Qdrant

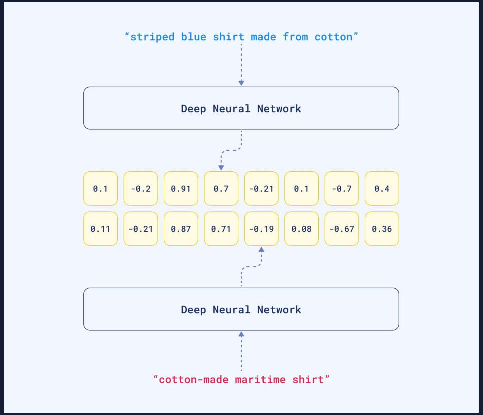
The many shades of text search in vector retrieval

Agenda

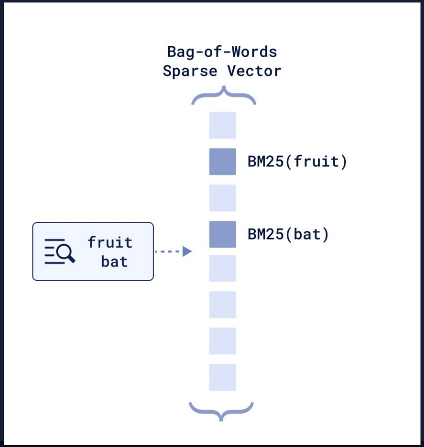
Qdrant is not a dedicated text search engine; it is a vector similarity search engine. Instead of treating vector search as an add-on to text search, we see text search as an extension of vector search.

