Codebase Sustenance

It is a continuous process that ensures the proper functioning of a product and enhances its lifespan by providing updates and maintenance.

It involves optimizing maintenance strategies, identifying and resolving potential issues, and continuously improving the product or system to meet the changing needs of customers and the marketplace.

The goal of sustenance engineering is to ensure that a product or system remains efficient and effective over the long term.

**This involves not only fixing user identified problems as they arise but also proactively identifying potential issues before they become major problems.**

Code sustenance refers to the process of maintaining and improving a software codebase over time to ensure its stability, reliability, and longevity. It involves activities such as bug fixing, performance optimization, code refactoring, adding new features, and adapting the codebase to meet changing requirements.

* Bug fixing: Identifying and resolving software defects or issues reported by users or detected through testing.
* Performance optimization: Analyzing and enhancing the codebase to improve its speed, efficiency, and scalability.
* Refactoring: Restructuring the codebase to improve its readability, maintainability, and adherence to coding standards.
* Feature enhancement: Adding new features or functionality to meet user requirements or business needs.
* Security updates: Applying patches and updates to address security vulnerabilities and ensure the codebase remains secure.

Links:

<https://blog.aspiresys.com/software-product-engineering/producteering/sustenance-lifecycle/>

<https://thinkpalm.com/blogs/how-sustenance-engineering-helps-organizations-ensure-constant-product-upgrades-maintenance-2/>

Open Source LLMs

1. BLOOM (BigScience Large Open-science Open-access Multilingual Language Model)

With its 176 billion parameters (larger than OpenAI’s GPT-3), BLOOM can generate text in 46 natural languages and 13 programming languages. It is trained on 1.6TB of text data

The architecture of BLOOM shares similarities with GPT3 (auto-regressive model for **next token prediction),**but has been trained in 46 different languages and 13 programming languages. It consists of a decoder-only architecture with several embedding layers and multi-headed attention layers.

1. LLaMA (by Meta)

LLaMA is available at several sizes (7B, 13B, 33B, and 65B parameters)

LLaMA 65B and LLaMA 33B are trained on 1.4 trillion tokens. Its smallest model, LLaMA 7B, is trained on one trillion tokens.

Like other large language models, LLaMA works by taking a sequence of words as an input and predicts a next word to recursively generate text. To train this model, they chose text from the 20 languages with the most speakers, focusing on those with Latin and Cyrillic alphabets.

As a foundation model, LLaMA is designed to be versatile and can be applied to many different use cases, versus a fine-tuned model that is designed for a specific task.

[Alpaca](https://crfm.stanford.edu/2023/03/13/alpaca.html) - A model fine-tuned from the LLaMA 7B model on 52K instruction-following demonstrations.

1. Falcon 40B (by TII)

Falcon-40B is a 40B parameters causal decoder-only model built by [TII](https://www.tii.ae/) and trained on 1,000B tokens of RefinedWeb enhanced with curated corpora.

* It is the best open-source model currently available. Falcon-40B outperforms LLaMA, StableLM, RedPajama, MPT, etc. See the OpenLLM Leaderboard.
* It is made available under a permissive Apache 2.0 license allowing for commercial use, without any royalties or restrictions.
* This is a raw, pretrained model, which should be further finetuned for most usecases.
* It has a finetuned version better suited to taking generic instructions in a chat format, called Falcon-40B-Instruct. Falcon-40B-Instruct is a 40B parameters causal decoder-only model built by TII based on Falcon-40B and finetuned on a mixture of Baize. It is made available under the Apache 2.0 license. It is ready-to-use chat/instruct model based on Falcon-40B

<https://github.com/Hannibal046/Awesome-LLM>

<https://huggingface.co/tiiuae/falcon-40b>

<https://analyticsindiamag.com/14-open-source-llms-you-need-to-know/>

BEST OPEN SOURCE LLM RANKED BY HUGGINGFACE

<https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard>

LLMs for Coding or Code Generation

A code-generating LLM is trained on a more specialized dataset that includes code repositories, technical forums, coding platforms, documentation of various products and general web data that’s useful for this purpose.

