Xorbits computing framework in Data Processing and Vector search for Large Language Models

SongMing Zhou

February 17, 2024

1 Introduction

Over the past two years, the emergence of AI tools like ChatGPT has vividly demonstrated the vast potential of AI programs and services across various fields. However, how does a machine, fundamentally capable of understanding only binary codes (0 and 1) and operating large language models, possess such extensive knowledge, despite its considerable flaws?

The answer lies in the way we humans feed information to these machines. Although machines lack the ability to perceive the world as humans do, they can process and understand the data we provide in the form of numerical vectors. This is where embeddings come into play.

Embeddings essentially describe entities such as words as vectors in a multidimensional space. In other words, it's a sequence of numbers, for example, [.89, .65, .45, ...]. This offers a smart way to convert categorical entities into numerical representations, allowing us to interpret them in various ways, such as evaluating their similarity by organizing them in a multidimensional space.

In today's data analysis field, Python is highly popular, with libraries like Pandas, Numpy, and Scikit-learn covering areas including table data processing, matrix numerical calculations, and machine learning. They are rich in interfaces and easy to use. However, as the volume of data grows rapidly, these libraries, which can only run on a single machine, face significant performance challenges, especially when dealing with tens of millions or even billions of data points. The memory of a single machine is insufficient to hold all the data. Xorbits seems to be born to solve these problems. As a computational framework, it enables the parallel and accelerated execution of Python computations using multi-core, heterogeneous, and distributed technologies. In this report, we will explore Xorbits computating framework and analyze how it handles large amounts of data.

2 Large Language Model

Before we start to explore the Xorbits computing framework, we first need to know something about LLM, and then we will know why computing framework is important to LLM.

2.1 What is Large Language Model

A Large Language Model (LLM) is an enormous deep learning model that is pre-trained on vast amounts of data. Its foundation, the transformer, consists of a series of neural networks equipped with encoders and decoders that feature self-attention mechanisms. These encoders and decoders extract meanings from a sequence of texts and understand the relationships between words and phrases within them.

Transformer LLMs are capable of undergoing unsupervised training, but a more precise explanation is that transformers can perform autonomous learning. Through this process, transformers learn to grasp basic grammar, language, and knowledge.

Unlike earlier Recurrent Neural Networks (RNNs) that processed inputs sequentially, transformers handle the entire sequence in parallel. This allows data scientists to train transformer-based LLMs using GPUs, significantly reducing training times.

With the transformer neural network architecture, you can work with extremely large models, which typically have hundreds of billions of parameters. These massive models are capable of ingesting vast amounts of data, typically sourced from the internet, but they can also draw from sources like the Common Crawl, which contains over 500 billion web pages, and Wikipedia, which boasts about 57 million pages.

2.2 Why Large Language Model is that Important

Large language models are remarkably versatile. A single model can perform completely different tasks, such as answering questions, summarizing documents, translating languages, and completing sentences. LLMs have the potential to disrupt content creation as well as how people use search engines and virtual assistants.

Despite not being perfect, LLMs demonstrate an extraordinary ability to make predictions based on relatively small amounts of prompts or inputs. LLMs can be used in generative artificial intelligence to produce content based on inputs in human language.

LLMs are immense. They can consider billions of parameters and have numerous potential uses. Here are some examples:

OpenAI's GPT-3 model has 175 billion parameters. Similar products like ChatGPT can identify patterns in data and generate natural and readable outputs. While the scale of Claude 2 is not disclosed, the model can input up to 100,000 tokens per prompt, meaning it can process hundreds of pages of technical documents, or even entire books. AI21 Labs' Jurassic-1 model has 178 billion parameters and a token vocabulary consisting of 250,000 words, along with similar conversational capabilities. Cohere's Command model has similar functionalities and can work in over 100 different languages. LightOn's Paradigm offers a foundational model and claims its functionalities surpass those of GPT-3. All these LLMs come with APIs, allowing developers to create unique generative artificial intelligence applications.

