Unlock Highly Relevant Search with AI

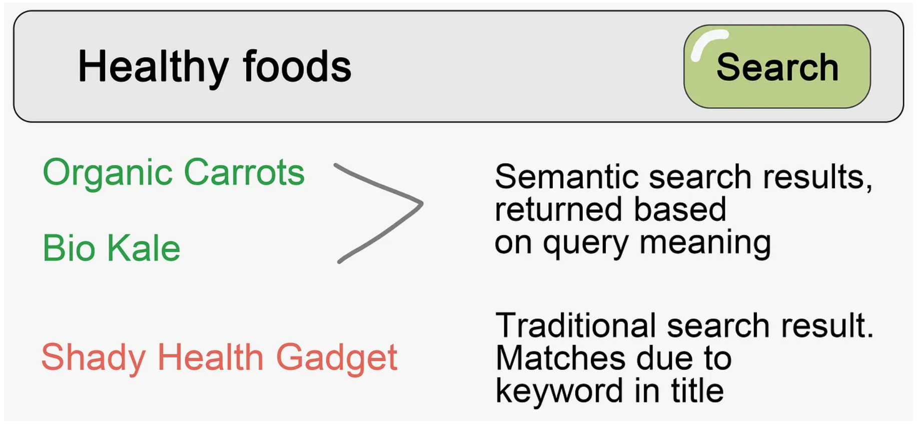
BYTEBYTEGO
NOV 30, 2023 · PAID

183 12 12 Share

We are considering launching a new 'How We Built This' series where we take a behind-the-scenes look at how innovative companies have created scalable, high-performing systems and architectures. Let us know if this is something you'd be interested in reading!

In today's issue, we are fortunate to host guest contributor Marcelo Wiermann, Head of Engineering at Y Combinator startup Cococart. He'll be sharing insights into Cococart's use of semantic search to power new in-store experiences.

How agile teams can leverage LLMs, Vector Databases, and friends to quickly launch cutting-edge semantic search experiences for fame & profit


Healthy foods
Search
Organic Carrots
Bio Kale
Semantic search results, returned based on query meaning
Shady Health Gadget
Traditional search result. Matches due to keyword in title

It's remarkable how so many things are made better with great search. Google made it easy for normal folks to find whatever they needed online, no matter how obscure. IntelliJ IDEA's fuzzy matching and symbol search helped programmers forget the

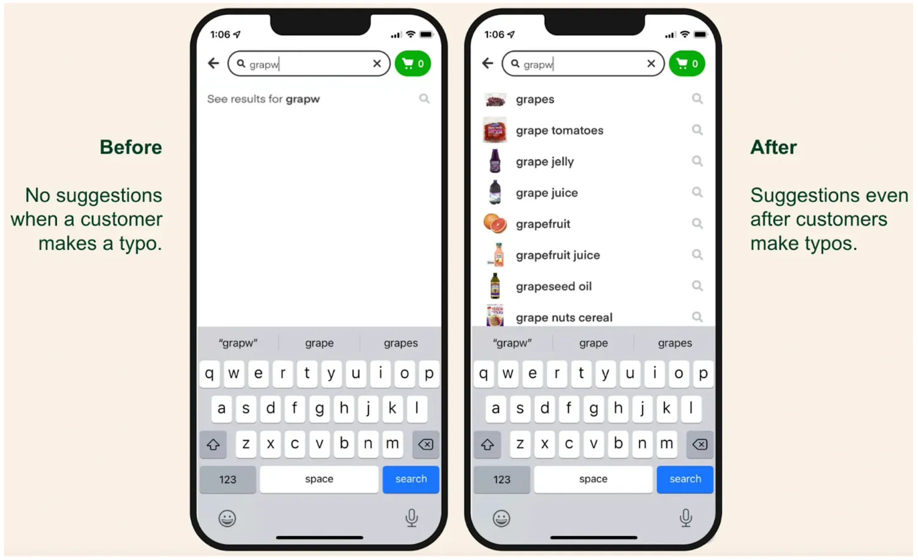
directory structure of their code bases. AirTag added advanced spatial location capabilities to my cat. A well-crafted discovery feature can add that "wow" factor that iconic, habit-forming products have.

In this post, I'll cover how a fast-moving team can leverage Large Language Models (LLMs), Vector Databases, Machine Learning, and other technologies to create a wow-inspiring search and discovery experience with startup budget and time constraints.

Semantic Search

Semantic Search is a search method for surfacing highly relevant results based on the **meaning** of the query, context, and content. It goes beyond simple keyword indexing or filtering. It allows users to find things more naturally and with better support for nuance than highly sophisticated but rigid traditional relevancy methods. In practice, it feels like the difference between asking a real person or talking to a machine.

Tech companies across the world are racing to incorporate these capabilities into their existing products. Instacart published an extensive article on how they added semantic deduplication to their search experience. Other companies implementing some form of semantic search include eBay, Shopee, Ikea, Walmart, and many more.

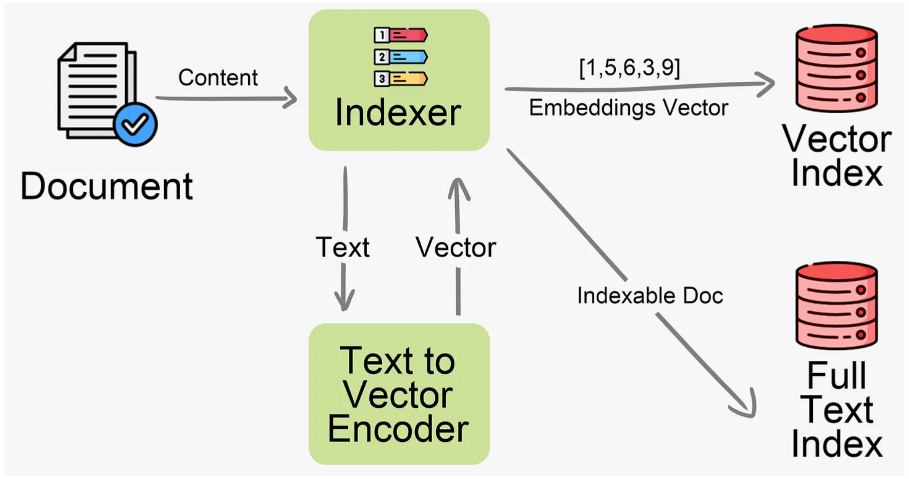


Source: Instacart

The motivation for embracing semantic search is simple: more relevant results lead to happier customers and more revenue. Discovery, relevancy, and trustworthiness are some of the hardest problems to solve in e-commerce. An entire ecosystem of solutions exists to help companies address these challenges.

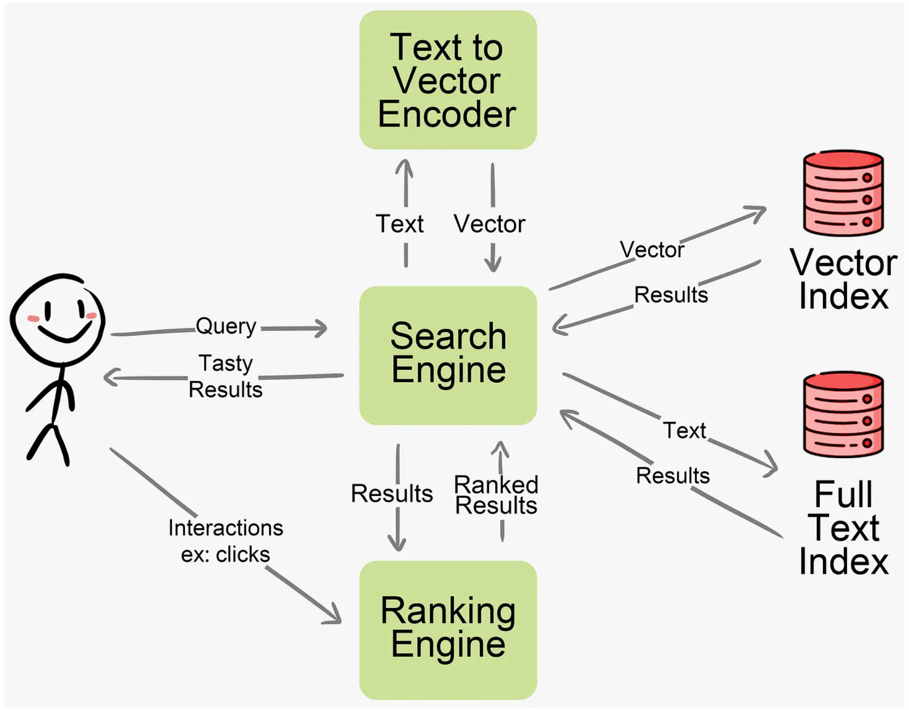
Many solutions today rely on *document embeddings* - representing meaning as vectors. Since semantic search alone may not provide sufficient relevant hits, traditional full-text search is often used to supplement resuts. A feedback loop based on user interactions (clickes, likes, etc.) provides input to continuously improve relevancy.

The key processes are: **indexing**, **querying**, and **tracking**



Document indexing

Indexing is done by converting a document's content to an embeddings vector through a text-to-vector encoder (e.g. OpenAI's *Embeddings* API). The vectors are inserted into a vector database (e.g. Qdrant, Milvus, Pinecone). Text-to-vector encoding models like sentence-transformers convert text snippets into numeric vector representations that capture semantic meaning and similarities between text. Documents are also indexed in a traditional full-text search engine (e.g. Elasticsearch)



Query and feedback loop

Querying relies on encoding incoming queries into vectors, preferably using the same encoder as indexing. These vectors are used to query the vector database. These results are combined with traditional full-text search results and re-ranked for improved relevancy. This combination of semantic and full-text search is referred to as "hybrid search".

Search result re-ranking relies on a mix of machine learning models and heuristics to optimize relevance. It is often a complex problem.

Tracking involves capturing important user interactions - e.g. clicking on results, liking items, etc. These events are used to update the machine learning models that power re-ranking. This creates a feedback loop, leveraging user behaviors to continuously improve search relevancy.

MVP

Most companies, especially startups given their different constraints, should not copy the architectures of large tech behemoths. However, we can borrow a few tricks to implement a reasonably good solution on a limited budget and timeline.

The approach here combines semantic search and full-text results, re-ranking for relevancy, and tracking users - a *hybrid search* (full text + semantic) and *learn-to-rank* (closed loop re-ranking) system.

Vector similarity search will be handled by Qdrant, a fast vector database with great developer experience. Qdrant is easy to get started with, has recently raised funding, and has an actively engaged team improving the product based on community feedback. This makes it a great starting point for exploring vector search.

Text embeddings will be generated by OpenAI's Embeddings API. Their text-embedding-ada-002 model provides high quality embeddings. There are alternatives like Cohere or even self-hosted models, like all-MiniLM-L6-v2 that could work too. I recommend starting with cloud-hosted ones for simplicity though.

ElasticSearch (ES), a fully-featured, mature search system, will handle full-text search. Interestingly, Elastic seems to be deeply interested in vector search capabilities. Definitely something to keep an eye on.

Re-ranking will be handled by Metarank. Search re-ranking is a complex problem, so I was pleasantly surprised to find this relatively user-friendly open source solution available. As the re-ranking task can be somewhat complex, especially when fine-tuning models, I'd advise getting basic indexing and search functionality working first. Once that foundation is in place, re-ranking can be added on top. This allows for an iterative approach - get simple search running, then make improvements by integrating re-ranking. Start modestly before building up to more advanced relevancy ranking approaches over time.

NOTE: the reason we typically combine semantic search and full-text results is that semantic search may not provide sufficient results. Your mileage may vary. Feel free to experiment with different combinations!

Indexing

The first step is to **index items** for search. Items could be anything - products, locations, sellers, images, etc. Items are stored in the main database (MongoDB in

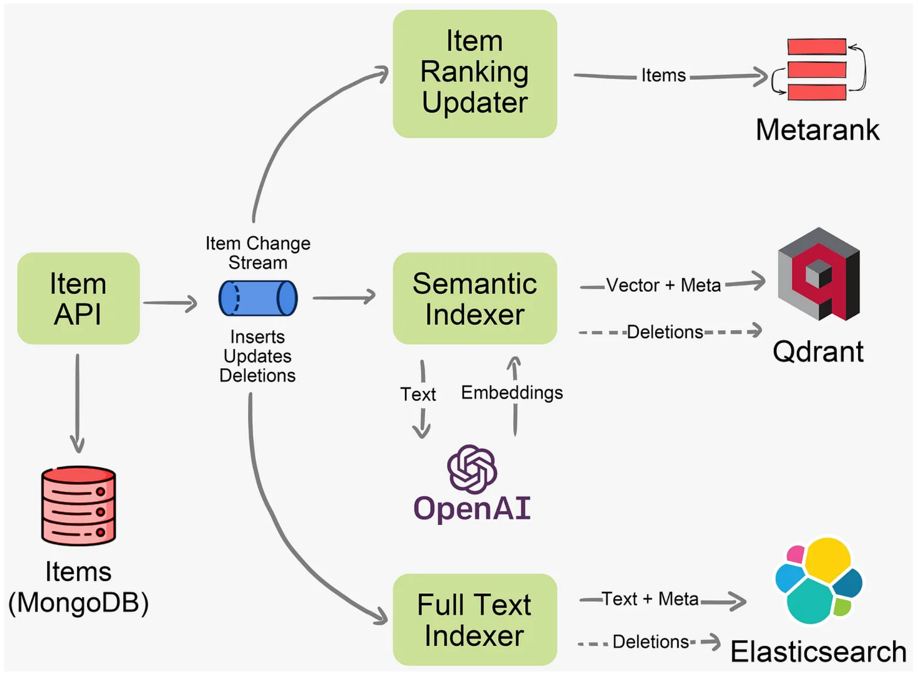
this example), which serves as the source of truth.

To index items, we will push changes (inserts, updates, and deletions) into a **Kafka** topic, which will then be consumed by multiple processors. Redis queues would work too. This is a pull-type architecture, and it is quite robust in production. You could call each system directly from the Items API, but this can easily lead to partially indexed items.

Three processors consume this topic:

- **Full Text Indexer**: updates the full-text index (ElasticSearch in our example).
- **Semantic Indexer**: assembles text strings from item changes (e.g., product titles + descriptions), encodes them into vector embeddings, and saves them to the vector database (Qdrant).
- **Item Ranking Updater**: updates Metarank's item metadata.

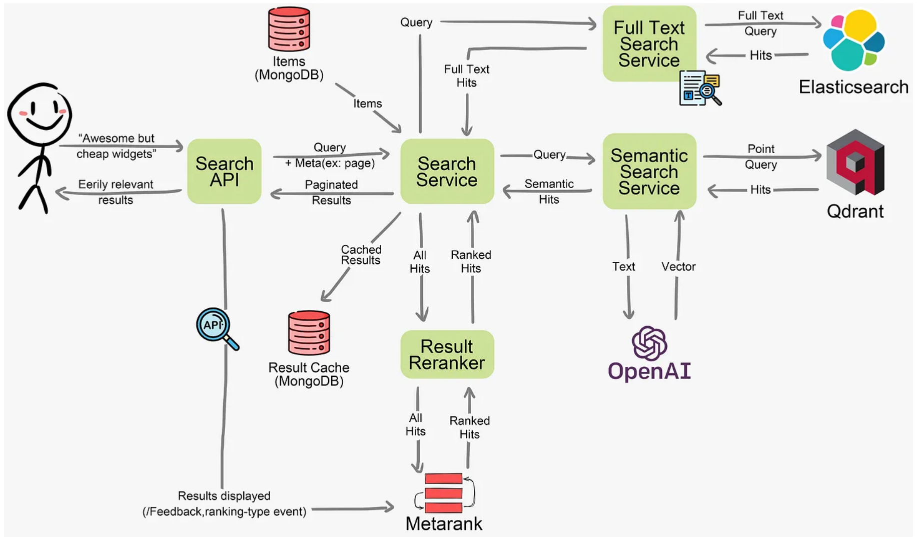
Note that none of these systems will generally contain the full item data, which is usually unnecessary (and costly)



Item indexing

Querying

This is where things get interesting



Querying

Our Search API works in four major processes:

- **Search**: the first step consists in sending the query to our two providers - Full Text (ElasticSearch) and Semantic (Qdrant).
  - The semantic search provider needs to first convert the query into vector embeddings, much like the semantic indexer. Vector similarity search works by taking the vector embedding of the search query text, then identifying items indexed with text vector embeddings most similar based on geometric comparisons and distance calculations between vectors in the high-dimensional space. This allows matching semantic relevance, not just keywords.
  - This is a good place to perform some pre-filtering, such as applying simple criteria (e.g., category IDs), geographical radius, etc.

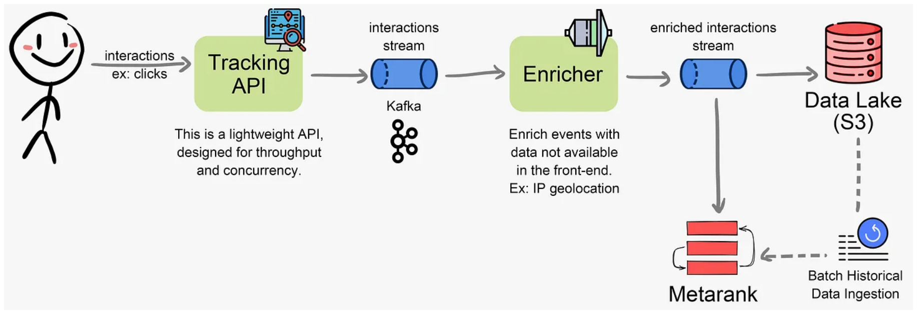
- **Re-ranking**: once the search service collects all hits, it routes the full set to the Result Re-ranker, which re-orders them for relevancy using Metarank.
  - Metarank does not work without some initial data. You can solve this cold start problem by having a simple ranker as a fallback. I had good results with just adding the top 3 semantic hits at the top and the full-text ones after. This generally doesn't look half bad, and it's just for a little while as real users start using the system. For a more sophisticated approach, check out this amazing article by Metarank's co-founder.
- **Result Generation**: search hits do not generally contain all item information. To return a complete, human-readable result set, we will "hydrate" our ranked hit list into a ranked item list.
  - This is a good place to cache the result list with a relatively short TTL. This helps a lot with things like **pagination** and **sorting**.
  - You may also want to store **search session** metadata here. This is usually **not** the same as a user session. This allows you to group search and result queries later for analysis and ML, and is a parameter that we can pass to Metarank to improve ranking relevancy.
- **Result Tracking**: once we have a fully hydrated and paginated set of results, we will update Metarank with the exact list of items we are showing users.
  - This could also be accomplished from the front end if we want to keep this API simple and fast.

NOTE: I mapped the different components for the sake of clarity. If you are operating in a small team or startup, I would not recommend creating tons of microservices. Keeping things simple early on allows you to iterate quickly.

Tracking

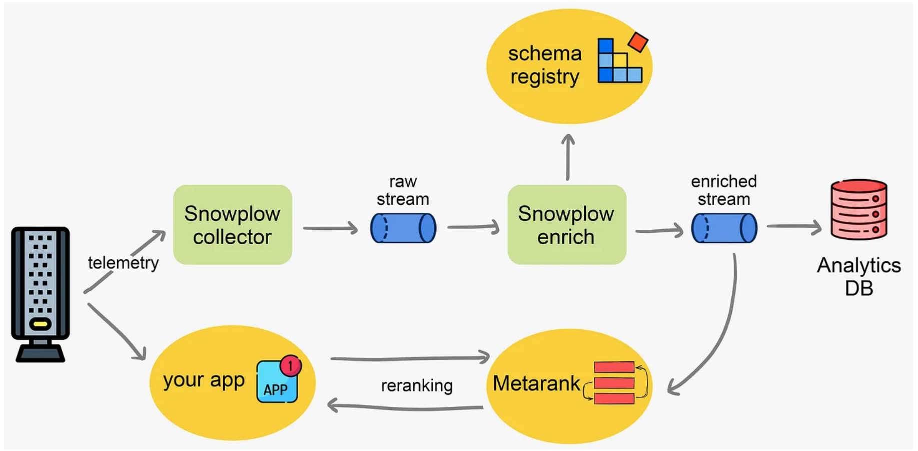
We need to track how users interact with our results so that we can create a *learn-to-rank* feedback loop. Examples of tracked interactions include viewing results, clicking them, liking them, etc. At a minimum, result clicks should be tracked.

