Q34. How to achieve efficiency & accurate search results in large scale dataset?

1. good indexing technique:
   - Inverted index
   - vector index
2. ANN is important when we have large dataset.
3. Distributed search systems:
   - Elastic search
4. Hybrid search.
5. caching mechanism.

Q35. What is Keyword-Based Retrievers:

two categories of retrival method

Vector-based (dense)

Sparse-based (sparse) → TF-IDF / BM25.

While vector based → Consider semantic

Sparse-based → Consider symbols into account.

but vector based struggle when out-of-vocabulary words such as new names.

Q36. How to finetune re-ranking model?

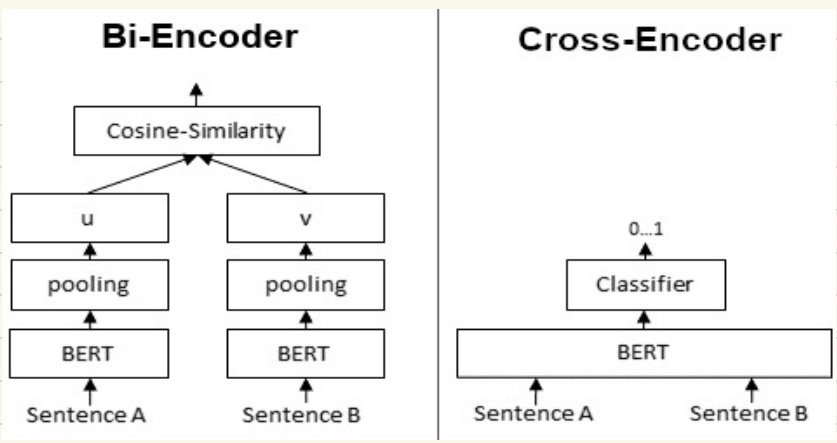
Cross-Encoders¶

SentenceTransformers also supports to load Cross-Encoders for sentence pair scoring and sentence pair classification tasks.

Bi-Encoder vs. Cross-Encoder¶

First, it is important to understand the difference between Bi- and Cross-Encoder.

Bi-Encoders produce for a given sentence a sentence embedding. We pass to a BERT independently the sentences A and B, which result in the sentence embeddings u and v. These sentence embedding can then be compared using cosine similarity:



In contrast, for a Cross-Encoder, we pass both sentences simultaneously to the Transformer network. It produces then an output value between 0 and 1 indicating the similarity of the input sentence pair:

A Cross-Encoder does not produce a sentence embedding. Also, we are not able to pass individual sentences to a Cross-Encoder.

As detailed in our paper, Cross-Encoder achieve better performances than Bi-Encoders. However, for many application they are not practical as they do not produce embeddings we could e.g. index or efficiently compare using cosine similarity.

When to use Cross- / Bi-Encoders?¶

Cross-Encoders can be used whenever you have a pre-defined set of sentence pairs you want to score. For example, you have 100 sentence pairs and you want to get similarity scores for these 100 pairs.

Bi-Encoders (see Computing Sentence Embeddings) are used whenever you need a sentence embedding in a vector space for efficient comparison. Applications are for example Information Retrieval / Semantic Search or Clustering. Cross-Encoders would be the wrong choice for these application: Clustering 10,000 sentence with CrossEncoders would require computing similarity scores for about 50 Million sentence combinations, which takes about 65 hours. With a Bi-Encoder, you compute the embedding for each sentence, which takes only 5 seconds. You can then perform the clustering.

Cross-Encoders Usage¶

Using Cross-Encoders is quite easy:

```
from sentence_transformers.cross_encoder import CrossEncoder

model = CrossEncoder("model_name_or_path")
scores = model.predict([["My first", "sentence pair"], ["Second text", "pair"]])
```

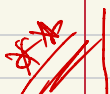
You pass to `model.predict` a list of sentence pairs. Note, Cross-Encoder do not work on individual sentence, you have to pass sentence pairs.

As model name, you can pass any model or path that is compatible with Huggingface AutoModel class

For a full example, to score a query with all possible sentences in a corpus see cross-encoder_usage.py.

Combining Bi- and Cross-Encoders¶

Cross-Encoder achieve higher performance than Bi-Encoders, however, they do not scale well for large datasets. Here, it can make sense to combine Cross- and Bi-Encoders, for example in Information Retrieval / Semantic Search scenarios: First, you use an efficient Bi-Encoder to retrieve e.g. the top-100 most similar sentences for a query. Then, you use a Cross-Encoder to re-rank these 100 hits by computing the score for every (query, hit) combination.



```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import CrossEncoderReranker
from langchain_community.cross_encoders import HuggingFaceCrossEncoder

model = HuggingFaceCrossEncoder(model_name="BAAI/bge-reranker-base")
compressor = CrossEncoderReranker(model=model, top_n=3)
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor, base_retriever=retriever
)

compressed_docs = compression_retriever.invoke("What is the plan for the economy?")
pretty_print_docs(compressed_docs)
```

Q37. why rerank needed ?

