ONDC
Open Network for Digital Commerce | presents

Build for Bharat

Supported by #startupindia

Sponsors

Google Cloud
ANTLER
protean Change is growth
Paytm

Powered by

H2S

Team Name: Catindexers

Team Leader Name: Risan Raja

Team Member Names:Risan Raja

Problem Statement Category: Scalable Solution

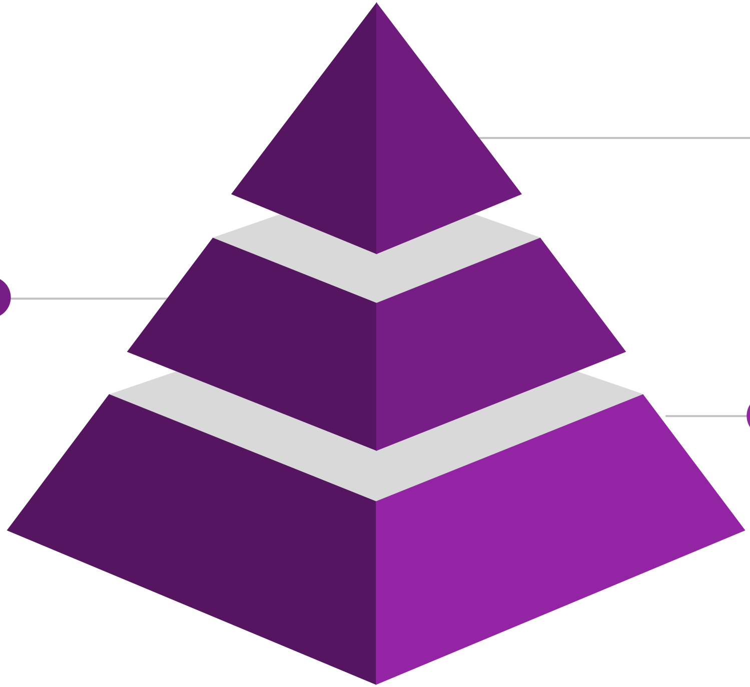
Problem Statement:

- Merchant catalogs can be of different types, as per the category e.g. grocery, fashion, electronics, etc;
- Each catalog will have multiple items with each item (SKU) defined using attribute key/value pairs that defines a particular aspect of the item (e.g. colour, product name, price, etc.);
- Optimal catalog search, using structured query or unstructured text, requires an efficient indexing engine that can support either type of search for catalogs of any size;
- Unstructured queries typically use an inverted index (e.g. elasticsearch) that facilitates efficient retrieval of documents associated with the search query;
- Inverted index can also be used to engineer prompts for catalog LLMs.

Table of Contents

Table of Contents

Destructuring the Problem Statement



Retrieval

1 Large Scale Retrieval with zero shot performance across multiple domains. Easily Scalable and Data Privacy conscious

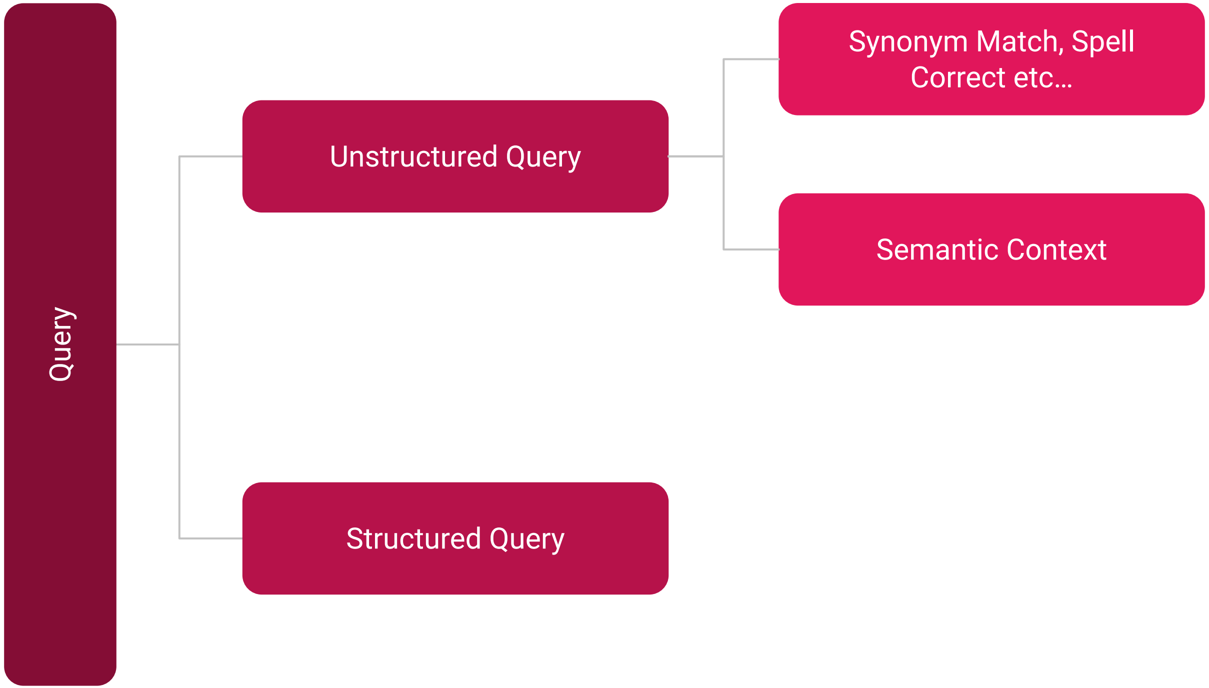
Query Engine

2 Information Retrieval, IR. Borrowing from a few principles the query engine needs to understand the Search Intent, Semantic Context all without the availability/ability to log user interactions extensively.

Indexing

3 As an Ecommerce Protocol which **exists** in the middle, the proposed solution needs to be readily adaptable, extensible and be less dependent on the schema.

Destructuring the Retrieval Paradigm



Neural Search: Beyond the Limitations of Inverted Indexes

Key Issues Addressed

Limited information
Inverted indexes lack the semantic understanding needed for precise results. Neural search bridges this gap by learning both term frequency and semantic relationships.

Static algorithms
Fixed algorithms like BM25 struggle to adapt to evolving user intent. Neural search employs dynamic representations learned from data, ensuring continual improvement and personalization.

Limited customization
Traditional methods offer minimal customization options. Neural search embraces flexible approaches like RAG and LLang Chain Agents, allowing seamless tailoring to specific needs.

Ensure discoverability for all
A decentralized system cannot keep favoring large vendors. To level the playing field, this engine maintains minimal dependency on Marketplace SEO optimization.

Utilize Existing Infrastructure
Buyer apps should be able to utilize the potential of their user's data for personalized searches. Easily adapting clickstream data and vectors, they predict user intent and seamlessly combine product and user profiles for targeted recommendations.

