Q. Why Instruction Tune LLMs?

LLMs are pre-trained using self-supervised learning on a massive corpus of written content. Instruction tuning improves large language models (LLMs) by teaching them to better follow instructions and engage in conversations. Normally, LLMs like Meta's Llama 2, OpenAI's GPT, and others are trained to predict the next word in a sequence based on a large amount of text. This training process helps the models generate fluent sentences but doesn't necessarily make them good at responding to specific requests. For example, if you ask, "How do I bake bread?" a base model might reply with something like "in a home oven," which isn't helpful.

Training these models from scratch to follow instructions is extremely resource-intensive due to their size. Instead, fine-tuning, especially through instruction tuning, is much more efficient. This method involves giving the model examples of user requests and ideal responses. The model then learns to generate useful answers to prompts, like giving step-by-step instructions for baking bread, instead of just predicting the next word.

By fine-tuning the model with instruction-like data, the model becomes more aligned with human needs, making it more reliable and helpful in real-world tasks.

Q. How does instruction tuning work?

Instruction tuning works by fine-tuning large language models (LLMs) on a labelled dataset of tasks that involve following instructions. This improves the model's ability to understand and respond to user requests, making it require less in-context information to generate useful answers. Instruction datasets can be created by humans or another LLM.

Each sample in the dataset has three parts: an instruction (like "translate this sentence from English to Spanish"), additional context (optional information to help the model, like a passage to read), and the desired output (the correct response). The model learns to match its answers with the target outputs, which helps it become better at following instructions overall.

The 2022 Google Research paper "Finetuned Language Models are Zero-Shot Learners" explains that instruction tuning combines the best aspects of pre-training and prompt engineering. By teaching the model to handle different instruction tasks, it reduces the need for complex prompt designs or examples.

Adding more tasks to the instruction tuning process improves the model's performance, even on tasks it hasn't seen before. This makes the model more flexible and better at understanding and following a wide variety of instructions.

Q. What is Prompt Engineering?

Prompt engineering involves refining the prompts a person inputs into generative AI tools, like ChatGPT or DALL-E, to create text or images. Anyone can do this using natural language to get more specific or improved results. It's also a technique AI engineers use to optimize large language models (LLMs) by crafting targeted or recommended prompts.

For instance, if you're using ChatGPT to draft a professional summary for your resume, you might start with a command like, "Write a sample professional summary for a marketing analyst." This approach can also be used with text-to-image models like DALL-E.

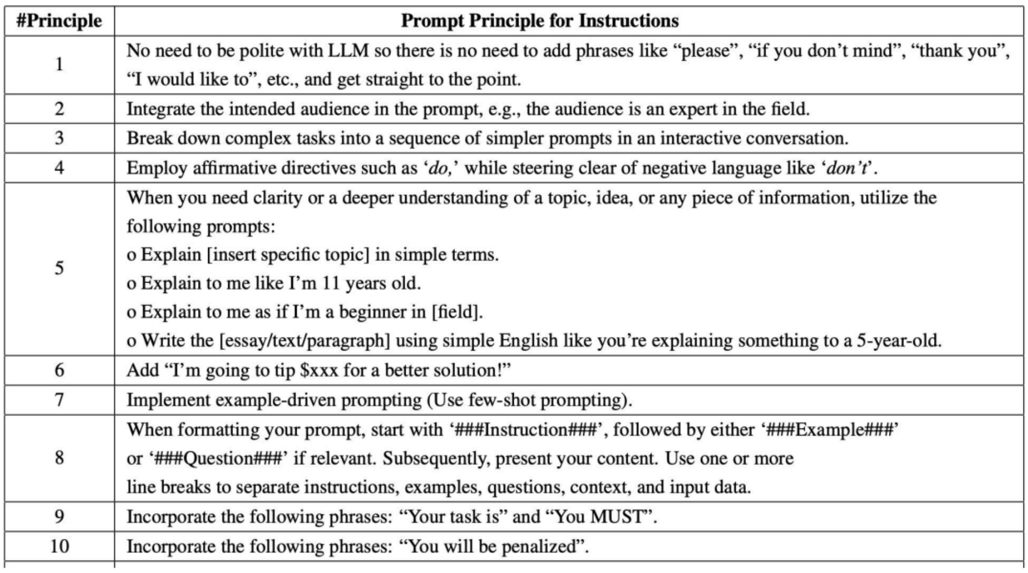
The Fine Art of Prompt Engineering

• Prompt: Natural language text describing the task that an AI should perform.