Text search in Qdrant

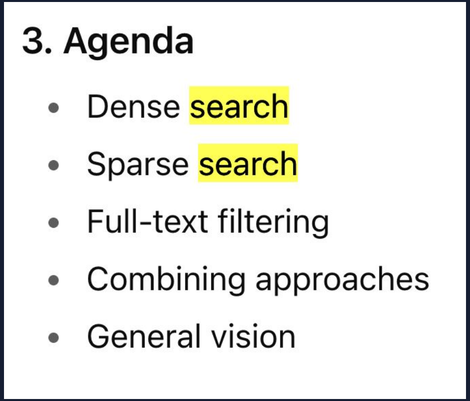
Dense Vectors



Sparse Vectors



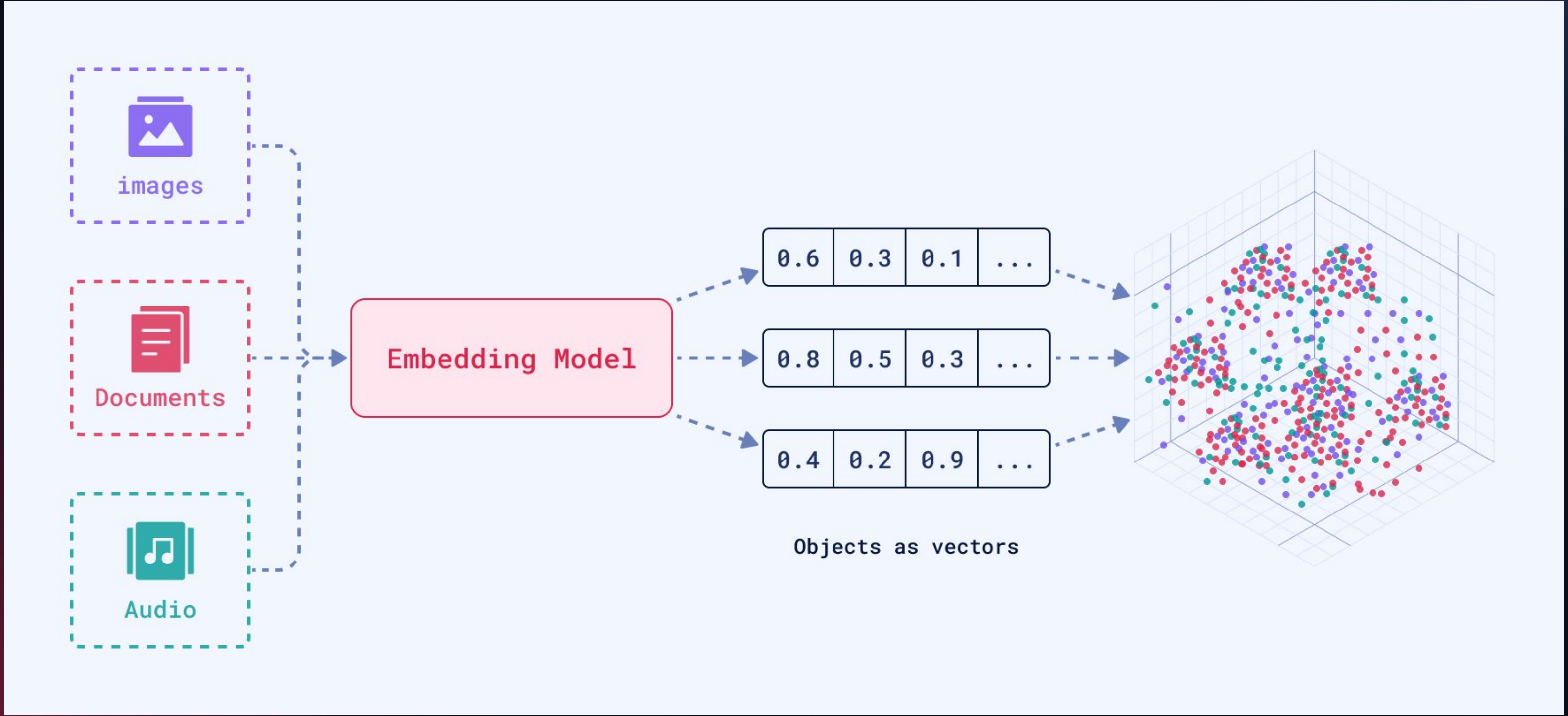
Full Text Filters



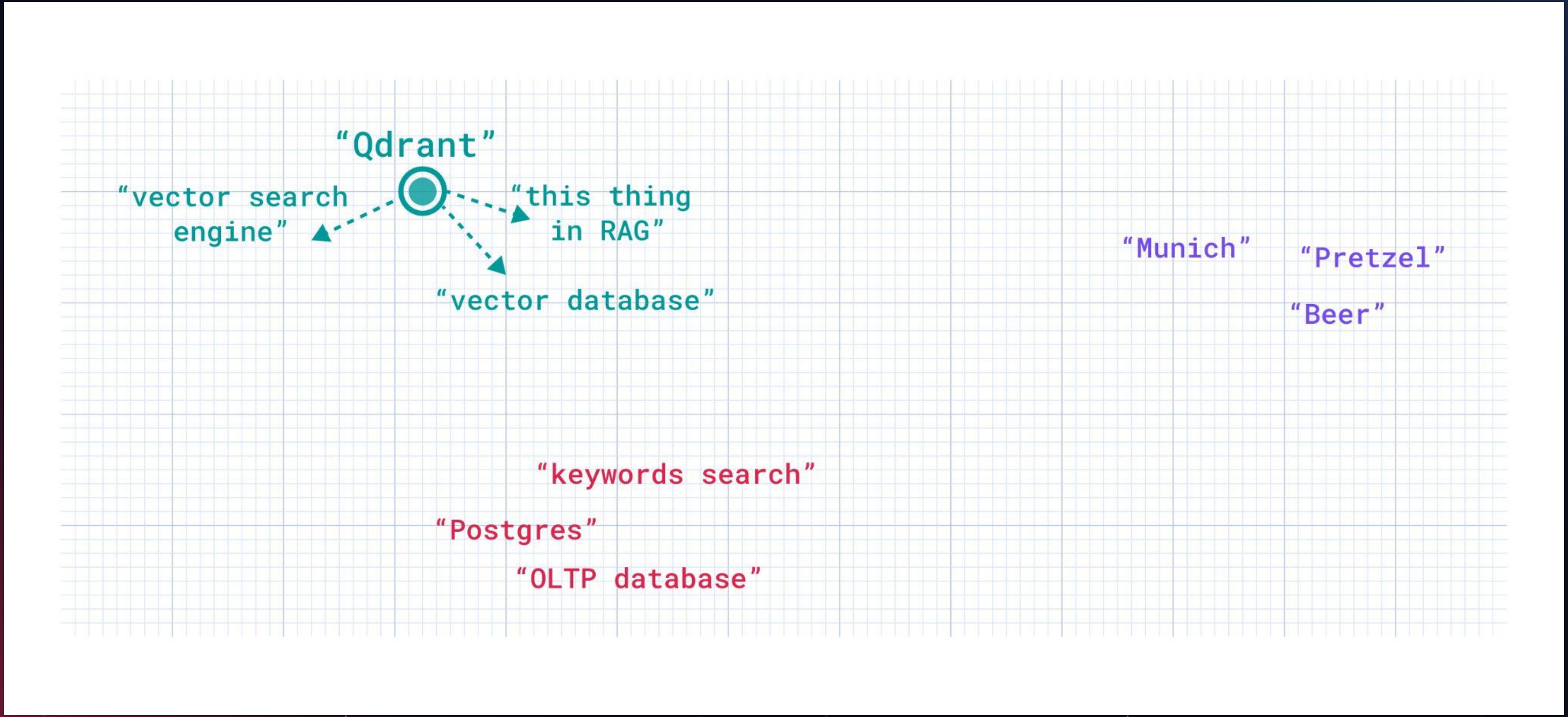
3. Agenda

- Dense search
- Sparse search
- Full-text filtering
- Combining approaches
- General vision