Neural Search: Beyond the Limitations of Inverted Indexes

Traditional inverted indexes are falling short. Their reliance on keywords alone hinders comprehension and adaptability. This submission utilizes cutting-edge neural search paradigms to:

Unlock Semantic Understanding

Go beyond keywords to capture meaning, offering relevant and nuanced results.

Embrace Dynamic Adaptations:

Learn from data, constantly improving relevance and tailoring results to user intent.

Empower Customization:

Leverage flexible models like RAG and LLang Chain Agents, easily adapting to your specific needs.

Enhanced User Experience

Find information faster and easier.

Improved Search Accuracy

Get relevant results, boosting trust and satisfaction.

Future-proof Scalability

Handle growing data volumes effortlessly.

The Open Challenge: Optimizing Search in a Decentralized ONDC World

- Deep dive into PIM systems revealed performance limitations were not mainly constrained by structured query retrieval.
- While queries move smoothly, bottlenecks seldom arise from **network latency** and **data quality**.
- High-quality data, often from bigger vendors, boasts more attributes, potentially biasing search results in their favor.
- Additionally, ONDC's **diverse** vendor landscape requires flexible indexing over strict data conformity, further challenging performance optimization.
- While structure is no longer the hurdle, network speed, **data richness**, and **adaptable** indexing emerge as the new frontiers for efficient catalog retrieval.

> "Structured query will remain relevant in the future, but its "future-readiness" hinges on embracing these adaptations and advancements. By incorporating AI, knowledge graphs, and prioritizing user-friendliness, it can overcome limitations and evolve into a more inclusive, accessible, and powerful tool for accessing and understanding information."

Catalog Indexing Engine

TL;DR Performance Metrics

TensorRT TensorFlow asyncio APACHE ARROW

Thanks to all these guys, they deserve most of the credit

Dataset Used & Simulation Setup

Dataset

Size
The dataset used for prototyping and benchmarking contains over **100** unique key attributes across **300K+** unique products.

Specialization: Fashion
This had the highest cardinality within the available dataset.

Simulation Setup

Taxonomy Resolution
Following the ONDC protocol reference, all L0, L1, L2, L3 and L4 domain(RET) codes were converted from taxonomy specification available on GitHub.

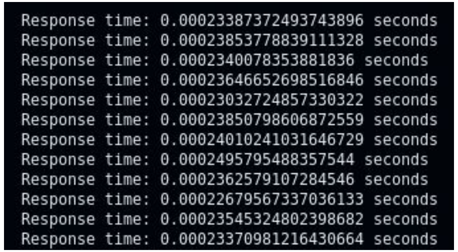
Modeling Real-World Conditions
To simulate real world scenario the catalogs were randomly assigned to 1000 vendors and then hosted on a NoSql Server [**MongoDB**]

Dataset had limited or no unique product ID, hence a product SKU code was randomly generated.

Source: https://github.com/adya0109/cataloguing_products

High-Performance Product Indexing and Retrieval with GPU Acceleration

Local response time within GKE cluster



Ultra-Fast Indexing

Processed **23,200** records per minute, completing full dataset indexing in under **15** minutes on a single node (i9-13900K, 32GB RAM, RTX4090 GPU).

Exceptional Search Performance

Can easily handle **5,000** rps with an response time ranging from of 0.2 ms to 5ms*.

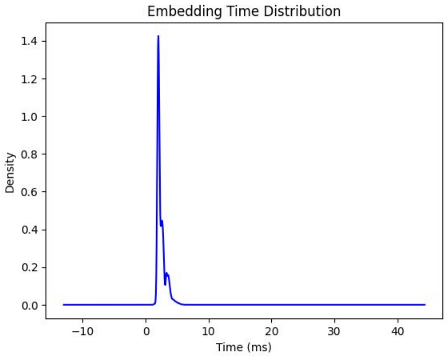
Robustness and Accuracy

Database was able to segment the data on multiple key categories without any payload indexing.

Embracing Parallelism

All the components that are implemented are natively compiled to leverage SIMD parallelism. Even Qdrant[Rust] utilizes all cores natively for all indexing operations

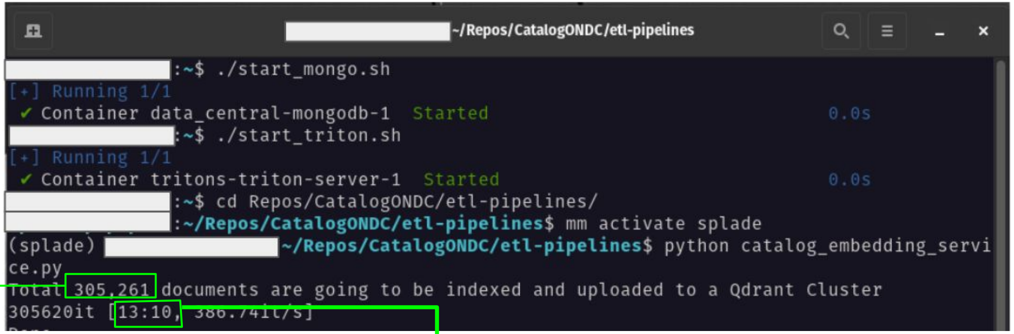
Fun Fact !
Contrary to normal CPU workloads, GPUs excel when more data is pushed, thus as an added bonus the performance of the stack scales linearly to a healthy extent until memory saturation



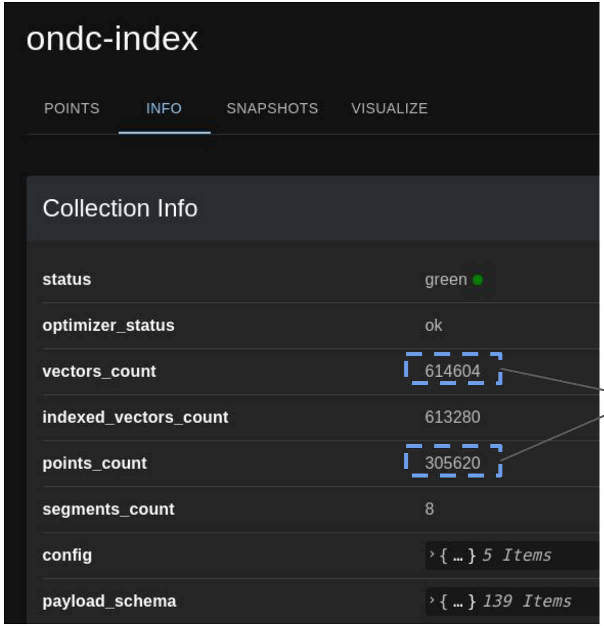
Source: https://github.com/adya0109/cataloguing_products

*This metric does not account for network latency.

Implementation Screenshots



Total Documents = $305,621$ Total Time = $13min\ 10s$



2 vec/ doc
Dense + Sparse

GPU Acceleration: Effortlessly

But why?

Why opt for a GPU-based Deployment and not use a SOTA Embedding API Service?