Because code-generating LLMs are integrated with our IDE, they fully grasp the context of your code (comments, function names, and variable names), and use this contextual information to further improve the suggestions they make.

1. Copilot : Copilot is based on GPT-3. We need a github account for this.
2. StarCoder: The StarCoder models are 15.5B parameter models trained on 80+ programming languages from The Stack (The Stack is a collection of source code from repositories with various licenses), with opt-out requests excluded. The model uses Multi Query Attention, a context window of 8192 tokens, and was trained using the Fill-in-the-Middle objective on 1 trillion tokens.

**The model was trained on GitHub code**. As such it is not an instruction model and commands like "Write a function that computes the square root." do not work well. However, by using the Tech Assistant prompt you can turn it into a capable technical assistant.

**Limitations of LLM for Code generation**:

* **Hallucinations of code-generating LLMs**

As we know LLMs hallucinate sometimes, and code-generating LLMs are not immune to that. And as with other LLM hallucinations, the code generated may look great and work okay, but it may not do the right thing – creating subtle bugs that are hard to find.

**For example** – If we provided this prompt: *# function to select the lowest two even numbers that sum to 17*

This of course is an impossible task, since no two even numbers could sum up to 17.

But Copilot continued quite confidently to suggest a code, which is obviously Wrong as it is Impossible.

RAG : Retrieval Augmented Generation

Different from Standard seq2seq model, meaning it takes in one sequence and outputs a corresponding sequence.

There is an intermediary step though, which differentiates and elevates RAG above the usual seq2seq methods. Rather than passing the input directly to the generator, RAG instead uses the input to retrieve a set of relevant documents, in our case from Wikipedia.

Rag model developed by researchers at facebook and is recently open sourced in the hugging face transformers library. The idea of this model is to augment language models with context so instead of just using the input sequence, x,(Note: x is same as query given by us) to generate output text ,y, you would also prepend retrieve document, z, to the input sequence x so the

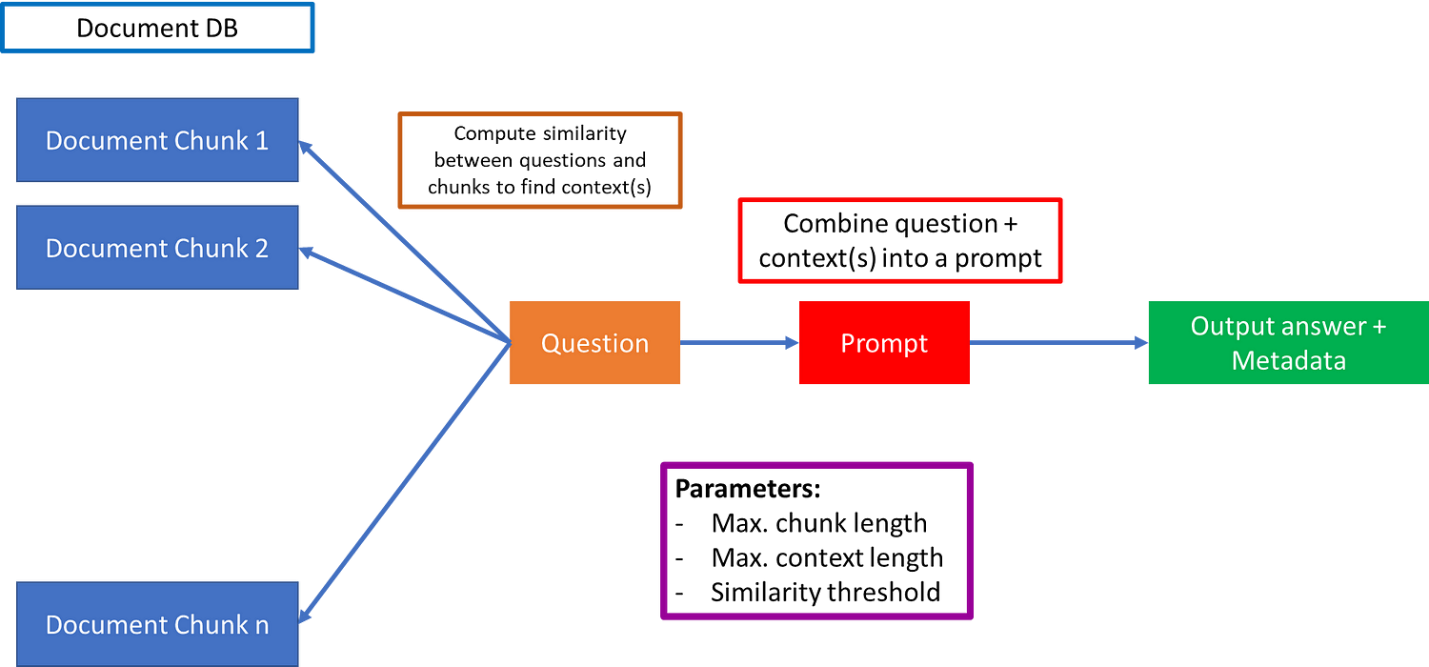
generated text, y, is a product of the input of x as well as retrieved document z.

