



Search: a new era

David Pilato | [@dadoonet](#)

Agenda

Commercial

- "Classic" search and its limitations
- ML model and usage
- Vector search or hybrid search in Elasticsearch
- OpenAI's ChatGPT or LLMs with Elasticsearch

Elasticsearch

You Know, for Search



Elasticsearch

Lucene



66

These are not the droids
you are looking for.

```
GET /_analyze
{
  "char_filter": [ "html_strip" ],
  "tokenizer": "standard",
  "filter": [ "lowercase", "stop", "snowball" ],
  "text": "These are <em>not</em> the droids
          you are looking for."
}
```

These are `not` the `droids you are looking` for.

```
{ "tokens": [{
  "token": "droid",
  "start_offset": 27, "end_offset": 33,
  "type": "<ALPHANUM>", "position": 4
},{
  "token": "you",
  "start_offset": 34, "end_offset": 37,
  "type": "<ALPHANUM>", "position": 5
}, {
  "token": "look",
  "start_offset": 42, "end_offset": 49,
  "type": "<ALPHANUM>", "position": 7
}]}
```




Semantic
search
≠
Literal
matches



similarweb

**YOU'RE COMPARING
APPLES TO NECTARINES**





TODAY

🔍 *X-wing starfighter squadron*

TOMORROW

🔍 *What ships and crews do I need to destroy an almost finished death star?
Or is there a secret weakness?*

Elasticsearch

You Know, for Search

Elasticsearch

You Know, for **Vector** Search



What is a
Vector?



