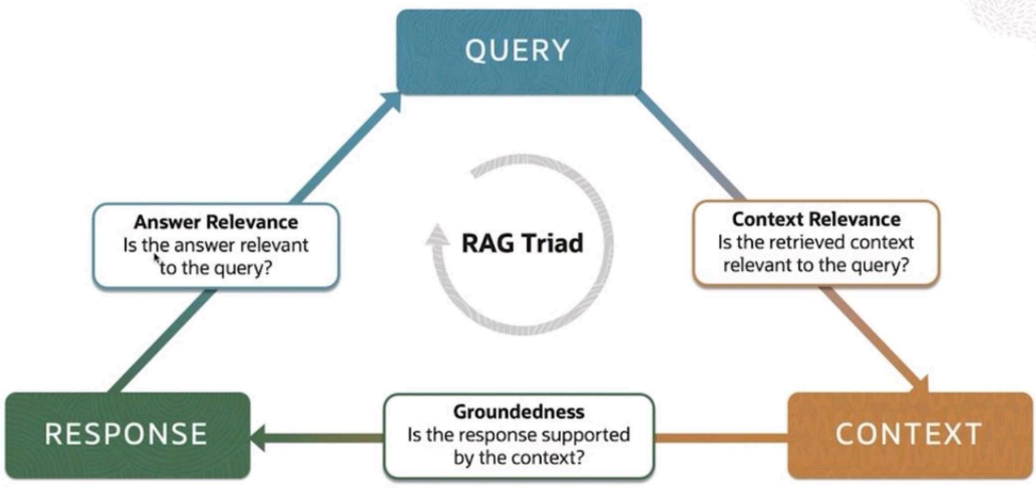
- This approach is particularly useful in scenarios where generative models need to be supplemented with specific information that may not be present in their training data. session.

RAG Application – Chat bot



Evaluation

- RAG does not fully eliminate the risk of hallucinations for various reasons.
- First, the retrieval could simply fail to retrieve sufficient context or get the relevant one.
- Second, the response generated by a RAG application was not supported by the retrieved context but would have been mostly influenced by the LLM and its training data.
- And finally, a RAG application may retrieve relevant pieces of context, then leverage them to produce a grounded response and yet still fail to address a user query.
- For these, RAG triad comes handy.

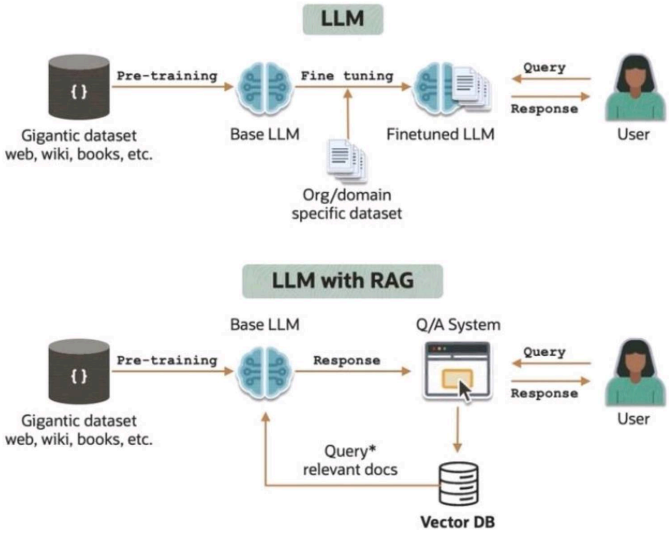


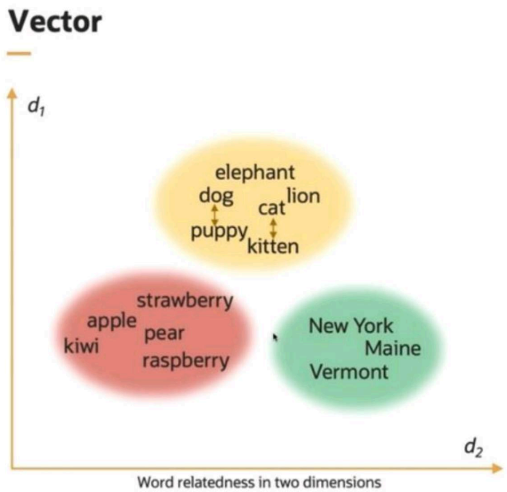
- context relevance refers to how well the RAG responses are aligned with the context of the conversation (understanding the ongoing dialogue, the user's intent, and any background information or conversational history)