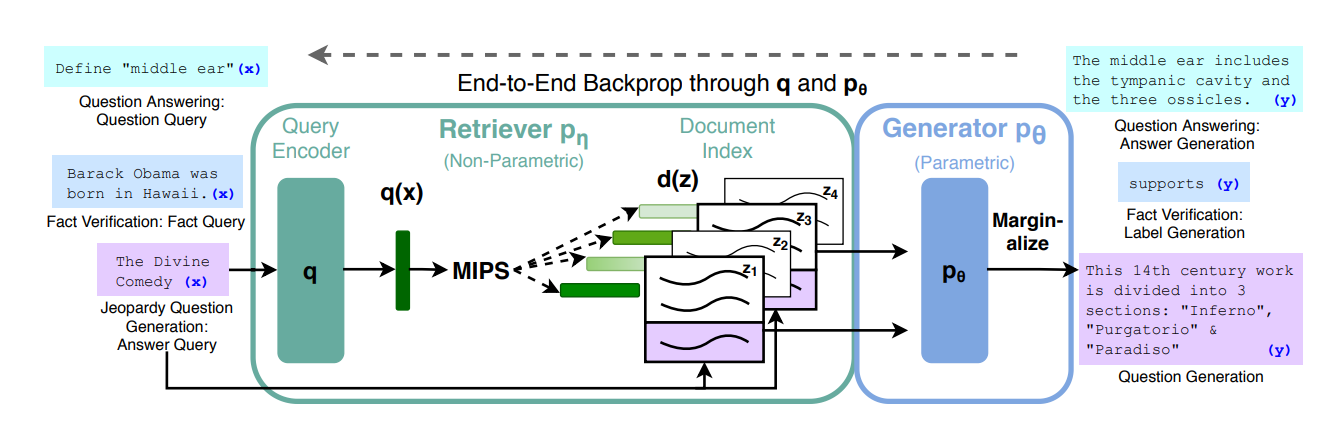
RAG thus has two sources of knowledge: the knowledge that seq2seq models store in their parameters (parametric memory) and the knowledge stored in the corpus from which RAG retrieves passages (nonparametric memory, ex: documents are stored in Non parametric memory).

These two sources complement each other. We found that RAG uses its nonparametric memory to “cue” the seq2seq model into generating correct responses, essentially combining the flexibility of the “closed-book” or parametric-only approach with the performance of “open-book” or retrieval-based methods.