• The quality of the prompts, understanding of what you want, and asking for it correctly all go into the art and science of prompting.

• Prompt Engineering: Process of structuring text that can be interpreted and understood by a generative AI model.

• Many people are under the mistaken belief that prompting is easy and anyone can do it.



#Principle	Prompt Principle for Instructions
1	No need to be polite with LLM so there is no need to add phrases like "please", "if you don't mind", "thank you", "I would like to", etc., and get straight to the point.
2	Integrate the intended audience in the prompt, e.g., the audience is an expert in the field.
3	Break down complex tasks into a sequence of simpler prompts in an interactive conversation.
4	Employ affirmative directives such as '*do,*' while steering clear of negative language like '*don't*'.
5	When you need clarity or a deeper understanding of a topic, idea, or any piece of information, utilize the following prompts: o Explain [insert specific topic] in simple terms. o Explain to me like I'm 11 years old. o Explain to me as if I'm a beginner in [field]. o Write the [essay/text/paragraph] using simple English like you're explaining something to a 5-year-old.
6	Add "I'm going to tip $xxx for a better solution!"
7	Implement example-driven prompting (Use few-shot prompting).
8	When formatting your prompt, start with '###Instruction###', followed by either '###Example###' or '###Question###' if relevant. Subsequently, present your content. Use one or more line breaks to separate instructions, examples, questions, context, and input data.
9	Incorporate the following phrases: "Your task is" and "You MUST".
10	Incorporate the following phrases: "You will be penalized".

Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4 https://arxiv.org/abs/2312.16171v1

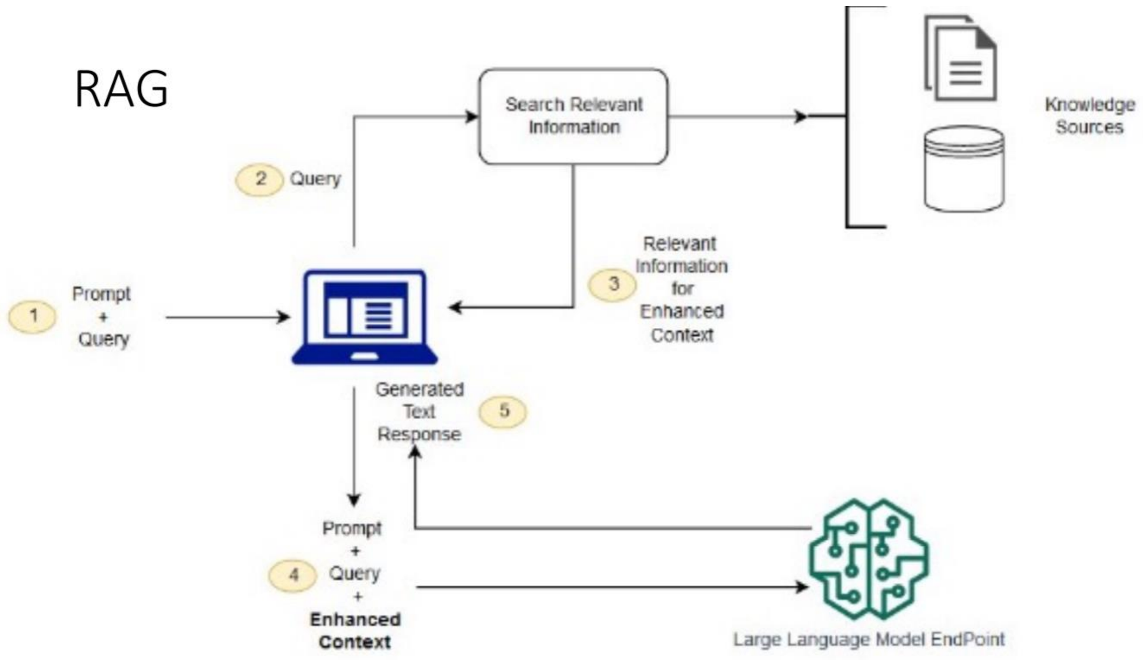
Q. What is RAG?