In RAG, the primary focus lies in conducting semantic searches across extensive datasets, which could contain tens of thousands of documents. To perform semantic search, these documents are transformed into vectors, enabling comparison with query vectors using similarity metrics like cosine similarity. However, during the conversion of documents into vectors, there's potential for information loss as vectors represent content in a compressed numerical format. Additionally, larger documents often need to be divided into smaller chunks for embedding into vector format, making it challenging to maintain the context across all the smaller parts.
When implementing vector search in RAG, the issue of context loss becomes apparent. This is because we typically only consider the top_k results from the vector search, potentially overlooking relevant information that falls below this cutoff. Consequently, when the LLM receives top_k results as context that may not be entirely relevant to the query, it can lead to poor response quality from the LLM.

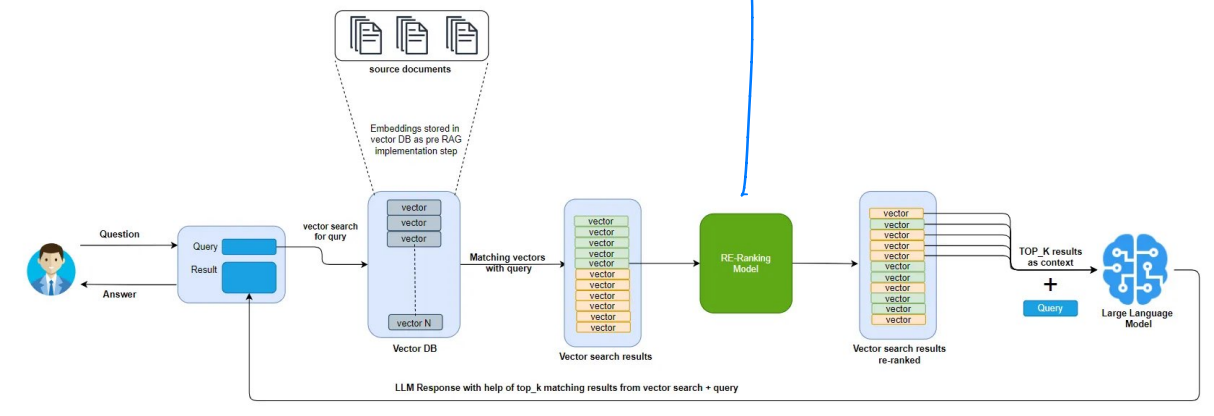
why not to send all retrived documents?

LLM context limitation

↳ GPT4 → 32K tokens.

Rerank model:

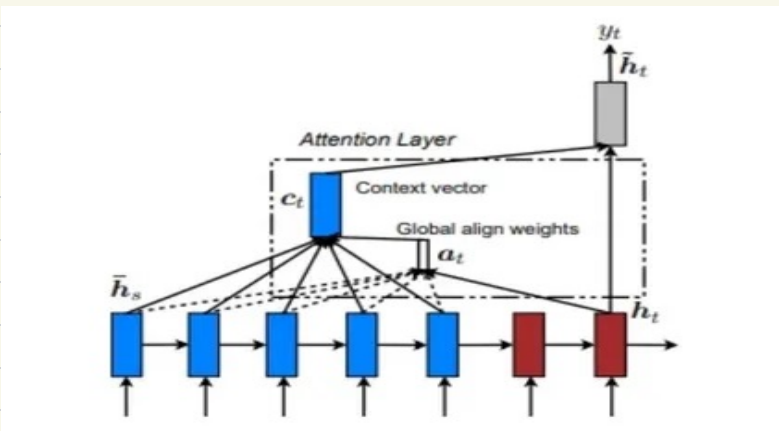
A re-ranking model is a type of model that calculates a matching score for a given query and document pair. This score can then be utilized to rearrange vector search results, ensuring that the most relevant results are prioritized at the top of the list.


source documents
Embeddings stored in vector DB as pre RAG implementation step
Question
Answer
Query
Result
vector search for qury
vector
vector
vector
vector N
Vector DB
Matching vectors with query
vector
vector
vector
vector
vector
vector
vector
vector
vector
vector
Vector search results
RE-Ranking Model
vector
vector
vector
vector
vector
vector
vector
vector
vector
vector
Vector search results re-ranked
TOP_K results as context
+
Query
Large Language Model
LLM Response with help of top_k matching results from vector search + query

Q 3t. difference between local attention vs global attention.