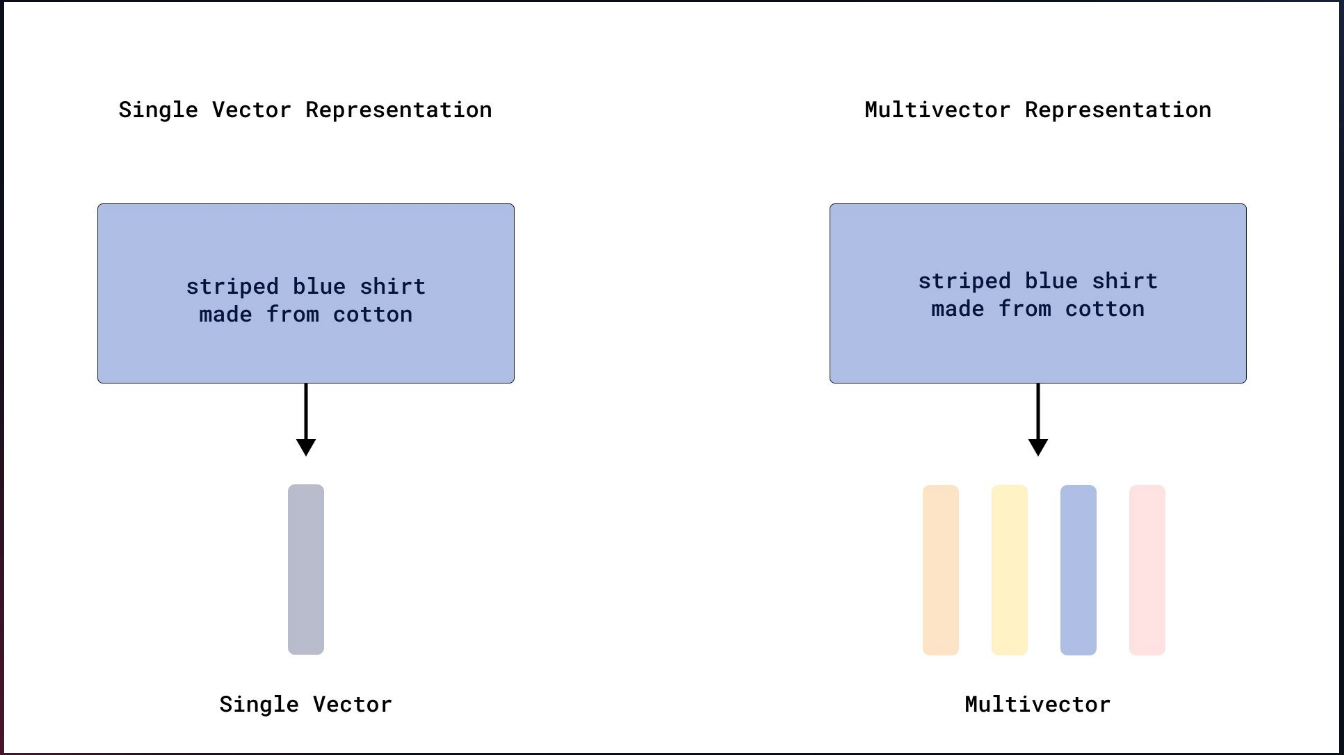
Dense Vectors



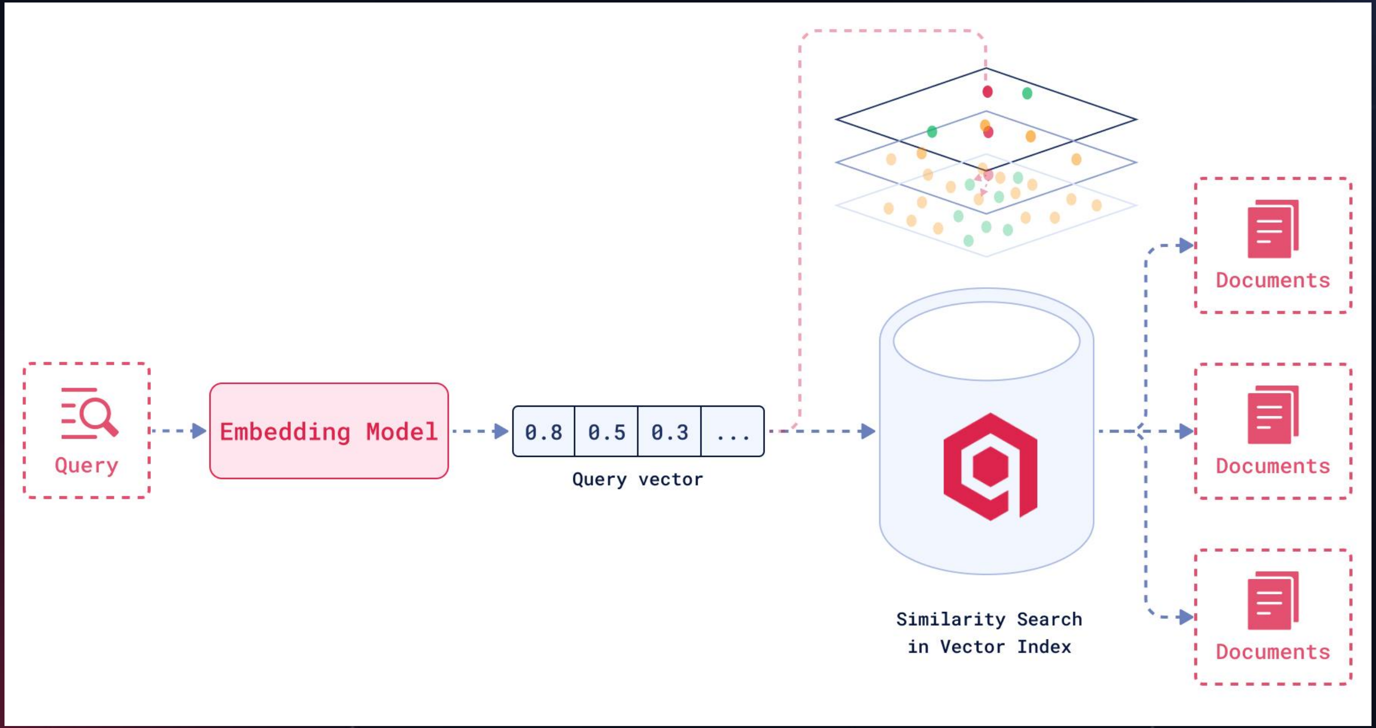
Dense Vectors for Text Search



Dense Vectors (for Text Search)


Single Vector Representation
Multivector Representation
striped blue shirt made from cotton
striped blue shirt made from cotton
Single Vector
Multivector

Dense Vectors in Qdrant



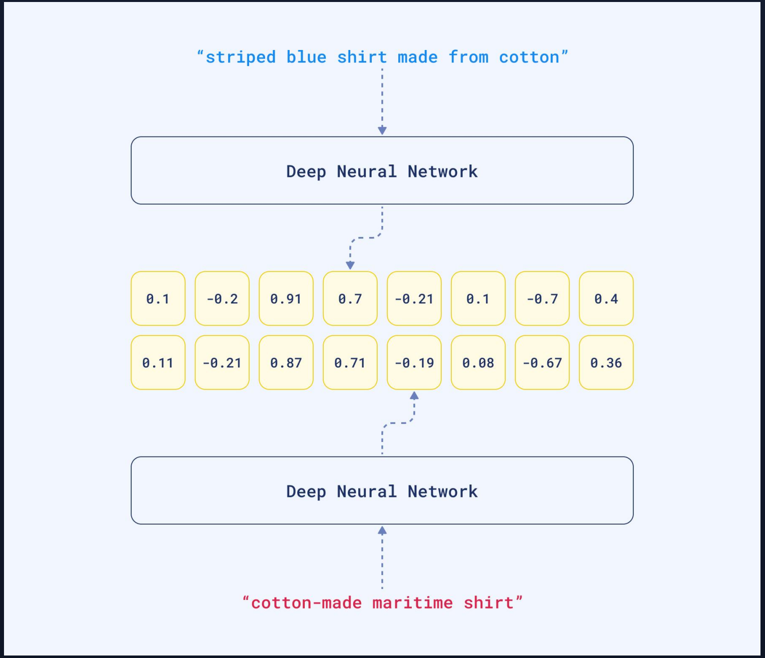
Dense Vectors for Text Search

What they should / could be used for:

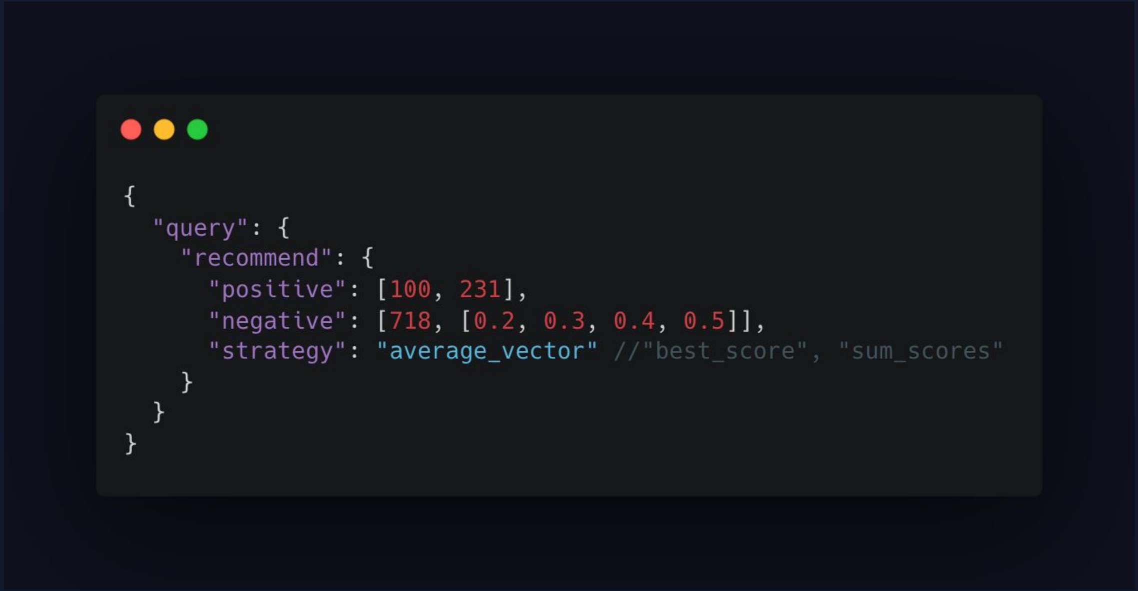
- Any kind of semantic similarity search (RAG, agentic memory, question-answering, duplicate detection, intent matching)
- Fuzzy matching of any kind (paraphrases, typos, synonyms)
- Cross-lingual search (English ↔ German, etc.)

