Search Quality Evaluation in the Era of Large Language Models: Dataset Generator + Vector Search Doctor

Speakers: Alessandro Benedetti + Nicolò Rinaldi

Search Solutions - 25th November 2025

Sease | Information Retrieval Applied

Who are we

ALESSANDRO BENEDETTI

Director + R&D Software Engineer @ Sease

- Born in **Tarquinia** (ancient Etruscan city in Italy)
- Master Degree in **Computer Science**
- Program Committee member for **ECIR, SIGIR** and **Desires**
- **Apache Solr Chair of the PMC + Apache Lucene/Solr committer**
- Elasticsearch/OpenSearch expert
- Semantic search, NLP, Machine Learning technologies passionate
- Beach Volleyball player and Snowboarder

Who am I?

NICOLÒ RINALDI

SOFTWARE ENGINEER/DATA SCIENTIST @ Sease

- Born in **Mirandola (Italy)**
- Master Degree in **Data Science** at the University of Padua
- Semantic search, NLP, Machine Learning, Information Retrieval and technologies passionate
- Tennis player and Mountain lover

- Headquarter in London/distributed
- Open-source Enthusiasts
- Apache Lucene/Solr experts
- Elasticsearch/OpenSearch experts
- Community Contributors
- Active Researchers

Sease
INFORMATION RETRIEVAL APPLIED

Hot Trends :

- Large Language Models Applications
- Vector-based (Neural) Search
- Natural Language Processing
- Learning To Rank
- Document Similarity
- Search Quality Evaluation
- Relevance Tuning

Mission

"Sease **mission** is to make **research** in **Information Retrieval** more accessible to an **industry** audience, transforming the **best research principles**, **ideas** and **implementations** from **academia** into **real-world** products."

"Firmly believing **Open Source** is the way, Sease puts strong effort in **contributing code back** to the **community**, **supporting public mailing lists** and evangelising R&D at **world class conferences**."

Tutorial material

rated-ranking-evaluator repo:

llm-search-quality-evaluation repo:

llm-search-quality-evaluation-tutorial repo:

Overview

1. Search Quality Evaluation & Dataset
2. Vector Search Doctor: exact
3. Vector Search Doctor: approximate
4. Key Takeaways & Q&A

Overview

1. Search Quality Evaluation & Dataset
2. Vector Search Doctor: exact
3. Vector Search Doctor: approximate
4. Key Takeaways & Q&A

Information Retrieval

"**Information retrieval (IR)** is the activity of obtaining information system resources relevant to an **information need** from a **collection of information resources.** Searches can be based on full-text or other content-based indexing. Information retrieval is the science of searching for information in a document, searching for documents themselves, and also searching for metadata that describe data, and for databases of texts, images or sounds."
Wikipedia

Query

Type to search

Corpus

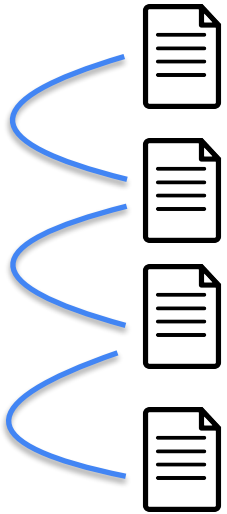
Search Relevance

MATCHING

- Identifying relevant documents from the corpus

RANKING

- Ordering the candidate documents retrieved by decreasing relevance

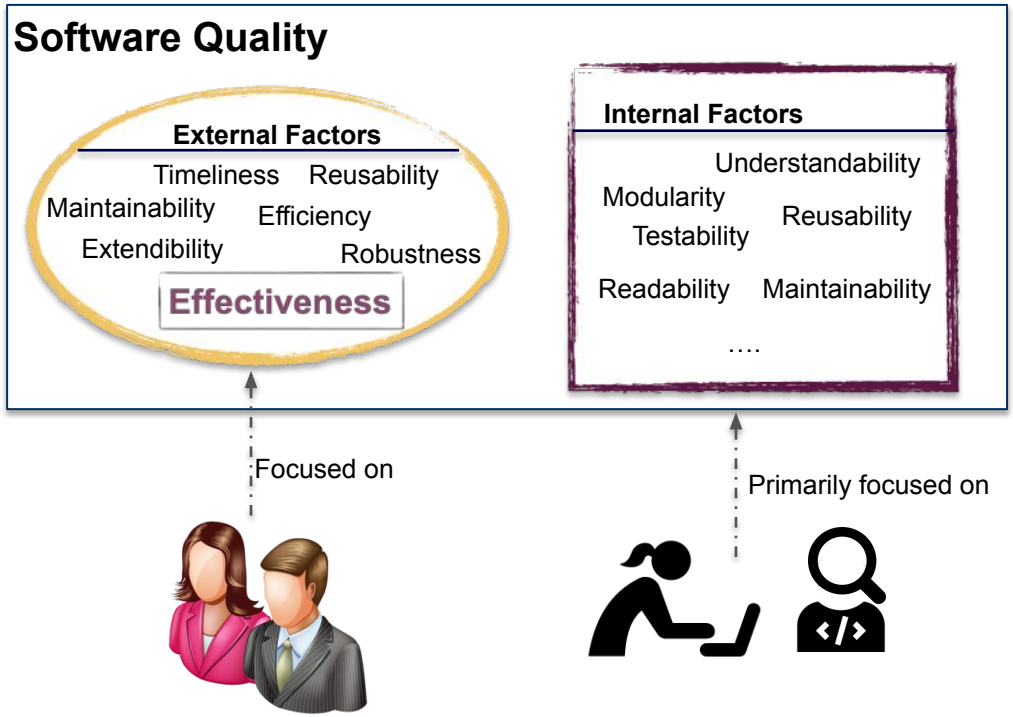


Search Quality Evaluation

Search Quality Evaluation is the activity of assessing how good a search system is.

Defining what **good** means depends on the interests of **who** (stakeholder, developer, ect...) is doing the evaluation.

So it is necessary to measure **multiple metrics** to cover all the aspects of the perceived quality and understand how the system is behaving.

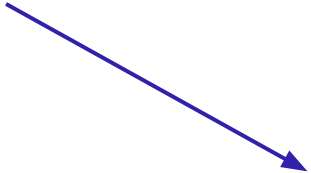


Relevance is hard!

In **Information Retrieval**, the **effectiveness** is the ability of the system to **retrieve relevant documents** while at the same time **suppressing the retrieval of non-relevant documents.**

You **need**:

- understanding user intent and domain
- understanding business stakeholders
- content curation
- search technologies expertise



- Relevance iterative tuning
- Continuous feedback
- Search quality evaluation
- Content refinement

Why measure search quality?

- You **can't improve** what you **can't measure**
- It opens a tangible **comparison between more query designs**
- It **reveals** dataset and model **weaknesses**
- It's the foundation for **long-term iteration**
- It unlocks **continuous improvement** through measurement

Image by juicy_fish on Freepik

Goal of Search Evaluation

- Assess how **effective** a search system is at retrieving relevant results for a given user query, i.e., **satisfy the users' information needs**.
- Measure the **alignment** between system **output** and **human judgment** of relevance.

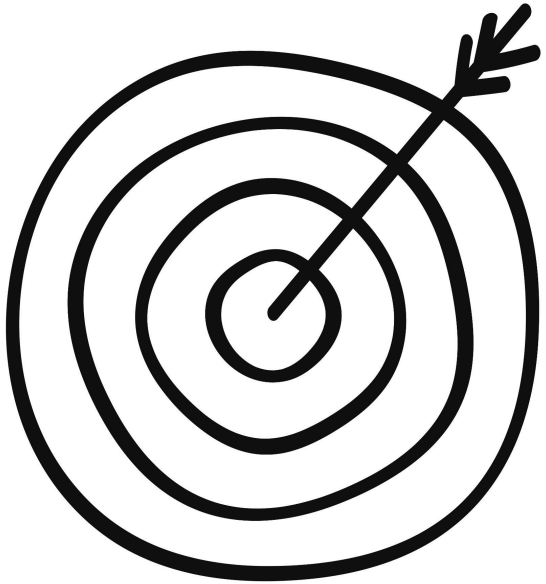


Image by juicy_fish on Freepik

Implicit and Explicit feedback

Implicit

- Feedback inferred **automatically** from user behavior.
- Clicks on search results, dwell time, scroll depth, query reformulations, skipped results.
- **Collected passively**: no extra effort from the user.
- Easy to gather at scale.
- Reflects real-world behavior.
- Continuous and unobtrusive.
- Noisy and ambiguous (e.g., click ≠ satisfaction)