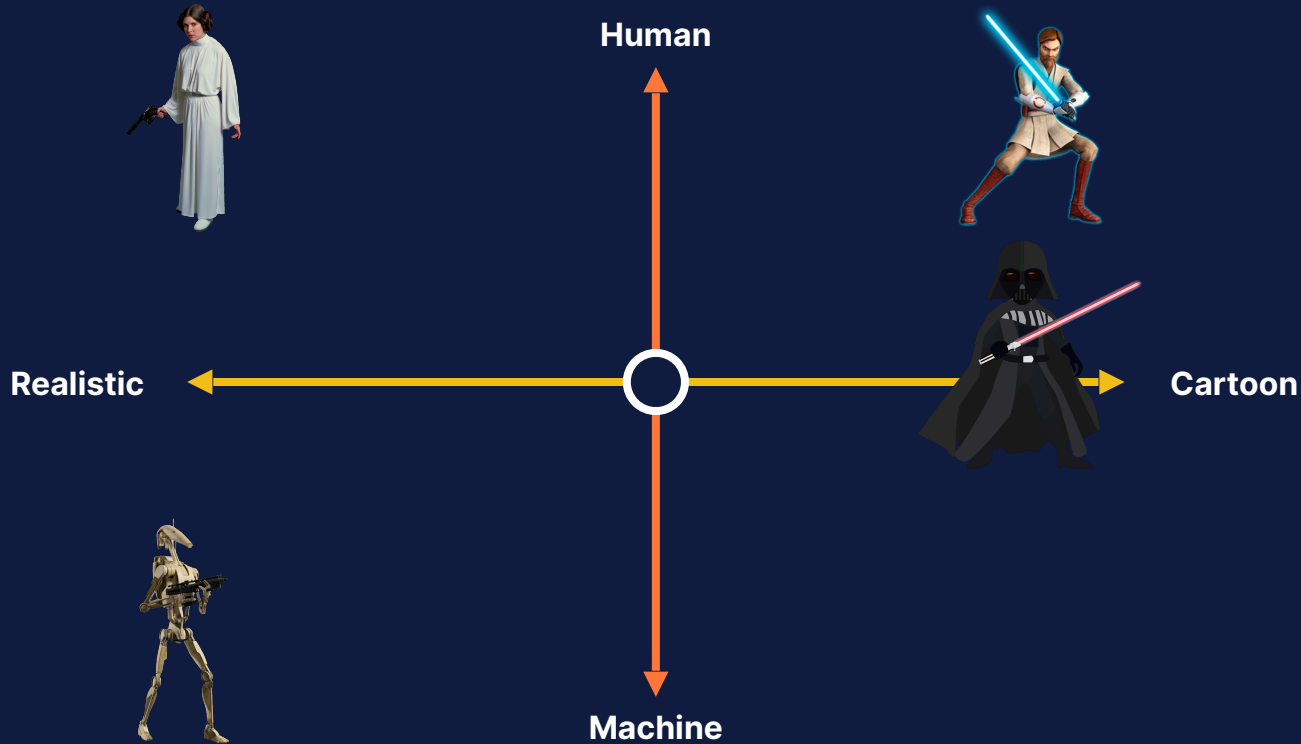
Embeddings represent your data



Example: 1-dimensional vector



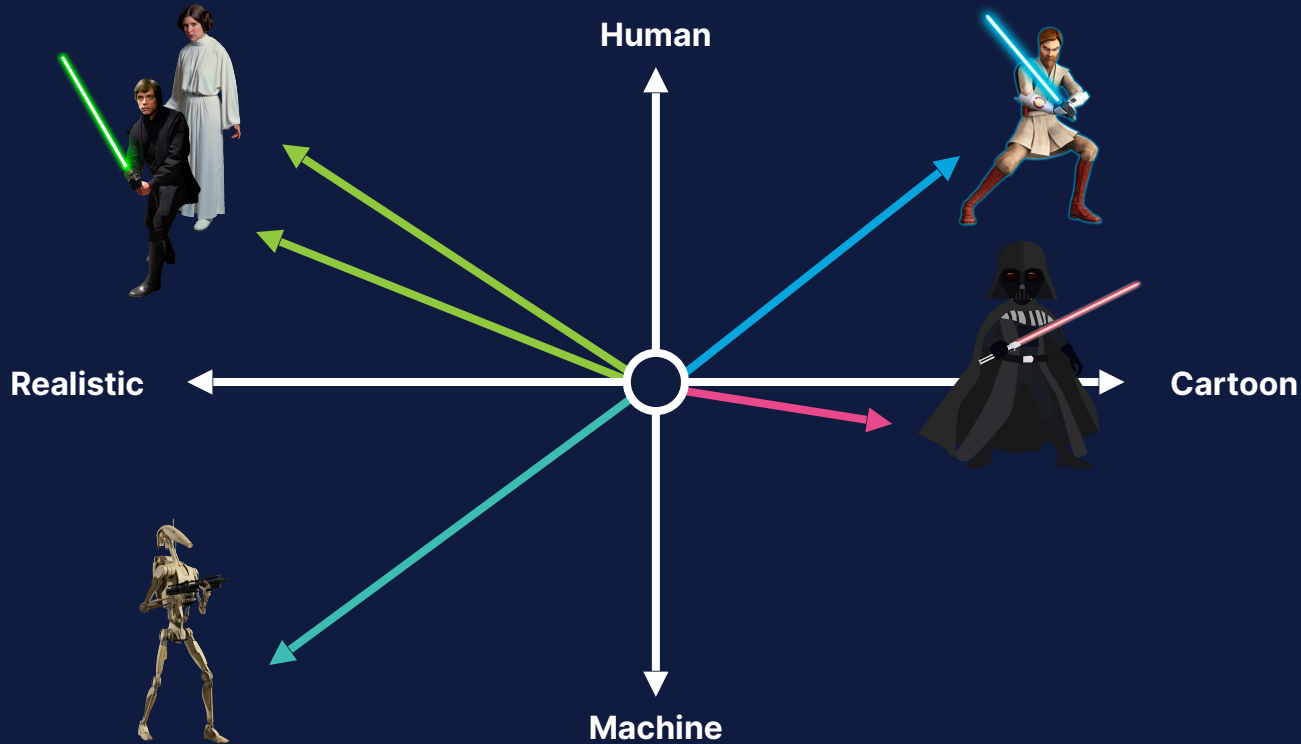
Character	Vector
	$[-1]$
	$[1]$




Multiple dimensions represent different data aspects



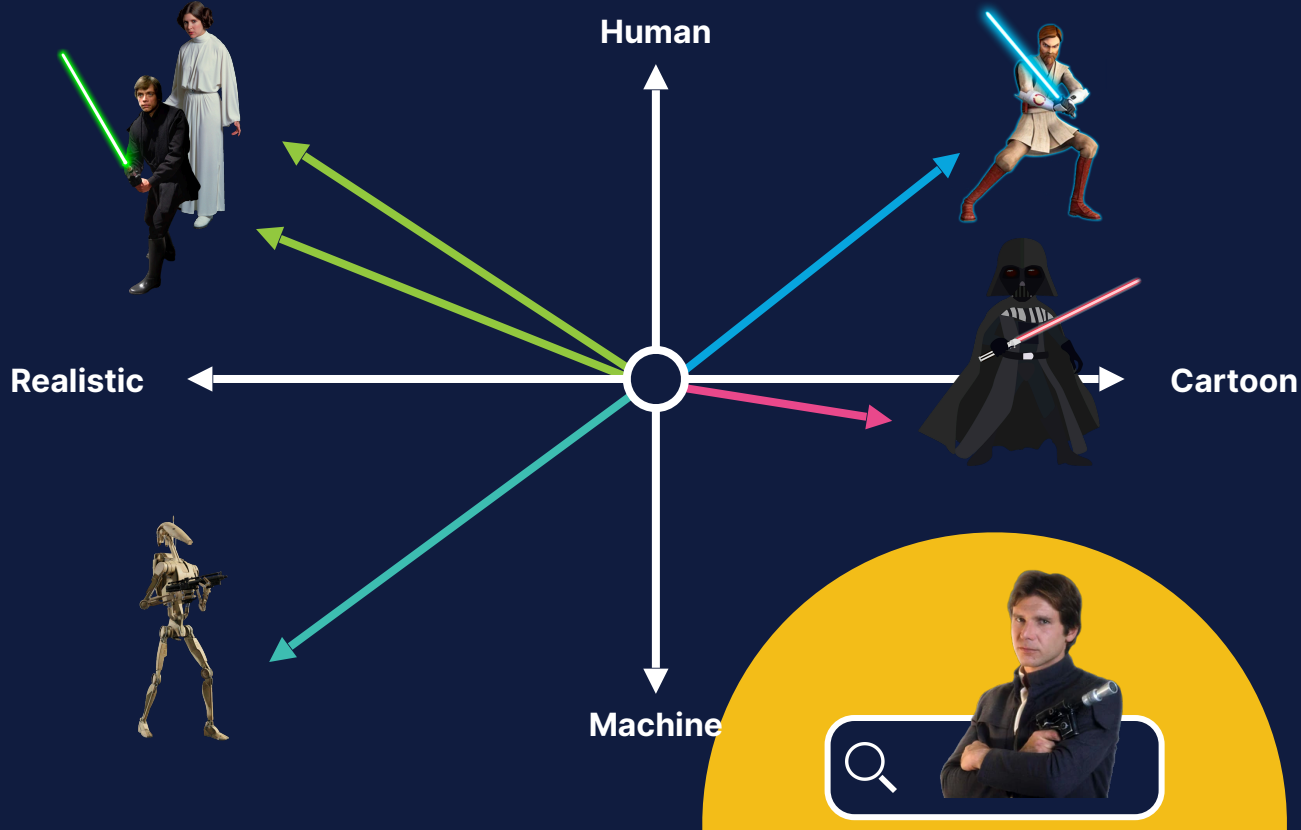
Character	Vector
	$[-1, 1]$
	$[1, 0]$

Similar data is grouped together



Character	Vector
	$[-1.0, 1.0]$
	$[1.0, 0.0]$
	$[-1.0, 0.8]$

Vector search ranks objects by similarity (~relevance) to the query



Rank	Result
Query	
1	
2	
3	
4	
5	

Choice of Embedding Model

Start with Off-the Shelf Models

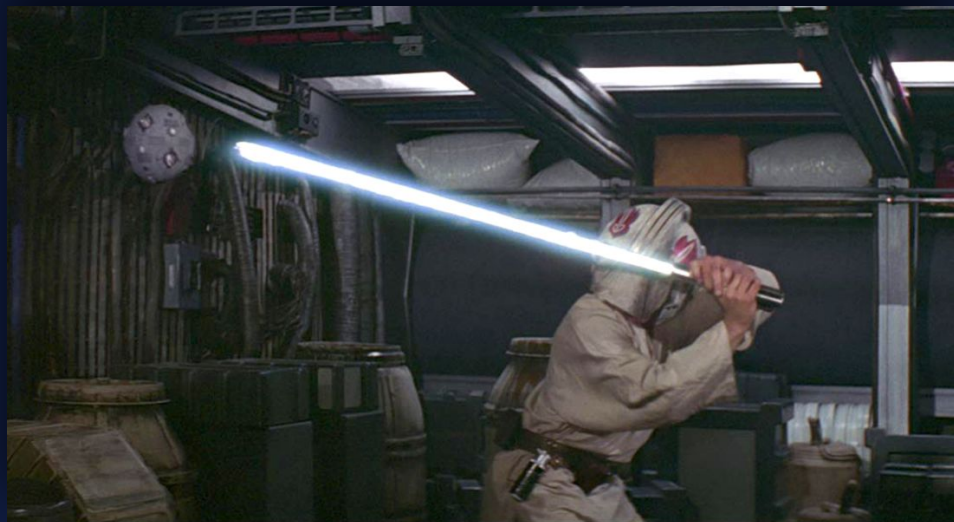
- Text data: Hugging Face (like Microsoft's E5)
- Images: OpenAI's CLIP

Extend to Higher Relevance

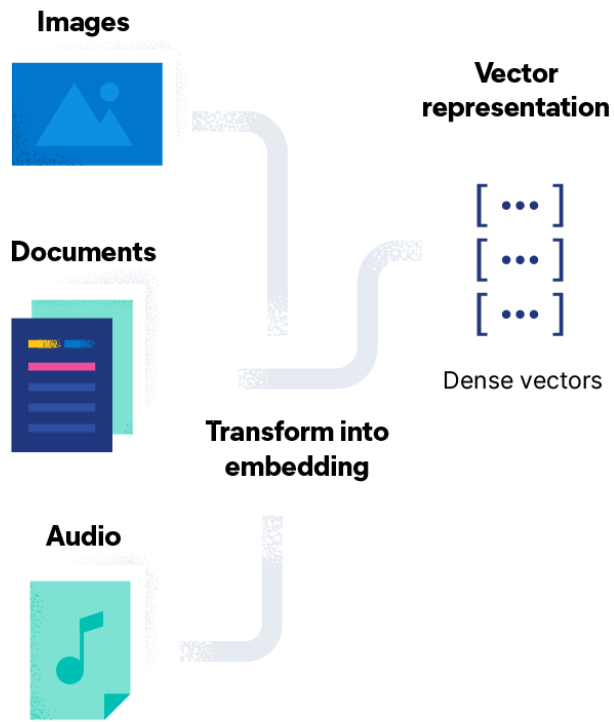
- Apply hybrid scoring
- Bring Your Own Model: requires expertise + labeled data