State of the art

Let

$$SearchLoad = 1000 \text{ Requests/sec}$$

$$AverageSearchQuery = 5 \text{ Tokens}$$

$$Total\ Tokens/sec = 5000 \text{ Tokens/sec}$$

Assume $1/3$ of all requests get served by on premise cache.

$$TotalTokens_{new}/sec \approx 3000 \text{ Tokens/sec}$$

$$Total\ Tokens/day = 3000 \times 24 \times 3600 \quad \text{Tokens/sec}$$

$$= 2.592 \times 10^8 \text{ Tokens/sec}$$

$$TotalCost/day \approx \$34$$

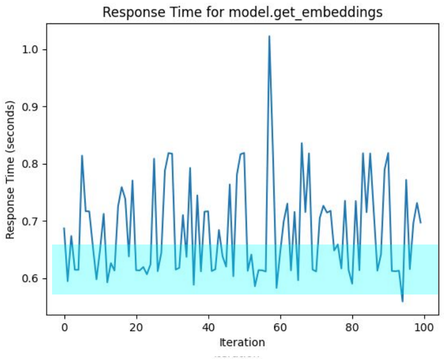
Recalculating this with $1500 Requests/sec$ we get $\$56.1$ with terrible latency and crappy user experience.

Why opt for a GPU-based Deployment and not use a SOTA Embedding API Service?

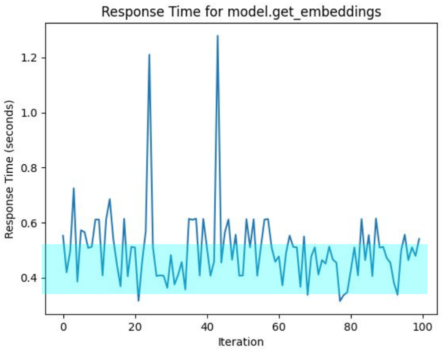
01	Latency	• API Latency Drag: Extra network calls and remote processing impacts responsiveness. • Control Lockout: Limited access to the internal workings hinders optimization for your specific latency needs.	Very Slow and Rate Limited
02	Open Source Scores Big! Beats Proprietary Models at Their Own Game.	• Lack of domain Adaptability or even the ability to fine tune the data based on existing data sources makes this a very isolated system.	Can't Fine Tune
03	No Sparse Retrieval	• Goodbye Slow Searches: Ditching pure dense retrieval speeds up searches and avoids inaccuracies. • Hybrid Hero: Combining dense and sparse techniques (Neural Sparse Retrieval) boosts efficiency and search quality, already benefiting many industries such as ELSER, OpenSearch.	Lack of Integrability

Although these model APIs excel in accuracy, scaling them for efficient distribution of large datasets might require exploring alternative solutions.

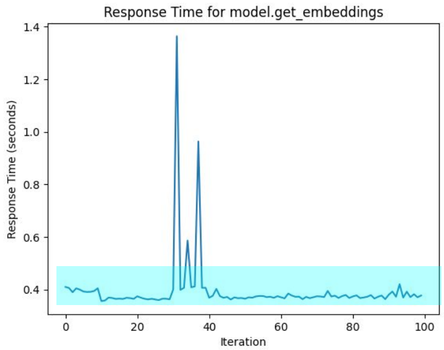
Response time of SOTA Embedding APIs



OpenAI-Large



OpenAI-small



Gecko-003

Average response time [API]	0.4 s
Proposed Soln Avg Response time	0.003 s

Proposed Solution is 130x faster

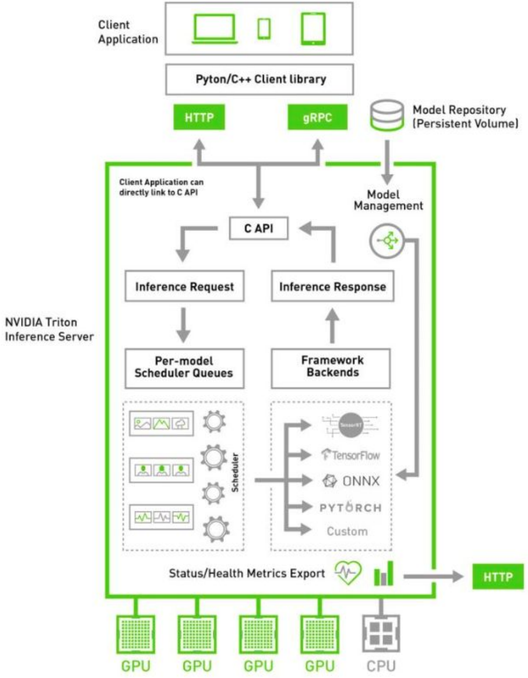
*These results were captured in a Compute Engine VM in "asia-south1b" region

NVIDIA TRITON INFERENCE SERVER Open Source 💗

Auto-Scaling
Deployed Via Google Model Registry for auto-scaling and managed inference.

Natively Compiled
Model execution is done through NVIDIA TensorRT compiled Engines.

Embarrassingly Parallel
Sparse and Dense Embeddings are computed in a single forward pass and assembled with its own response cache.

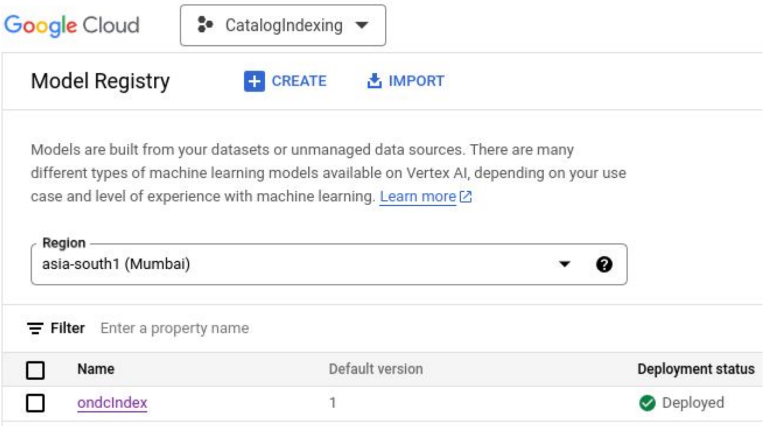


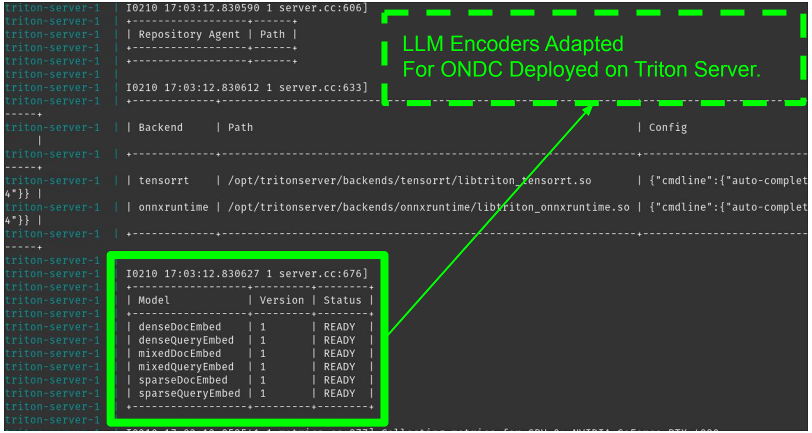
Customized Image
The base image is customized to add model polling and various performance improvements.