RAG stands for Retrieval-Augmented Generation, works by integrating two key components in natural language processing: a retrieval-based model and a generation-based model.
RAG is particularly useful for question answering, summarization, and tasks requiring factual correctness.

• RAG allows the use of the same model as a reasoning engine over new data provided in a prompt.

• This technique enables in-context learning without the need for expensive fine-tuning, empowering businesses to use LLMs more efficiently.

• RAG combines an information retrieval component with a text generator model.


RAG
Search Relevant Information
Knowledge Sources
2 Query
1 Prompt + Query
3 Relevant Information for Enhanced Context
Generated Text Response 5
4 Prompt + Query + Enhanced Context
Large Language Model EndPoint

RAG - Information retrieval components -

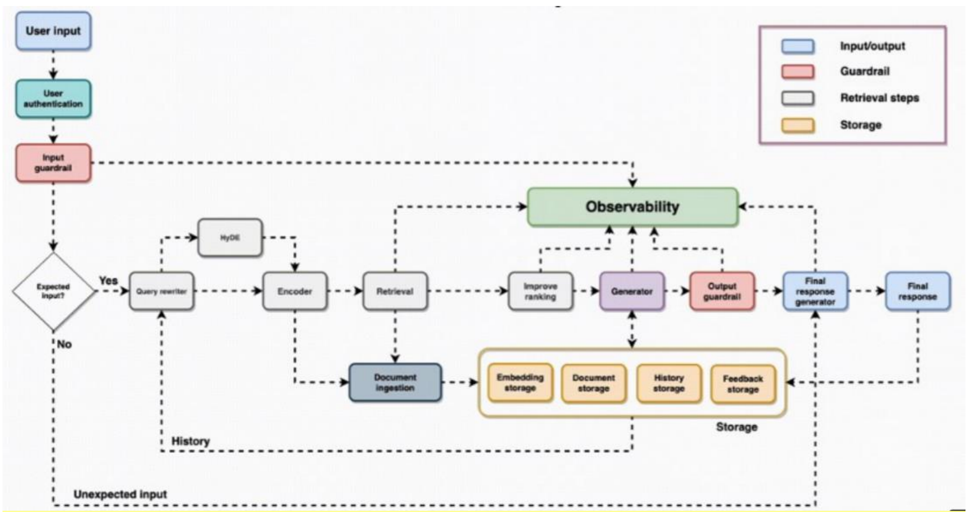
• The data are broken down into manageable chunks.

• Then they are converted into vector embeddings. The database indexes the embeddings for rapid retrieval.

• When a user queries the database, it computes similarity metrics between the chunk vectors and returns matches.

• Vector databases then precompute certain similarities between the vectors to further speed up the queries.

