LLM  
  
**what is LLM?**  
  
GPT is a large language model or an LLM that can generate human-like texts.  
\*GPT or Generative Pre-trained Transformer.  
  
LLM is an instance of something else called a foundation model.  
  
Foundation Model: Pre-trained on large amounts of unlabeled and self-supervised data; meaning the model learns from patterns in the data in a way that produces generalized and adaptable output.  
  
LLM are instances of foundation models applied specifically to text and text-like things(like code).

Large language models are trained on large datasets of text, such as books, articles, and conversations.  
  
Large means these can be trained on gigabytes in size and trained on enormous amounts of text data, petabytes of data.   
  
Ex: Document of 1GB in size; stores 178 million words  
1 PB = 1 million GB  
  
LLMs are also among the biggest models when it comes to parameter count.  
  
A parameter is a value that the model can change independently as it learns, and the more parameters a model has, the more complex it can be.  
  
Ex: GPT-3, pre-trained on a corpus of actually 45 terabytes of data, and it uses 175 billion ML parameters.  
  
How do they work:  
  
**LLM = DATA + ARCHITECTURE + TRAINING**  
  
Data: text data as discussed above  
  
Architecture: This is a neural network and for GPT that is a transformer.  
  
And the transformer architecture enables the model to handle sequences of data like sentences or lines of code.  
  
Transformers are designed to understand the context of each word in a sentence by considering it in relation to every other word.  
  
This allows the model to build a comprehensive understanding of the sentence structure and meaning of the words within it.  
  
**TRAINING:**  
  
And then this architecture is trained on all of this large amount of data.  
  
Now, during training, the model learns to predict the next word in a sentence.  
  
Example: “the sky is..” it starts off with a random guess, “the sky is bug.”   
  
But with each iteration, the model adjusts its internal parameters to reduce the difference between its predictions and actual outcomes.  
  
The model keeps doing this gradually improving its word predictions until it can reliably generate coherent sentences like forget about “bug”, it can figure out its “blue”.  
  
Now, the model can be fine-tuned on a smaller, more specific dataset.  
  
Here the model refines its understanding to be able to perform this specific task more accurately.  
  
Fine-tuning is what allows a general language model to become an expert at a specific task.  
  
OK, so how does this all fit into number 3, business applications?

**Business Applications:**Example 1: Well, for customer service applications, businesses can use LLMs to create intelligent chatbots that can handle a variety of customer queries, freeing up human agents for more complex issues.  
  
Example 2: Another good field, is content creation. That can benefit from LLMs which can help generate articles, emails, social media posts, and even YouTube video scripts.   
  
Example 3: LLMs can even contribute to software development, and they can do that by helping to generate and review code.  
  
As large language models continue to evolve, we’re bound to discover more innovative applications.

**What are transformers?**

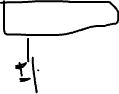
Specifically, I used a GPT-3, or a generative pre-trained transformer model. 3 means third generation.  
  
GPT-3 is an auto-regressive model that produces text that looks like it was written by a human.  
  
GPT-3 can write poetry, craft emails and evidently come up with its own jokes.  
  
GPT-3 is just one example of a transformer.

Something that transforms from one sequence into another.   
  
A language translation is a great example.  
  
Ex: why did the banana cross the road translate into French..  
  
Transformer consists of two parts: Encoder and decoder  
  
Encoder works on the input sequence  
  
Decoder Operates on the target output sequence.  
  
Transformers work is through sequence-to-sequence learning, where transformer takes sequence of tokens; like words in a sentence and predicts the next word in the output sequence.  
  
It does this through iterating through encoder layers, so the encoder generates encoding that define which part of the input sequence are relevant to each other and then passes these encodings to the next encoder layer.  
  
The decoder takes all of these encodings and uses their derived context to generate the output sequence.  
  
Transformers are a form of semi-supervised learning  
  
By semi-supervised, we mean that they are pre-trained in an unsupervised manner with a large, unlabeled data set, and then they’re fine-tuned through supervised training to get then to perform better.  
  
what makes transformers different than neural networks and RNN( which takes data in a sequential order) whereas transformers do not process the data in a order because it uses a mechanism called attention mechanism.  
  
This mechanism provides context around items in the input sequence, so rather than starting our translation with the word “why” because it’s at the start of the sentence(why did the banana cross the road translate into French). The transformer attempts to identify the context that bring meaning in each word in the sequence.  
  
This attention mechanism that gives transformers a huge leg up over other algorithms like RNN which must run in sequence.  
  
Transformers run multiple sequences in parallel and this vastly speeds up training times.  
  