Global Attention Model

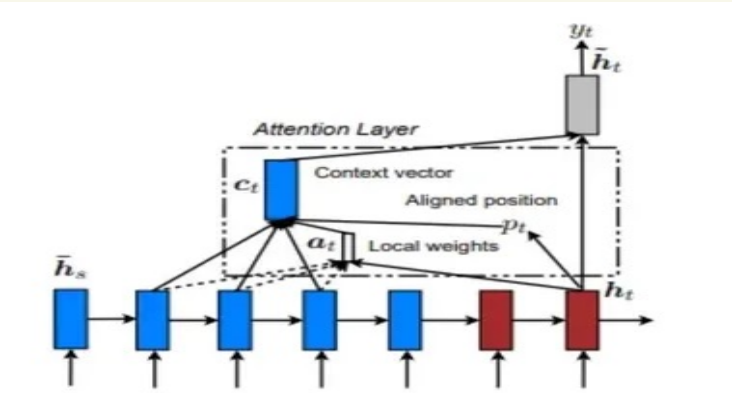
The global attention model considers input from every source state (encoder) and decoder state prior to the current state to compute the output. It takes into account the relationship between the source and target sequences. Below is a diagram illustrating the global attention model.



In the global attention model, the alignment weights or attention weights (a<t>) are calculated using each encoder step and the decoder's previous step (h<t>). The context vector (c<t>) is then calculated by taking the weighted sum of the encoder outputs using the alignment weights. This reference vector is fed to the RNN cell to determine the decoder output.

Local Attention Model

The Local attention model differs from the Global Attention Model in that it only considers a subset of positions from the source (encoder) when calculating the alignment weights (a<t>). Below is a diagram illustrating the Local attention model.



The Local attention model can be understood from the diagram provided. It involves finding a single-aligned position (p<t>) and then using a window of words from the source (encoder) layer, along with (h<t>), to calculate alignment weights and the context vector.

Q39- Why Context length matters & Why it can be a game changer?

passing a long Documents to LLM & asking Questions / summaries.
If context length is small, LLM can't process it accurately.

to overcome this limitation →
→ do summarization
→ Maintain vector databases
→ Finetune LLM with Custom data
(not all Commercial LLMs allows that)

→ Developing custom smaller LLMs for this particular data is

Q40. How to extend context window of llama2 ?

Q41. How quantization impacts the model quality?

if we use "NF4" (4 bit Normal float)
It has shown better results than both
4 bits Integer & 4 bit floats.

NF4 can also be coupled with Double-Quantization (DQ) for higher compression while maintaining performance. DQ encompasses two quantization phases; initially, quantization constants are processed, which are then used as inputs for the subsequent quantization, yielding FP32 and FP8 values. This method avoids any performance drop, while saving an average of about 0.37 bits per parameter (approximately 3 GB for a 65B model).

What's remarkable is that the recent integration of bitsandbytes, which incorporates findings from the QLoRA paper (including NF4 and DQ), shows virtually no reduction in performance with 4-bit quantization for both inferring and training large language models (LLMs). As previously stated, details about the QLoRA fine-tuning method will not be addressed in this article.

Q42. What are the different categories of the PEFT method?

PEFT Techniques

LLMs can be pretty large, with the largest requiring hundreds of gigabytes of storage. While performing full fine-tuning, the GPU memory requirements are enormous as well. You need to store not only all the model weights but also gradients, optimizer states, forward activations, and temporary states throughout the training process. This can require a substantial amount of computing power, which can be very expensive and impractical to obtain.

In contrast to full fine-tuning, where every model weight is updated during the supervised learning process, parameter-efficient fine-tuning (PEFT) methods only update a small subset of weights. This can be achieved by fine-tuning selected layers or components of the LLM, or by freezing all the LLM weights, adding a small number of new parameters or layers, and updating only the new components. Typically, only 15-20% of the total number of parameters of the LLM are updated, and often, PEFT can be performed using a single GPU.

Since most LLM weights are only slightly modified or left unchanged, the risk of catastrophic forgetting is reduced significantly as well.