Problem

training vs actual use-case

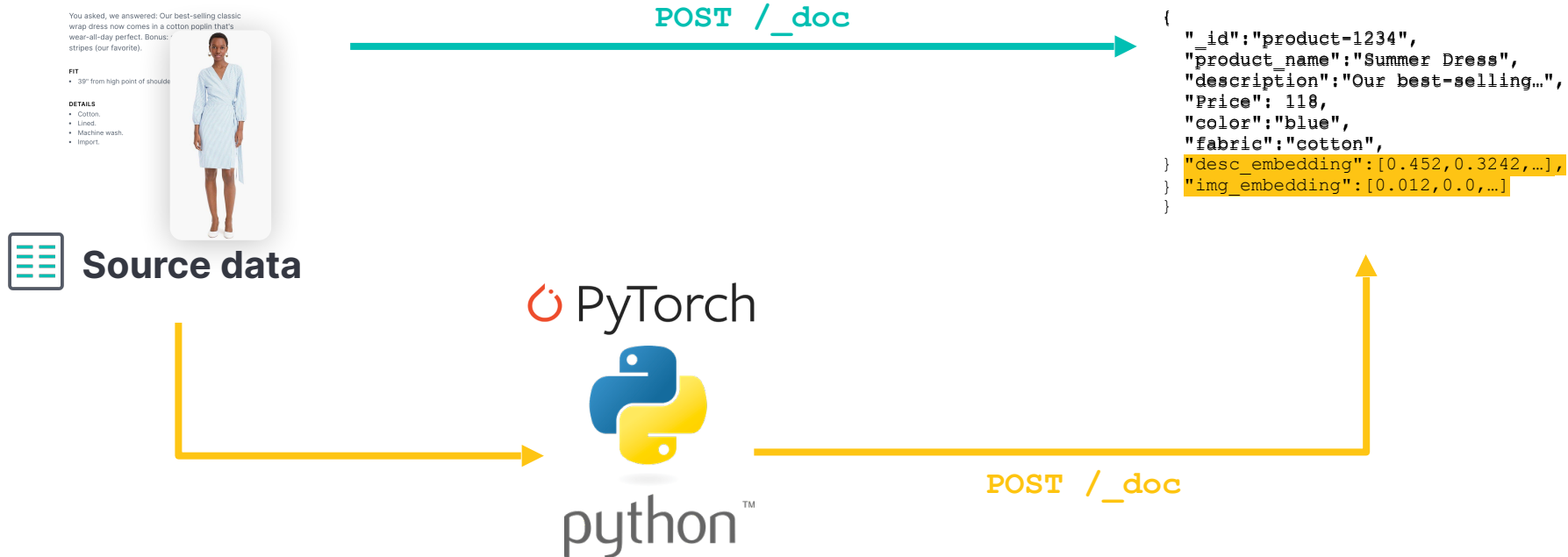


Architecture of Vector Search



How do you index **vectors**?

Data Ingestion and Embedding Generation



With Elastic ML

You asked, we answered: Our best-selling classic wrap dress now comes in a cotton poplin that's wear-all-day perfect. Bonus: stripes (our favorite).

FIT

- 39" from high point of shoulder

DETAILS

- Cotton
- Lined
- Machine wash
- Import



Source data

POST /_doc

ML Inference pipelines

[Add inference pipeline](#)

Inference pipelines will be run as processors from the Enterprise Search Ingest Pipeline

ml-inference-embedding-generation [Actions](#)

Deployed `pytorch` `text_embedding`

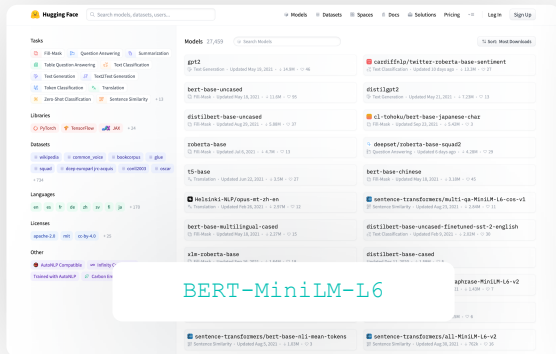
ml-inference-emational-analysis [Actions](#)

Deployed `pytorch` `text_classification`

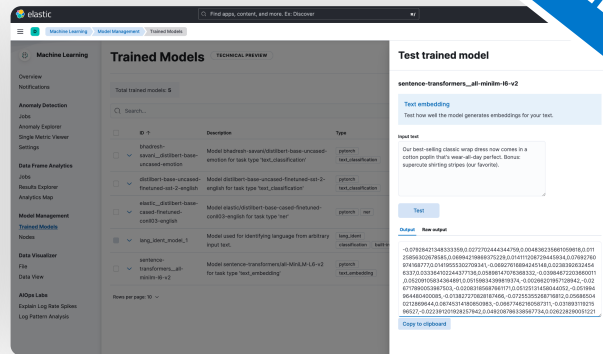
[Learn more about deploying ML models in Elastic](#)

```
{
  "_id": "product-1234",
  "product_name": "Summer Dress",
  "description": "Our best-selling...",
  "Price": 118,
  "color": "blue",
  "fabric": "cotton",
  "desc_embedding": [0.452, 0.3242, ...]
}
```

Eland Imports PyTorch Models



```
$ eland import hub_model
--url https://cluster_URL --hub-
model-id BERT-MiniLM-L6 --task-
type text_embedding --start
```



PyTorch



Load it



Manage models

Select the appropriate model

Elastic's range of supported NLP models

- **Fill mask model**

Mask some of the words in a sentence and predict words that replace masks

- **Named entity recognition model**

NLP method that extracts information from text

- **Text embedding model**

Represent individual words as numerical vectors in a predefined vector space

- **Text classification model**

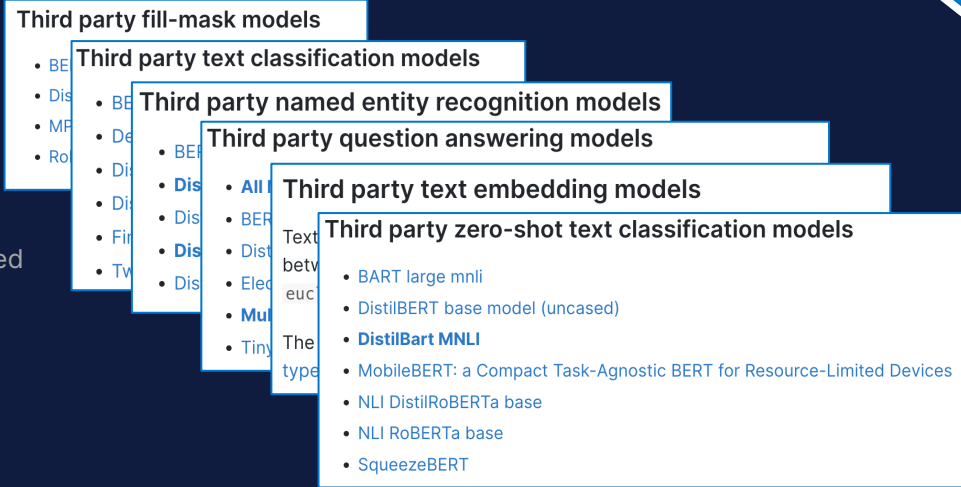
Assign a set of predefined categories to open-ended text

- **Question answering model**

Model that can answer questions given some or no context

- **Zero-shot text classification model**

Model trained on a set of labeled examples, that is able to classify previously unseen examples



How do you search **vectors**?

Vector Query

🔍 summer clothes ✕ 

 PyTorch



python™

```
GET product-catalog/_search
{
  "query" : {
    "bool" : {
      "must" : [{
        "knn" : {
          "field": "desc_embedding",
          "num_candidates": 50,
          "query_vector": [0.123, 0.244, ...]
        }
      ]
    },
    "filter" : {
      "term" : {
        "department": "women"
      }
    }
  }
},
"size": 10
}
```

Vector Query

🔍 summer clothes



Transformer model



```
GET product-catalog/_search
{
  "query" : {
    "bool" : {
      "must" : [{
        "knn" : {
          "field" : "desc_embedding",
          "num_candidates" : 50,
          "query_vector_builder" : {
            "text_embedding" : {
              "model_text" : "summer clothes",
              "model_id" : <text-embedding-model>
            }
          }
        }
      ]
    },
    "filter" : {
      "term" : {
        "department" : "women"
      }
    }
  }
},
  "size" : 10
}
```