Explicit

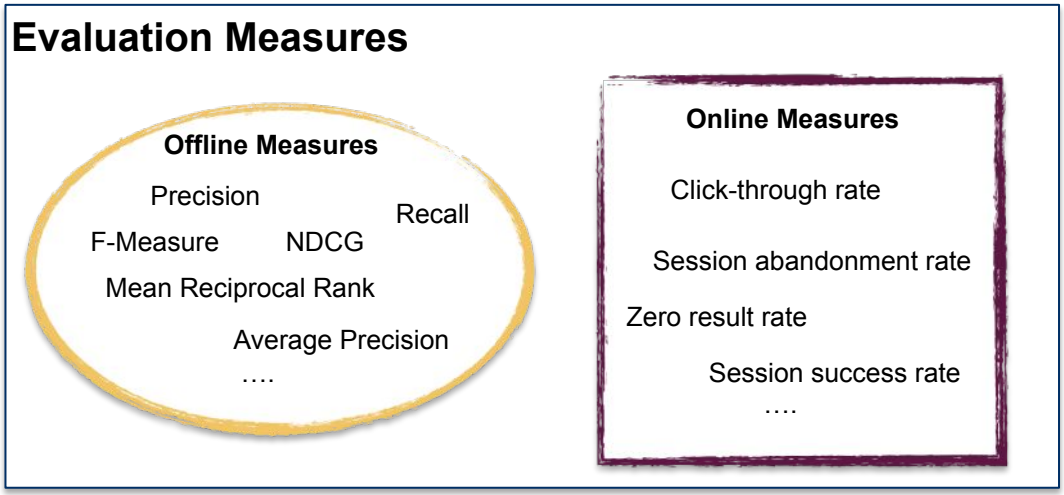
- Feedback directly provided by users in a **deliberate** way.
- Ratings (e.g., 1–5 stars), relevance judgments, thumbs up/down, written comments, likes.
- **Collected actively**: user must intentionally provide feedback.
- High precision and interpretability.
- Direct measure of perceived relevance.
- Expensive and time-consuming.
- Hard to scale and may suffer from subjectivity.

Evaluation Measures

Evaluation measures for an information retrieval system formalise how well a search system satisfies its user information needs.

Measures are generally split into two categories: **online** and **offline measures**.

We'll start from **online measures.**



Online Measures

Using online testing can lead to many **advantages**:

- The **reliability** of the results: we directly observe the user behaviour.
- The **interpretability** of the results: we directly observe the impact of the model in terms of online metrics the business cares.
- The possibility to observe the **model behavior**: we can see how the user interact with the model and figure out how to improve it.

Signals to measure

- Click Through Rates (views, downloads, add to cart ...)
- Sale/Revenue Rates
- Dwell time (time spent on a search result after the click)
- Query reformulations/ Bounce rates
- ....

Recommendation: monitor the signals more important to your business!

Business Advantages

Using online testing can lead to many **advantages**:

- The **reliability** of the results: we directly observe the user behaviour.
- The **interpretability** of the results: we directly observe the impact of the model in terms of online metrics the business cares.
- The possibility to observe the **model behavior**: we can see how the user interact with the model and figure out how to improve it.

Offline measures

Advantages:

- Find **anomalies in data**, like: weird distribution of the features, strange collected values, ...
- Check how the **ranking function performs** before using it in production: implement improvements, fix bugs, tune parameters, ...
- Save **time** and **money**. Put in production a bad ranking function can worse the user experience on the website.

Offline measures: some examples

Usually in IR we are interested in the top k results, that the meaning of the “@k” in the following metrics: they consider just the top k results in the computation

- Normalized Discounted Cumulative Gain at k (**nDCG@k**)
- Mean Reciprocal Rank at k (**MRR@k**)
- Mean Average Precision at k (**MAP@k**)
- Recall at k
- Precision at k

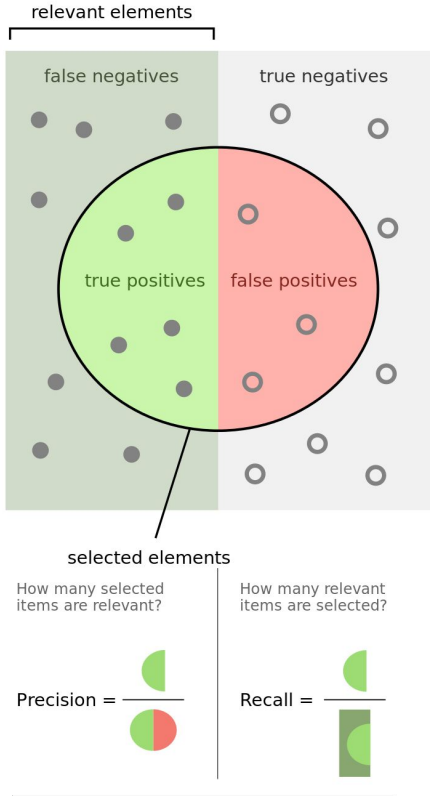
Precision

- **precision** = Ratio of relevant results among all the search results returned
- **precision@K** = Ratio of relevant results among the top-K search results returned

What happens if :

precision@k ↘ means fewer <relevant results> in the top-K

precision@k ↗ means more <relevant results> in the top-K



Normalized Discounted Cumulative Gain

- DCG@K = Discounted Cumulative Gain@K

$$\mathrm{DCG}_K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

relevance weight

result position

It measures the usefulness, or gain, of a document based on its position in the result list.

	Model1	Model2	Model3	Ideal
	1	2	2	4
	2	3	4	3
	3	2	3	2
	4	4	2	2
	2	1	1	1
	0	0	0	0
	0	0	0	0
DCG	14.01	15.76	17.64	22.60
NDCG	0.62	0.70	0.78	1.0

Normalised Discounted Cumulative Gain

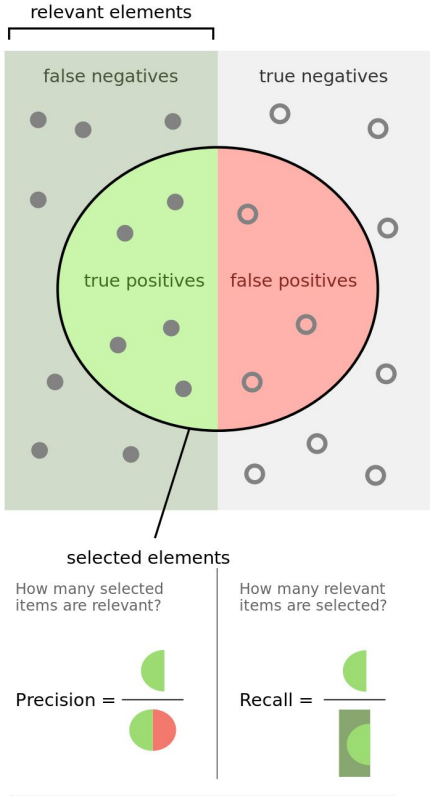
- NDCG@K = DCG@K/ Ideal DCG@K
- It is in the range [0,1]

Recall

- **recall** = Ratio of relevant results found among all the relevant results
  i.e.
  Out of all the relevance results, how many have you found?

- **recall@k** = Ratio of all the relevant results, you found in the top-K
  i.e.
  Out of all the relevance results, how many have you found in the top-K?

  It means how much the topK is covering all relevant results. It can be useful to understand if you need to increase the size of pages, for recall intensive applications.