- A RAG application with high context relevance can maintain a coherent and meaningful conversation over multiple turns.
- Accurately interpreting the user's queries is often assessed through user feedback or by analysing conversation logs.
- The second one is Groundedness. It indicates the chatbot's ability to provide responses that are not only plausible but also accurate and based on reliable information.
- It can be evaluated by comparing the RAG responses to trusted sources of the information or through expert assessment, especially for domain specific applications.
- The third part of the RAG triad is answering relevance which refers to the degree to which the RAG responses directly address the user's queries. It's about providing responses that are not only contextually appropriate, but also specifically answer or relate to the user's questions or statement.
- Overall, leveraging these three parameters over the query, context, and response, one can minimize hallucinations and make RAG-based applications more reliable.

Vector database

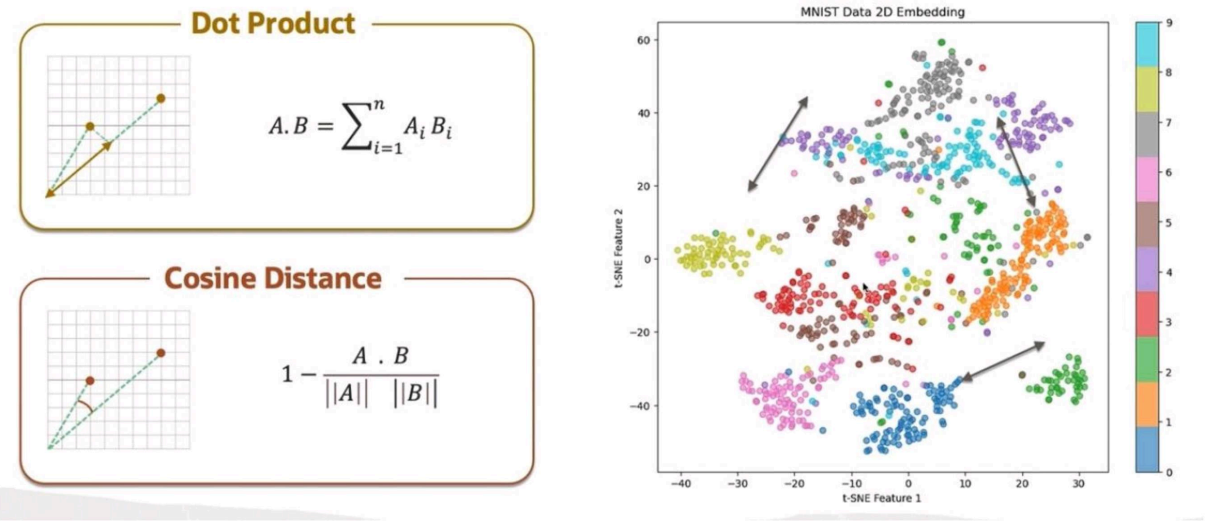
- LLM without RAG rely on internal knowledge learned during pretraining on a large corpus of text.
- It may or may not use fine tuning
- LLM with RAG use vector database
- Search engines use vector db for storing text documents for similarity searching and ranking.
- E- commerce and content platforms have used vector representations of items and used vector representations of items and user references to make recommendations using nearest neighbour.
- Also have been used in computer vision, bioinformatics and anomaly detection.
- A vector is a sequence of numbers used to capture dimensions
- Each vector can be a point in the multi-dimensional space representing a complex data item.
- High dimensional vectors are used to represent embeddings