It is similar to In-context Retrieval-Augmented Learning (IC-RALM)  used by GPT3

How RAG works?





* The first part is splitting multiple documents into manageable chunks, the associated parameter is the maximum chunk length.

It is available as a  Hugging Face transformer library.

the query encoder is going to be used to

encode these queries. So anytime we have an x that we're using

say it's just a sequence with a mask at the

end of it we're going to encode that as

a query and then use it to go find the most similar documents in the non-parametric encoded these wikipedia sequences .

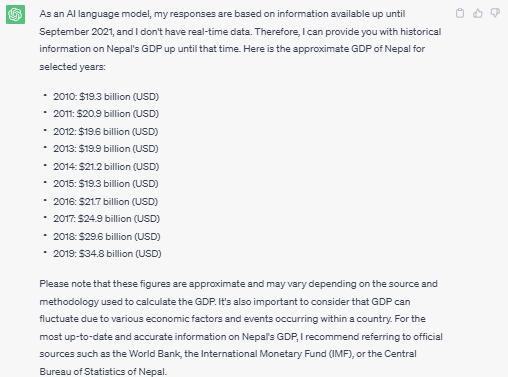
So a core idea to this is that we're not going to be training the document encoder at all.

Examples of RAG:

1. EM-GPT: It is the RAG version of ChatGPT (EM-GPT) that is able to query financial documents from IMF in the backend. Buolt by EM Alpha company.

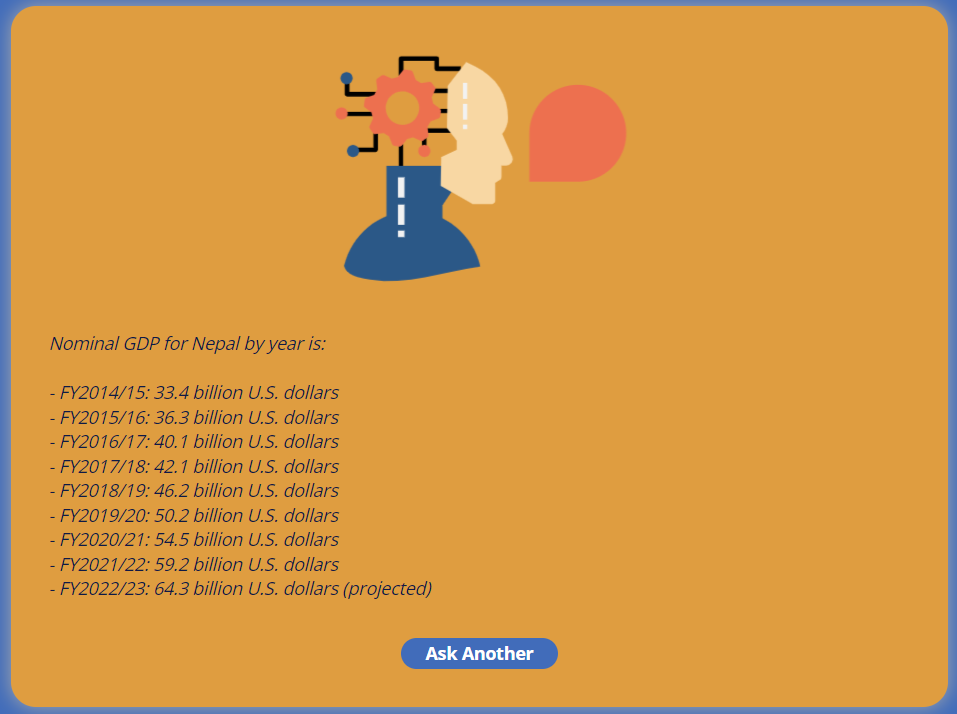
Example: Prompt is the question “What is Nepal’s GDP by year?”:

Answer by CHATGPT:



ChatGPT only returns the GDP until 2019 and it says that if you want more information, look at IMF. But if you want to figure out where this data is located in the IMF website, it is hard and you need to have an idea about where documents are stored on the website.