Vector Search components

Search

Query

`kNN`

Index

Mapping

`dense_vector`

Generate

Embedding

Text embedding
model

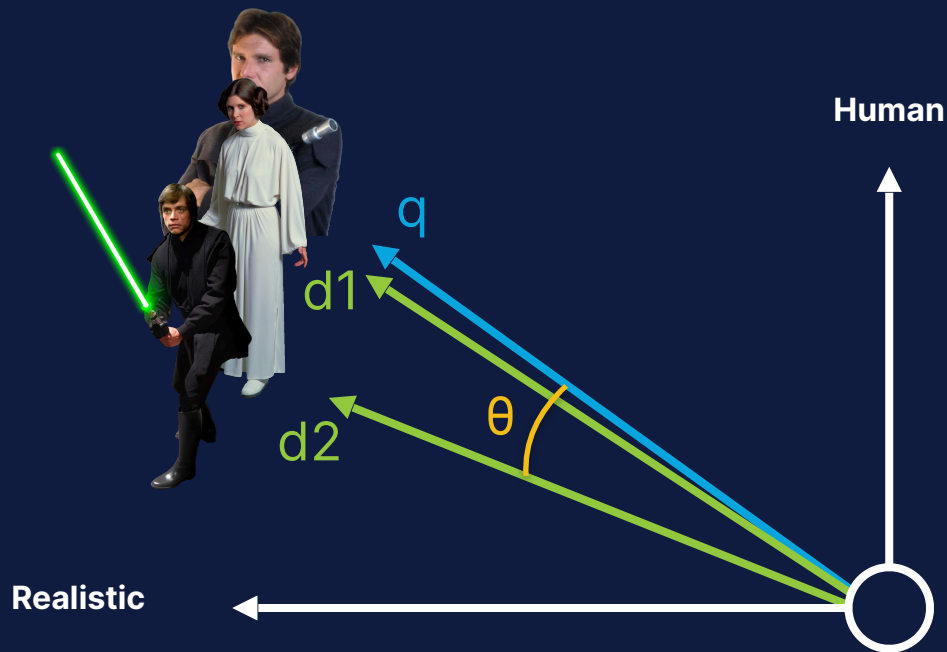
(3rd party, local, in Elasticsearch)



But how does it
really work?



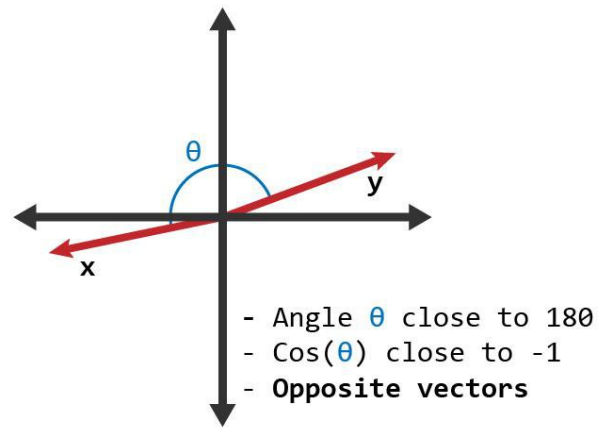
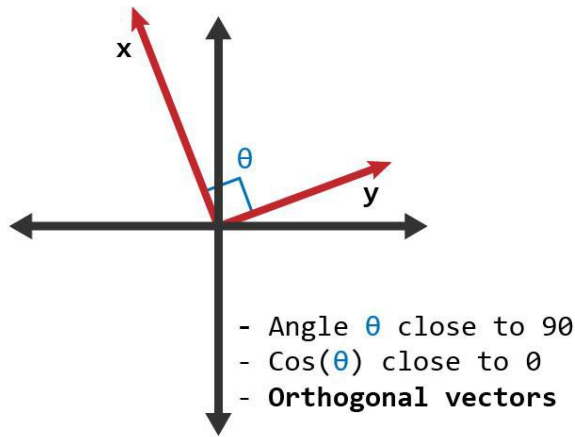
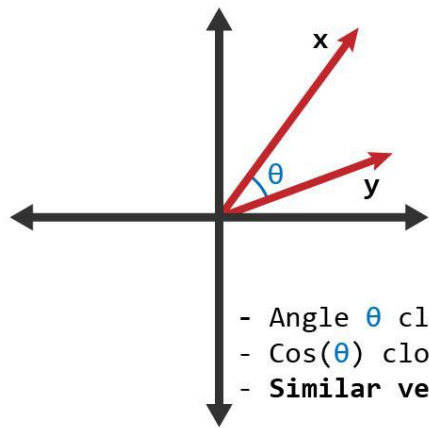
Similarity: cosine (cosine)



$$\cos(\theta) = \frac{\vec{q} \times \vec{d}}{|\vec{q}| \times |\vec{d}|}$$

$$\text{_score} = \frac{1 + \cos(\theta)}{2}$$

Similarity: cosine (cosine)

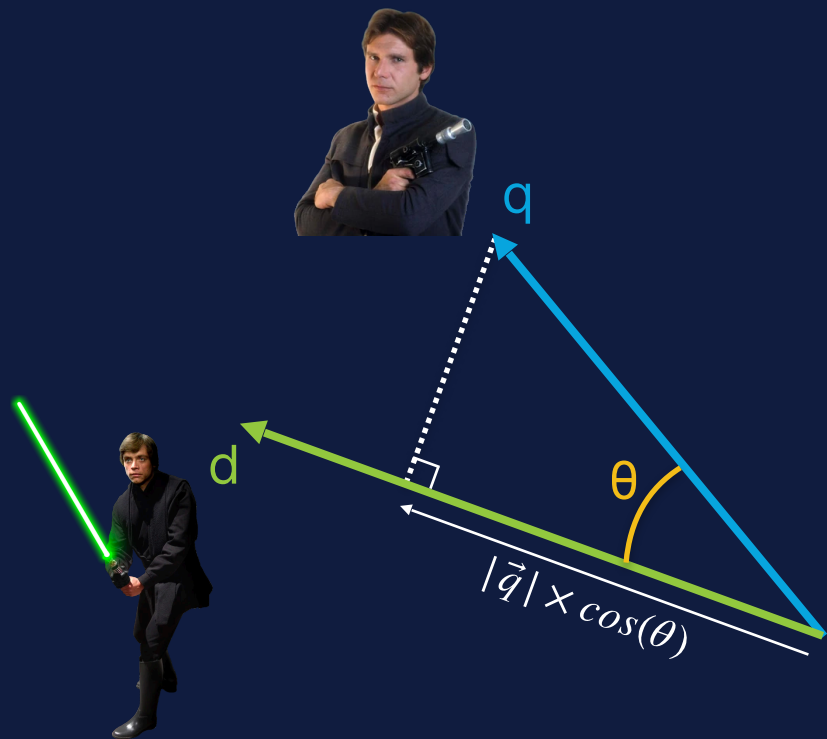


$$\text{_score} = \frac{1 + 1}{2} = 1$$

$$\text{_score} = \frac{1 + 0}{2} = 0.5$$

$$\text{_score} = \frac{1 - 1}{2} = 0$$

Similarity: Dot Product (`dot_product`)

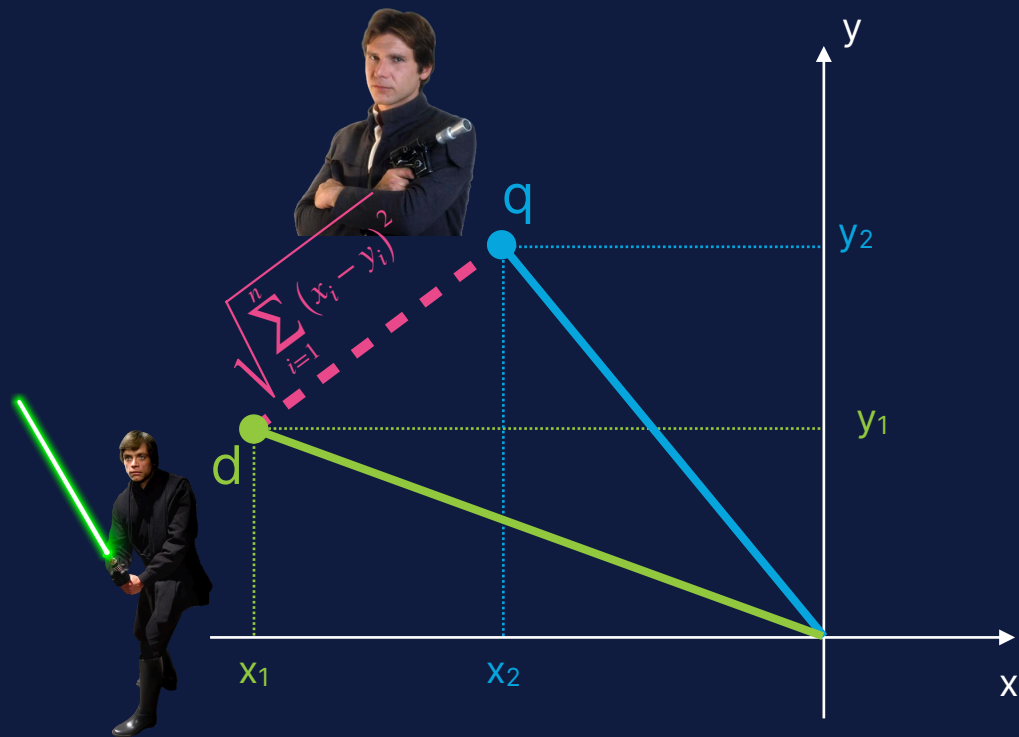


$$\vec{q} \times \vec{d} = |\vec{q}| \times \cos(\theta) \times |\vec{d}|$$

$$\text{_score}_{float} = \frac{1 + \text{dot_product}(q, d)}{2}$$

$$\text{_score}_{byte} = \frac{0.5 + \text{dot_product}(q, d)}{32768 \times \text{dims}}$$