Input  
  
Encoder  
  
Decoder  
  
Output  
  
  
  
Transformers examples:  
  
Document summaries-summarizes, writing blog posts  
Translation  
playing chess  
performing image processing that even rivals the capabilities of convolutional neural networks  
  
Transformer is a powerful deep learning model because of the attention mechanism which can be parallelized and it's getting over time.



References:  
  
IBM Technology YouTube Channel

Transformers Indepth Architecture by Krish Naik:  
  
**Attention Is All You Need – article  
  
Jay Alammar – The Illustrated Transformer  
  
Ex: converting one sentence into another language  
  
Input 🡪 Transformer ->Output  
  
  
In Transformer we have 🡪 Encoder -> Decoder**



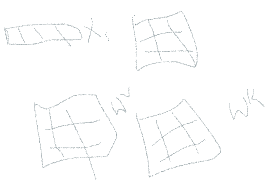
**The encoders that are present : 6 and 6 decoders ( According to attention all you need; 6 encoders are giving good results). --🡪consider this as Hyperparameter tuning step  
  
Insider the Encoder, we’ve : Feed Forward Neural Network and Self Attention  
Encoder**



Feed Forward  
|  
Self-Attention

I/p = Je suis   
  
First, the input is converted into some dimensions using an embedding technique like word2vec and so on..  
  
Here the input converted into vectors of size 512 by using word2vec according to paper.  
  
Then its passed to the self-attention layer like as a 512 vector dimensions. In RNN we give one at a time but where as here we give it parallely.

How the self-attention model works?  
  
Example: The animal didn’t cross the street because it was too tired.  
  
It means 🡪 street or animal🡪 a human can understand easily but what about machine?

Self-attention in detail:  
  
Input Thinking  
  
Now we’ll create 3 weights(wq,wk,Wv) and these weights value will be randomly initialized based on back propagation.   
  
Input Thinking  
  
word2vec Embedding X1

(Converted into 512 dimension)--  
  
  
Vector X1 \* WQ= q1 64 dimension …Queries  
vector x1 \* WK= k1 ..keys

Vector x1\* wv= v1….values

Like ANN calculation ( weight \* bias \* loss) -🡪read this later  
  
what are the query, key, values vector?  
  
For thinking, we will calculate q1\*k1 =112also, q1\*k2 =96-🡪score  
  
Step 1: creating queries, keys, values  
Step 2: Calculate Score  
Step 3: Divide by 8 ( as we’ve 64 dimensions; whatever q1\*k1 we’re getting, we’ll divide by 8 root of dk(queries)….  
Step 4: Softmax activation function (q1\*k1/8 + q1\* k2/8=1..for making it to one)  
Step 5: Softmax \* Value = we get two different vectors= v1+v2= z1  
  
X



Thinking \* Wq ( size is initialized randomly) = Q (64D)  
Machine  
Go  
 \* Wk = K

\*Wv=V

In the back propagation, these weights will get updated. Encode to decoder it goes Loss function->Optimizer🡪updating weights

If we observe it clearly, we’re passing queries, keys and values…(wq,wk,wv)..as a single head attention…where **its** giving importance to animals and not to tired  
  
So instead we’ll use multi head attention:

Having different attention heads to give importance to each word  
  
We’re gonna use 8 heads according to paper; we’ll be getting 8 z values(z0 to z7)

Then combine all z values (z0+z1+z2+z3+z4+z5+z6+z7); before passing it to feed forward; we will initialize another weight and multiply with sum of z values..then we will get final z value which will be passed to feed forward neural network.

Now after using multi -head; we can see its giving importance to both animal and tired too for “it” word.  
  
Before giving input to the encoder; embeddings x1 \* positional encoding(words are out of order) t1 = embedding with time signal t1 ( which tells if two words are near or not) and also to get the ordering of the words…  
  
After self-attention; apply normalization which does two functions: because sometimes self attentions might not work; so we give it to add & normalize..like residual connection: where if two hidden layers not working properly they give it to layer where it works…where we can skip some layers where its not important.  
  
Add & Normalize {output Z + Input X}  
  
How Decoder works:  
  
output of the encoder will be given to decoder  
  
Decoder has self-attention; add& Normalize; Encoder-decoder attention  
  
Encoder-decoder attention is similar to self -attention..the output of the encoder is given to encoder-decoder attention  
  
Then what we give to self -attention in the decoder end: after encoder output is given dencoder-decoder attenmtion; whatever the output we get is given to Self-attention to   
  
output of decoder is given to linear and then to softmax; Thinking in Hindi is Sochana that we got  
  
Then that sochana is given to the self-attention model; as its goes up then it combines with encoder output  
  
In decoder we give it one by one where in encoder we give it parallely.  
  