No downtime during update
Changing the model binary for a better model will not result in downtime as the server is configured to reload itself and serve without interruption.

Source: https://github.com/triton-inference-server/server

NVIDIA TRITON INFERENCE SERVER

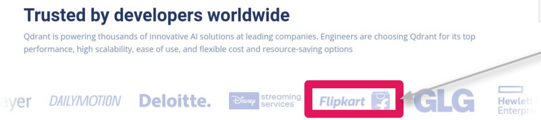




Deployed in Google Cloud as API

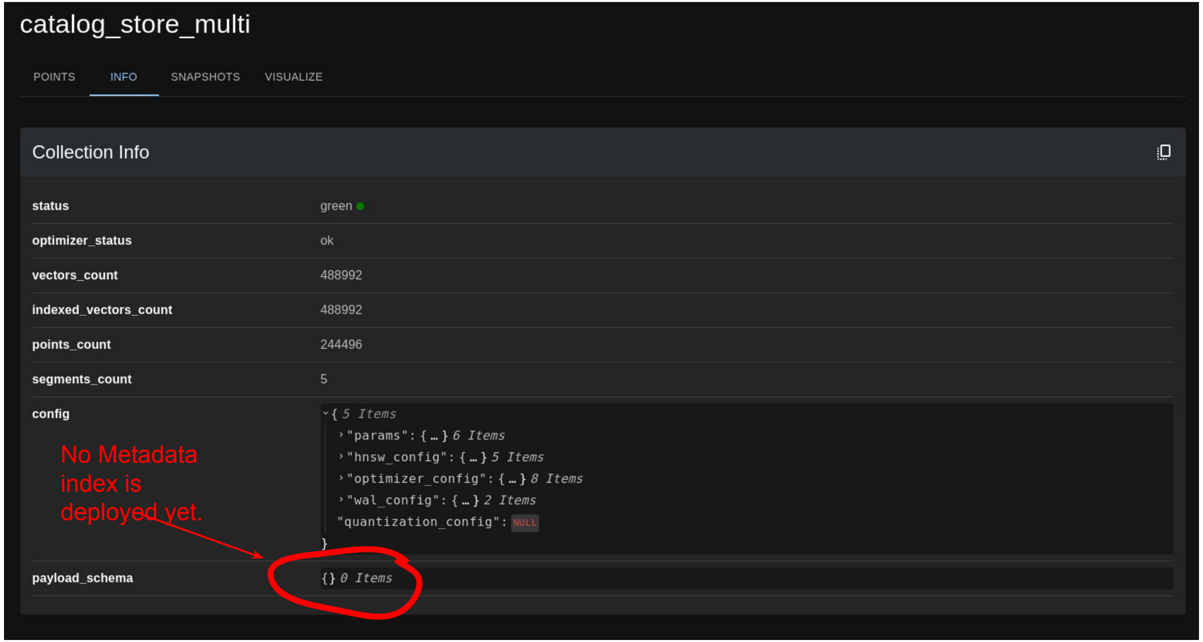
Qdrant VectorDB

Used for Ecommerce in India



- Developed in Rust. .
- Open Source 💗 and one of the fastest vectorDB.
- Text Search Index aka BM25 out of the box support.
- Also Hybrid Vector Search is natively supported i.e Dense + Sparse Vector Search.
- Comes with batteries included for recommendation, product discovery etc.
- Is also integrated with multiple Langchain and Llama Index tools for downstream configurations
- Uses HNSW indexing and also supports all indexing features that a NoSQL Database has multiple vector per endpoint support.
- Google Marketplace and Google GKE Implementation available.

I have not indexed the metadata purposefully to demonstrate the ability of my embedding models to learn product categories without any additional data.



The Database has already learnt the domain grouping without any keyword index.

Sparse Neural Engine:
Neural Power Overtakes Keywords

Example:
To show industry wide adoption of this new frontier.
This model is not open source and requires an explicit licence purchase for use.

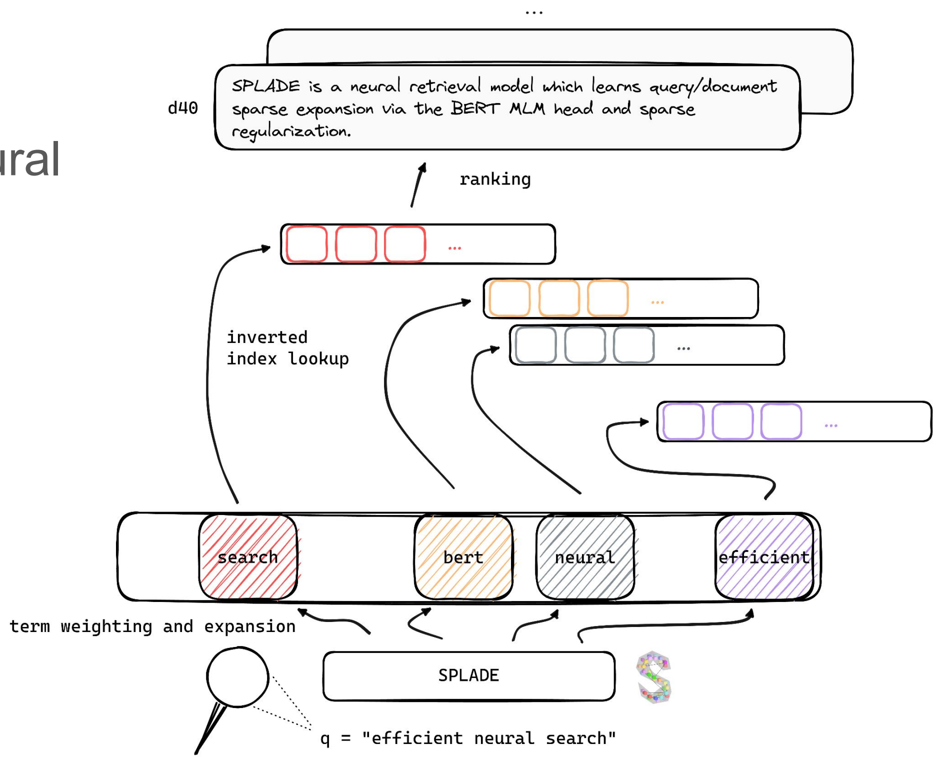
ELSER - ElasticSearch's improvement over BM25

- Elastic Learned Sparse Encoder (ELSER) is a pre-trained text expansion model for semantic search.
- It improves search relevance by capturing relationships between words and understanding context.
- ELSER outperforms other models in zero-shot retrieval tasks.
- Overall the optimized V2 model ingested at a max rate of 26 docs/s, compared with the ELSER V1 max rate of 14 docs/s from the ELSER V1 benchmark, resulting in a 90% increase in throughput.