Similarity: Euclidean distance (l2_norm)



$$l2_norm_{q,d} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$
$$_score = \frac{1}{1 + (l2_norm_{q,d})^2}$$

Brute Force



Hierarchical Navigable Small Worlds (HNSW)

One popular approach



HNSW: a layered approach that simplifies access to the nearest neighbor



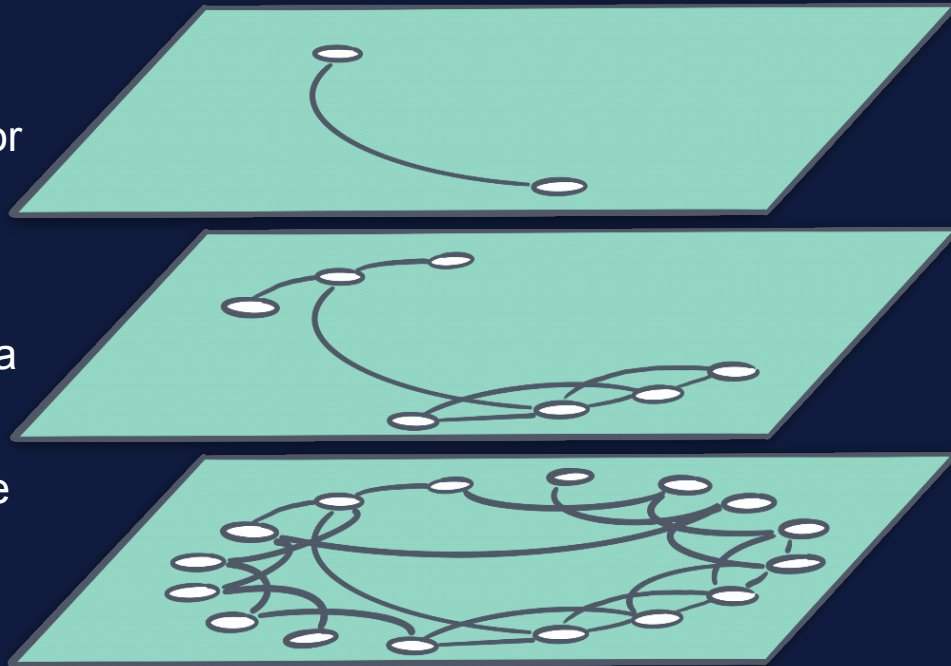
Tiered: from coarse to fine approximation over a few steps



Balance: Bartering a little accuracy for a lot of scalability



Speed: Excellent query latency on large scale indices



Elasticsearch + Lucene = fast progress ❤️

Increase max number of vector dims to 2048 #95257

Increase the max vector dims to 4096 #99682

Merged

mayya-sharipova merged 2 commits into `elastic:main` from `mayya-sharipova:increase_vector_dims_4096`

Conversation 5

Commits 2

Checks 0

Files changed 8



mayya-sharipova commented on Sep 19

Contributor ...

No description provided.

Increase the max vector dims to 4096

✗ 3f97c5f

mayya-sharipova added `>enhancement` `:Search/Vectors` `v8.11.0` labels on Sep 19

Scaling Vector Search

Vector search

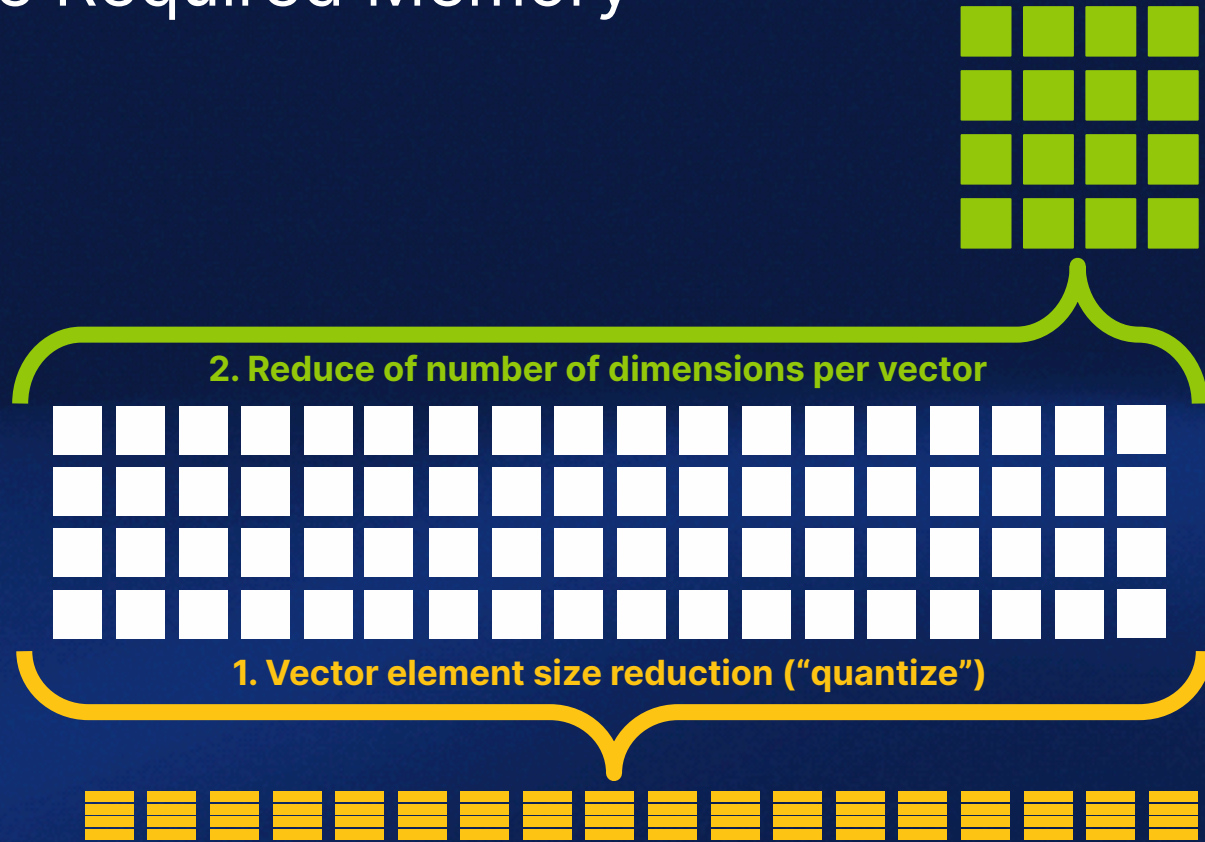
1. Needs lots of memory
2. Indexing is slower
3. Merging is slow