Dense vector search unique (requires owning the vector index)

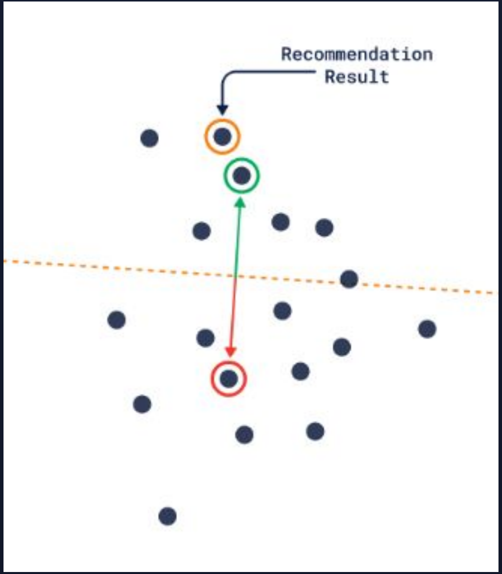
- Discovery search
- Recommendations



Text Search in Qdrant: Recommend

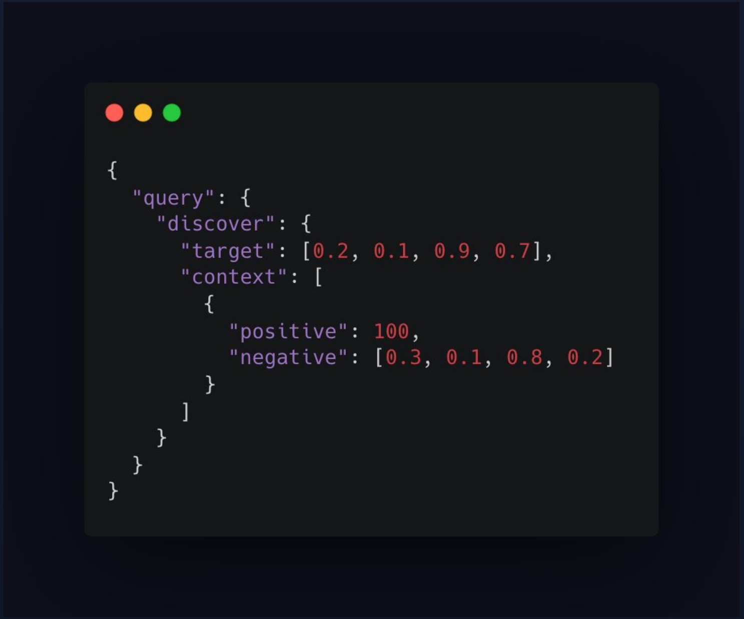


```
{
  "query": {
    "recommend": {
      "positive": [100, 231],
      "negative": [718, [0.2, 0.3, 0.4, 0.5]],
      "strategy": "average_vector" //"best_score", "sum_scores"
    }
  }
}
```

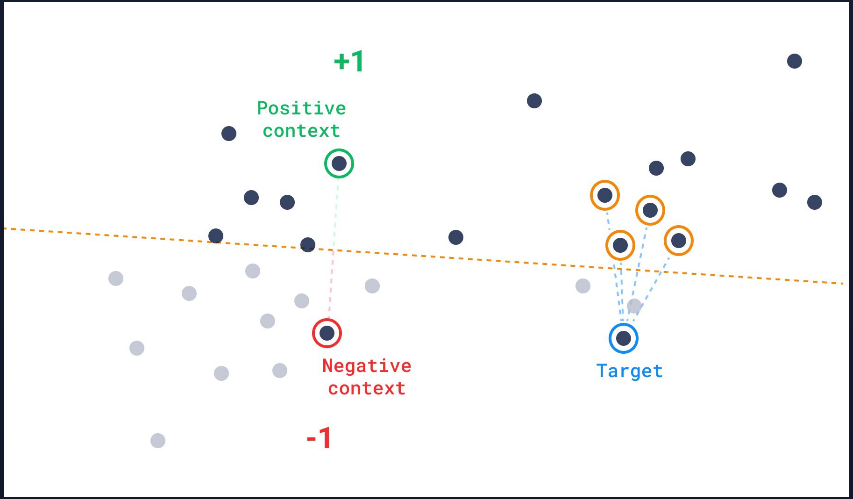


- I'd like to try Bavarian cuisine, but nothing too heavy
- Show me sci-fi movies, but not dystopian ones
- Articles about applied AI, but not about chatbots

Text Search in Qdrant: Discovery



```
{
  "query": {
    "discover": {
      "target": [0.2, 0.1, 0.9, 0.7],
      "context": [
        {
          "positive": 100,
          "negative": [0.3, 0.1, 0.8, 0.2]
        }
      ]
    }
  }
}
```



I want to discover an article on modern search → more like 'AI-powered search,' but not 'RAG is dead.'
(It could even be in the style of writing on topics unrelated to search as a positive/negative example.)

Dense Vectors for Text Search

What they should / could be used for:

- Any kind of semantic similarity search (RAG, agentic memory, question–answering, duplicate detection, intent matching)
- Fuzzy matching of any kind (paraphrases, typos, synonyms)
- Cross-lingual search (English ↔ German, etc.)

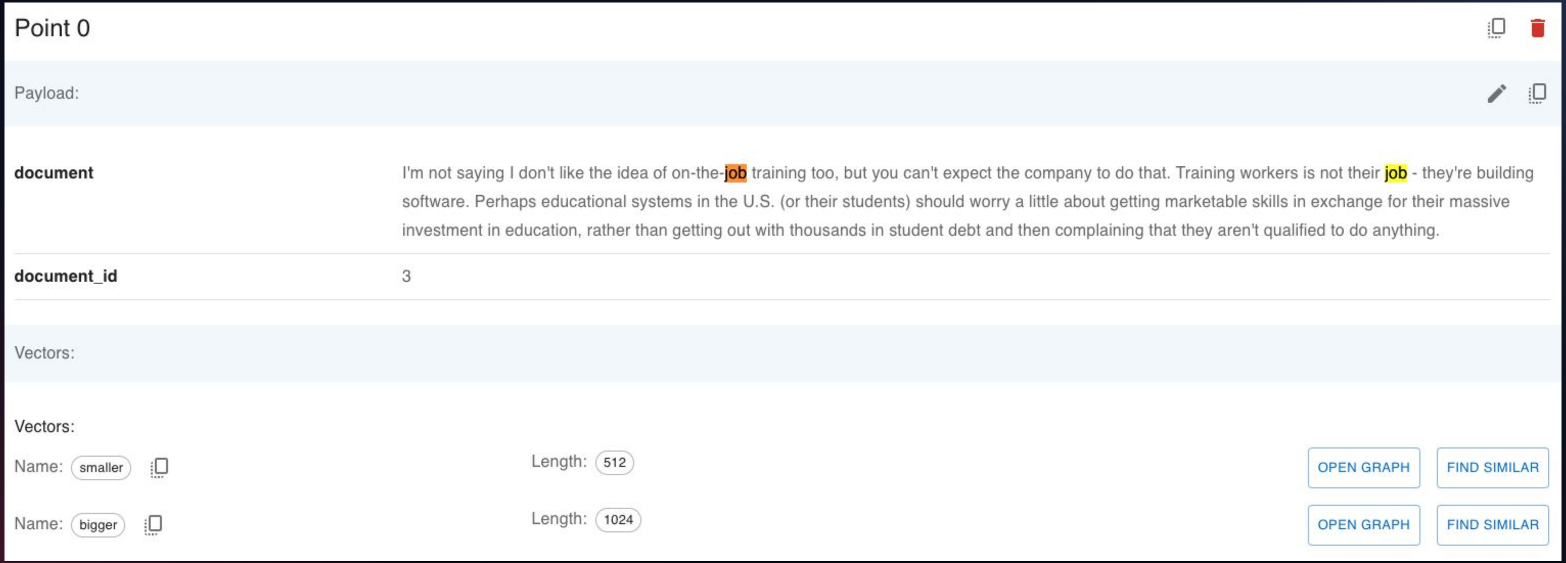
Dense vector search unique
(requires owning the vector index):