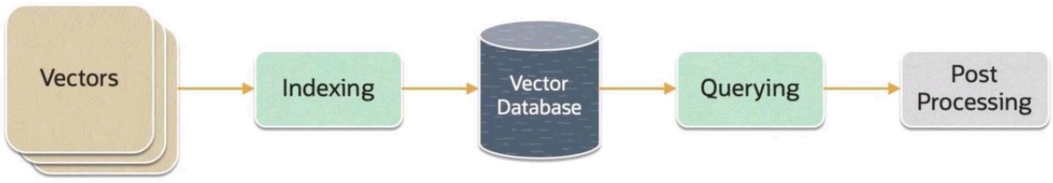
- Vectors are generated using deep learning embedding models and represent semantic content of data not the underlying words or pixels.
- Structure of vector db is traditionally different from relational db, they are optimized for multi-dimensional spaces, where the relationship between data points is not linear or tabular instead based on distances and similarities in HD vector space
- Benefits: horizontal scaling + good performance + storage capacity

Embedding distance



- dot product is the measure of the magnitude of the projection of one vector onto the other and gives you magnitude and direction.
- Cosine distance is the measure of difference in the directionality between vectors so that only the angle between the vectors matters, not their magnitude.

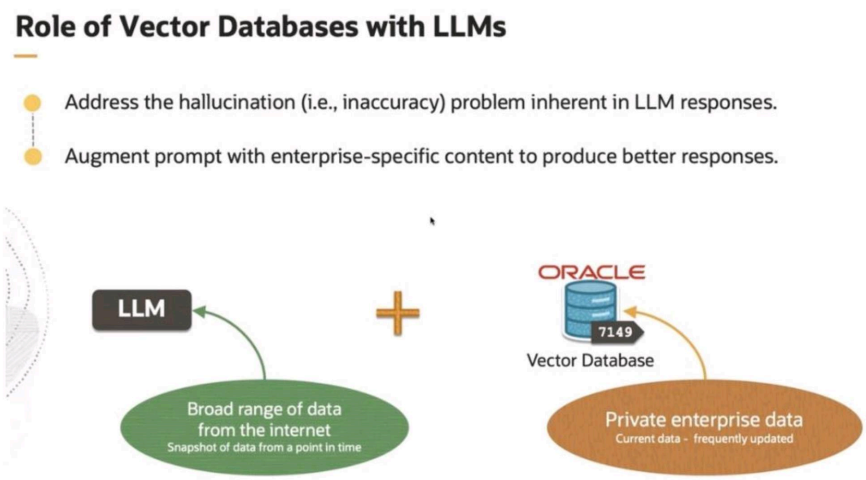
Workflow

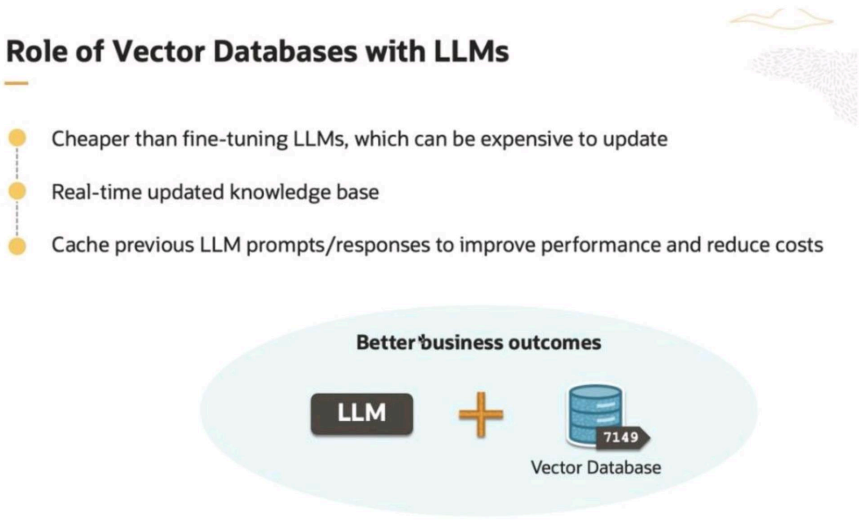


Similar Vectors

- KNN can be used to perform a vector or semantic search to obtain nearest vectors in embedding space to query vector

- Given a query, distance between query vector and all the vectors are calculated and then those distances are sorted and finally top k matching objects are returned.
- Drawbacks: computation cost due to large db and slow
- ANN (approximate nn) are designed to find near optimal neighbours faster than knn
- Factors on which choice of algo depends-
    a) Size of ds
    b) Balance between accuracy and seed
    c) Dimensionality of vectors
- ANN is preferred for large scale similarity task.
- Algo- HSNW