* Continuous improvements in Lucene + Elasticsearch

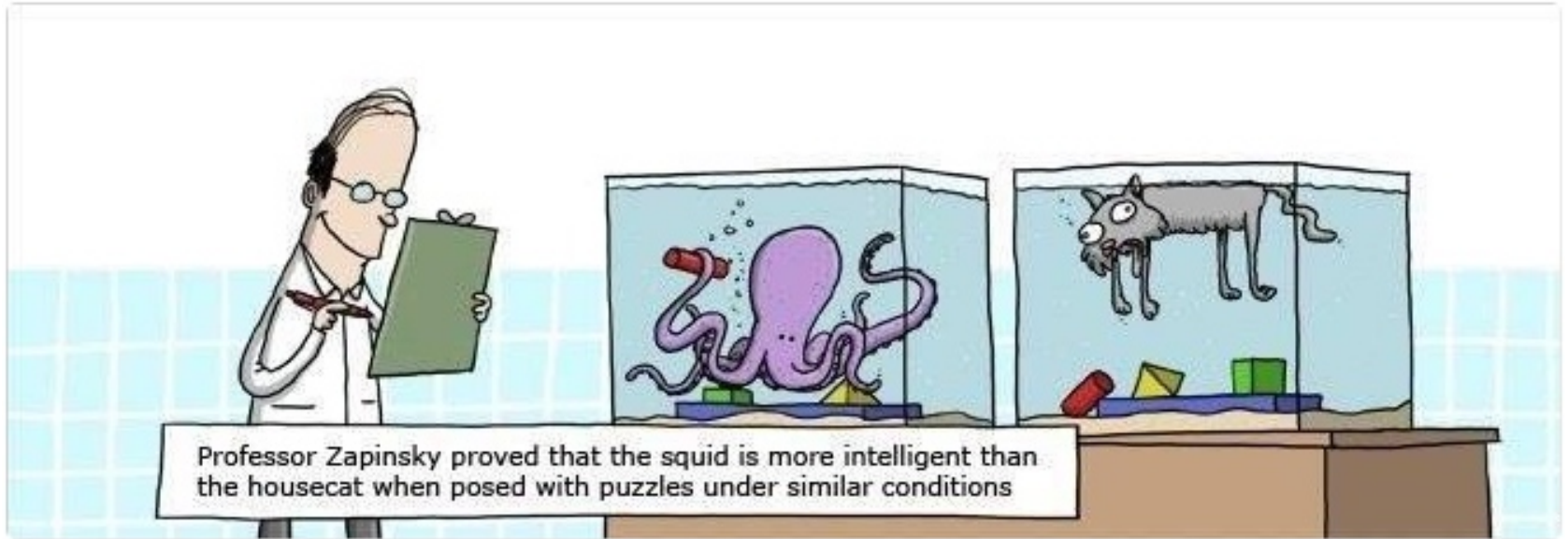
Best practices

1. Avoid searches during indexing
2. Exclude vectors from `_source`
3. Reduce vector dimensionality
4. Use byte rather than float

Reduce Required Memory



Benchmarking



https://github.com/erikbern/ann-benchmarks

Add Elasticsearch KNN #401



erikbern merged 1 commit into `erikbern:main` from `ceh-forks:elasticsearch` last week



Conversation 5



Commits 1



Checks 34



Files changed 4



ceh commented last week · edited ▾

Contributor



Add an implementation for [Elasticsearch KNN search](#). Fixes [#298](#).

- Supports specifying `index_options` via arg groups, with one entry for Elasticsearch's [default settings](#) for M and EF.
- Supports specifying `num_candidates` via query args.

What do you think @erikbern, @alexklibisz?

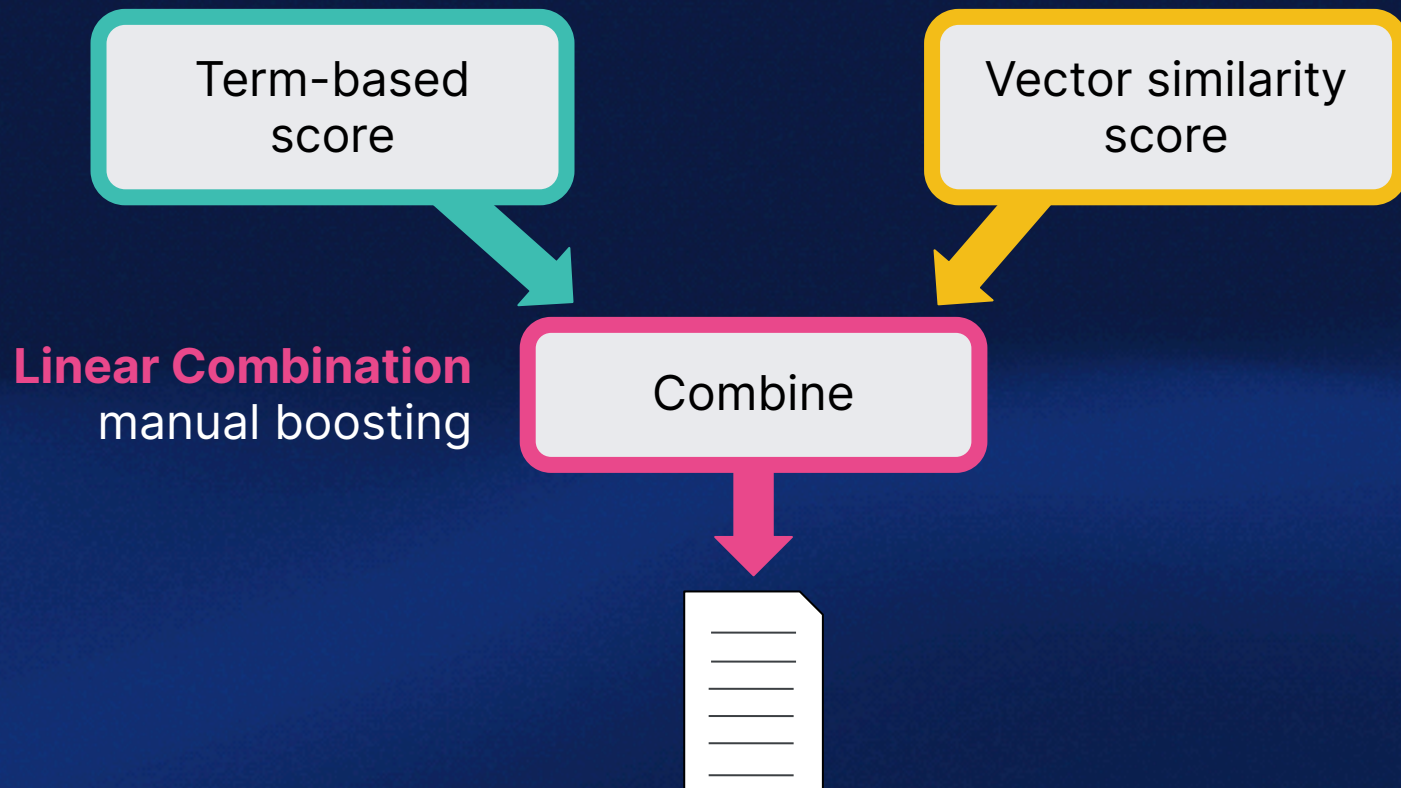


1

Elasticsearch

You Know, for **Hybrid** Search

Hybrid scoring



```
GET product-catalog/_search
```

```
{  
  "query" : {  
    "bool" : {  
      "must" : [{  
        "match": {  
          "description": {  
            "query": "summer clothes",  
            "boost": 0.9  
          }  
        }  
      }  
    }, {  
      "knn": {  
        "field": "desc_embedding",  
        "query_vector": [0.123, 0.244, ...],  
        "num_candidates": 50,  
        "boost": 0.1,  
        "filter": {  
          "term": {  
            "department": "women"  
          }  
        }  
      }  
    }  
  }  
}, {  
  "filter" : {  
    "range" : { "price": { "lte": 30 } }  
  }  
}  
}
```

summer clothes

pre-filter

post-filter

```
GET product-catalog/_search
{
  "query" : {
    "bool" : {
      "must" : [{
        "match": {
          "description": {
            "query": "summer clothes",
            "boost": 0.9
          }
        }
      ]
    }, {
      "knn": {
        "field": "image-vector",
        "query_vector": [54, 10, -2],
        "num_candidates": 50,
        "boost": 0.1
      }
    }, {
      "knn": {
        "field": "title-vector",
        "query_vector": [1, 20, -52, 23, 10],
        "num_candidates": 10,
        "boost": 0.5
      }
    }
  ]
}
```

ELSER

Elastic Learned Sparse Encoder

text_expansion

Not BM25 or (dense) vector

Sparse vector like BM25

Stored as inverted index

Commercial

Machine Learning Inference Pipelines

Inference pipelines will be run as processors from the Enterprise Search Ingest Pipeline

New Improve your results with ELSER ✕

ELSER (Elastic Learned Sparse Encoder) is our **new trained machine learning model** designed to efficiently use context in natural language queries. This model delivers better results than BM25 without further training on your data.