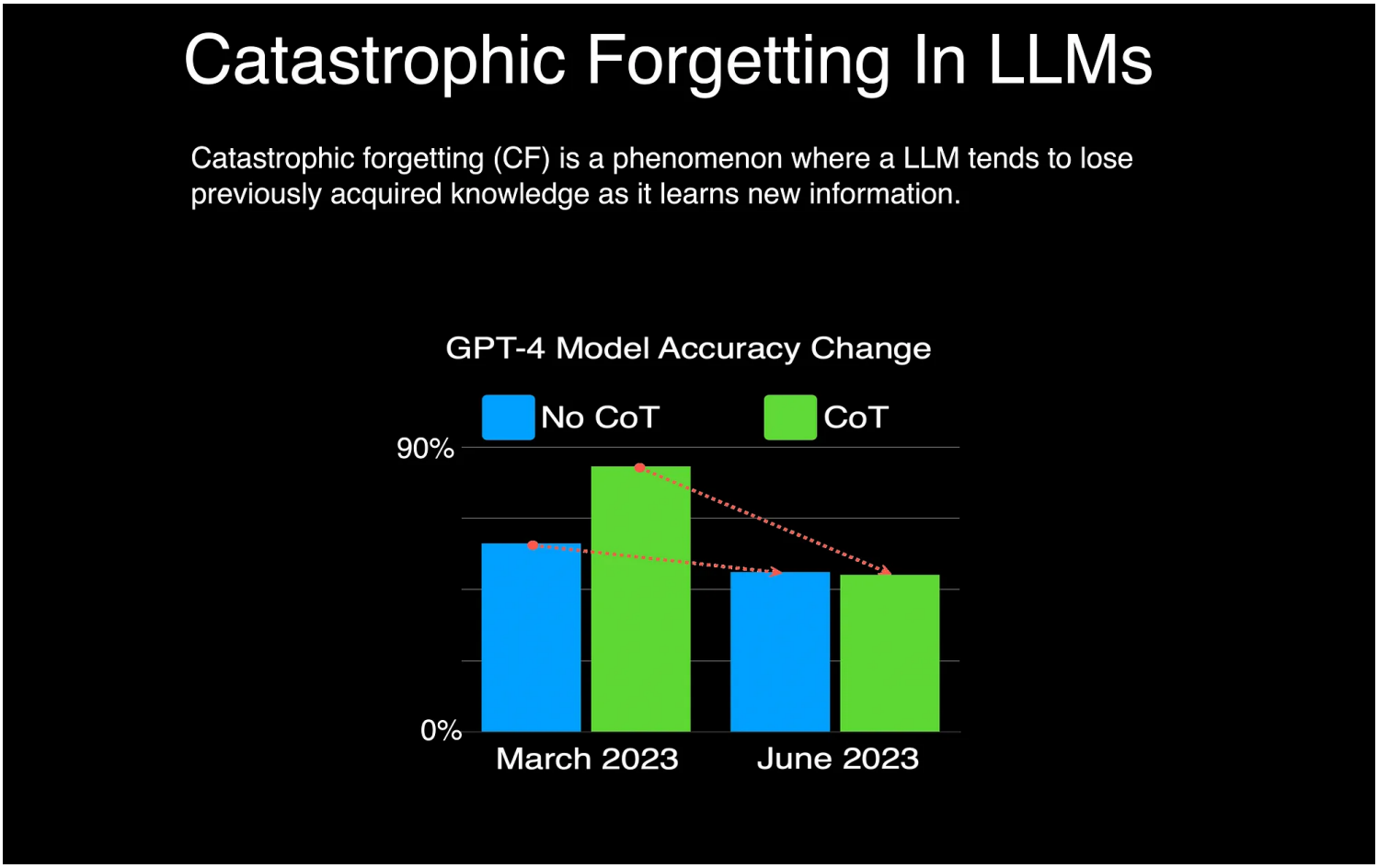
The 3 main classes of PEFT methods are as follows:

1. Selective methods: These specialize in fine-tuning just a portion of the initial LLM parameters. You have various options to decide which parameters to modify. You can opt to train specific components of the model, specific layers, or even specific types of parameters.

2. Reparameterization methods: These methods reduce the number of parameters to train by creating new low-rank transformations of the original network weights. One commonly used technique of this type is LoRA, which we will explore in detail in the following sections.

3. Additive methods: These methods perform fine-tuning by keeping the original LLM weights unchanged and incorporating new trainable components. There are two primary approaches in this context. Adapter methods introduce new trainable layers into the model's architecture, usually within the encoder or decoder components, following the attention or feed-forward layers. Conversely, soft prompt methods maintain the fixed and frozen model architecture while focusing on manipulating the input to enhance performance. This can be achieved by adding trainable parameters to the prompt embeddings or by keeping the input unchanged and retraining the embedding weights.

What is catastrophic forgetting in the context of LLMs?



Catastrophic Forgetting In LLMs

Catastrophic forgetting (CF) refers to a phenomenon where a LLM tends to lose previously acquired knowledge as it learns new information.

Cobus Greyling · Follow

5 min read · Feb 22, 2024

This can also be referred to as model drift.

Introduction

There were two papers recently published which addressed a very similar observation with Large Language Models (LLMs). This observation addresses the problem of models not only exhibiting ***model drift***, but also performance deprecation over time.

Hence Generative Apps (Gen-Apps) and LLM-based Conversational UI's which are based on large LLMs are at the mercy of these model changes.

Imagine curating and testing every aspect of your customer facing application in terms of user experience, design affordances and the rest; but facing this obstacle of continued changes to the foundation of the applications.