2.3 How do Large Language Models Work

A key factor in how LLMs operate is the way they represent words. Early machine learning models used numerical tables to represent each word. However, this form of representation failed to recognize the relationships between words, such as words with similar meanings. This limitation was overcome by using multi-dimensional vectors (often called word embeddings) to represent words, thereby allowing words with similar contextual meanings or other relationships to be close to each other in vector space.

With word embeddings, transformers preprocess text into numerical representations through encoders, understanding the context of words and phrases with similar meanings, as well as other relationships between words, such as parts of speech. Then, LLMs can apply this linguistic knowledge through decoders to generate unique outputs.

2.4 What kind of applications does LLM has

Large Language Models (LLMs) have a wide range of practical applications.

2.4.1 Copywriting

Beyond GPT-3 and ChatGPT, models like Claude, Llama 2, Cohere Command, and Jurassic can also produce original copy. AI21's Wordspice suggests changes to original sentences to improve style and voice.

2.4.2 Knowledge-Based Question Answering

This technology, often referred to as Knowledge-Intensive Natural Language Processing (KI-NLP), involves LLMs that can assist in answering specific questions through information in digital archives. An example is AI21 Studio Playground's capability to answer general knowledge questions.

2.4.3 Text Classification

Using clustering, LLMs can categorize texts with similar meanings or sentiments. Applications include measuring customer sentiment, determining relationships between texts, and document searching.

2.4.4 Code Generation

LLMs excel in generating code based on natural language prompts. Examples include Amazon CodeWhisperer and OpenAI's Codex used in GitHub Copilot, capable of coding in Python, JavaScript, Ruby, and several other programming languages. Other coding applications include creating SQL queries, writing shell commands, and website design.

2.4.5 Text Generation

Similar to code generation, text generation can complete incomplete sentences, write product documentation, or, like Alexa Create, compose a short children's story.

3 Practical application of the Xorbits and the exploration of Vector search

3.1 Creating Vector embeddings

In the introduction, we've already mentioned that for computing systems operating large language models, numerical vector embeddings play a crucial role. But how do distributed computing frameworks generate vector embeddings from text?

Let's illustrate this with an example of how we use the SentenceTransformer model to encode textual data into dense vectors.

Recent research in NLP machine learning has primarily focused on training deep language models. In this process, neural networks take a large corpus of text as input and create mathematical representations of words in the form of vectors. The method of creating these vectors involves grouping words with similar meanings and contextual appearances together, represented by similar vectors. For example, we can also take the average of all word vectors to create a vector for the entire text (such as a query, sentence, or paragraph).

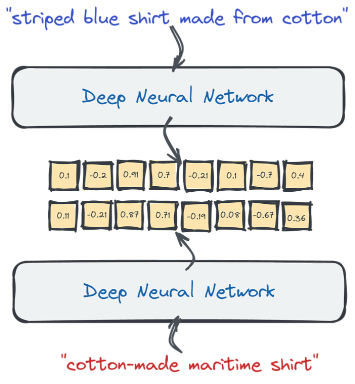


Figure 1: Input queries contain different words, but they are still converted into similar vector representations, because the neural encoder can capture the meaning of the sentences. That feature can capture synonyms but also different languages.

Vector search is the process of finding similar objects based on embedding similarity. The advantage is that you don't have to design and train neural networks yourself. Many pre-trained models are available, whether on HuggingFace or through libraries like SentenceTransformers. However, if you prefer not to deal directly with neural models, you can also create embeddings using SaaS tools, such as the co.embed API.

3.2 Exploratory Data Analysis

After conducting a series of data cleaning processes on the original dataset, I extracted 100,000 rows from the original data to perform Exploratory Data Analysis (EDA). This was done to compare the performance differences in generating embedding vectors between traditional pandas and Xorbits.pandas.