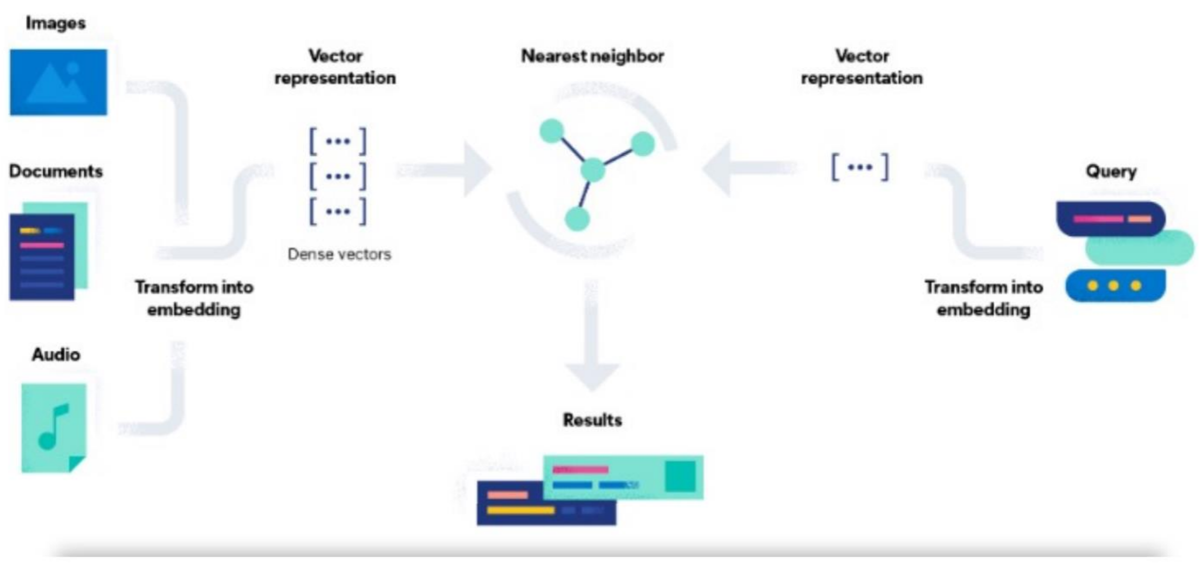
Architecture of Enterprise RAG



Q. What are Vector Databases?

• A vector database is a database that stores information as vectors, which are numerical representations of data objects, also known as vector embeddings.

• A vector database is different from a vector search library or vector index: it is a data management solution that enables metadata storage and filtering, is scalable, allows for dynamic data changes, performs backups, and offers security features.



Vector database pipeline

Q. How Vector Databases works?

• A vector database works by using algorithms to index and query vector embeddings.

• The algorithms enable approximate nearest neighbour (ANN) search through hashing or graph-based search.

• To retrieve information, an ANN search finds a query’s nearest vector neighbour.

• Less computationally intensive than a KNN search (known nearest neighbour, or true k nearest neighbour algorithm), an approximate nearest neighbour search is also less accurate.

Pre Processing and Post Processing

• Post-processing: The final step in a vector database pipeline is sometimes postprocessing, or post-filtering, during which the vector database will use a different similarity measure to re-rank the nearest neighbours.

• At this stage, the database will filter the query's nearest neighbours identified in the search based on their metadata.

• Some vector databases may apply filters before running a vector search. In this case, it is referred to as preprocessing or pre-filtering.

Q. What is PEFT?

Parameter-Efficient Fine-Tuning (PEFT), methods fine-tune only a small subset of additional model parameters while keeping the majority of the pretrained LLM's parameters frozen. This significantly reduces computational and storage requirements and mitigates the problem of catastrophic forgetting, which often occurs with full fine-tuning of LLMs. By Attaching an adapter with an additional set of weights that modifies the output of the original model.

PEFT Method – LoRA (Low-Rank Adaptation)

1. Freeze most of the original LLM weights.
2. Inject 2 low rank (generally, r = 1,2,4 or 8) decomposition matrices **into each layer** of Transformer architecture.
   - A – dimension (d * r)
   - B – dimension (r * k)
3. Train the weights of matrices A and B.

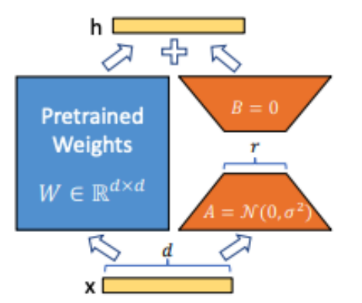


Figure 1: Our reparametrization. We only train A and B.

https://arxiv.org/pdf/2106.09685.pdf

PEFT Method – LoRA (Low-Rank Adaptation)

• LoRA is applied to any subset of weight matrices in a neural network to reduce the number of

trainable parameters.