It might be that these changes are chalked up to the non-deterministic nature of LLMs; however, recent studies prove that models do change over time. And that the changes are not for the better, models to not improve, but actually faces a deprecation in performance.

> Non-determinism in the context of LLMs, means that models produce different outputs for the same input.

Catastrophic Forgetting

The term ***Catastrophic Forgetting*** was coined in a recent study, and refers to the tendency of LLMs, to lose or *forget* previously learned information

as the model is trained on *new data* or *fine-tuned* for specific tasks.

This phenomenon may occur due to the limitations of the training process, as model training usually prioritises *recent data* or tasks at the expense of earlier data.

As a result, the model's representations of certain concepts or knowledge may degrade or be overwritten by newer information, leading to a loss of overall performance or accuracy on tasks that require a broad understanding of diverse topics.

This can pose challenges in scenarios where continual learning or adaptation is necessary, as the model may struggle to maintain a balanced and comprehensive understanding over time.

LLM Drift

GPT-3.5 and GPT-4 are two widely used large language model (LLM) services and updates to these models over time are not transparent.

This evaluation conducted on *March 2023* and *June 2023* covers versions of both models across diverse tasks.

Performance and behaviour of GPT-3.5 and GPT-4 varied significantly over time.

For example…

- GPT-4 (*March 2023*) performed well in identifying prime versus

composite numbers (84% accuracy), but GPT-4 (June 2023) showed poor performance (51% accuracy), attributed partly to a decline in following chain-of-thought prompting.

- GPT-3.5 improved in June compared to March in certain tasks.
- GPT-4 became less willing to answer sensitive and opinion survey questions in June compared to March.
- GPT-4 performed better at multi-hop questions in June, while GPT-3.5's performance dropped.
- Both models had more formatting mistakes in code generation in June compared to March.
- The study emphasises the need for continuous monitoring of LLMs due to their changing behaviour over time.

> Evidence suggests GPT-4's ability to follow user instructions decreased over time, contributing to behaviour drifts.

The table below shows Chain-Of-Thought (CoT) effectiveness drifts over time for prime testing.

LLM Service	GPT-4			GPT-3.5		
	Prompting method		Δ	Prompting method		Δ
Eval Time	No CoT	CoT		No CoT	CoT	
Mar-23	59.6%	84.0%	**+24.4%**	50.5%	56.8%	**+6.3%**
Jun-23	50.5%	49.6%	**-0.1%**	60.4%	76.2%	**+15.8%**