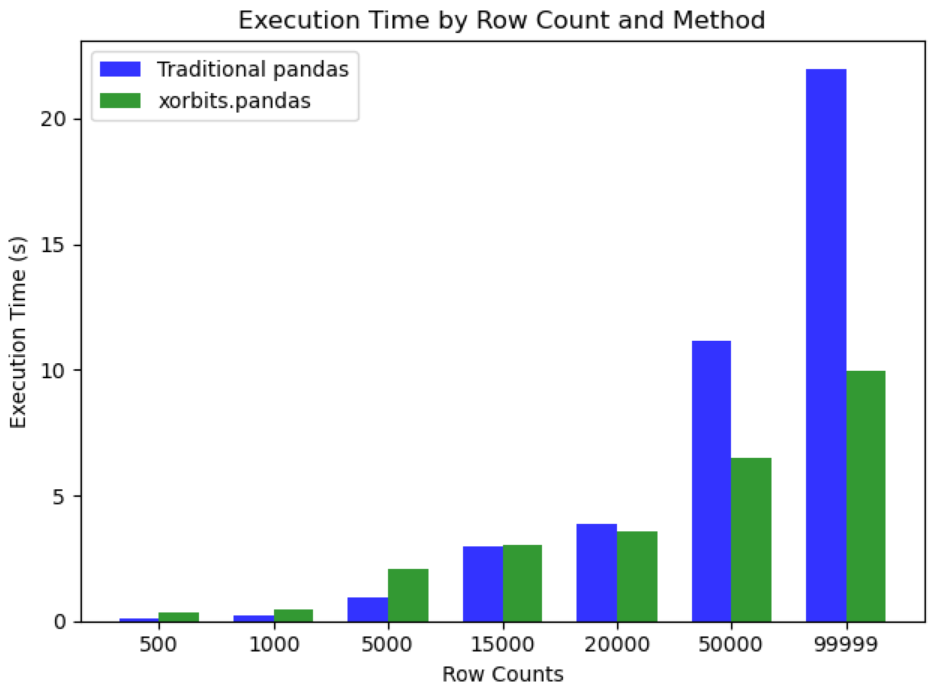


Figure 2: Execution Time Figure

I compared the processing results for datasets containing 500, 1,000, 5,000, 15,000, 20,000, 50,000, and 99,999 rows, respectively. It was not difficult to observe that traditional pandas has a certain advantage in handling small datasets and generating vector embeddings on a single-core basis. However, as we move forward in time, the datasets we need to process are becoming increasingly larger, and this is where tensor-based parallel computing frameworks begin to shine. From the exploratory analysis graphs, it's clear that Xorbits' performance begins to surpass traditional pandas when the dataset size exceeds 15,000 rows. When the dataset size reaches close to 100,000 rows, Xorbits' performance is almost three times superior to that of traditional pandas.

3.3 How Xorbits computes in parallel

Although users use Xorbits almost the same as Numpy or Pandas, the implementation behind it has been rewritten with parallel logic.

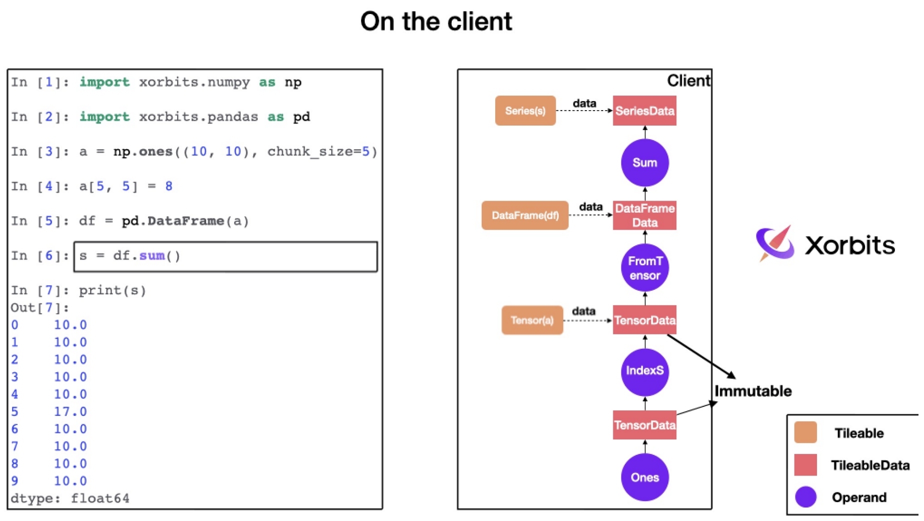
Xorbits uses a divide-and-conquer approach to perform computing tasks in parallel, which is mainly divided into three steps:

1. The client builds a calculation graph and submits it to the server.
2. The server-side calculation graph will be processed into multiple small execution units (called chunks in Xorbits).
3. The scheduler distributes tasks to workers for execution.

To better understand how Xorbits performs distributed computing, let's consider the following example:

3.3.1 Build a computational graph

When a user calls a function of Xorbits, a calculation graph will be incrementally constructed behind the scenes. Each time the function is called, one or more nodes will be added to the graph, and each node corresponds to an operator.



```
In [1]: import xorbits.numpy as np

In [2]: import xorbits.pandas as pd

In [3]: a = np.ones((10, 10), chunk_size=5)

In [4]: a[5, 5] = 8

In [5]: df = pd.DataFrame(a)

In [6]: s = df.sum()

In [7]: print(s)
Out[7]:
0     10.0
1     10.0
2     10.0
3     10.0
4     10.0
5     17.0
6     10.0
7     10.0
8     10.0
9     10.0
dtype: float64
```

Figure 3: Xorbits composition process

Within the Xorbits framework, these operations do not execute immediately but build a computational graph instead. This graph describes the entire process of data from its creation to transformation. Each time a user performs an operation (like changing a value or summing up), the graph adds one or multiple nodes.

When the Xorbits framework determines the need to compute the value of a node (for instance, when the user wants to output a result), it does not just compute that one node but sends the entire computational graph to the server-side. Once the server receives the computational graph, it performs the computations based on the dependencies and optimization strategies within the graph. This approach allows for multiple optimizations, such as:

- **Lazy Evaluation**: Computations are only executed when results are needed, which allows the framework to optimize the entire computation process, reducing unnecessary operations.
- **Parallel and Distributed Computing**: The server can execute operations in the computational graph across multiple processors or machines, enhancing efficiency.
- **Resource Management**: The server can manage memory and other resources more effectively since it has a view of the entire computational graph.
- **Automatic Differentiation**: For machine learning and optimization problems, the computational graph makes implementing automatic differentiation easier.

3.3.2 Decomposed calculation diagram

When the server receives the calculation graph, there will be a series of operations. The most important step is to decompose the calculation graph into a more fine-grained graph. In this fine-grained graph, all nodes are executable units. This process is It is called tiling in Xorbits. During the tiling process, the calculation graph will adjust the decomposition granularity according to the size of the data and some specified parameters, and a single operator may also be decomposed into multiple operators. In this example, the original input data is divided into four small pieces of data, and each piece of data becomes a separate node, which does not affect each other and can be executed in parallel.

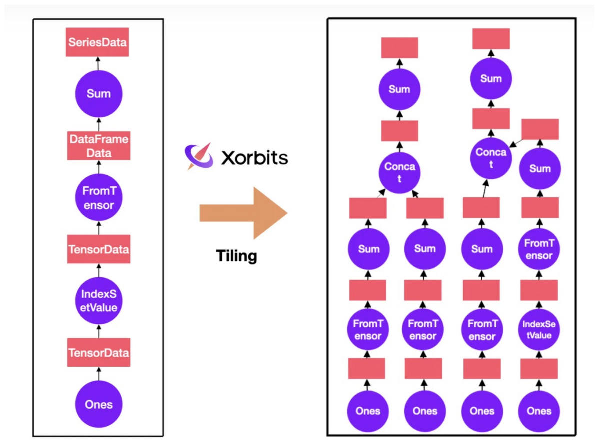


Figure 4: Tiling diagram

The above figure shows the comparison of calculation graphs before and after tiling. Observing the fine-grained graph on the right, we can see that an operator like "setvalues" will only affect one of the four small pieces of data, so only one branch in the fine-grained graph has this operator node inserted. , this feature makes the construction of computational graphs more precise and flexible.