My Engine does 389 docs/sec

PS:The next slide reveals the secret

Splade: The Neural Retrieval Model



Introducing SPLADE v2

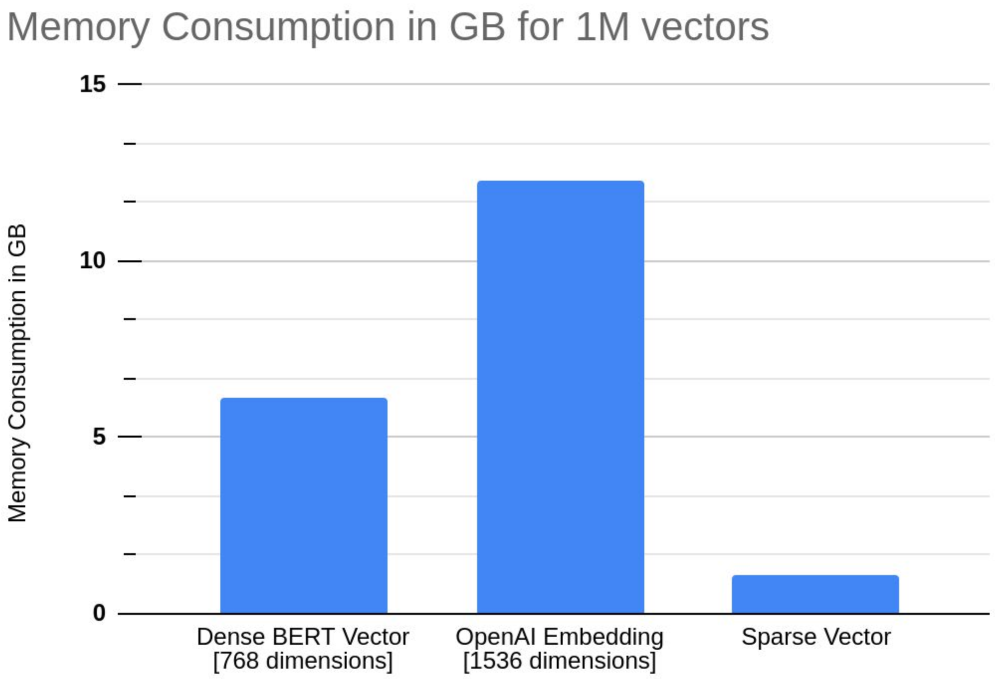
- Inverted Index v2.0 (unofficially 😉).
- Algorithms like BM25(Lucene) focus on keyword filtering using tf-idf(term frequency distribution)
- BM25 struggles to map contextual meaning or lexical similarities. This is easily achieved by a dense embedding algorithm similar to the OpenAI offering however, the tradeoff is in search space storage complexity and computational complexity.
- SPLADE generates Sparse Embeddings i.e it generates vectors of 30522 dimensions whereas an advanced multimodal embedding model generates 1536 dimensionional vectors.
- But most of it is zero, so effectively storing it in sparse matrix the storage space is reduced exponentially.

Table 1: Evaluation on MS MARCO passage retrieval (dev set) and TREC DL 2019.

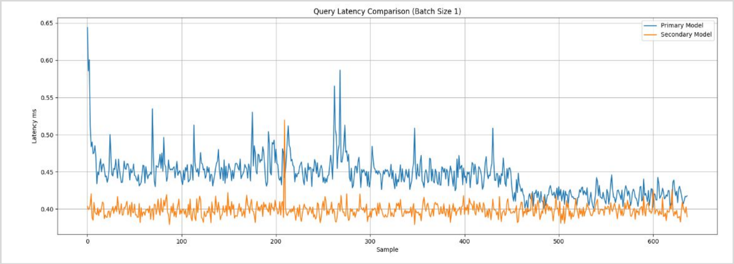
model	MS MARCO dev MRR@10	MS MARCO dev R@1000	TREC DL 2019 NDCG@10	TREC DL 2019 R@1000
Dense retrieval				
Siamese (ours)	0.312	0.941	0.637	0.711
ANCE [29]	0.330	0.959	0.648	-
TCT-ColBERT [16]	0.359	0.970	0.719	0.760
TAS-B [11]	0.347	0.978	0.717	0.843
RocketQA [24]	0.370	0.979	-	-
Sparse retrieval				
BM25	0.184	0.853	0.506	0.745
DeepCT [4]	0.243	0.913	0.551	0.756
doc2query-T5 [20]	0.277	0.947	0.642	0.827
SparTerm [1]	0.279	0.925	-	-
COIL-tok [9]	0.341	0.949	0.660	-
DeepImpact [18]	0.326	0.948	0.695	-
SPLADE [8]	0.322	0.955	0.665	0.813
Our methods				
SPLADE-max	0.340	0.965	0.684	0.851
SPLADE-doc	0.322	0.946	0.667	0.747
DistilSPLADE-max	0.368	0.979	0.729	0.865

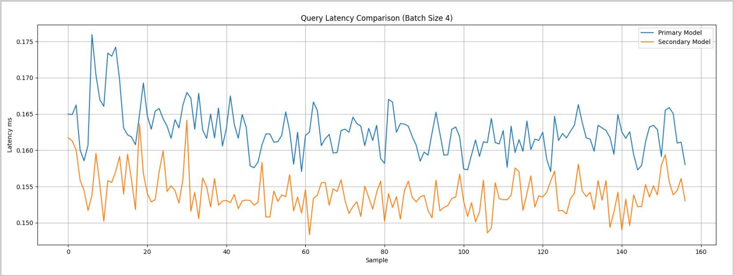
on the MS MARCO dev set as well as TREC DL 2019 queries; (2) the results are competitive with state-of-the-art dense retrieval methods.

SPLADE v2: Why Sparse Vectors can be helpful?



Performance boost of 2.6x when search load increases.





- Models that use GPUs 💞 higher loads which is what a scalable system wants.
- Triton inference Server has been configured to use dynamic batching.
- When requests arrive it starts dynamically batching and the throughput goes 4x.
- Implementing Splade directly is very slow in fact the indexing performance is about 10-15 docs/sec.
- My adaptation increases that speed to **20x** of it.
- SPLADE is a class of models which can be adapted to any use-case. I customized by fusing a few of their attention layers and enabled mixed precision execution which was later optimized using a TensorRT engine.

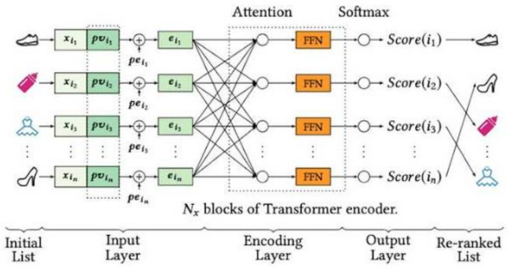
Data Privacy

Downstream Search Personalization

Consumers of this search retrieval have the flexibility to further personalize the search results of their user without the need to share any user data with the indexing system.