What happens if recall@k : means that your top-K is covering a smaller ratio of all relevant

Offline vs Online

- Both Offline/Online evaluations are vital for a business

Offline

- doesn't affect production
- allows research and experimentation of wild ideas
- reduces the number of Online experiments to run
- Fast, repeatable experiments

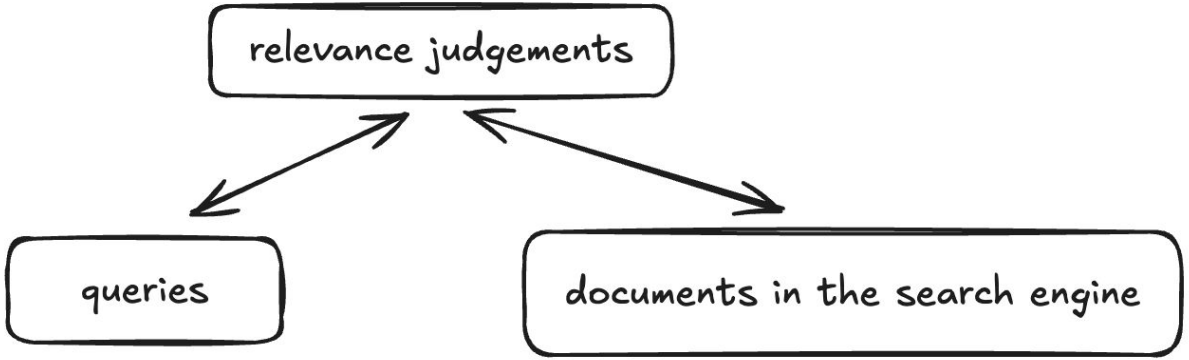
Online

- measures improvements/ regressions with real users
- isolates the benefits coming from the relevance model
- Slower, requires live traffic

Offline search quality evaluation data

In the scenario of offline search quality evaluation, the main obstacle is the lack of **queries** and **relevance judgements**.

Without labeled **query–document pairs**, it becomes difficult to assess retrieval quality.



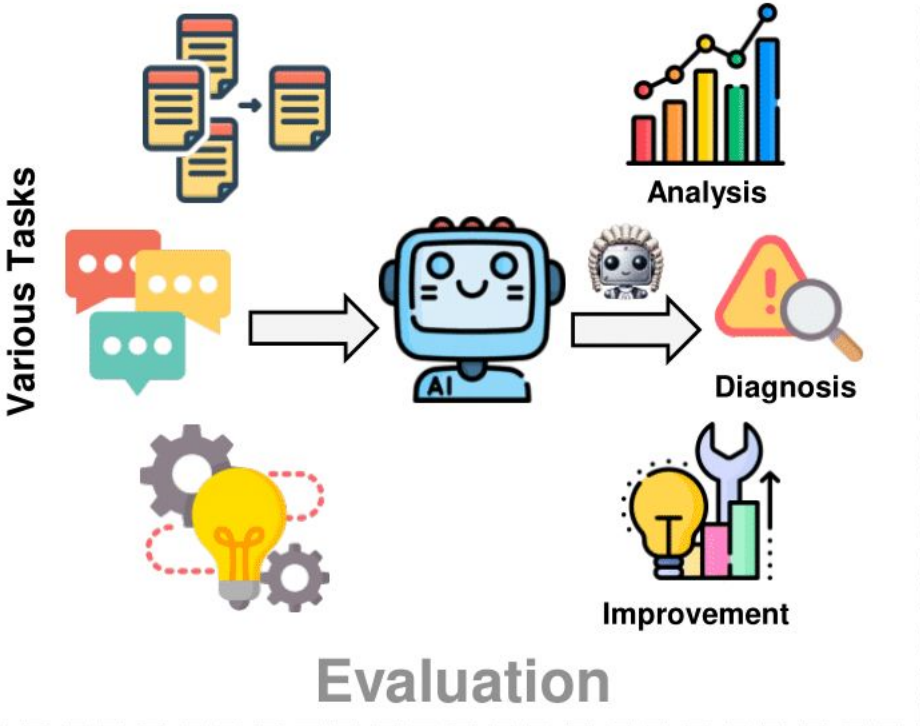
The Challenge of Obtaining Labeled Data

- **Expensive & time-consuming:** Requires expert annotators.
- **Accuracy:** Labels become less accurate due to the fact that labeling is a boring job and requires focus for a long period of time.
- **Subjective:** Relevance may depend on context, user intent, or domain.
- **Scalability issue:** Hard to label enough data for large corpora or many queries.

Image by vectorjuice on Freepik

Why LLMs change the game for synthetic dataset generation

LLM-as-a judge for evaluation



Project/Tutorial goals

Our goal is to make you life easier trying to address the following questions:

- What can I do if I want to **test my search engine** but I don't have queries and ratings?
- Is there a **straightforward way to leverage LLM** without creating my own pipeline?
- What **embedding model** should I use on my corpus?
- Is the **vector search implementation** I'm using good enough?

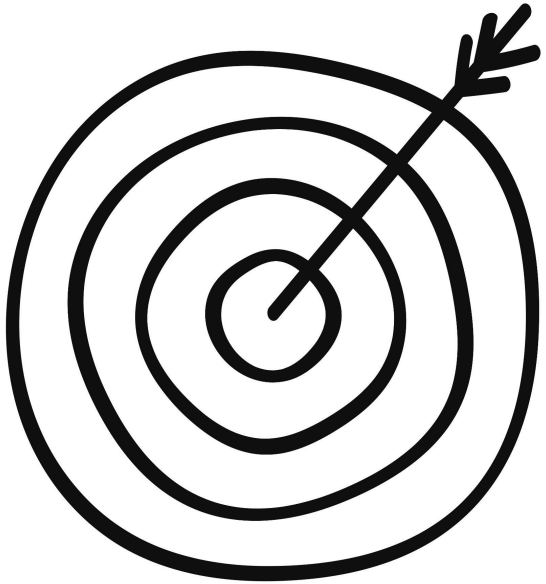
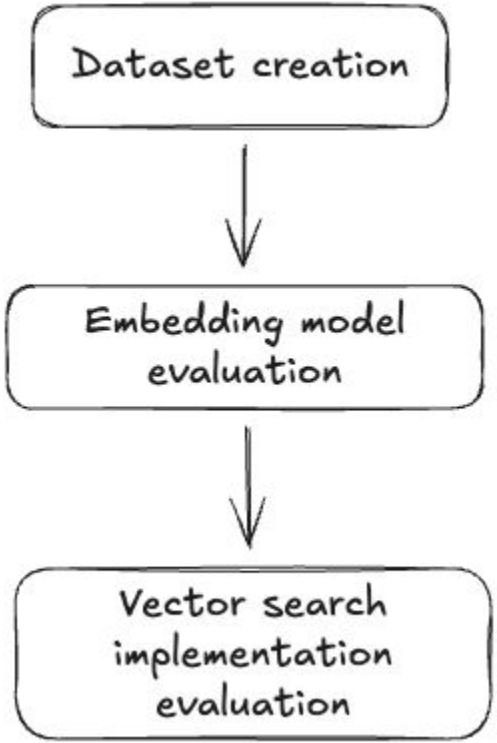


Image by juicy_fish on Freepik

Overview of the our solution



Our first tool: Dataset generator

Our team

DANIELE ANTUZI
R&D SOFTWARE ENGINEER
SEARCH CONSULTANT

NICOLÒ RINALDI
SOFTWARE ENGINEER/DATA SCIENTIST

NAZERKE SEIDAN
IR/ML
SOFTWARE ENGINEER

Assumption: you have only the documents indexed

Generate queries

Generate relevance labels (with rationale)

Functionalities

Query Input & Data Management

- Possibility to add a base of user queries if collected (e.g., with logs)
- Possibility to enable periodical caching of generated relevance judgements and queries:
  - minimal waste on crashes
  - reproducibility

Interpretability

- Possibility to enable LLM “explainability”
- Possibility to enrich the set of document scored for each query using a query template and retrieve more documents directly from your search engine