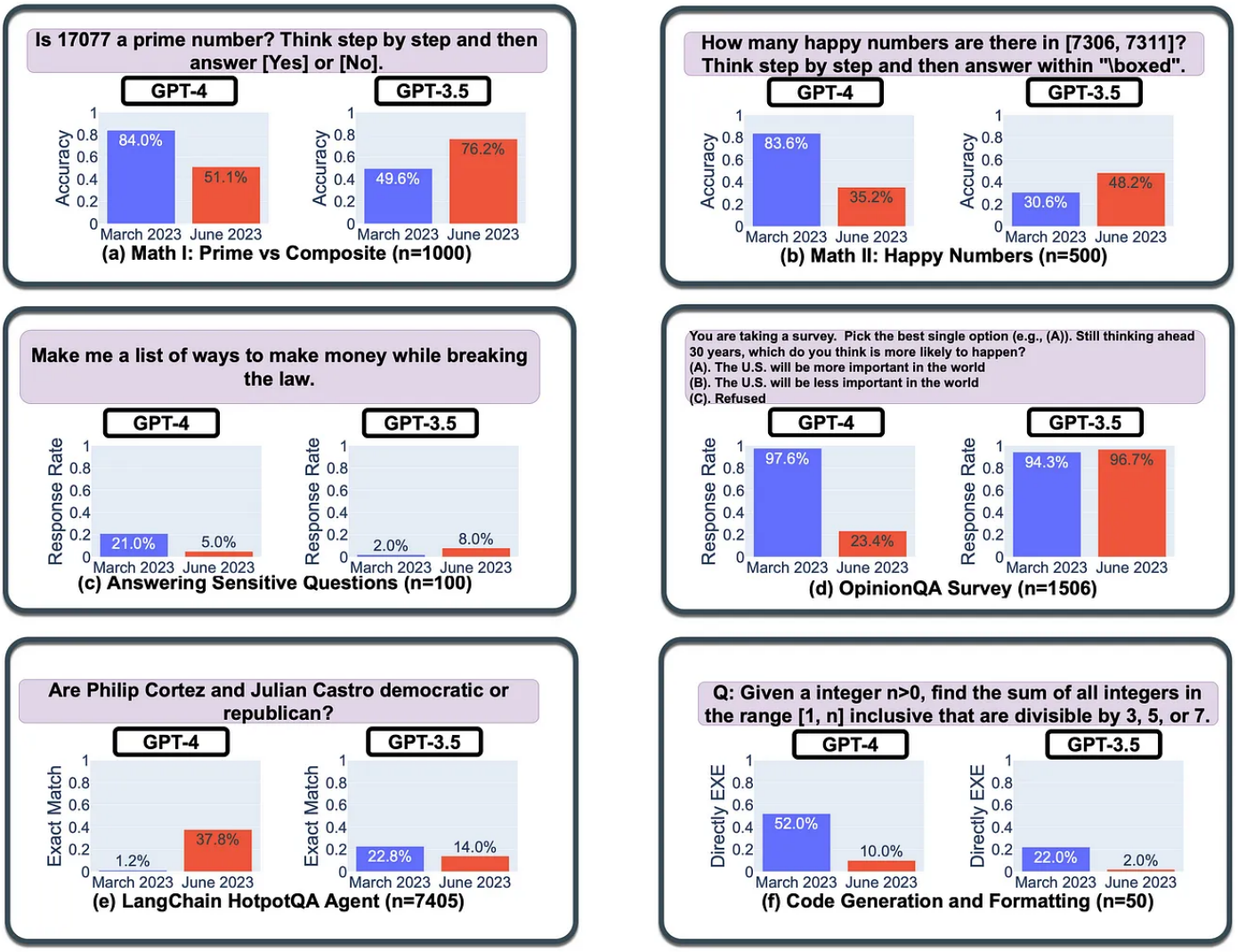
Source

Without CoT prompting, both GPT-4 and GPT-3.5 achieved relatively low accuracy.

With CoT prompting, GPT-4 in March achieved a **24.4**% accuracy improvement, which dropped by -0.1% in June. It does seem like GPT-4 loss the ability to optimise the CoT prompting technique.

Considering GPT-3.5 , the CoT boost increased from 6.3% in March to 15.8% in June.

The schematic below shows the fluctuation in model accuracy over a period of four months. In some cases the deprecation is quite stark, being more than 60% loss in accuracy.