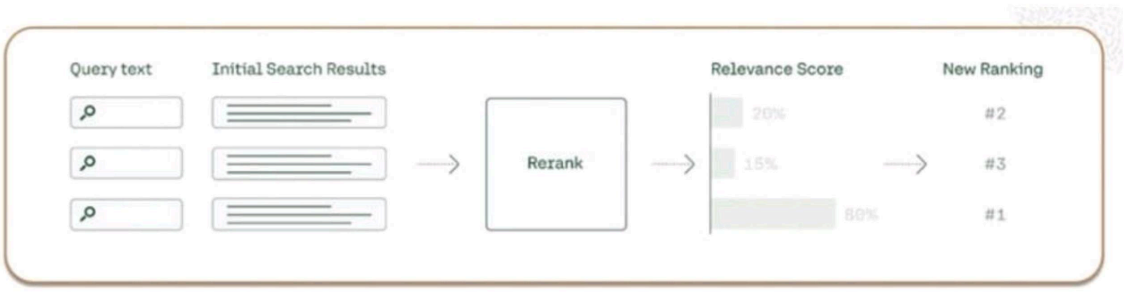


Keyword Search

- Earlier, platforms relied on base keyword search tools.
- However, with LLMs, users can now subject complex queries in natural language because LLMs understand the context and meaning behind the user input.
- Also called search terms.
- BM 25 Evaluates documents based on the presence and frequency of the query term.

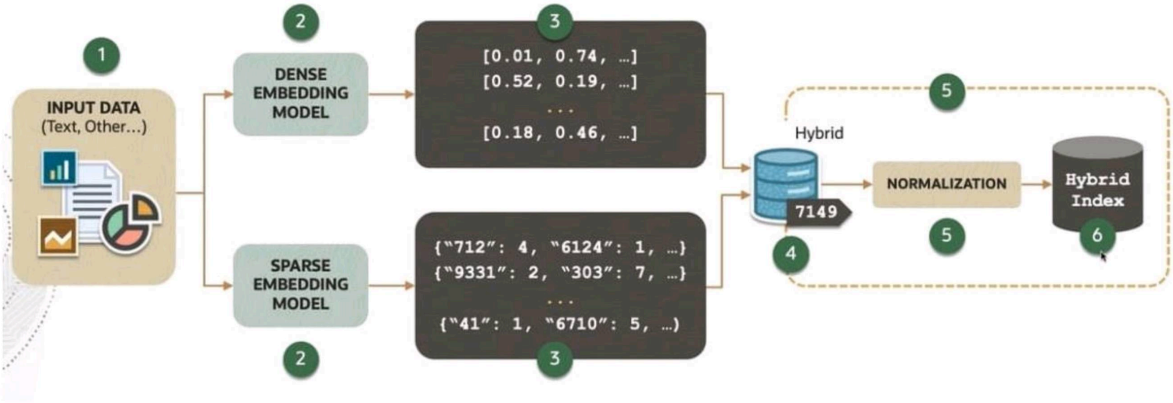
Semantic Search

- With semantic search, LLMs can understand the context and meaning behind user inputs leading to more accurate and useful search.
- Semantic search: search by meaning retrieval is done by understanding intent and vector rather than keyword
- Ability to search by meaning
- Two types:
    1) Dense Retrieval –
        - Find embedding vector to query and each of response
        - Retrieve closest vectors
        - Relies on embeddings of vectors of both queries and documents to identify and rank relevant document for a given query
        - Enables the retrieval system to understand and match based on the contextual similarities between queries and documents.
    2) Rerank –
        - Takes an initially retrieved set of items, images, search results, docs and reorders them to improve the relevance or quality of results
        - Often done after an initial ranking has been performed by fast basic retrieval method



- Assigns a relevance score to (query, response) pairs from initial search results
- High relevance score pairs are more likely to be correct
- Implemented through a trained LLM