- Discovery search
- Recommendations

What they are not suitable for:

- Exact key term/number/ID matching (e.g., error code 404)
- Control over matching rules (proximity, must/should matches)
- Controlled synonym/vocabulary expansion (unless fine-tuned/custom trained)

(Text) Filtering


Point 0

Payload:

document	I'm not saying I don't like the idea of on-the-job training too, but you can't expect the company to do that. Training workers is not their job - they're building software. Perhaps educational systems in the U.S. (or their students) should worry a little about getting marketable skills in exchange for their massive investment in education, rather than getting out with thousands in student debt and then complaining that they aren't qualified to do anything.
document_id	3

Vectors:

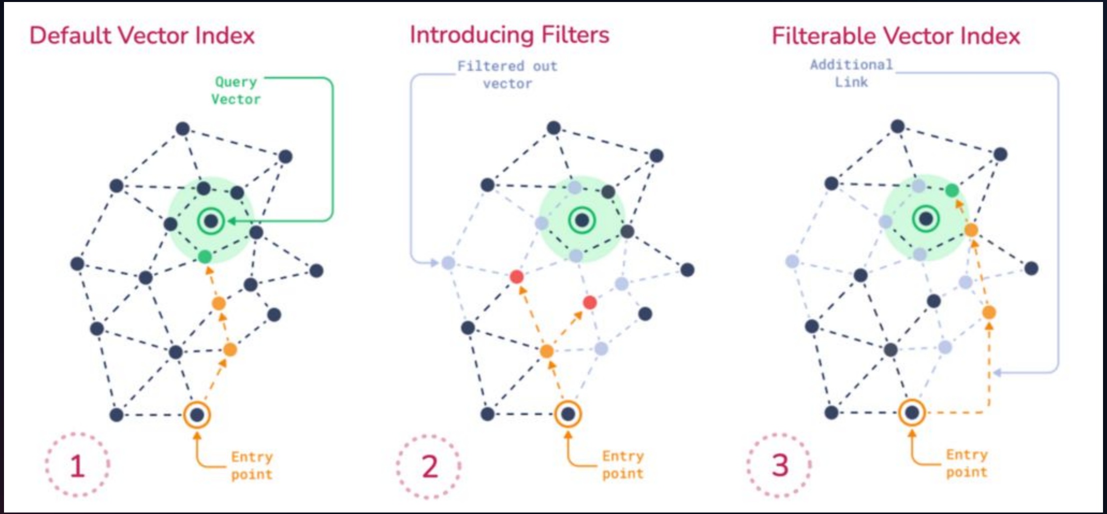
Vectors:

Name: smaller Length: 512

Name: bigger Length: 1024

In Qdrant, full-text filtering gives a **binary answer (there/not there)** when checking documents against the query, while **leaving the final ranking step untouched**.
It's built on top of metadata (called **payload**)

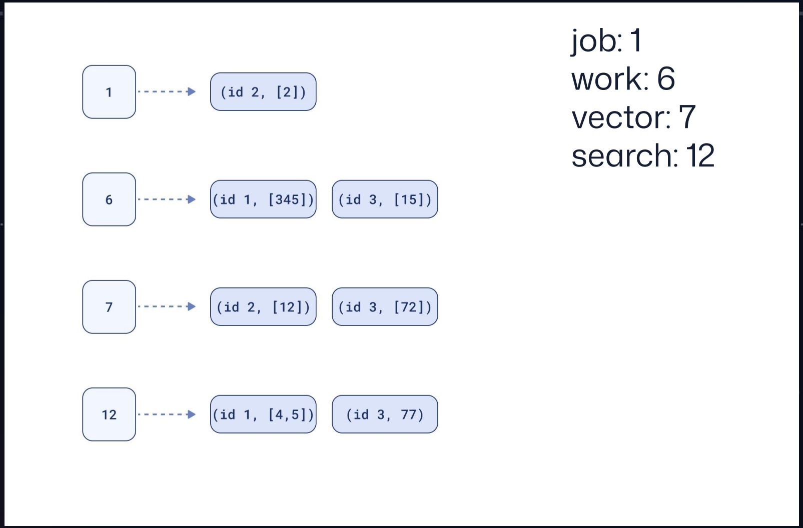
(Text) Filtering + Vector Search



(Text) filtering affects vector search more than you think: many vector indexes weren't designed as-is for filtering conditions.

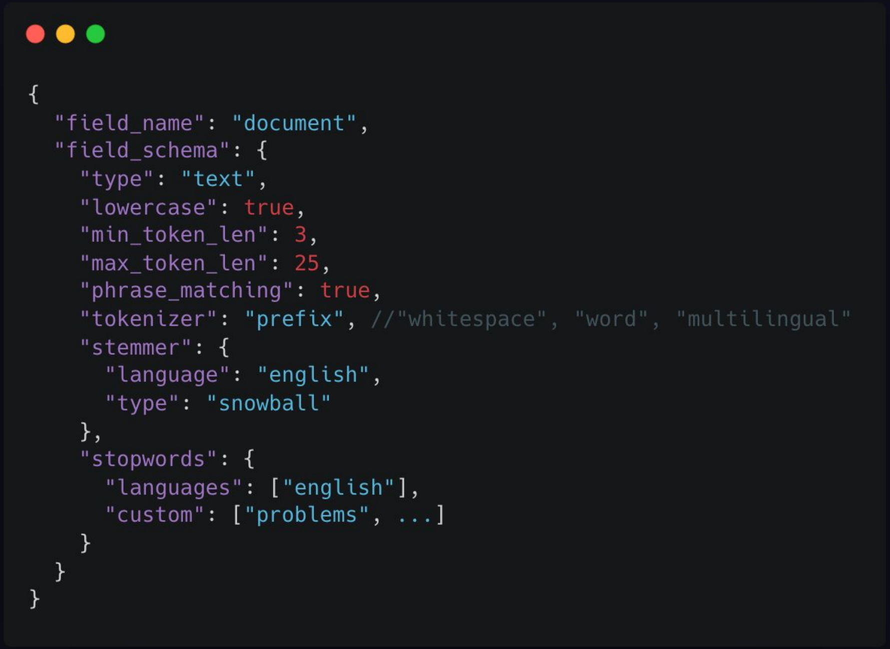
That's why Qdrant has **filterable HNSW**: HNSW vector index adapted for filtering conditions.

Payload index = filters index



Payload index (for **filtering**) is an inverted index storing positions (but no weights, so no ranking).
It must be created **before indexing** data into HNSW, **to make filterable HNSW work**!