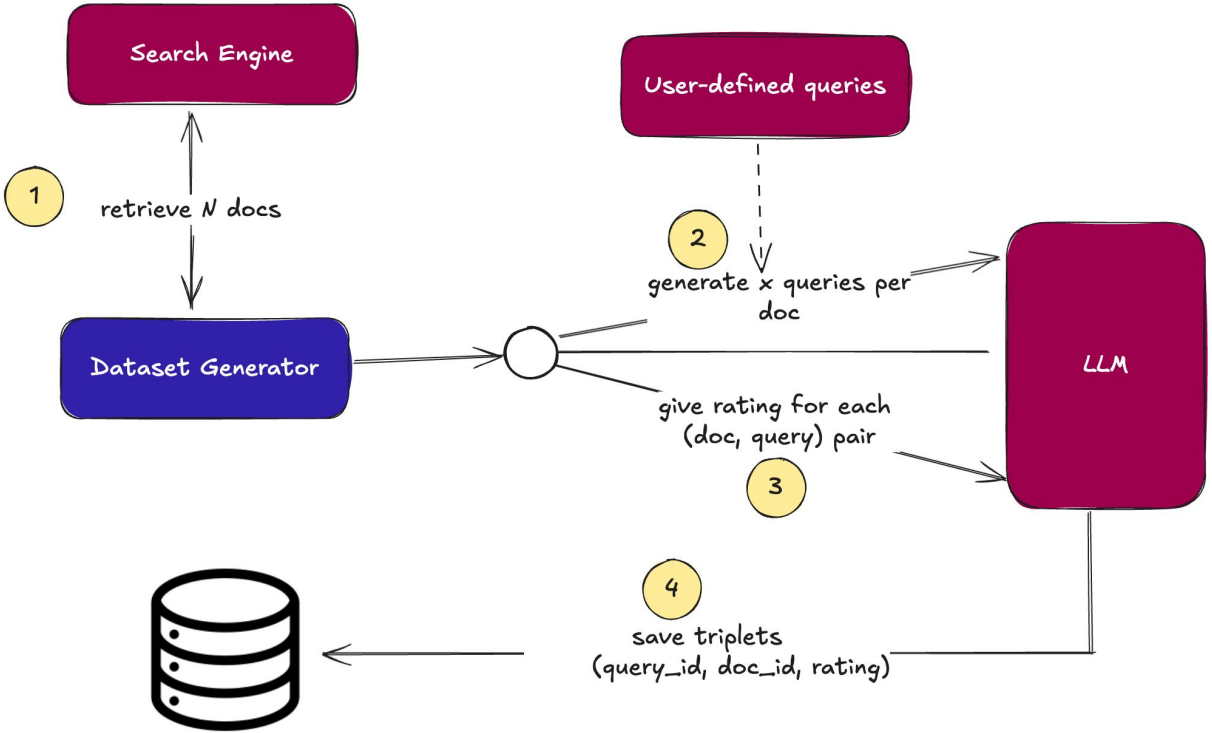
Output formats & Integration

quepid

mteb

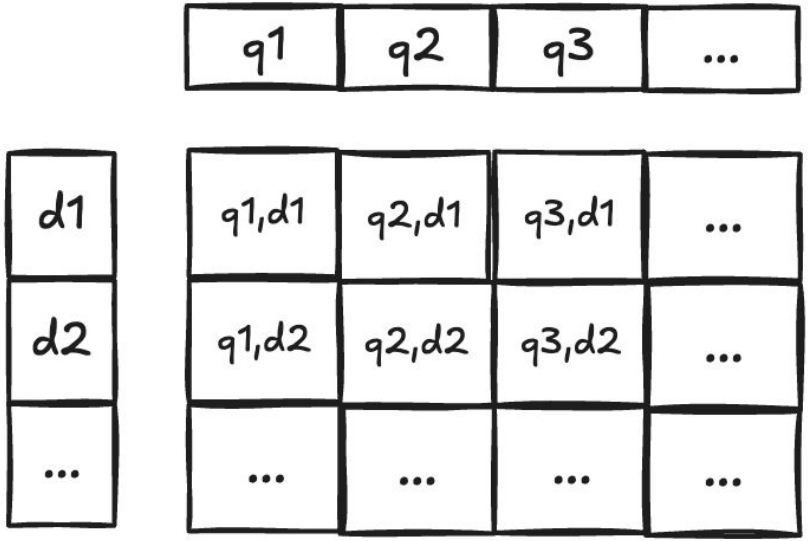
RRE

First step: query generation



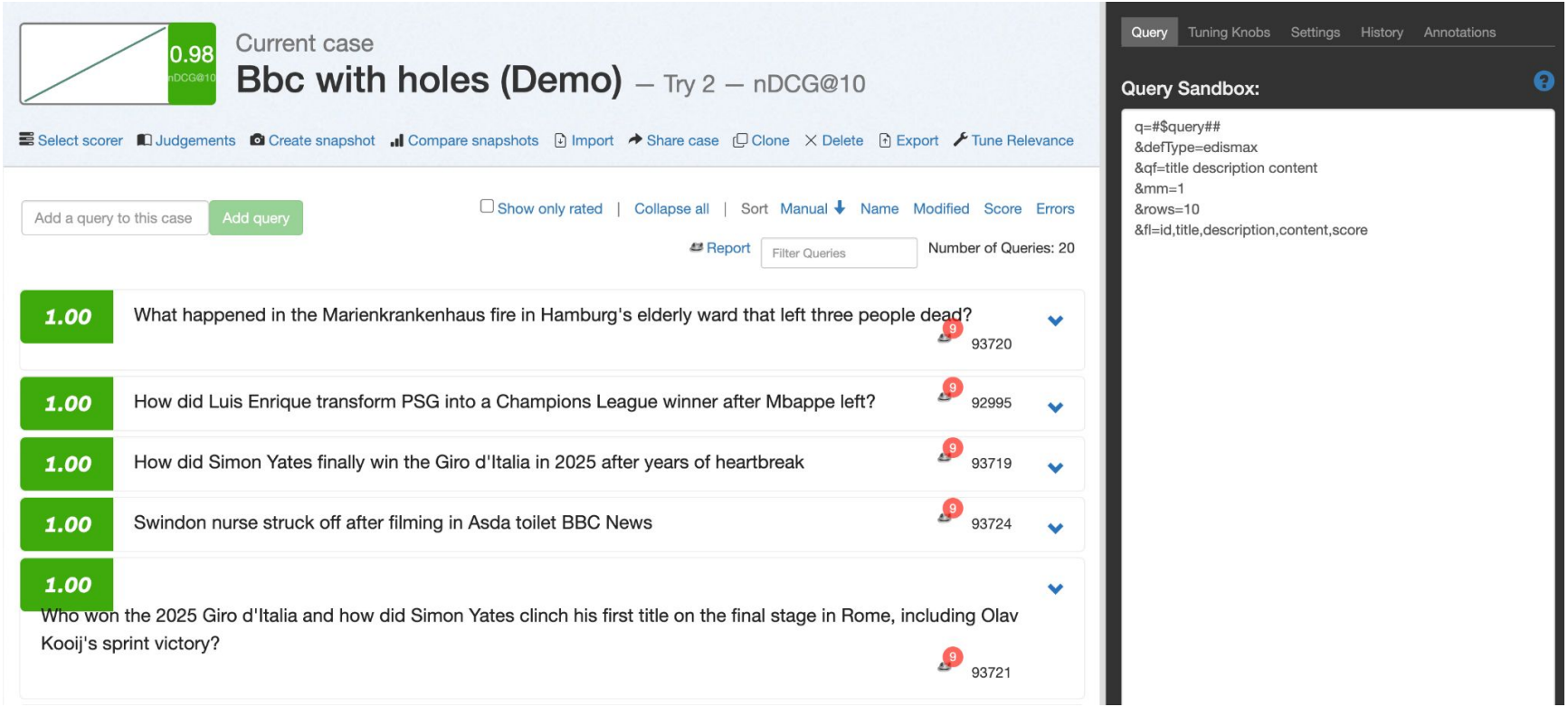
About cartesian product

3

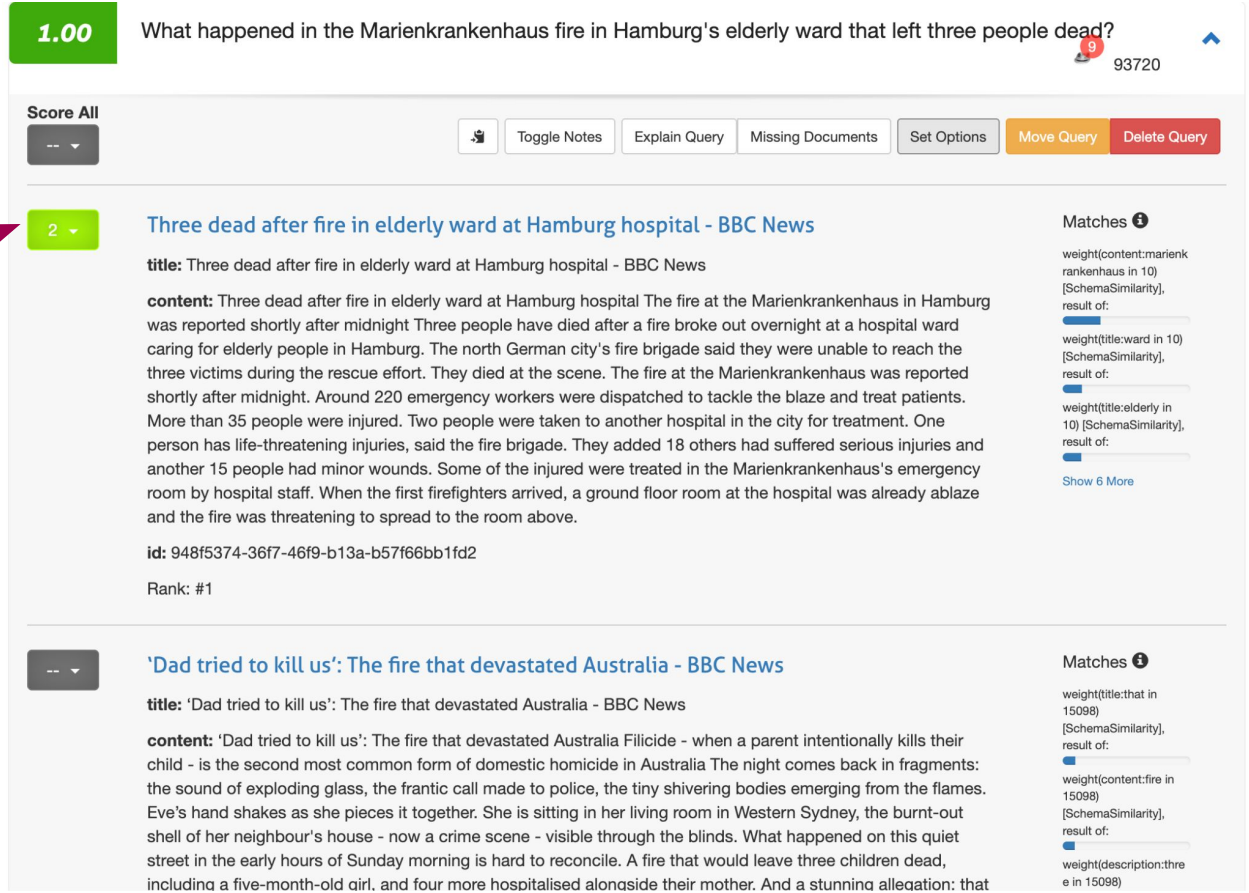


Recommendation: disable this if not strictly needed, since it increases consistently the number of LLM calls!

Second step: retrieval with query template



Only first one is rated!

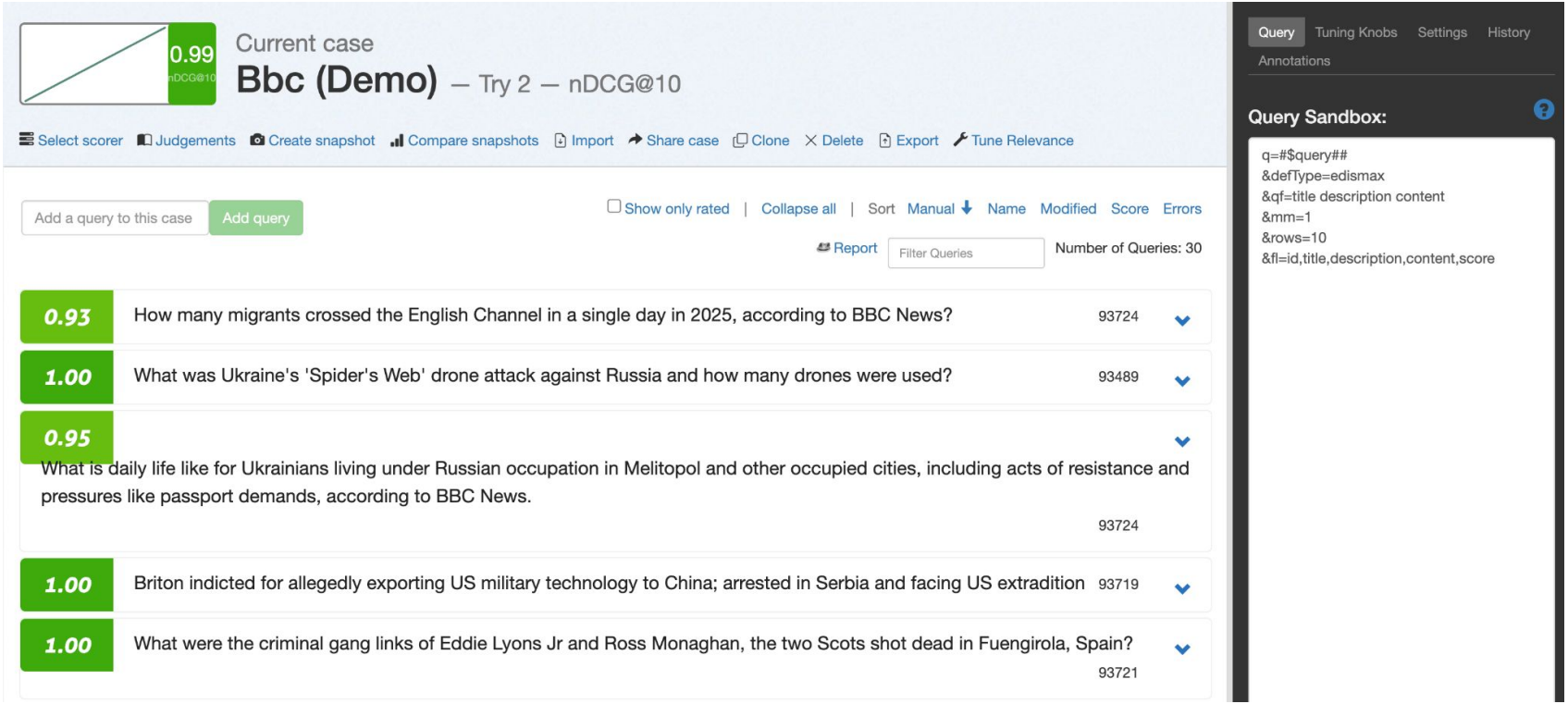


Why using a template is important

- Documents in input may be a subset of entire corpus
- Generated queries may retrieve un-rated documents
- You can run the queries and use the LLM to fill the gaps

In this way, you can also visualize your results with Quepid for example.

After using the template: all retrieved docs are rated!



Input

Configuration file

```
query_template: "templates/template_solr.json"          <- be careful: this file must follow RRE guidelines
search_engine_type: "solr"                              <- for now: support just for Solr
collection_name: "testcore"
search_engine_url: "http://localhost:8983/solr/"        <- instance MUST be up and running
documents_filter:                                       # if any
  - genre:
      - "horror"
number_of_docs: 100
doc_fields:
  - "title"
  - "content"
queries: "queries.txt"                                  # if any
generate_queries_from_documents: true                   <- if not set: generation enabled
num_queries_needed: 10
relevance_scale: "graded"                               <- graded ({0, 1, 2}), binary ({0, 1})
llm_configuration_file: "configs/dataset_generator/llm_config.yaml"
max_query_terms: 5                                      <- if not set: let LLM decide
output_format: "quepid"                                 <- “rre”, “quepid”, “mteb”
output_destination: "resources"
save_llm_explanation: true
llm_explanation_destination: "resources/rating_explanation.json"
datastore_autosave_every_n_updates: 50                   <- if not set: autosave disabled
enable_cartesian_product: false                         <- if not set: cartesian product enabled
```

Query template file structure

```
"queries": [
  {
    "template": "only_q.json",
    "placeholders": {
      "$query": "fender"
    }
  },
  {
```

```
{
  "template": "filter_by_language.json",
  "placeholders": {
    "$query": "Fender",
    "$lang": "eng"
}
```