For Example

Buyer Apps could retrain a simple attention layer to effectively predict or model their user's actions by combining with ONDC Index. This is totally more feature richful as ONDC Index focuses more on discovery whereas the buyer app could focus on customizability.



Encoded Querying Interface

If a buyer app wants to query the indexing engine and does not want to share their unstructured query data of their user base eg. incognito search etc. This is still possible and was one of the first few modules that I developed. **TL;DR** Before every embedding call there is a tokenization step, this module written by HuggingFace runs on python or rust and it takes roughly 600µs. Using TF Lite, I wrote a tokenizer which is compatible with this search system which can be integrated into any mobile,web or server runtime to encode search query at **1/5th** of the speed i.e at 50µs and just 1.9mb static size which goes to mere Kilobytes when served through a gzip compression layer.

This to a very good extend ensures ONDC Index to be isolated and still be a viable indexing solution that respects the data privacy protocol of it s

Artefacts and Assumptions

Artefacts

- Neural Embedding Engine
    - **Designed** & **Deployed** on Google Cloud Model Repository using a custom image.
    - `Docker-compose` Folder within the repository contains the needed script to launch the complete runtime as services within a development environment.
- Database Runtime
    - Available to deploy on Kubernetes cluster or use a managed deployment from Vendor
- Query Engine
    - Even though the model is completely a custom solution, I have taken good efforts to integrate it with GCP services, Thus comes with added benefits of to be able to use Google Vertex APIs for querying which is included with the deployment.

Assumptions

- Data needs to exist.
- Dataset used in prototype doesn't confirm to the enums protocol in the ONDC layer but I can confidently attest that this is not a limitation.
- This solution aims to serve as an index not as a repository of catalogs, the architecture detailed earlier will show a data sink which can be an in-house data lake warehouse or it can live within each vendors ecosystem.

Datasets and Model Files

Datasets

The GitHub Repository contains a `datasets` folder which has the entire **816mb** dataset which is compressed using `7zip`. This file can be loaded into memory as it is after decompression or used to import data into any `ODM` Database to recreate the prototype dataset.

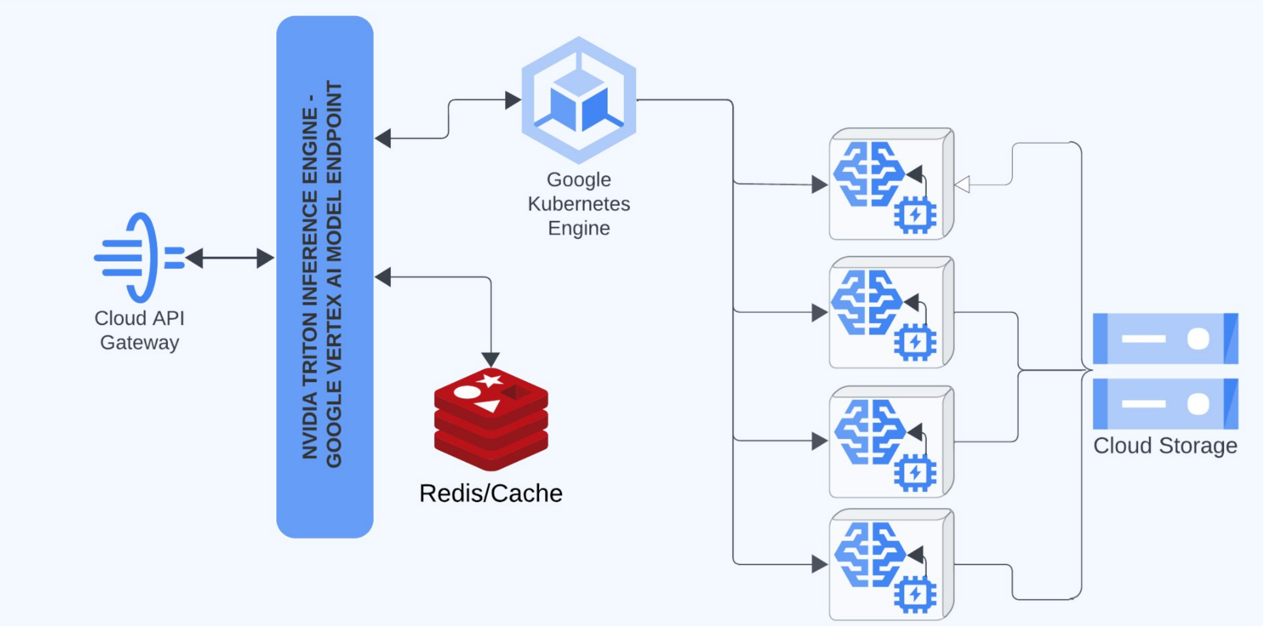
Model Files

Folder named `model_repository` is hosted within the same repository and is also hosted within a cloud storage bucket. This folder is large and is approximately `1GB` in size.