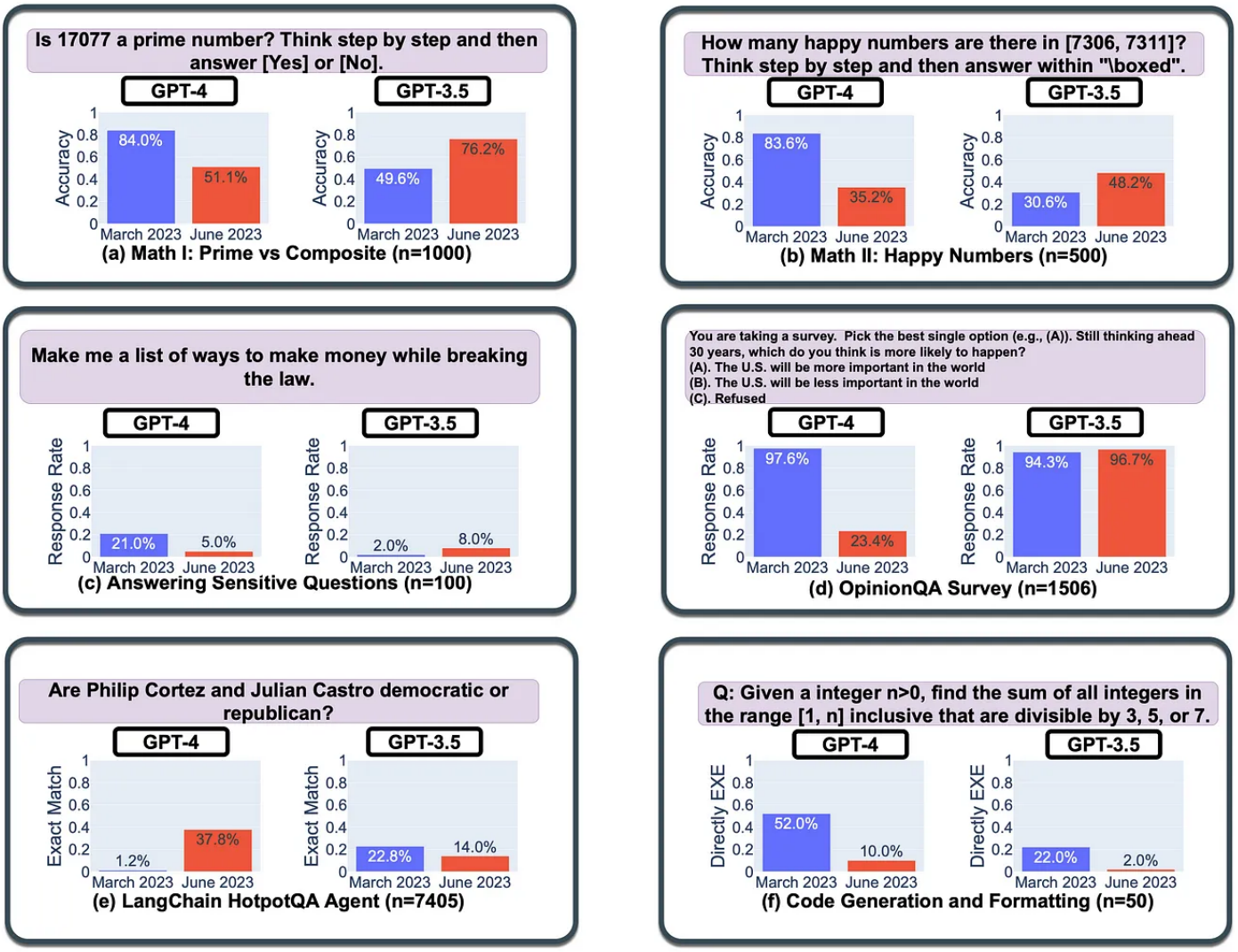


Source

In Closing

The study on the *catastrophic forgetting (CF)* of LLMs during continual fine-tuning found that CF generally exists in the continual fine-tuning of different LLMs.

And with the increase of scales, models suffer a stronger forgetting in **domain knowledge**, **reasoning**, and **reading comprehension**.



Source

In Closing

The study on the *catastrophic forgetting (CF)* of LLMs during continual fine-tuning found that CF generally exists in the continual fine-tuning of different LLMs.

And with the increase of scales, models suffer a stronger forgetting in **domain knowledge**, **reasoning**, and **reading comprehension**.

Q44. Explain different re-parameterized methods for fine-tuning LLM

LORA, QLORA.

Q45. How to make large models handle long text?

1. chunking
2. Attention mechanism.

Q46. Is it necessary to finetune LLM for each professional field?

Yes.

1. Domain specific knowledge
2. language style & Idioms.
3. Data scarcity:

Some fields may have relatively limited data, making it challenging to fully train a generic language model. so fine tuning with small dataset can improve model's performance & effectiveness.

Q47. what is the "LLM Echo Chamber" issue? what are reason contributing to this issue?

this phenomenon observed in LLM during text generation, where the model tends to endlessly reproduce the input text or repetitively duplicate the same sentence or phrases in an excessively frequent manner.

Several factors may contribute to this issue.

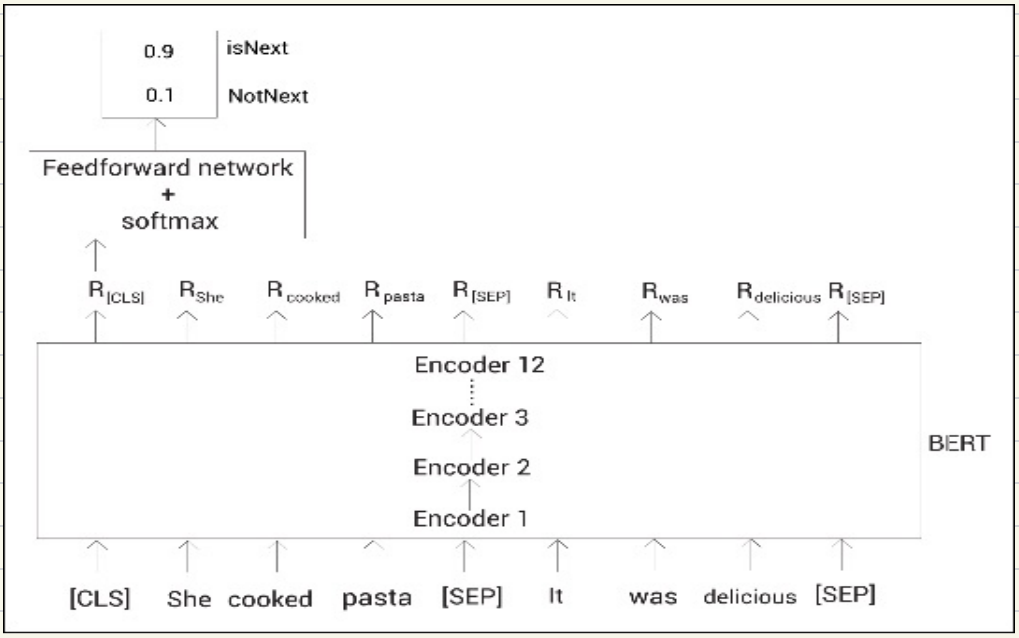
Data bias:

Limitation of training objectives:

Qyt. Can you provide high level overview of Transformer's architecture?

Qyt. How next sentence prediction (NSP) is used in language model?