only_q.json

```
{
  "q" : "$query"
}
```

filter_by_language.json

```
{
  "q" : "$query",
  "fq" : "language:$lang"
}
```

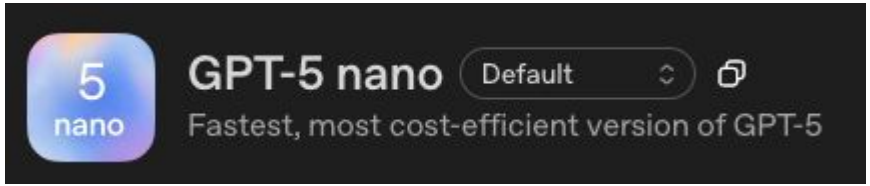
Example of llm_config.yml file

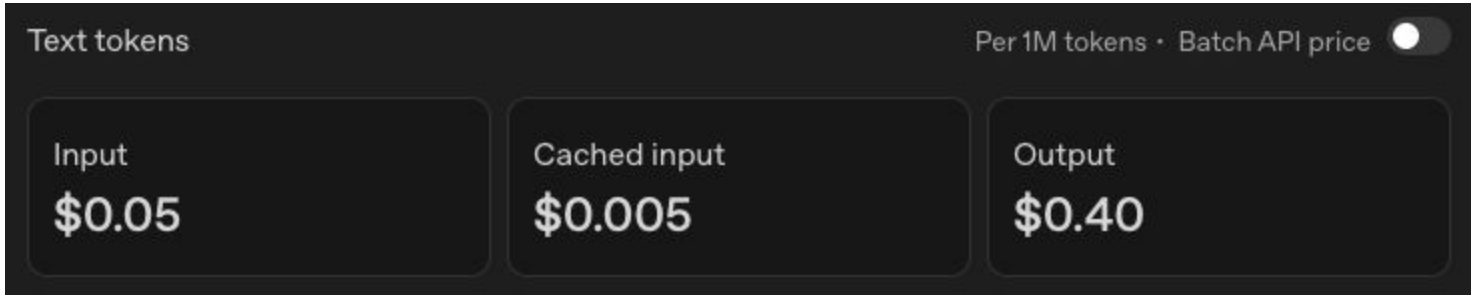
```
# OpenAI LLM
name: openai            # supported: openai, gemini

# Chat model name
model: gpt-5-nano-2025-08-07

# Maximum number of tokens the model may return
max_tokens: 2000

# Environment variable where LLM API key is stored
api_key_env: OPENAI_API_KEY
```

Some cost analysis...

https://platform.openai.com/docs/models/gpt-5-nano

Reference model cost





Search engine setup

Statistics: we have on average 1,056 words each document in the field “content” (the one we are going to use in this cost analysis) → # tokens = 1408 by the 4/3 rule of thumb

Solr

Queries generation process

Input tokens:

- **Internal prompt length** impacts the cost (max_query_terms)
- **Internal schema** for structured output (more reliable): dealt internally by OpenAI, let's say 100 tokens

Output tokens:

- **Generated query** with:
  - Query terms = Null → # tokens = ~25
  - Query terms = n → # tokens = n · 4/3

Rating generation process

Input tokens:

- **Internal prompt length** impacts the cost (explainability)
- **Internal schema** for structured output (more reliable): dealt internally by OpenAI, let's say 100 tokens
- Query with max_query_terms: # tokens = ~25 + 3 prefix

Output tokens:

- **Score**:
  - with explanation → # tokens = 6
  - with explanation → # tokens = 15 + **explanation tokens**

Cost analysis

Setup configuration:

- num_queries_needed: 100
- No queries (manually added input queries)
- No max_query_terms
- No explanation
- number_of_docs: 200
- No cartesian product
- Template to retrieve 10 documents (rows=10)

Operations:

- Queries generation: 100
- Ratings generations: 10 each query

Total tokens:

- Input:
  - Queries generation: 1723 each call = 172.300
  - Ratings generations: 1591 each call = 1.591.000
  - Total: 1.763.300
- Output:
  - Queries generation: 25 each call = 2.500
  - Ratings generations: 6 each call = 6000
  - Total: 8500
- TOTAL COST: 0.092$

How to run Dataset Generator

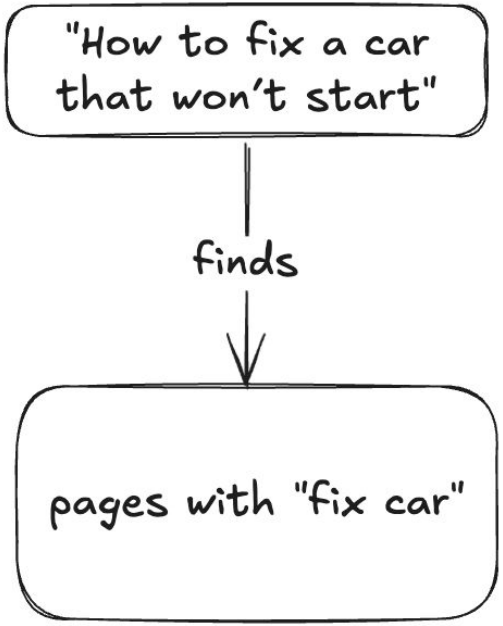
```
Step 1: ensure that your search engine is up and running

Step 2: clone rated-ranking-evaluator repo

Step 3: cd rre-tools

Step 4: prepare configuration file

Step 5: uv run dataset_generator

That's it!
```

Demo time!

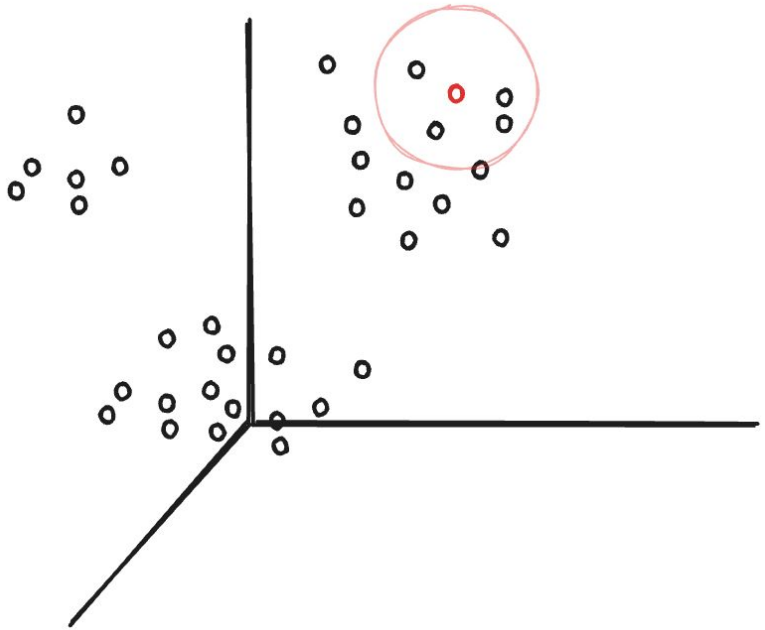
Overview

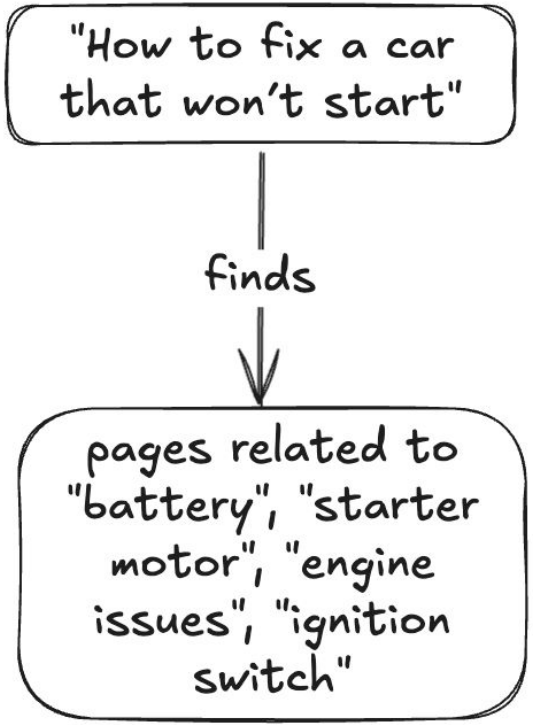
1. Search Quality Evaluation & Dataset
2. Vector Search Doctor: exact
3. Vector Search Doctor: approximate
4. Key Takeaways & Q&A