Payload index setup



```
{
  "field_name": "document",
  "field_schema": {
    "type": "text",
    "lowercase": true,
    "min_token_len": 3,
    "max_token_len": 25,
    "phrase_matching": true,
    "tokenizer": "prefix", //"whitespace", "word", "multilingual"
    "stemmer": {
      "language": "english",
      "type": "snowball"
    },
    "stopwords": {
      "languages": ["english"],
      "custom": ["problems", ...]
    }
  }
}
```

What can we do with text filters in Qdrant

- Word match (by prefix / full word / stem)
- Phrase match (by prefix / full word / stem)

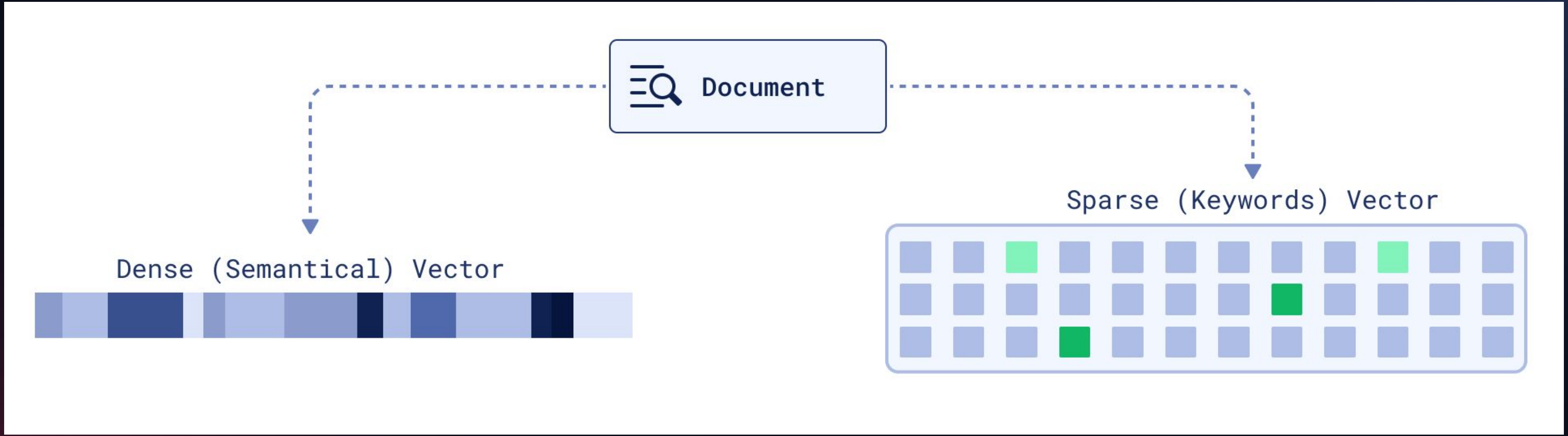
Controlling matching rules with should / shouldn’t, must / mustn’t

What we might introduce in the next versions: proximity rules (not more than 2 words apart)

As all of this makes sense when combined with vector search: generic problems that vectors alone can’t solve.

But full-text filters in Qdrant can’t rank documents –
they only select a subset based on conditions.
If you want keyword matching and ranking by relevance in one step,
you need **sparse vectors**.

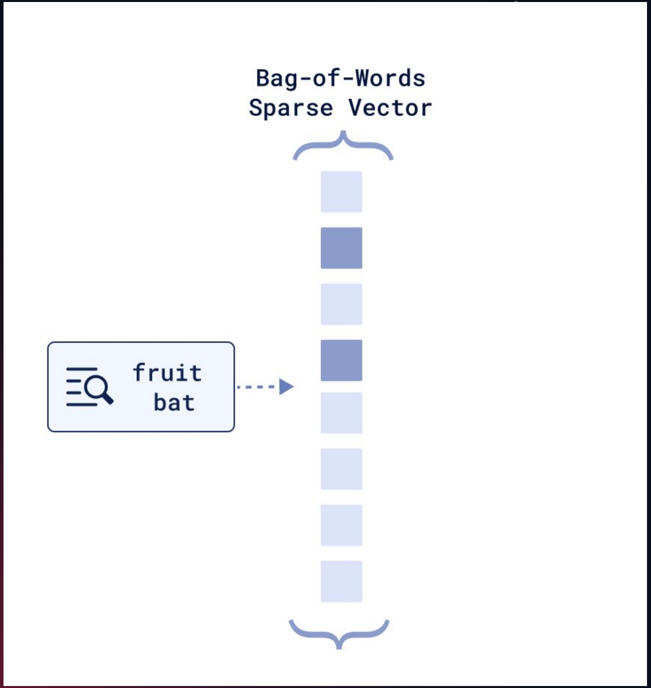
Sparse Vectors



```
         M1   M2   M3   M4   M5   M6   M7   M8   M9   M10
User_1: [ 0,   0,   0,   0,   0,   0,   1,   5,   0,   0 ]
User_2: [ 0,   0,   0,   0,   4,   0,   0,   2,   0,   0 ]
```

```
{
    "indexes": [1, 3, 5, 7],
    "values": [0.1, 0.2, 0.3, 0.4]
}
```

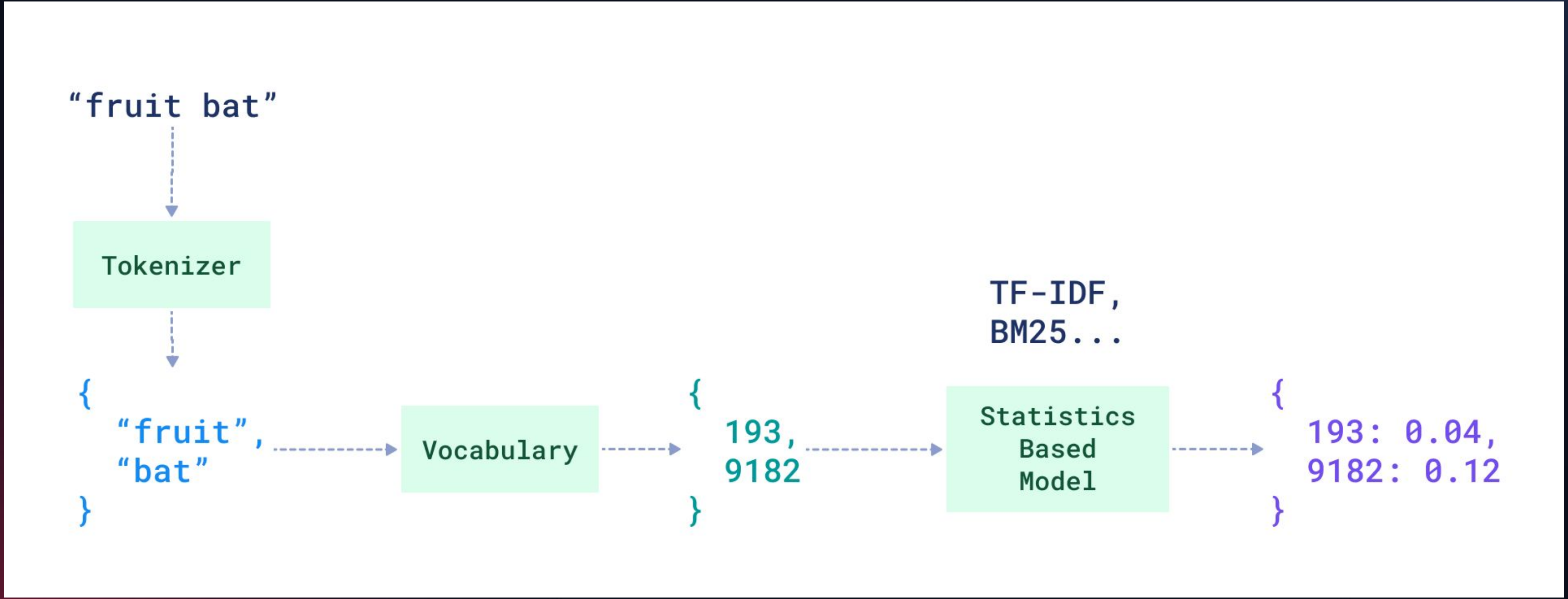
Sparse Vectors for Text Retrieval



Indices (non-zero dimensions) → words in a vocabulary which was built over a dataset / corpus