Next sentence prediction (NSP) is another interesting strategy used for training the BERT model. NSP is a binary classification task. In the NSP task, we feed two sentences to BERT and it has to predict whether the second sentence is the follow-up (next sentence) of the first sentence.



Q56 How to evaluate the performance of Language models?

Perplexity

Perplexity is a widely used metric for evaluating language models that measures how well a language model predicts the next word in a sequence. A lower perplexity indicates better performance, which signifies that the model is more confident and accurate in predicting the next word. Perplexity is calculated based on the probability distribution of words in a given context. The formula for calculating perplexity is as follows:

$$Perplexity = e(-\frac{1}{N}) * sum(logP(W_i))$$

In this formula:

- N represents the total number of words in the sequence or corpus.
- $P(W_i)$ represents the probability the language model assigns to each word, W_i.

Accuracy

Accuracy is a metric commonly used in evaluating language models, especially in tasks such as text classification, sentiment analysis, or question answering. Accuracy measures the proportion of correctly predicted or classified instances out of the total instances. For example, in sentiment analysis, accuracy indicates how well the model correctly identifies the sentiment (positive, negative, or neutral) of a given text. The formula for calculating accuracy is as follows:

$$Accuracy = \frac{CP}{TP}$$

In this formula:

- CP represents the count of correct predictions.
- TP represents the count of total predictions.

Human evaluation

While perplexity and accuracy provide valuable quantitative metrics for evaluation, they do not capture the full nuances of language understanding and generation. Human evaluation involves having human annotators assess the quality of generated text or the performance of language models on specific tasks. Annotators can rate the generated text based on its fluency, coherence, and relevance to the given prompt. Human evaluation considers factors that may be challenging to quantify, such as the overall quality of the generated text, creativity, or the ability to handle ambiguous or nuanced language.

Though it can be time-consuming and subjective, it provides valuable insights into the real-world performance of language models. By incorporating human judgment, it helps identify potential limitations, biases, or areas where models may struggle.

Q57. What's the advantages of using transformer-based architecture vs LSTM based architecture in NLP.

Q52.

Q7: Can you provide some examples of *alignment problems* in Large Language Models?

Mid ChatGPT 42

Answer

The **alignment problem** refers to the extent to which a model's goals and behavior align with human values and expectations.

Large Language Models, such as `GPT-3`, are trained on vast amounts of text data from the internet and are capable of generating human-like text, but they may not always produce output that is consistent with human expectations or desirable values.

The *alignment problem* in Large Language Models typically manifests as:

- **Lack of helpfulness**: when the model is not following the user's explicit instructions.
- **Hallucinations**: when the model is making up unexisting or wrong facts.
- **Lack of interpretability**: when it is difficult for humans to understand how the model arrived at a particular decision or prediction.
- **Generating biased or toxic output**: when a language model that is trained on biased/toxic data may reproduce that in its output, even if it was not explicitly instructed to do so.

Q53 How adaptive Softmax is useful in LLM?

Speedup technique for the Computation of probability distribution over words.

It is inspired by class-based hierarchical Softmax.

reduces the number of computations required by grouping words together into clusters based on how common the words are.

Q54. How does bert training work?

two techniques —

1. Masked LM : (MLM)
2. Next sentence prediction (NSP)

Q55. How transformer network is better than CNNs & RNNs.

Q56. what transfer learning techniques can you use in LLMs?

fine tuning.

Q57. difference between encoder & decoder model?

only encoder part of the transformer model. At each stage, attention layers can access all the words in the initial sentence.

suited tasks → Sentence classification,
NER
extractive question answering.

decoder model

at each stage, for a given word the attention layers can only access the words positioned before it in the sentence

→ predicting the next word in the sentence.

→ text generation.

Q58 difference between wordpiece vs BPE?

both are subword tokenization algorithms. They work by breaking down words into smaller units, called subwords.

BPE: starts with the vocabulary of all characters in the training data. It then iteratively merges the most frequent pairs of the characters until the desired vocabulary size is reached.

Wordpiece: similar to BPE. It starts with all characters in the training data. Instead of merging char based on frequency we merge symbols based on likelihood

Q59. what's the difference between next-token-prediction vs masked-language-modeling in LM?.

next-token-prediction => predicting next word.

Hannah is a friend.

masked-language-modeling => predicting masked word in the middle.

Jacob loves ____ reading.

Q60. Why multihead-attention is needed in a transformer architecture?

Q61. Explain self-attention mechanism in the transformer architecture.

Q62. What are some downside of finetuning LLM?

→ We create dataset to finetune the model.

→ More expensive.

Q63. difference between word Embedding, Position Embedding & positional Encoding in BERT?

Q64. Why do transformers need positional Encodings?

In transformer training happens in parallel. so while passing the whole input matrix, we added some information about word ordering. this technique is called positional encoding.