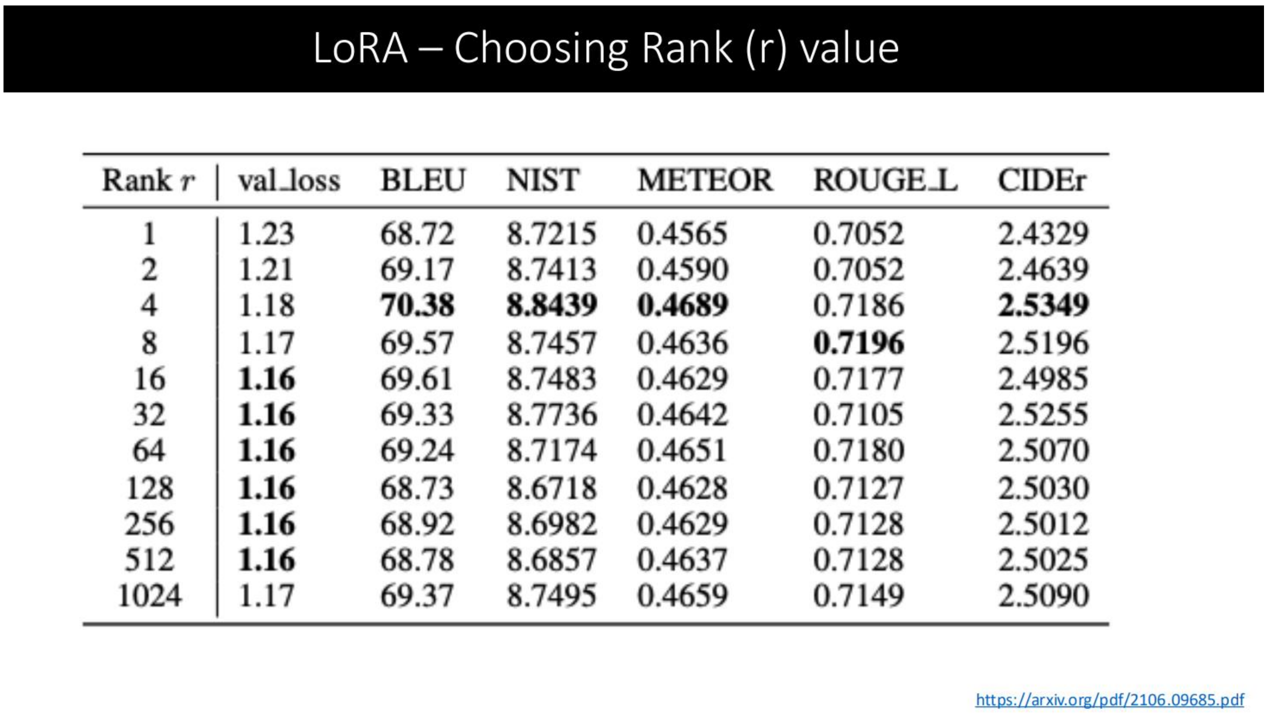
• In the Transformer architecture, there are four weight matrices in the self-attention module

(Wq, Wk, Wv, Wo) and two in the MLP module.

• Only the 4 weights matrices are adapted and are treated as single matrix of dimension dmodel * dmodel.

If the weights have a dimension d = 512, Then instead of training d*k weights i.e., 512 * 64 = 32, 768 weights

We just train 512 * 8 + 8 * 64 = 4096. (assuming r = 8). That is whopping 86% reduction.



LoRA – Choosing Rank (r) value

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

https://arxiv.org/pdf/2106.09685.pdf

LoRA Advantages

• Since the pre-trained LLM is not touch and only the new layers are trained, catastrophic forgetting is less of a concern.

• Saves space as the model is trained only for few parameters. It could be in range of few MBs instead of GBs.

• Training is faster, memory-efficient, and inference costs are lower in a trade-off model with slightly reduced performance.

• Compared to GPT-3 175B fine-tuned model, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times.

• A pre-trained model can be shared and used to build many small LoRA modules for different tasks.

• We can freeze the shared model and efficiently switch tasks by replacing the matrices.