Answer By EM-GPT:



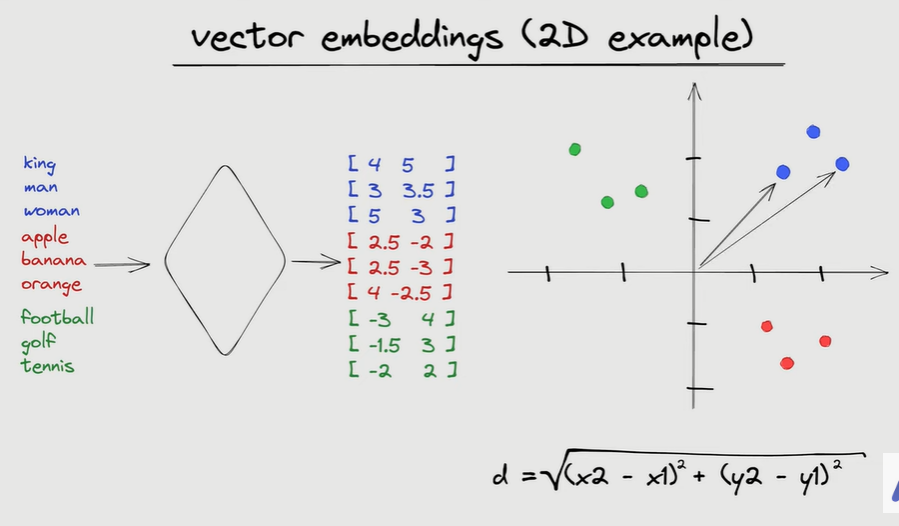
CONCLUSION:

RAG offers a great way to use LLMs powered on custom documents. While I have had people asking me which LLM to use and whether or not to fine-tune or completely train models over custom documents, **the role of engineering the sync between LLMs and vector search is underrated.**

Industry specific apps that use vector search powered LLMs (i.e. RAG apps) over their custom documents can be first movers and edge out their competition and **completely avoid the fine-tuning the model by using a simple vector search along with existing pretrained LLM.**

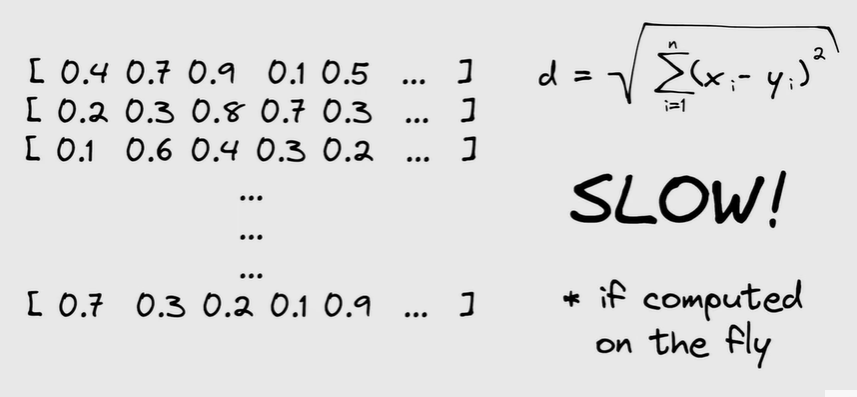
Vector database:

2 components ------- Vector Embeddings are created , and , them vector Indexes are assigned to them



**Vector Embeddings may be created using NN . The embedding allows us to look for semantically similar words.**

**Since thus searching works on the principle on Nearest Neighbor, it requires to calculate Eucledian distance for all vectors before pouring out the result. This is obviously slow. Hence Vector Indexes were developed.**



**Index is a Data structure that facilitates the search process and makes it faster. Indexes are needed for efficient search.**

