Importing necessary liberies

In [1]:
```python
from annoy import AnnoyIndex
import numpy as np
import pandas as pd
import re
import cohere
```

In [2]:
```python
API_KEY = "52ibaZZU3WoKzduJrpEI2eNhyEikplQ3QTLDd9BP"
co = cohere.Client(API_KEY)
```

important data

In [3]:
```python
text = """This is a fascinating time in the study and application of large language models. New advancements are

In this guide, I share my analysis of the current architectural best practices for data-informed language model

Overview
In nearly all practical applications of large language models (LLM’s), there are instances in which you want the

At a high level, there are two primary methods for referencing specific data:

Insert data as context in the model prompt, and direct the response to utilize that information
Fine-tune a model, by providing hundreds or thousands of prompt <> completion pairs
Shortcomings of Knowledge Retrieval for Existing LLM’s
Both of these methods have significant shortcomings in isolation.

For the context-based approach:

Models have a limited context size, with the latest `davinci-003` model only able to process up to 4,000 tokens
Processing more tokens equates to longer processing times. In customer-facing scenarios, this impairs the user e
Processing more tokens equates to higher API costs, and may not lead to more accurate responses if the informati
For the fine-tuning approach:

Generating prompt <> completion pairs is time-consuming and potentially expensive.
Many repositories from which you want to reference information are quite large. For example, if your applicatio
Some external data sources change quickly. For example, it is not optimal to retrain a customer support model ba
Best practices around fine-tuning are still being developed. LLM's themselves can be used to assist with the gen
The Solution, Simplified

The design above goes by various names, most commonly "retrieval-augmented generation" or "RETRO". Links & rela

RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
RETRO: Improving language models by retrieving from trillions of tokens
REALM: Retrieval-Augmented Language Model Pre-Training
Retrieval-augmented generation a) retrieves relevant data from outside of the language model (non-parametric) an

Retrieval


The retrieval of relevant information is worth further explanation. As you can see, data may come from multiple

Language model embeddings are numerical representations of concepts in text and seem to have endless uses. Here


Back to the flow — when a user submits a question, an LLM processes the message in multiple ways, but the key st

Augmentation

Building the prompt using the relevant text chunks is straightforward. The prompt begins with some basic prompt

Finally, the combined prompt is sent to the large language model. An answer is parsed from the completion and pa

That’s it! While this is a simple version of the design, it’s inexpensive, accurate, and perfect for many lightw

Advanced Design

I want to take a moment to discuss several research developments that may enter into the retrieval-augmented gen

Generate-then-Read Pipelines

This category of approaches involves processing the user input with an LLM before retrieving relevant data.

Basically, a user’s question lacks some of the relevance patterns that an informative answer will display. For e

A similar approach titled generate-then-read (GenRead) builds on the practice by implementing a clustering algo

Improved Data Structures for LLM Indexing & Response Synthesis
```

```
The GPT Index project is excellent and worth a read. It utilizes a collection of data structures both created by

List Index - Each node represents a text chunk, otherwise unaltered. In the default setup, all nodes are combine

Vector Store Index - This is equivalent to the simple design that I explained in the previous section. Each tex

Keyword Index - This supports a quick and efficient lexical search for particular strings.

Tree Index - This is extremely useful when your data is organized into hierarchies. Consider a clinical documen


GPT Index offers composability of indices, meaning you can build indices on top of other indices. For example, i
Expanded Context Size

Some of the approaches outlined in this post sound "hacky" because they involve workarounds to the relatively sm

GPT-4 is anticipated within the next 1-3 months. It is rumored to have a larger context size.
This paper from the folks at Google AI features a number of explorations of engineering tradeoffs. One of the co
A new state space model architecture scales ~linearly with context size instead of quadratically like seen in t
In my opinion, advancements in context size will scale alongside demands for more data retrieval; in other words

Persisting State (e.g. Conversation History)

When LLM's are presented to the user in a conversational form, a major challenge is maintaining that conversatio

An overview of the relevant strategies is beyond the scope of this post; for an example of a recent code demons
```