 Deploy

[Learn more](#) 

 Add Inference Pipeline

[Learn more about deploying Machine Learning models in Elastic](#) 



```
PUT /_inference/sparse_embedding/my_elsar_model
{
  "service": "elsar",
  "service_settings": {
    "num_allocations": 1,
    "num_threads": 1
  },
  "task_settings": {}
}
```



```
PUT /_inference/text_embedding/openai_embeddings
{
  "service": "openai",
  "service_settings": {
    "api_key": "<api_key>"
  },
  "task_settings": {
    "model": "text-embedding-ada-002"
  }
}
```



```
PUT /_inference/text_embedding/hugging_face_embeddings
{
  "service": "hugging_face",
  "service_settings": {
    "api_key": "<access_token>",
    "url": "<url_endpoint>"
  }
}
```

```
POST /_inference/sparse_embedding/my_elses_model
{
  "input": [
    "These are not the droids you are looking for.",
    "Obi-Wan never told you what happened to your father."
  ]
}
```



```
{
  "sparse_embedding": [{
    "lucas": 0.50047517,
    "ship": 0.29860738,
    "dragon": 0.5300422,
    "quest": 0.5974301,
    "dr": 2.1055143,
    "space": 0.49377063,
    "robot": 0.40398192,
    ...
  ]
}
```


Hybrid ranking

Commercial

Term-based
score

Vector similarity
score

ELSER
score

Reciprocal Rank Fusion (RRF)
blend multiple
ranking methods

Combine



```
GET product-catalog/_search
{
  "sub_searches": [
    {
      "query": {
        "match": {...}
      }
    },
    {
      "query": {
        "text_expansion": {...}
      }
    }
  ],
  "knn": {...},
  "rank": {
    "rrf": {
      "window_size": 50,
      "rank_constant": 20
    }
  }
}
```

BM25f

+

ELSER

+

Vector



Hybrid Ranking

Reciprocal Rank Fusion (RRF)

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k+r(d)}$$

D - set of docs

R - set of rankings as permutation on 1..|D|

k - typically set to 60 by default

Ranking Algorithm 1

Doc	Score	r(d)	k+r(d)
A	1	1	61
B	0.7	2	62
C	0.5	3	63
D	0.2	4	64
E	0.01	5	65

Ranking Algorithm 2

Doc	Score	r(d)	k+r(d)
C	1,341	1	61
A	739	2	62
F	732	3	63
G	192	4	64
H	183	5	65



Doc	RRF Score
A	$1/61 + 1/62 = 0,0325$
C	$1/63 + 1/61 = 0,0323$
B	$1/62 = 0,0161$
F	$1/63 = 0,0159$
D	$1/64 = 0,0156$

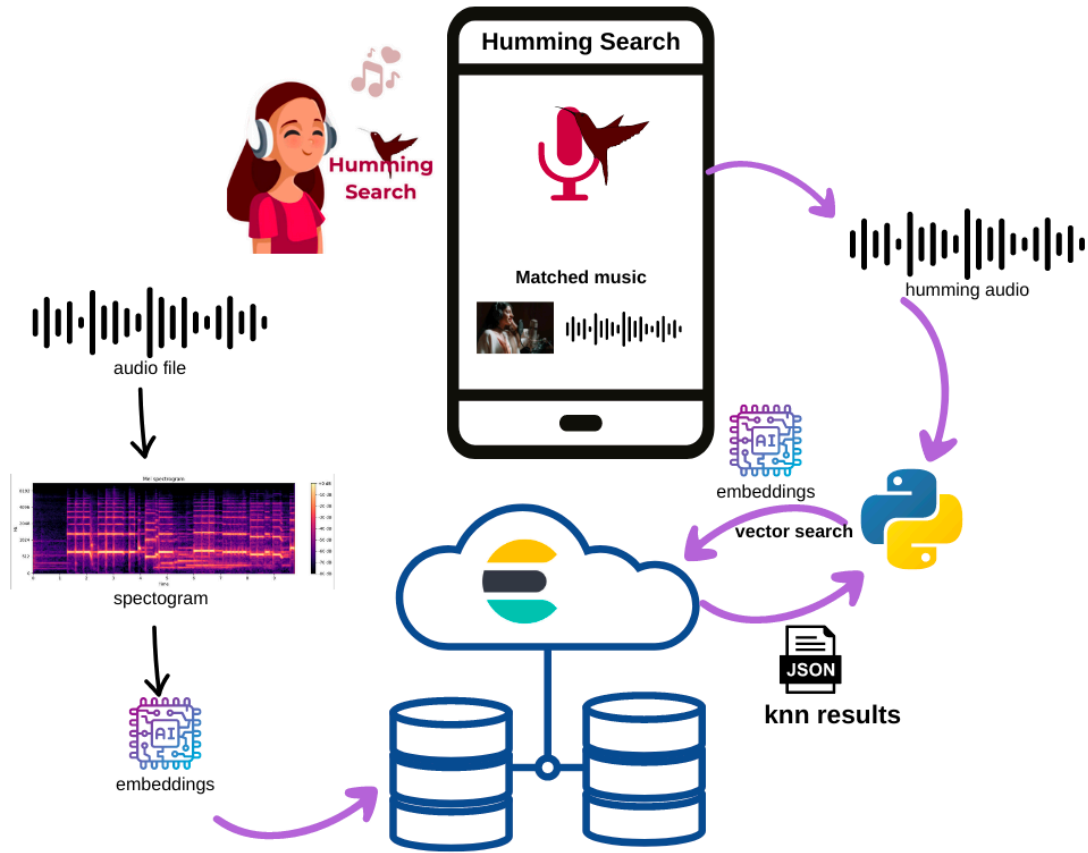


<https://djdadoo.pilato.fr/>

Anniversaire **Lucas** - 25 ans



16/09/2023



<https://github.com/dadoonet/music-search/>

ChatGPT

Elastic and LLM

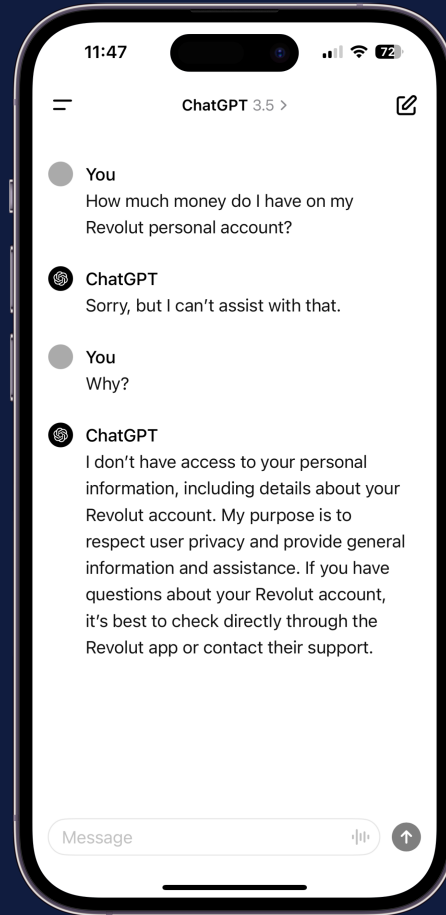
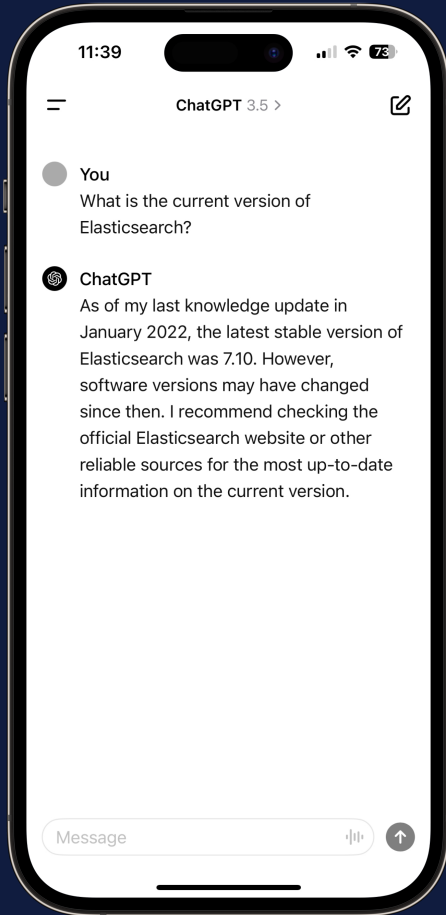
Gen AI