Until we get End of Statement.  
  
**Vector Database:**  
  
Vector Database, is a type of database that is used in various machine learning use cases. They are specialized for the storage and retrieval of vector data.  
  
Data: Traditional DB: Structured and Unstructured; Vector DB: Vector data  
  
Search: Traditional DB: Predefined criteria for search; Based on the context or vector distance  
  
Use Cases: Image and Video Processing, NLP, LLM, Recommendation Systems 🡪 in all of these cases, we are talking about embedding to be stored and retrieved.  
  
embeddings are the numerical presentation of the data such as text, video, image, audio.  
  
because for computer it is not understandable so we need to convert it into numerics  
  
we use embedding functions to generate these numeric representations like word2vec, Glove, BERT.  
  
  
  
Ex: The dog is running on the hill 🡪 Embedding function/model -🡪 0,45|9,34|….  
  
Databases:  
  
Chroma: Open source, deploy on cloud or on-premise, optimized for use cases of audio data, support integration with various API and frameworks such as PyTorch and TF  
  
Pinecone: Cloud based, suitable for many ML use cases, Easy integration with cloud platforms, Elasticsearch, GPT models, etc  
  
milvus: open source, support integration with frameworks as PyTorch and TF, Fast retrieval for similarity search in large vectors.  
  
  
  
**LANGCHAIN TUTORIAL  
  
VECTOR DATABASE - CODEBASICS**  
  
EX: Employees at Apple; Calories in Apple  
How does Google determine whether it’s a company or a fruit?  
It does it by using SEMANTIC SEARCH  
  
What is Semantic Search?  
Semantic Search is a set of search engine capabilities, which includes understanding words from the searcher’s intent and their search context.  
In simple, the process search engines use to try to understand the intent and contextual meaning of your search query to give you results that match what you had in mind.  
For example: if you search for “wedding dresses”, the words related to that might include “wedding”, ”cake”, ”bride”, and “dream”  
  
To do the semantic search, it uses a concept called “Embedding”.  
  
A word embedding or sentence embedding is nothing but a numerical representation of text.  
  
How does Embedding work?  
  
Apple (context: Revenue of Apple)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Context: Revenue of Apple | Context: Calories in Apple | Context: Nutrition in Orange | Context:  Employees in Samsung |
| related\_to\_phones | 1 | 0 | 0 | 1 |
| is\_location | 0 | 0 | 0 | 0 |
| has\_stock | 1 | 0 | 0 | 1 |
| revenue | 82 | 0 | 0 | 76 |
| is\_fruit | 0 | 1 | 1 | 0 |
| calories | 0 | 95 | 50 | 0 |

when you have the embeddings for different words, looking at the embeddings we can say that the second Apple and the word orange are similar. The second and third vector is similar.  
Similarly, the first and fourth vectors are similar.  
  
Word2vec is a technique to represent a word in a numeric representation. In reality, complex statistical techniques are used to auto-derive the features of text

Word embedding techniques  
  
CBOW, Skip gram: Word2vec, GloVe, fastText

Based on transformer architecture: BERT, GPT

Based on LSTM: ELMo

when we do embeddings, we get millions of embedding vectors. We have to store them.   
  
The first thing that strikes our mind is storing them in a relational database(i.e. MySQL)

First we use articles related to apple company and fruits. At first, we generate Embeddings(e.g. using openai API) -🡪[0.45,0.34,1.2….0.08]-🡪 and then we will save that info into SQL database.  
  
When we do Google Search: Calories in Apple [we do embedding for the query also]

And then we compare the query embedding with the stored embedding and to retrieve the relevant documents using cosine similarity.  
  
When we have millions of records, that is when things get interesting  
  
[query vector] 🡪cosine similarity🡪 [stored vectors] [here we go linear search to combine one after another]

Computational will be high; and delay will be more  
  
So we use index database instead of a traditional storage

[millions of stored vectors]🡪Hashing Function🡪[creates buckets of similar vectors]  
  
if we do a search query now, using hashing function we will compare the query vector with buckets, and within buckets we do a linear search. This method is called LSH (locality sensitive hashing)  
  
Benefits of Vector Database  
  
Fast Search  
Optimized Storage  
  
  
Technical Architecture of LangChain:  
  
1. Convert text into vectors and store it in a vector database

2. convert query into vector and compare it with the stored

3. CSVLoader of langchain, Hugging Face for Embeddings, FAISS for vector database, retrievalQA of langchain, google palm for LLM  
  
Google Palm – LLM  
Google PaLM -Free;easy to use  
Meta-LLaMA-Free  
OpenAI-GPT4-Paid