This is accomplished by sending these events to a lightweight tracking API, which quickly pushes them to a Kafka topic (a Redis queue or similar would work as well). Since not all data is available in the front end, we may want to first enrich events with internal metadata. Finally, this enriched event is shipped to Metarank (directly reading from Kafka or through its HTTP API) and to a data lake for future analysis


interactions ex: clicks
Tracking API
This is a lightweight API, designed for throughput and concurrency.
interactions stream
Kafka
Enricher
Enrich events with data not available in the front-end. Ex: IP geolocation
enriched interactions stream
Data Lake (S3)
Metarank
Batch Historical Data Ingestion

Interaction tracking

I am also keeping this at a high level since building such data pipelines by hand nowadays is usually not recommended, especially for small teams. There are plenty of great tools available (e.g., Segment, Rudderstack, Fivetran, dbt, Snowplow, etc.) that can help you build a strong data pipeline or plug Metarank into an existing one. For example, here is a chart showing Metarank's native integration with Snowplow:



Native Metarank and Snowplow integration. Source: metarank.ai

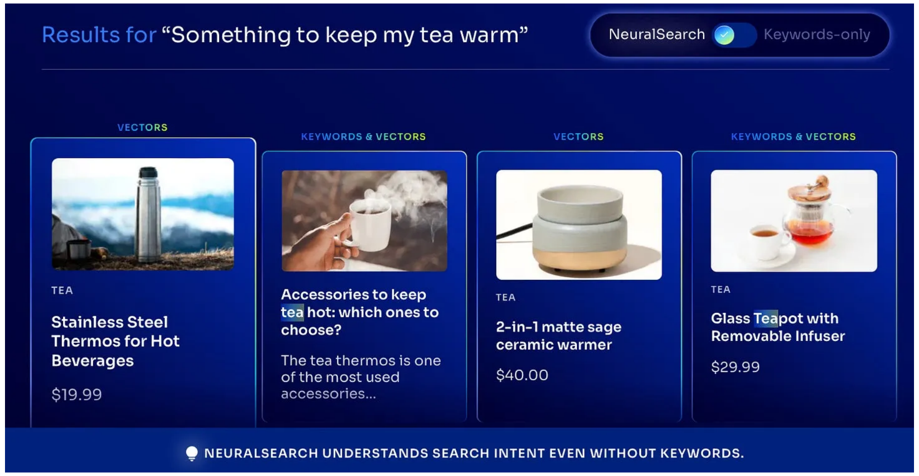
NOTE: Building a full learn-to-rank system adds significant complexity. I suggest only implementing this if you are committed to a comprehensive ranking approach. For many use cases, the simpler hybrid search system may be sufficient, especially early on.

What's Next

Creating great search experiences has never been easier. Semantic hybrid search is a solid first step towards providing search results beyond simple keyword matching. Customers are slowly but surely becoming used to this increased level of relevancy, which raises the bar for everyone.

Luckily for small teams, the tooling ecosystem is developing rapidly. Many search tools are striving for convergence. For example, Elasticsearch has an early vector search capability, and Qdrant added more filtering functions (like match and geo-radius). Metarank ships with a semantic similarity recommendation model. There may soon be an all-in-one system with good baseline performance across the board.

You can also skip much of this work and use Algolia NeuralSearch if you have a small catalog, a moderate traffic (e.g., B2B SaaS dashboard), or want to quickly ship a proof of concept to test user response. It's a power, fully hosted solution from a company focused on search.


Results for "Something to keep my tea warm"
NeuralSearch
Keywords-only
VECTORS
TEA
Stainless Steel Thermos for Hot Beverages
$19.99
KEYWORDS & VECTORS
Accessories to keep tea hot: which ones to choose?
The tea thermos is one of the most used accessories...
VECTORS
TEA
2-in-1 matte sage ceramic warmer
$40.00
KEYWORDS & VECTORS
TEA
Glass Teapot with Removable Infuser
$29.99
NEURALSEARCH UNDERSTANDS SEARCH INTENT EVEN WITHOUT KEYWORDS.

Source: Algolia

There are many options to get started, and few excuses left to provide your users a *meh* search experience!

183 Likes · 12 Restacks

12 Comments

Write a comment...

Tapshil Nov 30, 2023 Liked by Alex Xu

Yes i am interested in 'How we built this' series. Thanks.

LIKE (11) REPLY SHARE

Artur Nov 30, 2023 Liked by Alex Xu

> Let us know if this is something you'd be interested in reading!

OMG yes yes yes!

LIKE (4) REPLY SHARE

10 more comments...