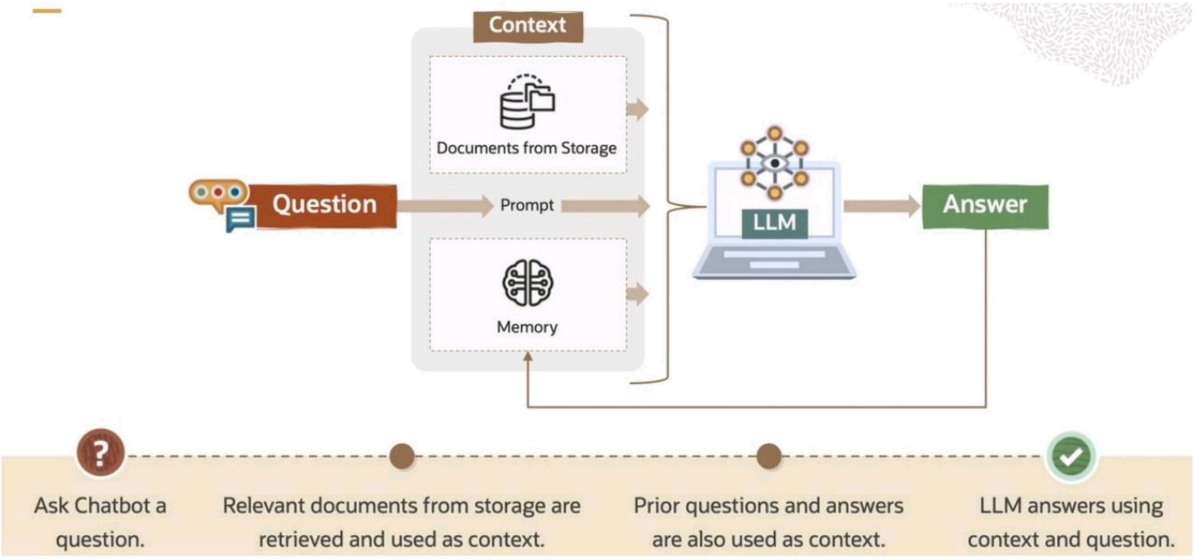
Hybrid search



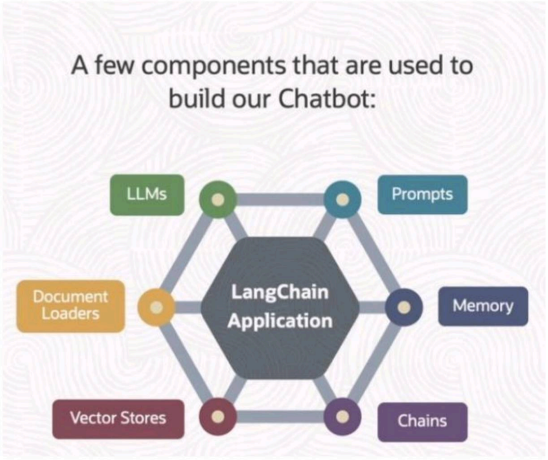
- Sparse (keyword) + Dense
- Specificity of keyword + Relevance of semantic
- Keyword search quickly narrow down the pool of relevant documents from a large corpus and then apply vector search within this subset to identify the most semantically relevant doc based on embedding
- Part of input is processed by dense where model generates dense vector with continuous values. D.E. captures semantic meaning in high dimensional space.
- Another part of input is processed by sparse model which creates sparse vector representation which are typically high but with many 0 values.
- Sparse vector represent presence of frequency of certain word or features
- Vectors by both are then fed into hybrid component which is a vector db service supporting hybrid search
- Alpha- parameter which control dense or sparse level in index.
- Before indexing, normalisation is done for prep of search operation typically ensuring that vector have unit length and standardise vector length making comparison possible.
- Normal vectors are stored in hybrid index- a data structure capable of handling both dense and sparse vectors.
- Hybrid Index enable efficient retrieval operations having strengths of both.