For more practical examples of Xorbits applications, one may refer to the literature provided by [3], which offers a detailed explanation and analysis of Xorbits.

3.4 Insert Articles into the Vector Database

3.4.1 The purpose of Vector Database

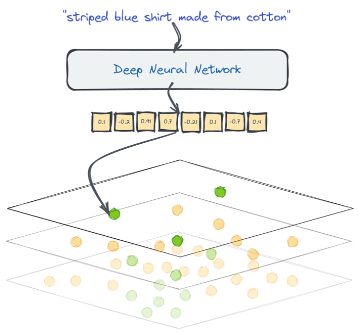


Figure 5: Vector search with Vecotr Database

Vector search becomes challenging when searching for similar documents in a large set. Calculating distances to every document is feasible for small sets but impractical for larger ones. Like using indexes in relational databases to avoid full scans, a vector database uses a graph structure to find close objects quickly without comparing every object, enhancing search efficiency.

3.4.2 What is Vector Search

Vector search is a way to find related objects that have similar characteristics using machine learning models that detect semantic relationships between objects in an index.

Solutions for vector search and recommendation are becoming more and more common. If you want to add a natural language text search on your site, create image search, or build a powerful recommendation system, you'll want to look into using vectors.

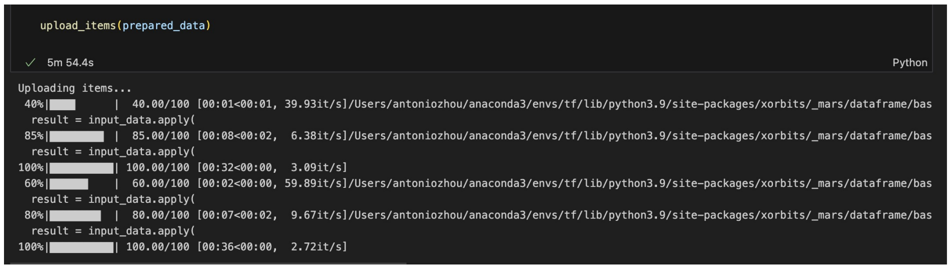
The research behind it has been decades in the making, but up until now building and scaling vector search has only been available to the largest of companies like Google, Amazon, and Netflix.

These companies have hired thousands of engineers and data scientists, and some have even developed their own computer chips to offer faster machine learning.

Today, just about any company can deploy vector-powered search and recommendations in a fraction of the time and price. Vector technologies unleash a whole new era for developers to build solutions that enable better search, recommendation, and prediction solutions.

Vector search is not the area we just started exploring, the following article can help you to enter this world faster: [2] and [4]

3.4.3 Uploading Embedding Vectors



```
upload_items(prepared_data)

✓ 5m 54.4s                                                                                     Python

Uploading items...
 40%|████      |  40.00/100 [00:01<00:01, 39.93it/s]/Users/antoniozhou/anaconda3/envs/tf/lib/python3.9/site-packages/xorbits/_mars/dataframe/bas
  result = input_data.apply(
 85%|████████▌ |  85.00/100 [00:08<00:02,  6.38it/s]/Users/antoniozhou/anaconda3/envs/tf/lib/python3.9/site-packages/xorbits/_mars/dataframe/bas
  result = input_data.apply(
100%|██████████| 100.00/100 [00:32<00:00,  3.09it/s]
 60%|██████    |  60.00/100 [00:02<00:00, 59.89it/s]/Users/antoniozhou/anaconda3/envs/tf/lib/python3.9/site-packages/xorbits/_mars/dataframe/bas
  result = input_data.apply(
 80%|████████  |  80.00/100 [00:07<00:02,  9.67it/s]/Users/antoniozhou/anaconda3/envs/tf/lib/python3.9/site-packages/xorbits/_mars/dataframe/bas
  result = input_data.apply(
100%|██████████| 100.00/100 [00:36<00:00,  2.72it/s]
```