From keywords to semantic understanding

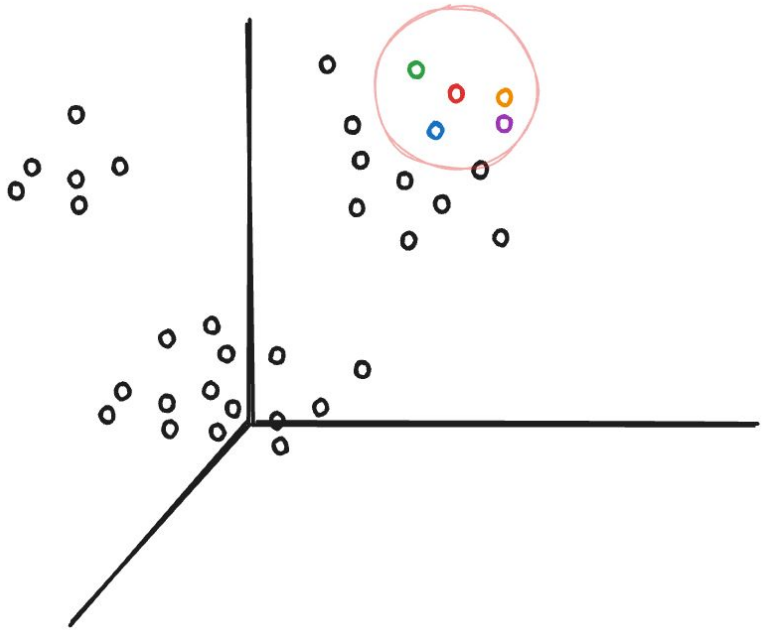


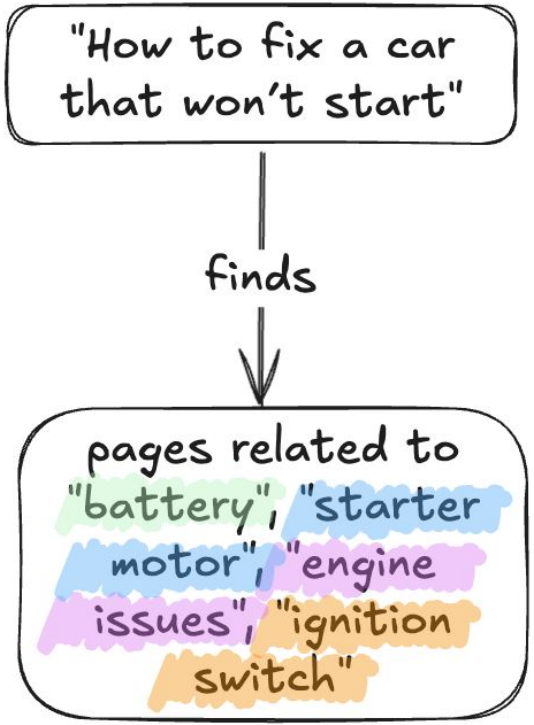
From keywords to semantic understanding



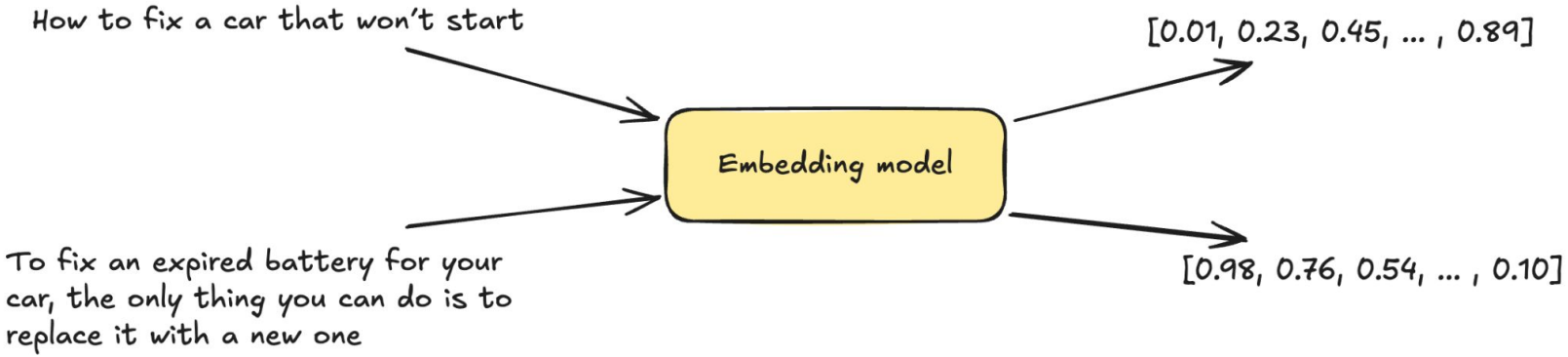


From keywords to semantic understanding

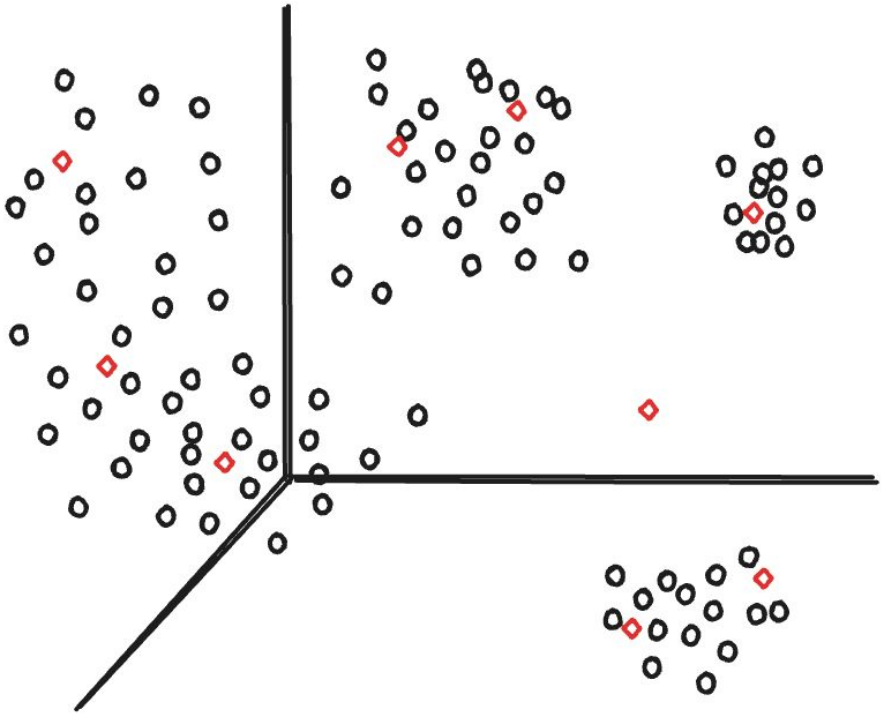




Embedding models: transforming text to vectors



Shared embedding space for queries and documents



Similarity metrics

$$dot\_product = < query\_emb, doc\_emb >$$

If vector are not normalized, the dot product depends heavily on the order of magnitude of the entries of the vector

$$cosine\_similarity = \frac{< query\_emb, doc\_emb >}{||query\_emb|| \cdot ||doc\_emb||}$$

More flexible, since it looks only at the angle between the vectors

Which one to pick? Usually **depends on the embedding model** you are using!

Problems of Exact Vector Search

- **Embedding model doesn’t suit your data**
  - Documents and queries **length mismatch**
  - **Domain-specific** terminology challenges

How do we choose the model properly? Hugging Face MTEB Leaderboard

"MTEB is a Python framework for evaluating embeddings and retrieval systems for both text and image. MTEB covers more than 1000 languages and diverse tasks, from classics like classification and clustering to use-case specialized tasks such as legal, code, or healthcare retrieval."

We are mostly interested in retrieval and reranking related tasks.

Why can’t we trust embedding models blindly?

- **Lack of Transparency**
- Embeddings reflect training **biases**, not objective meaning
- Context dependence and domain drift

Solution: evaluate the embedding model on your data

Why evaluate the embedding model on your data?

- **Leaderboard performance** ≠ **your dataset performance**
- **Domain-specific evaluation** necessity
- Possibility to **compare multiple embedding models** on your data and choose the most suited one

Vector Search Doctor: exact

Goal: understand if the Embedding model you choose fits your data

How does Vector Search Doctor do that?: computes the metrics for your dataset and retrieves the metrics from the MTEB leaderboard, so you can compare and see if your model choice is good or not for your data

Input

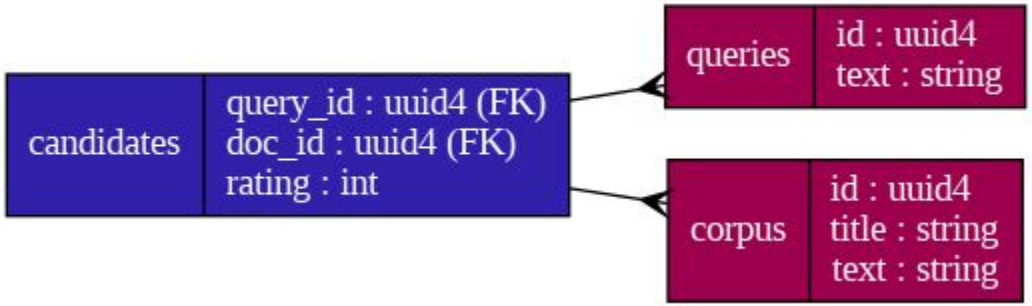
Configuration file

```
model_id: "sentence-transformers/all-MiniLM-L6-v2"
task_to_evaluate: "retrieval"                    <- “retrieval”, “reranking”
corpus_path: "resources/corpus.jsonl"
queries_path: "resources/queries.jsonl"
candidates_path: "resources/candidates.jsonl"
relevance_scale: "graded"                        <- “binary”, “graded”
output_dest: "resources"
embeddings_dest: "resources/embeddings"
```

About corpus, queries and candidates

This is the **format required** by **MTEB** framework to work with retrieval and reranking tasks.