Building LLM Applications

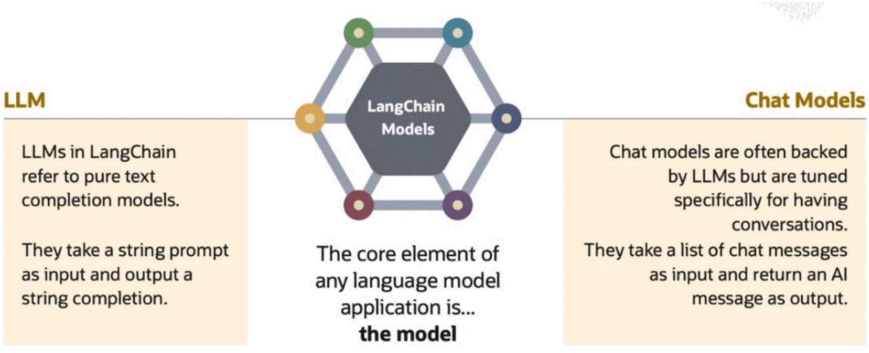
- OCI gen AI service allows us to use pretrained models or create and host fine-tuned customed models.
- LangChain provides a wrapper class for using OCI Gen AI service as an LLM in LangChain applications
- Chat bot architecture-



LangChain Components



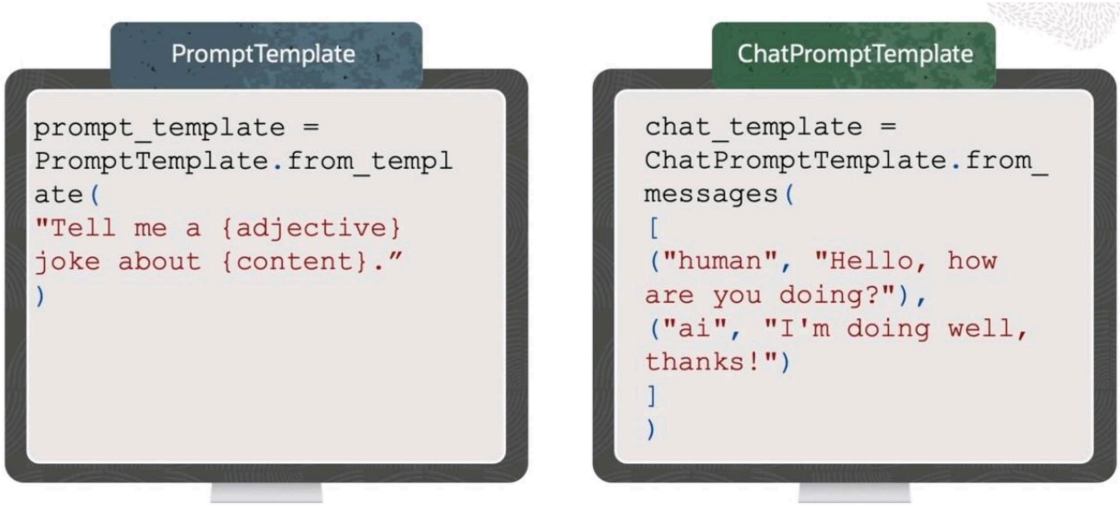
- A framework for developing applications powered by language models.
- Offers a multitude of components that help use build LLM-powered applications.

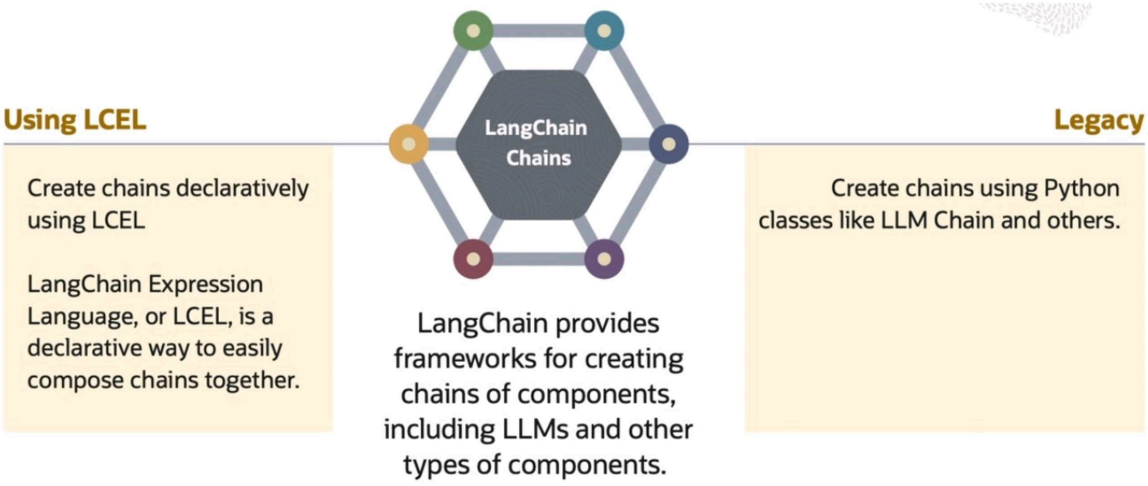


Prompt Templates

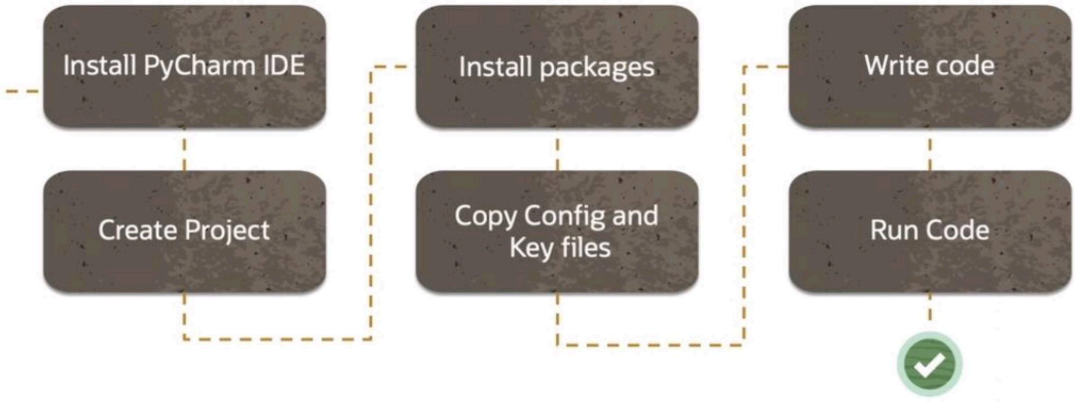
- Predefined recipes for generating prompts for language models.
- Lang model expect the prompt to either be a string or else a list of chat messages.
- String prompt- The template supports any number of variables, including no variables

- Chat prompt- a list of chat messages where each of them is associated with content and an additional parameter called role.