Figure 6: Uploading Embedding Vectors

After we have created the article embedding vectors using the Xorbits computing framework, the next step involves storing these embedding vectors in a vector database (specifically, the Pinecone vector database) for the purpose of swiftly querying vectors similar to the input vector. We will create an index within Pinecone, which leverages **various algorithms (such as approximate nearest neighbor search algorithms) to optimize the search process.** This enables rapid identification of the most similar items within vast datasets while minimizing the use of computational resources. Through this approach, Pinecone maintains efficient query performance even in very large datasets, making it suitable for applications requiring quick similarity matching, such as recommendation systems, content retrieval, and other machine learning-driven tasks.

3.5 Query the Index for Similar Articles

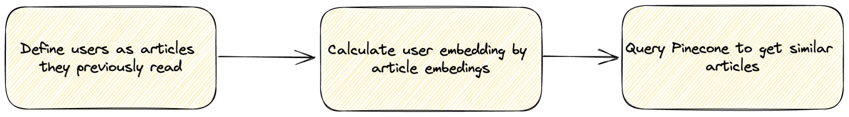
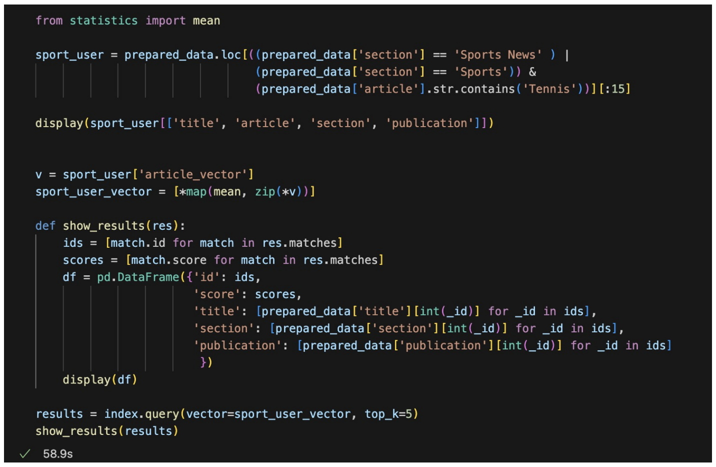

Define users as articles they previously read
Calculate user embedding by article embedings
Query Pinecone to get similar articles

Figure 7: query Embedding vectors process figure

How can we identify articles related to Tennis? Here, we employ a straightforward approach (though not necessarily highly precise): by extracting ten articles from the original data whose titles contain "Tennis" and whose sections fall under "Sports" or "Sports News". We then retrieve their embedding vectors and calculate an average vector from these embeddings, using this average vector as the search vector for Tennis.

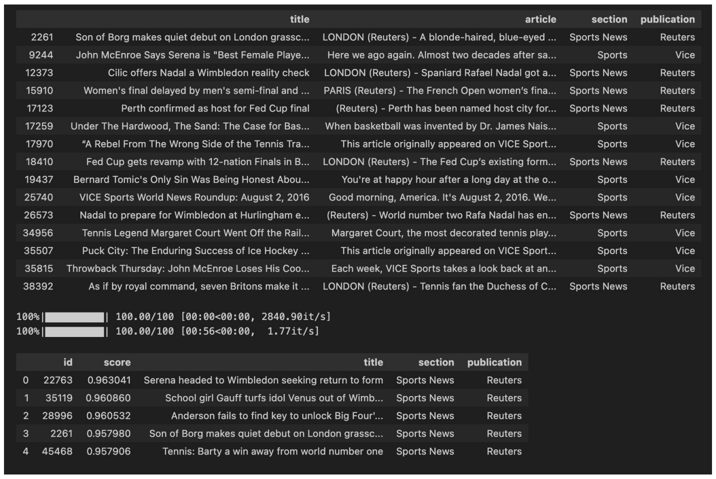
Here is the code and output we got from search:



```python
from statistics import mean

sport_user = prepared_data.loc[((prepared_data['section'] == 'Sports News' ) |
                                (prepared_data['section'] == 'Sports')) &
                                (prepared_data['article'].str.contains('Tennis'))][:15]

display(sport_user[['title', 'article', 'section', 'publication']])


v = sport_user['article_vector']
sport_user_vector = [*map(mean, zip(*v))]

def show_results(res):
    ids = [match.id for match in res.matches]
    scores = [match.score for match in res.matches]
    df = pd.DataFrame({'id': ids,
                       'score': scores,
                       'title': [prepared_data['title'][int(_id)] for _id in ids],
                       'section': [prepared_data['section'][int(_id)] for _id in ids],
                       'publication': [prepared_data['publication'][int(_id)] for _id in ids]
                        })
    display(df)

results = index.query(vector=sport_user_vector, top_k=5)
show_results(results)
```

✓ 58.9s

Figure 8: Query code figure



	title	article	section	publication
2261	Son of Borg makes quiet debut on London grassc...	LONDON (Reuters) - A blonde-haired, blue-eyed ...	Sports News	Reuters
9244	John McEnroe Says Serena is "Best Female Playe...	Here we ago again. Almost two decades after sa...	Sports	Vice
12373	Cilic offers Nadal a Wimbledon reality check	LONDON (Reuters) - Spaniard Rafael Nadal got a...	Sports News	Reuters
15910	Women's final delayed by men's semi-final and ...	PARIS (Reuters) - The French Open women's fina...	Sports News	Reuters
17123	Perth confirmed as host for Fed Cup final	(Reuters) - Perth has been named host city for...	Sports News	Reuters
17259	Under The Hardwood, The Sand: The Case for Bas...	When basketball was invented by Dr. James Nais...	Sports	Vice
17970	"A Rebel From The Wrong Side of the Tennis Tra...	This article originally appeared on VICE Sport...	Sports	Vice
18410	Fed Cup gets revamp with 12-nation Finals in B...	LONDON (Reuters) - The Fed Cup's existing form...	Sports News	Reuters
19437	Bernard Tomic's Only Sin Was Being Honest Abou...	You're at happy hour after a long day at the o...	Sports	Vice
25740	VICE Sports World News Roundup: August 2, 2016	Good morning, America. It's August 2, 2016. We...	Sports	Vice
26573	Nadal to prepare for Wimbledon at Hurlingham e...	(Reuters) - World number two Rafa Nadal has en...	Sports News	Reuters
34956	Tennis Legend Margaret Court Went Off the Rail...	Margaret Court, the most decorated tennis play...	Sports	Vice
35507	Puck City: The Enduring Success of Ice Hockey ...	This article originally appeared on VICE Sport...	Sports	Vice
35815	Throwback Thursday: John McEnroe Loses His Coo...	Each week, VICE Sports takes a look back at an...	Sports	Vice
38392	As if by royal command, seven Britons make it ...	LONDON (Reuters) - Tennis fan the Duchess of C...	Sports News	Reuters

```
100%|██████████| 100.00/100 [00:00<00:00, 2840.90it/s]
100%|██████████| 100.00/100 [00:56<00:00,  1.77it/s]
```

	id	score	title	section	publication
0	22763	0.963041	Serena headed to Wimbledon seeking return to form	Sports News	Reuters
1	35119	0.960860	School girl Gauff turfs idol Venus out of Wimb...	Sports News	Reuters
2	28996	0.960532	Anderson fails to find key to unlock Big Four'...	Sports News	Reuters
3	2261	0.957980	Son of Borg makes quiet debut on London grassc...	Sports News	Reuters
4	45468	0.957906	Tennis: Barty a win away from world number one	Sports News	Reuters

Figure 9: Query output figure

3.5.1 What kind of search algorithm could be included in Vector Database Index

Nowadays, there is a wide diversity of vector embedding models to process different data such as images, videos, and audio. There are also many freely available vector databases with vector embeddings and distance metrics that represent nearness or similarity between vectors.