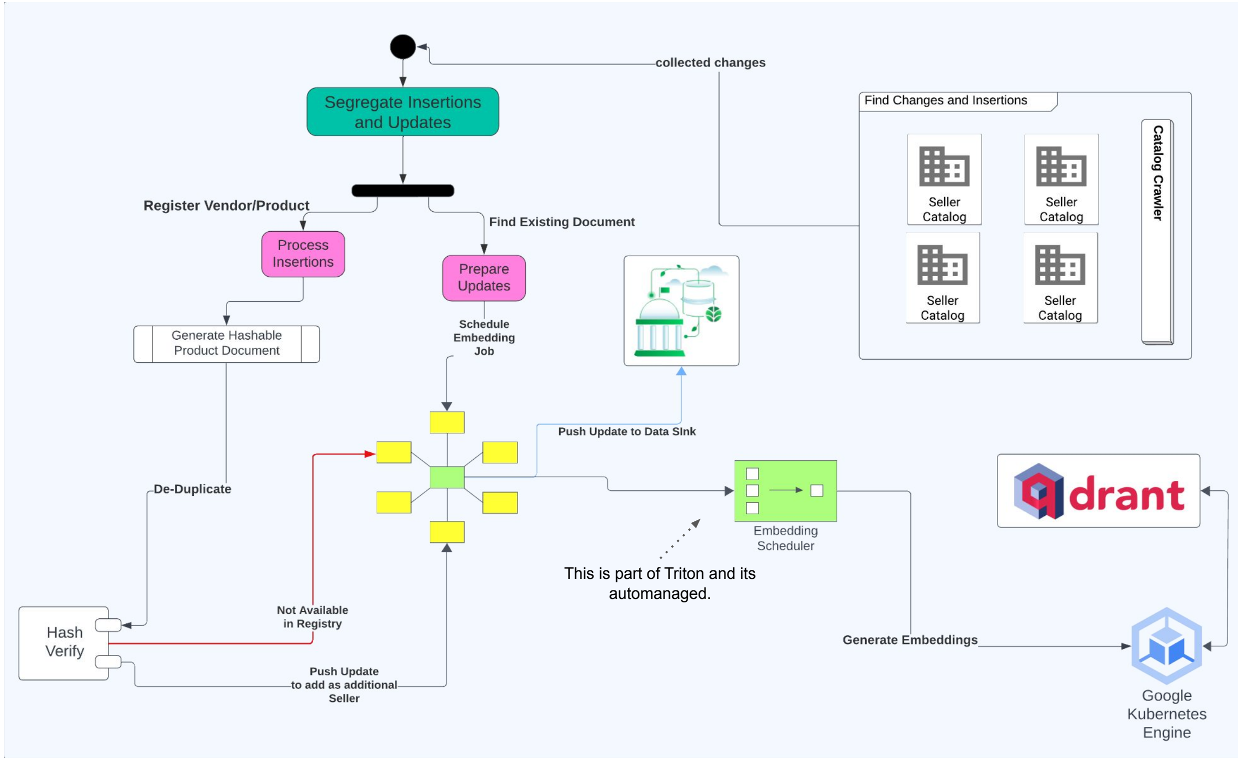
Customization and Deployment Options

Cloud Based Architecture blueprint leveraging Google Cloud Platform

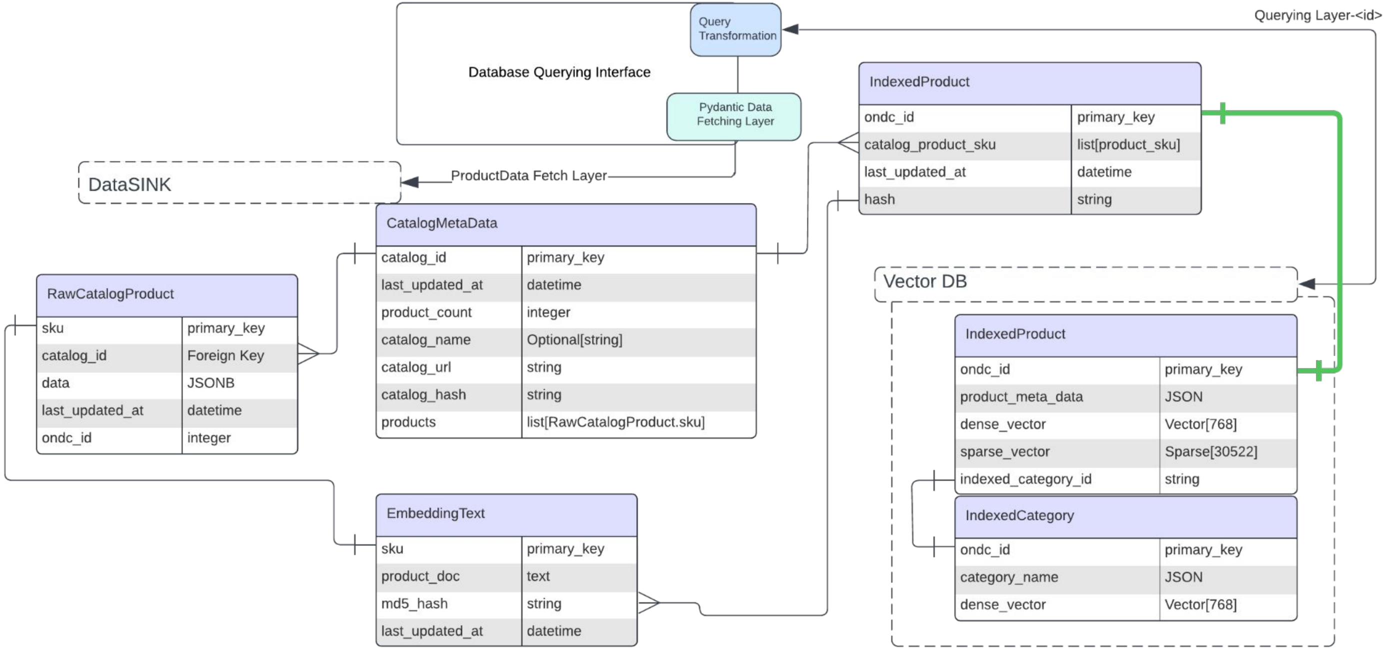
Embedding Engine Module

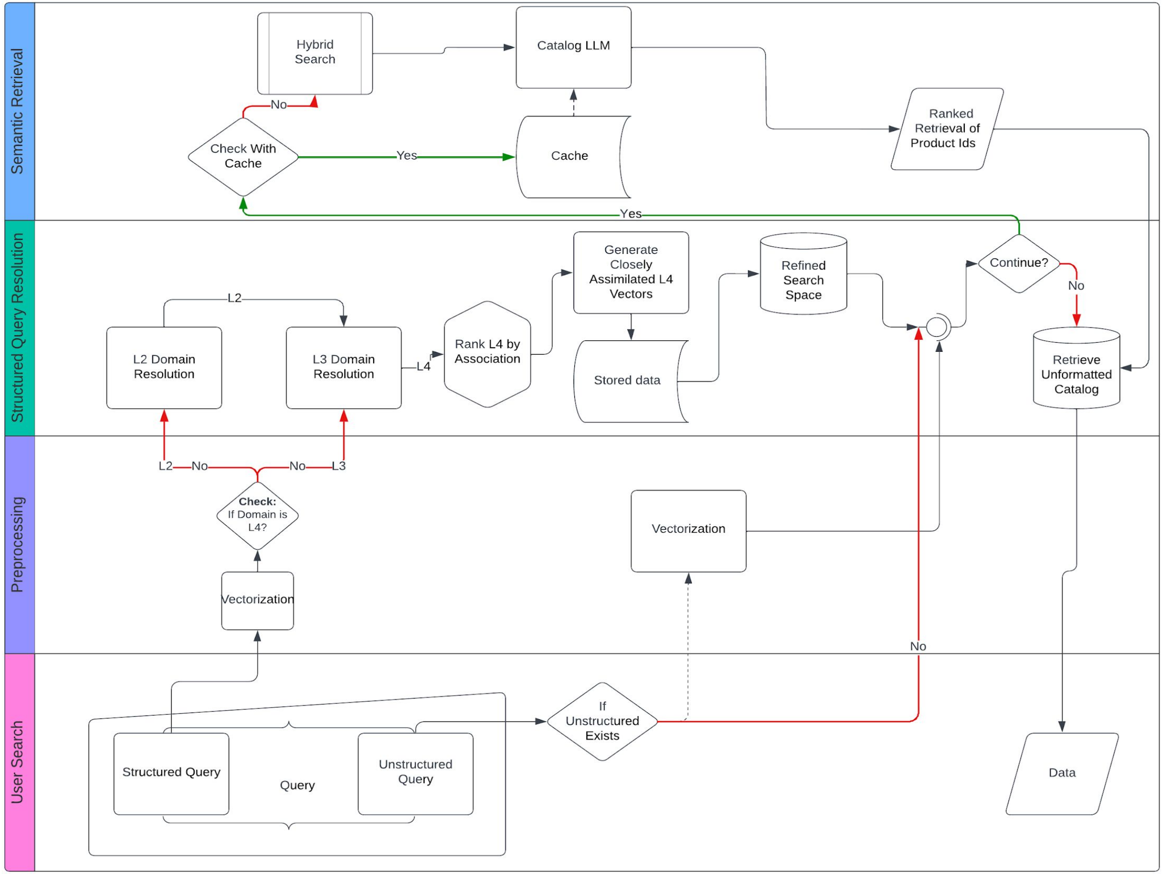


Catalogs ETL Pipeline

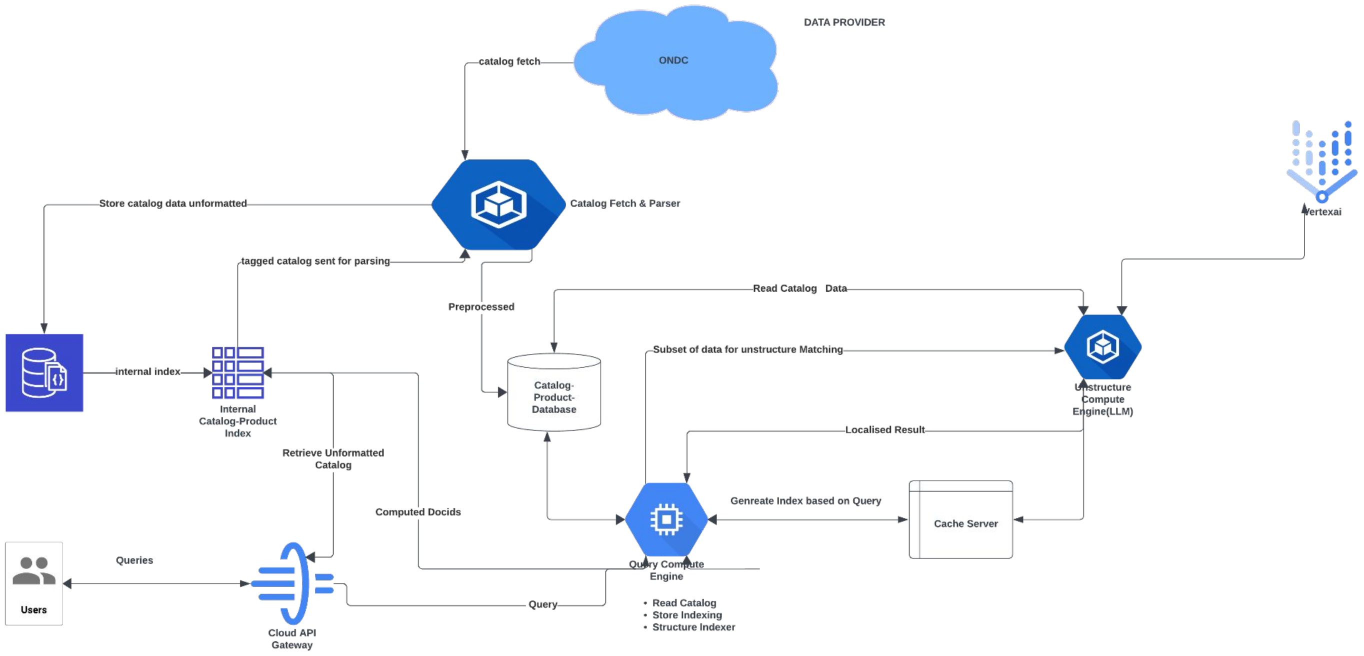


Generalized ER Model

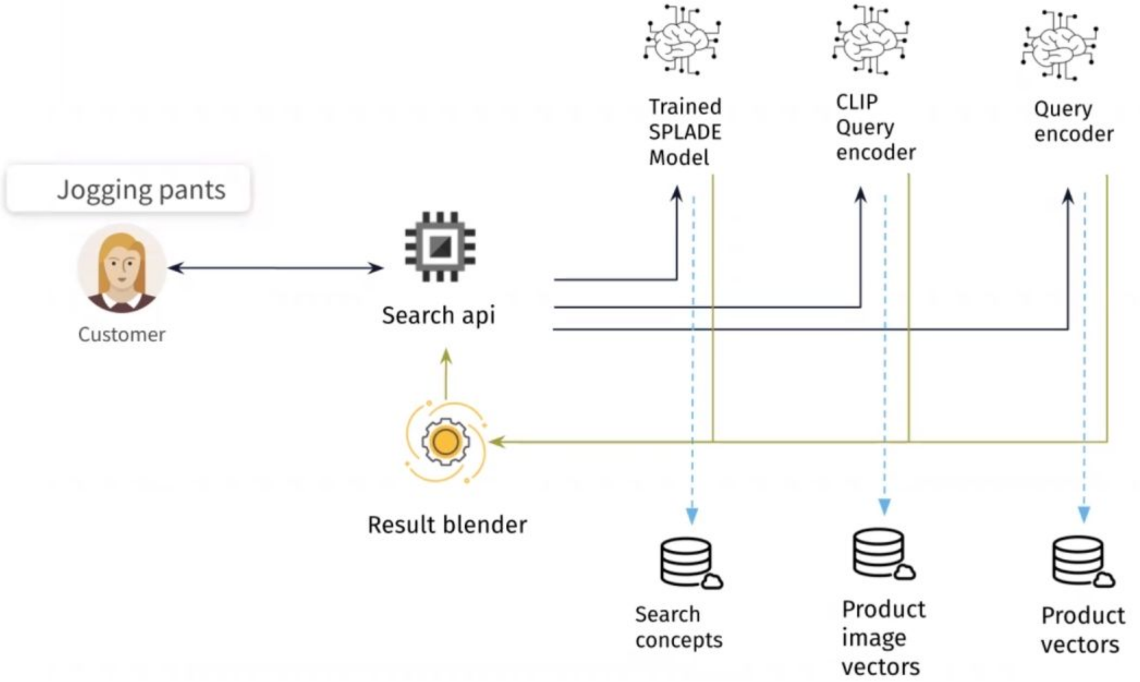



Semantic Retrieval
Structured Query Resolution
Preprocessing
User Search
Hybrid Search
Catalog LLM
Ranked Retrieval of Product Ids
No
Check With Cache
Yes
Cache
Yes
Generate Closely Assimilated L4 Vectors
Refined Search Space
Continue?
No
L2
L2 Domain Resolution
L3 Domain Resolution
L4
Rank L4 by Association
Stored data
Retrieve Unformatted Catalog
L2
No
No
L3
Check: If Domain is L4?
Vectorization
Vectorization
No
Structured Query
Query
Unstructured Query
If Unstructured Exists
Data

Brief Overview



Integration Blueprint for Neural Search system



Detailed Architecture Document

Click here

A picture speaks a thousand words! This is the reason I have embedded Flow diagrams for reference, However to cut short the length of an initial idea submission the detailed architecture document and recommendations are kept separate.

ONDC Open Network for Digital Commerce | presents

Build for Bharat

Supported by #startupindia

Sponsors Google Cloud ANTLER protean Change is growth Paytm

Powered by H2S HACK2SKILL

THANK YOU