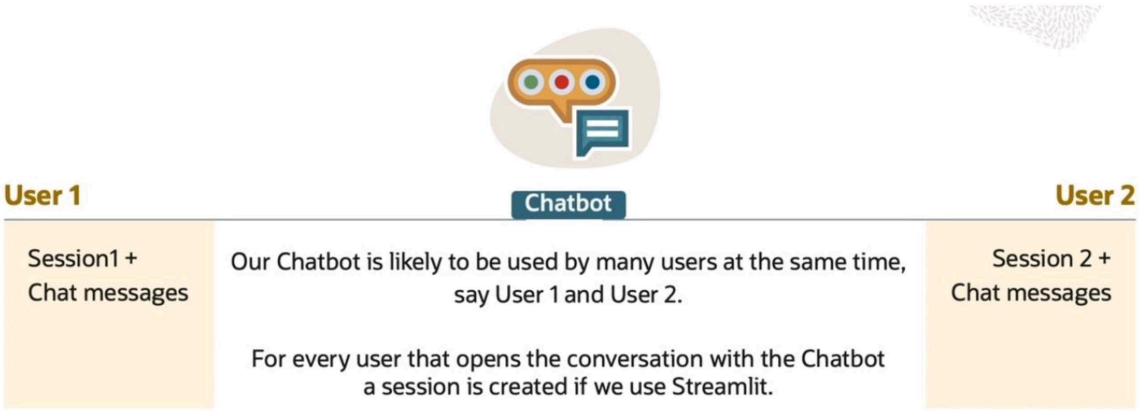


Setting up in Python



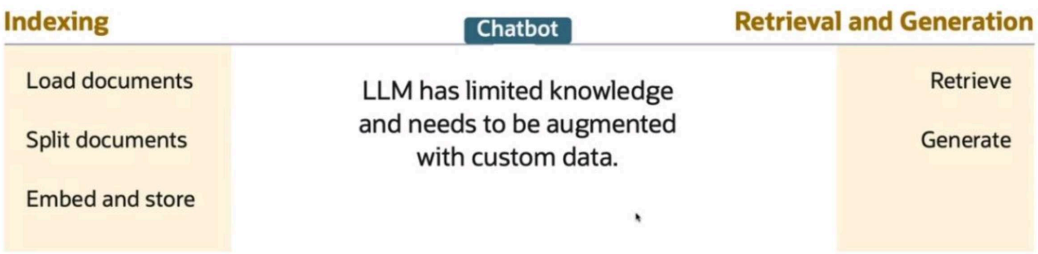
LangChain Memory

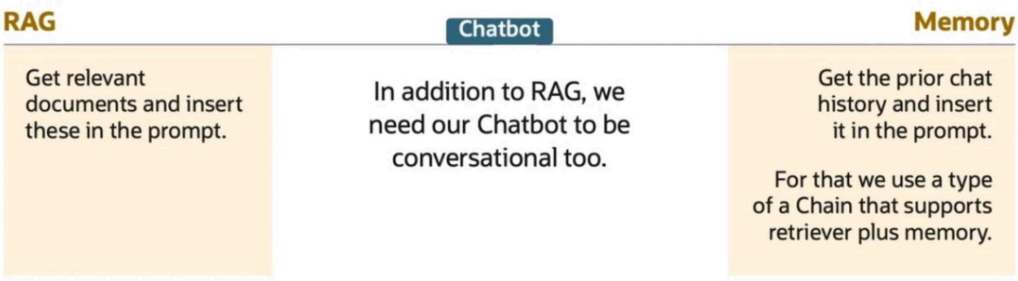
- Ability to store information from past interactions is memory.
- Chain interacts with memory twice in a run
- After user input and before chain execution: Reads from memory
- After core logic but before output write to memory
- LangChain offer various types of memory
- DSA built on top of chat messages decide what is returned from the memory, e.g., memory might return a succinct summary of past k messages.



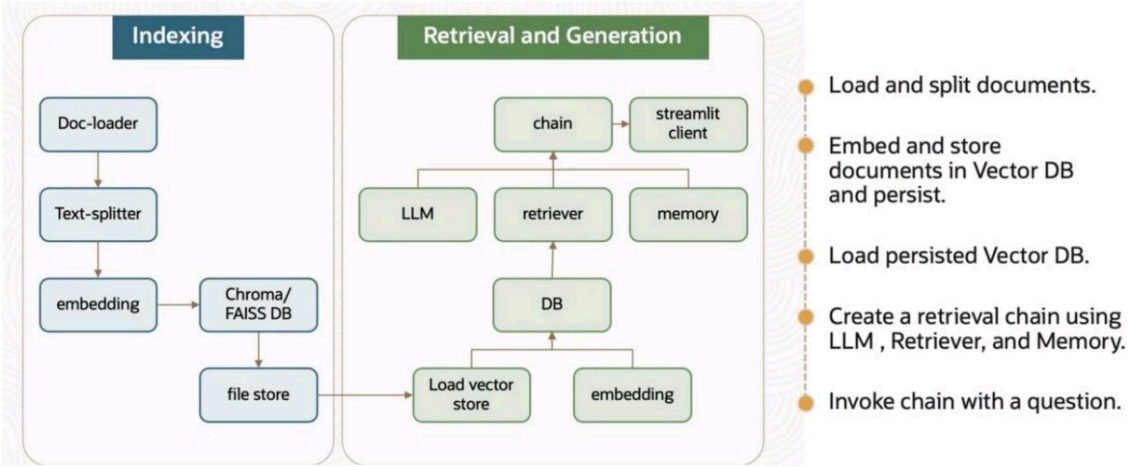
RAG with LangChain

- Fetching the custom information and inserting it into the model prompt is known as RAG
- For building AI applications that can reason about private data, the custom data needs to given to the model





Chatbot Architecture



Deployment