The easiest way to obtain this 3 files is the **export with mteb** format from dataset generator.



An example: corpus, queries and candidates

```
{"id": "d1", "title": "Helicopter Crashes in Colombian Drug War, Kills 20", "text": "BOGOTA, … army said."}
{"id": "d2", "title": "Police Use Taser on Python to Free Man", "text": "UNIONTOWN, Pa. … not let go."}
{"id": "d3", "title": "Zidane apologizes for head butt", "text": "French soccer … and sister."}
{"id": "d4", "title": "Iraqi PM Says Sticking to January Election Plan", "text": "Iraqi Prime … insurgent."}
{"id": "d5", "title": "Ga. Crematory Operator to Plead Guilty", "text": "Relatives … guilty plea."}
{"id": "d6", "title": "Long-driving Kuehne finally breaks out of 2004 slump", "text": "Hank Kuehne … Kuehne"}
```

```
{"id": "q1", "text": "What is the nature and purpose of the mission the helicopter was on?"}
{"id": "q2", "text": "How did the helicopter crash in the Colombian jungle?"}
{"id": "q3", "text": "What is the involvement of U.S. technology or personnel in the Colombian drug war?"}
{"id": "q4", "text": "Why did the police use a Taser on the python?"}
```

```
{"query_id": "q1", "doc_id": "d1", "rating": 2}
{"query_id": "q2", "doc_id": "d1", "rating": 2}
{"query_id": "q3", "doc_id": "d1", "rating": 2}
{"query_id": "q4", "doc_id": "d2", "rating": 2}
{"query_id": "q1", "doc_id": "d2", "rating": 0}
{"query_id": "q2", "doc_id": "d2", "rating": 0}
```

The models: HuggingFace Sentence Transformers

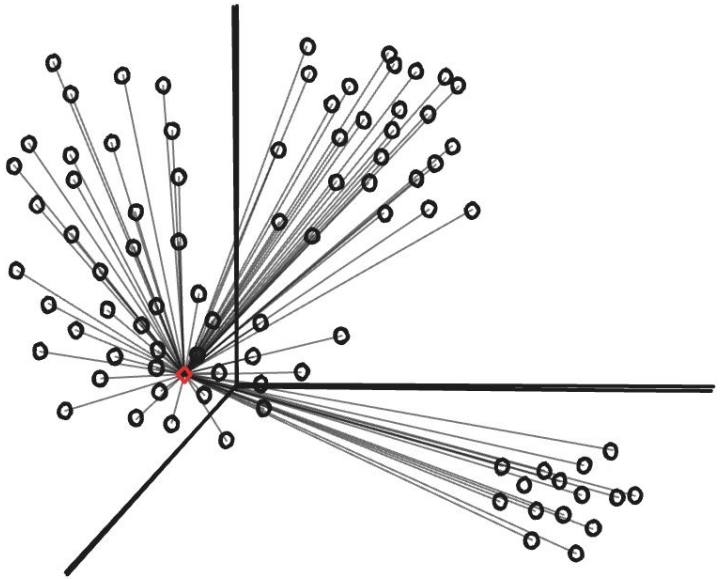
What we are going to use for this demo:
sentence-transformers/all-MiniLM-L12-v2

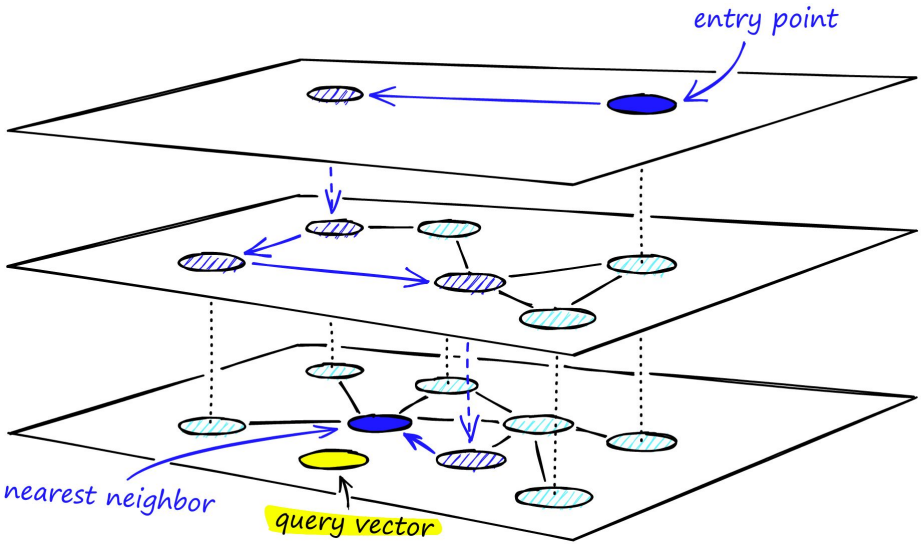
Demo time!

Overview

1. Search Quality Evaluation & Dataset
2. Vector Search Doctor: exact
3. Vector Search Doctor: approximate
4. Key Takeaways & Q&A

Exact vs. Approximate vector search





Why we use Approximate Nearest Neighbor (ANN)

Exact Vector Search

- Compute metrics between **all docs**
- Retrieves the **best possible solution**, but it's **slow**
- **Unfeasible in large dataset**

Approximate Vector Search

- Compute metrics between a **subset of the documents**
- Retrieves a **local minima** solution, but it's **faster**
- **Scalable**

Most common ANN algorithm: HNSW

HNSW (Hierarchical Small Navigable World) is a graph-based approximate nearest neighbor search technique used to search in a collection of vectors.

⚙️ **Key Parameters**

- **M** → maximum number of edges or connections that each node can have in the graph at each level of the hierarchy
- **ef_construction** → number of candidates considered during the index construction
- **ef_search** → number of neighbors evaluated during the search (query time)

ANN configuration issues: HNSW example

Parameter	⬆️Value Effect	⬇️Value Effect
`M`	+ Recall / + Memory / + Build time	– Recall / + Speed
`ef_construction`	+ Index accuracy / + Build time	– Index quality / + Build speed
`ef_search`	+ Query recall / + Latency	– Recall / + Speed

Vector Search Doctor: approximate

Goal: understand if the vector search algorithm implementation you chose (search engine) is decent in the approximation (and iterate)

How does Vector Search Doctor do that?: computes the metrics for your dataset, by retrieving the documents directly from your search engine leveraging rated-ranking-evaluator (RRE) and checking how far they are from the exact implementation

Input

Configuration file

```
query_template: "templates/only_vector.json"
search_engine_type: "solr"
collection_name: "testcore"
search_engine_url: "http://localhost:8983/solr/"
search_engine_version: "9.8.1"                          <- if not provided, set to “latest”
ratings_path: "resources/ratings.json"                  <- if not found, datastore is used
embeddings_folder: "resources/embeddings"
output_destination: "resources"
```

Template vs ANN configuration

The configuration of the search algorithm can come from both the template that you're using to query your search engine (e.g., for hybrid search) and the configuration of the ANN algorithm. The indication to test just the vector search algorithm is to use a plain template that just uses the vector to retrieve documents. The vector placeholder is set to "$vector".
E.g., for Solr:

```
{
  "q": "{!knn f=vector topK=10}$vector"
}
```

Demo time!

Overview

1. Search Quality Evaluation & Dataset
2. Vector Search Doctor: exact
3. Vector Search Doctor: approximate
4. Key Takeaways & Q&A

Key takeaways

- A working methodology for **evaluating vector search end-to-end**
- A possibility to **build your own evaluation dataset** in a few steps
- **Comparative performance data**: your setup vs. public benchmarks
- Concrete understanding of how to understand **where your search implementation may be failing**

Q&A

Sease | Information Retrieval Applied

THANK YOU

sease.io