There are also various algorithms which can be used to search a vector database to find similarity. These include:

- ANN (approximate nearest neighbor): an algorithm that uses distance algorithms to locate nearby vectors.
- kNN: an algorithm that uses proximity to make predictions about grouping.
- (SPTAG) Space partition tree and graph: a library for large scale approximate nearest neighbors.
- Faiss: Facebook's similarity search algorithm
- HNSW (hierarchical navigable small world): a multilayered graph approach for determining similarity.

There are tradeoffs between these different techniques and often you'll see multiple techniques being used to deliver results faster and with greater accuracy. These various techniques will deliver better results even for hard-to-process queries. We will write a future blog about these different techniques and tradeoffs.

Vector search is no longer a novelty for us; tech giants like Google and Amazon have been utilizing vector search for decades. It's said to be a crucial driver behind clicks, views, and sales across various platforms. However, it's only with the advent of FAISS that this technology has become more accessible. What sets FAISS apart? We'll delve into what makes FAISS unique in this report.

3.5.2 FAISS

FAISS, standing for Facebook AI Similarity Search, is a library written in C++ with a Python interface, offering various data structures and methods to enhance the efficiency of vector search. Vector search diverges from traditional search as it moves away from relying on inverted indexes and necessitates considering the distances between data points, rather than merely the value of individual dimensions. These vectors are often produced by deep learning models, with a vast array of pre-trained vectors readily available.

As a C++ library, FAISS serves as an excellent tool for rapidly conducting experiments. If you have a dataset, you can generate embeddings from data points using pre-trained models or fine-tune existing models, then create a FAISS index, and finally perform searches similar to examples. When compared to brute-force k-NN, the speed is impressively fast. Unfortunately, the index is only available within the process that created it, so integrating with some external services requires extra effort and the creation of custom interfaces. All vectors are also stored in the RAM of a single machine, hence, if the dataset is large and/or the embedding dimensions are high, a significant amount of available memory is needed, or one must settle for using virtual memory and facing slowdowns. Of course, there are optimizations available, such as Product Quantization (PQ), but at its core, FAISS is another heavyweight data structure that you need to maintain in some way.

Is FAISS perfect, then? Primarily optimized for x86 architecture, FAISS caters well to its widespread use in servers and personal computers. However, as ARM architecture becomes increasingly prevalent in servers and edge devices, the demand for ARM support grows, marking a frontier we'll need to explore in the coming years. How to transition vector search algorithms from x86 to ARM will be our focus in these years ahead.

If you want to know more about FAISS and learn FAISS, I really recommend you to check this Missing Mannuals [1]

References

[1] Faiss: The missing manuals. https://www.pinecone.io/learn/series/faiss/.

[2] An introduction to the power of vector search for beginners. https://hackernoon.com/an-introduction-to-the-power-of-vector-search-for-beginners, 2022.

[3] Practical applications and detailed analysis of xorbits. https://zhuanlan.zhihu.com/p/607200328, 2023.

[4] What is vector search? https://www.algolia.com/blog/ai/what-is-vector-search/?utm_source=google&utm_medium=paid_search&utm_campaign=rl_emea_search_dstalg_nb_dynamic&utm_content=blog_ai_dynamic&utm_term=&utm_region=emea&utm_model=nonbrand&utm_ag=rl&utm_camp_parent=slg&utm_2nd_camp=dstalg&_bt=677640514505&_bm=&_bn=g&gad_source=1&gclid=Cj0KCQiA5rGuBhCnARIsAN11vgSn6g4hkZwwABFxiUwTUzdVa_BevT7Z01F-GY8mAeYR6A68UHsS64kaAjckEALw_wcB, 2023.