Values → importance of these words in the encoded text affecting ranking

Sparse Vectors for Text Retrieval: Lexical



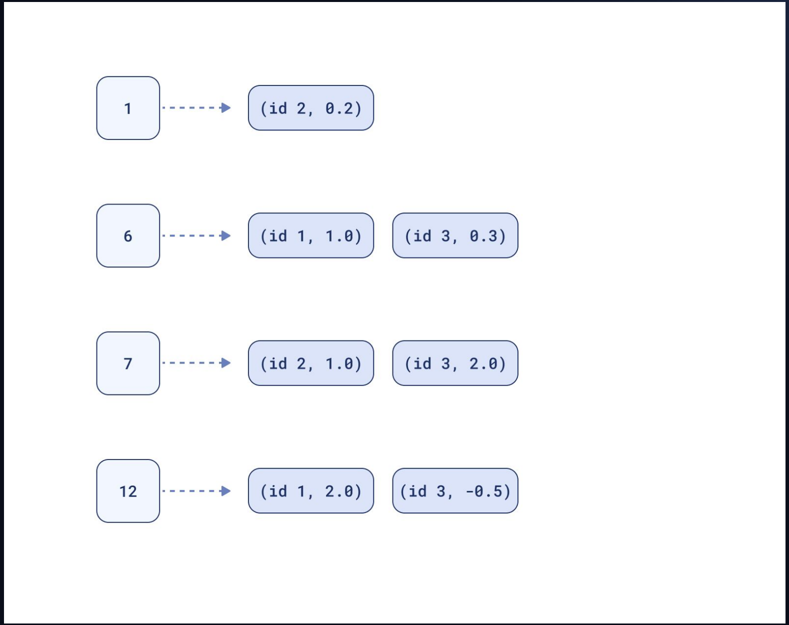
Sparse Retrieval in Qdrant

Done on an inverted index

Difference from payload index:
We store key term weights (but not their positions)

Storing weights makes ranking possible

For example, BM25-based ranking



BM25 in Qdrant

```
PUT /collections/{collection_name}
{
    "sparse_vectors": {
        "text": {
            "modifier": "idf"
        }
    }
}
```

```
PUT collections/{collection_name}/points
{
  "points": [
    {
      "id": 0,
      "payload": {"text": "I'm not saying I don't like the idea of on-the-job training too ... they
aren't qualified to do anything."},
      "vector": {
        "bm25_vector": {
          "model": "Qdrant/bm25",
          "text": "I'm not saying I don't like the idea of on-the-job training too ... they aren't
qualified to do anything.",
          "options": {
            "stopwords": {
              "custom": ["workers"],
              "languages": ["english"]
            }, // ... everything that we have for text index
            "b": 1.2,
            "k": 0.75,
            "avg_len": 50
          }
        }
      }
    },
    ...
    ]
```

Qdrant 1.15.3+

Yet TF-IDF–like retrieval doesn't capture the meaning of words in context.
If you want retrieval based on keywords and with semantic understanding, you might want to try **sparse neural retrieval.**

Sparse Neural Retrieval

Homonyms :

🔎 ”**vectors** in medicine”

⬆️ 📄 ”**vector** control strategies in public health”

⬇️ 📄 “advanced **vector** calculus for engineers”

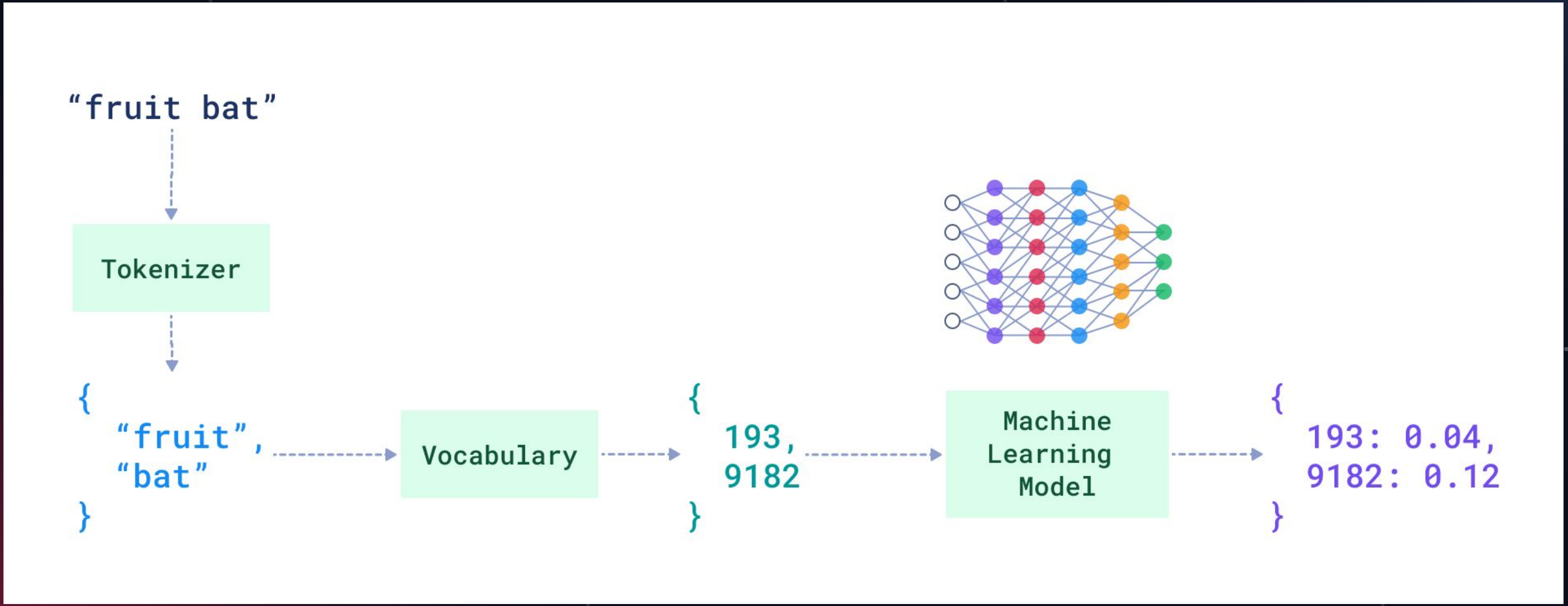
Synonyms:

🔎 ”**vectors** in medicine *health public search*”

⬆️ 📄 ”**vector** control strategies in **public health”**

⬇️ 📄 “advanced **vector** calculus for engineers”

Sparse Neural Retrieval



Sparse Neural Retrieval in Qdrant

`With miniCOIL (homonyms)`

🔎 ”**vectors** in medicine”

⬆️ 📄 ”**vector** control strategies in public health”

⬇️ 📄 “advanced **vector** calculus for engineers”

With SPLADE++ (synonyms)

🔎 ”**vectors** in medicine *health public search*”

⬆️ 📄 ”**vector** control strategies in **public health**”

⬇️ 📄 “advanced **vector** calculus for engineers”

miniCOIL - Qdrant's sparse neural retriever

You can use **SPLADE++** in Qdrant either with **FastEmbed** or with **CLoud Inference**

Sparse Vectors in Text Search

What they should/could be used for:

- Explainable text search retrieval with exact keyword matching

With sparse neural retrieval:

- Retrieval over synonyms / homonyms

What they’re not suitable for:

- Fuzzy matching / semantic similarity search / paraphrasing
- Discovery or recommendations (even though some studies exist)
- Custom ranking over the position of words in the text

Text search in Qdrant

Dense Vectors

- Semantic search
  Fuzzy search (typos, synonyms, etc.)
- Cross-lingual
- Discovery
- Recommendations

Sparse Vectors

- Ranking with control over exact keyword matches (but not keywords position)
- Synonyms / homonyms with sparse neural retrieval

Full Text Filters

Binary pre-filtering with hand-crafted matching conditions

No ranking!

Combining all the building blocks

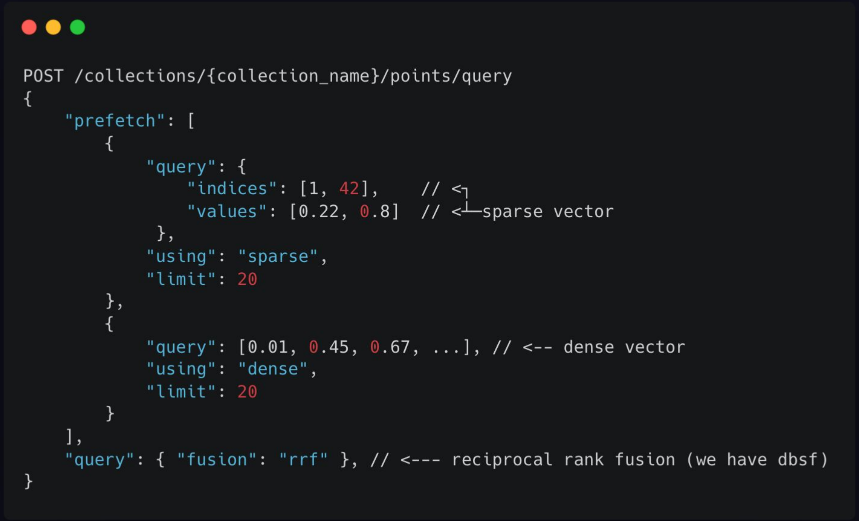
We usually call **hybrid search** = sparse + dense & fusion

But we saw that in practice, “hybrid search” also means other combinations, like:

- Re-ranking with dense vectors on top of sparse retrieval
- Re-ranking based on rules over filterable payload fields

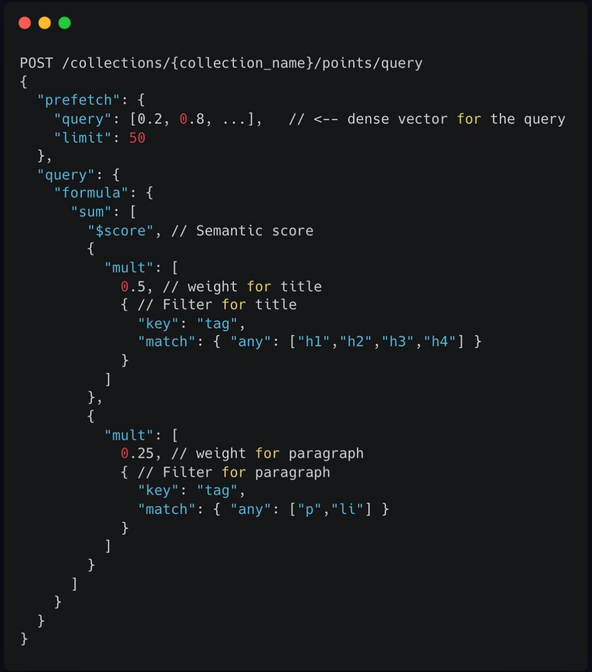
Sure, makes sense, it’s all possible!

Hybrid search: prefetches & fusion



```
POST /collections/{collection_name}/points/query
{
    "prefetch": [
        {
            "query": {
                "indices": [1, 42],    // <┐
                "values": [0.22, 0.8]  // <┴─sparse vector
             },
            "using": "sparse",
            "limit": 20
        },
        {
            "query": [0.01, 0.45, 0.67, ...], // <-- dense vector
            "using": "dense",
            "limit": 20
        }
    ],
    "query": { "fusion": "rrf" }, // <--- reciprocal rank fusion (we have dbsf)
}
```

Hybrid search: custom reranking



```
POST /collections/{collection_name}/points/query
{
  "prefetch": {
    "query": [0.2, 0.8, ...],   // <-- dense vector for the query
    "limit": 50
  },
  "query": {
    "formula": {
      "sum": [
        "$score", // Semantic score
        {
          "mult": [
            0.5, // weight for title
            { // Filter for title
              "key": "tag",
              "match": { "any": ["h1","h2","h3","h4"] }
            }
          ]
        },
        {
          "mult": [
            0.25, // weight for paragraph
            { // Filter for paragraph
              "key": "tag",
              "match": { "any": ["p","li"] }
            }
          ]
        }
      ]
    }
  }
}
```

Our position on text search
Based on three stages of text retrieval

Preprocessing

- Tokenization
- Stemming
- Lemmatization
- Custom synonymization (abbreviations)
- Custom typos correction

This part is problem-specific (doesn't yet have a unique solution) → should be minimally covered and left to the user.

Retrieval

- (Multivector) dense vector search at scale
- Vector search under strict conditions (filters)
- Sparse (neural) retrieval at scale

It should work – and work **fast**.

We're also responsible for **vector-search-specific** interfaces/tools:

- Relevance feedback
- Recommendations
- Discovery search

Reranking

- multivector/sparse/dense reranking
- Rules-based reranking (custom score boosting)
- LTR models, cross-encoders, rerankers (**external**)

This is also problem-specific: we provide the interfaces,
you apply them based on your task

Qdrant

High-Performance Vector Search at Scale

Create your first cluster on Qdrant Cloud.

Evgeniya Sukhodolskaya
Dev Advocate at Qdrant

Thank You