Search engines

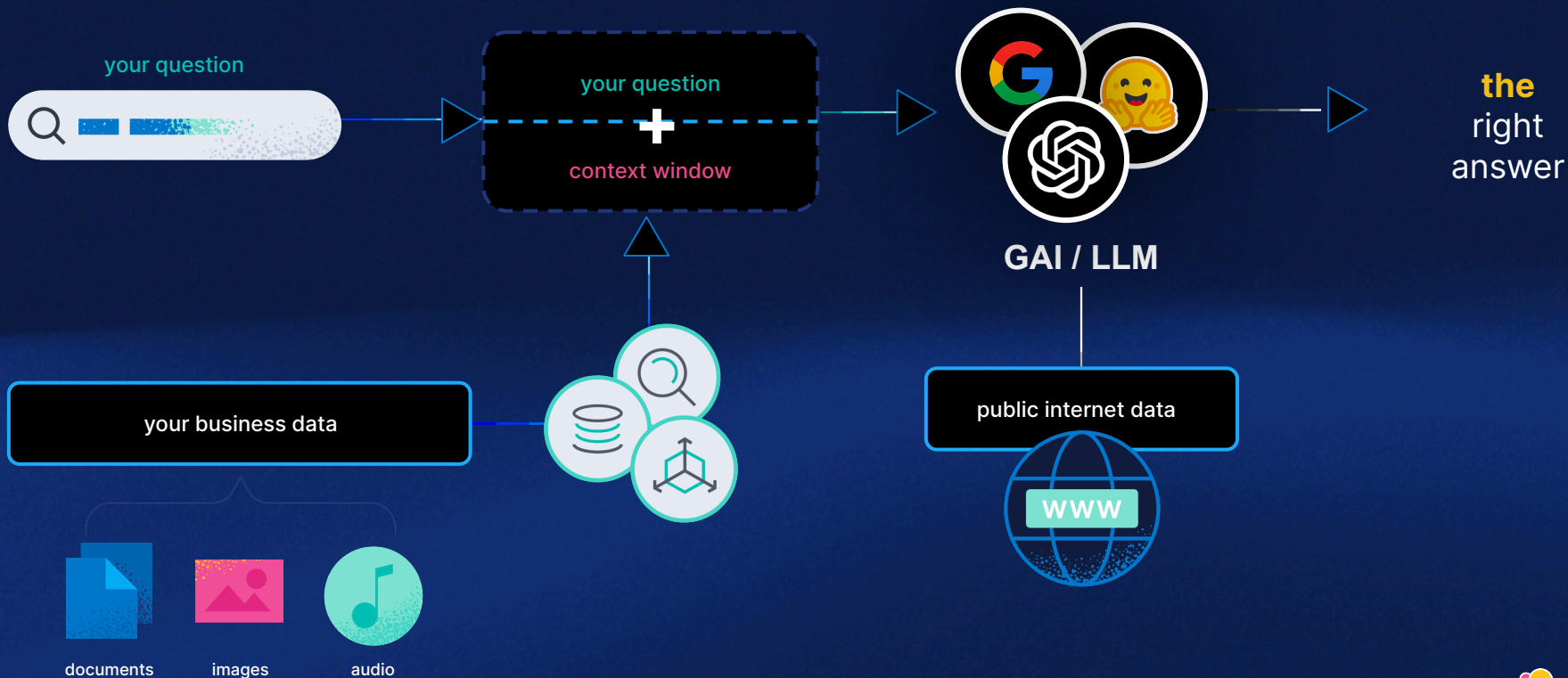


LLM: opportunities and limits





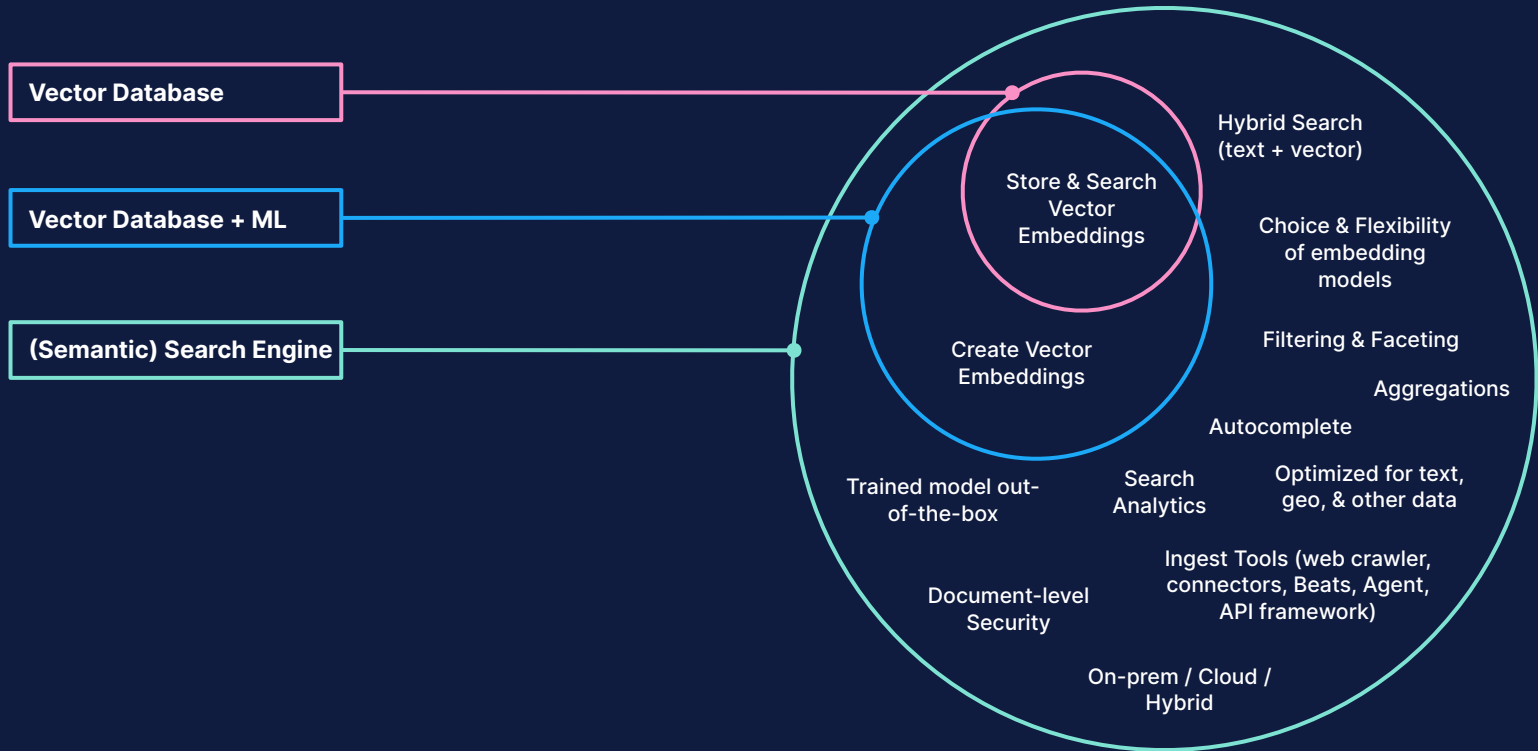
Retrieval Augmented Generation



Demo

Elastic + Azure OpenAI
AWS Bedrock
Google Vertex AI

Conclusion



Elasticsearch

You Know, for **Semantic** Search



Search: a new era

David Pilato | [@dadoonet](#)