CHUNKING

Split into a list of paragraphs

In [4]:
```
texts =text.split('\n\n')
```

Clean up to remove empty spaces and new lines

In [5]:
```
texts = np.array([t.strip(' \n') for t in texts])
```

Checking first 10 paragraphs

In [7]:
```
texts[:5]
```

Out[7]:
```
array(['This is a fascinating time in the study and application of large language models. New advancements are announced every day!',
       'In this guide, I share my analysis of the current architectural best practices for data-informed language model applications. This particular subdiscipline is experiencing phenomenal research interest even by the standards of large language models - in this guide, I cite 8 research papers and 4 software projects,  with a median initial publication date of November 22nd, 2022.',
       'Overview\nIn nearly all practical applications of large language models (LLM's), there are instances in which you want the language model to generate an answer based on specific data, rather than supplying a generic answer based on the model's training set. For example, a company chatbot should be able to reference specific articles on the corporate website, and an analysis tool for lawyers should be able to reference previous filings for the same case. The way in which this external data is introduced is a key design question.',
       'At a high level, there are two primary methods for referencing specific data:',
       'Insert data as context in the model prompt, and direct the response to utilize that information\nFine-tune a model, by providing hundreds or thousands of prompt <> completion pairs\nShortcomings of Knowledge Retrieval for Existing LLM's\nBoth of these methods have significant shortcomings in isolation.'],
      dtype='<U812')
```

Embeddings

In [8]:
```
response = co.embed(
    texts=texts.tolist()
).embeddings
```

Checking dimension

In [11]:
```
embeds = np.array(response)
embeds.shape
```

Out[11]:
```
(36, 4096)
```

Showing the Embeddings(Vector representations)

In [12]:
```
embeds
```

Out[12]:
```
array([[ 1.8261719 ,  1.3398438 ,  1.9511719 , ...,  0.29418945,
         0.5629883 ,  2.1503906 ],
       [ 1.1611328 , -0.05752563,  0.44995117, ...,  0.6870117 ,
        -1.8222656 ,  0.62402344],
       [-1.2568359 , -0.50097656, -1.1279297 , ...,  1.2841797 ,
        -0.06890869, -0.05160522],
       ...,
       [-1.0722656 , -2.1699219 ,  0.00843048, ...,  0.32739258,
        -1.0058594 , -0.89697266],
       [ 0.11981201, -0.13781738, -0.76953125, ...,  0.26513672,
        -0.6118164 ,  0.42089844],
       [ 0.5991211 ,  0.01847839,  0.5214844 , ...,  0.14221191,
        -0.6118164 ,  0.53808594]])
```

Create the search index

In [13]:
```python
search_index = AnnoyIndex(embeds.shape[1], 'angular')
# Add all the vectors to the search index
for i in range(len(embeds)):
    search_index.add_item(i, embeds[i])

search_index.build(10) # 10 trees
search_index.save('test.ann')
```

Out[13]: True

Search Function

In [18]:
```python
pd.set_option('display.max_colwidth', None)

def search(query):

  # Get the query's embedding
  query_embed = co.embed(texts=[query]).embeddings

  # Retrieve the nearest neighbors
  similar_item_ids = search_index.get_nns_by_vector(query_embed[0],1,

                                                    include_distances=True)
  # Format the results
  results = pd.DataFrame(data={'texts': texts[similar_item_ids[0]],
                              'distance': similar_item_ids[1]})

  print(texts[similar_item_ids[0]])

  return results
```

In [19]:
```python
query = "what are large language models"
search(query)
```

```
["Language model embeddings are numerical representations of concepts in text and seem to have endless uses. Her
e's how they work: an embeddings model converts text into a large, scored vector, which can be efficiently compa
red to other scored vectors to assist with recommendation, classification, and search (+more) tasks. We store th
e results of this computation into what I'll generically refer to as the search index & entity store - more adva
nced discussions on that below."]
```

Out[19]:

	texts	distance
0	Language model embeddings are numerical representations of concepts in text and seem to have endless uses. Here's how they work: an embeddings model converts text into a large, scored vector, which can be efficiently compared to other scored vectors to assist with recommendation, classification, and search (+more) tasks. We store the results of this computation into what I'll generically refer to as the search index & entity store - more advanced discussions on that below.	0.866105

In [ ]: