BLAVATNIK INSTITUTE
HARVARD MEDICAL SCHOOL

TRANSLATIONAL DATA SCIENCE
CELEHS
CENTER FOR A LEARNING HEALTH SYSTEM

Building R / Python pipelines for biomedical semantic search

Thomas Charlon, MEng / PhD
Tianxi Cai, ScD

Social Linux Expo 22x – Mar 8, 2025

About me

Computer Science Engineer from Paris (2013)

Bioinformatics PhD in Geneva in pharma industry (2019)

2 years independent research on NLP (twitter withheld content)

3 years wannabe founder, web app for real-estate market estimation

(Also some automated trading and storytelling)

Harvard Medical, CELEHS

Joined Prof. Cai January 2024 in CELEHS
Research associate (senior postdoc)

Translational data science, learning health system
Analysis of electronic health records (hospitals visits)

Some of the big collaborators: Mass General Brigham, Veterans Affairs

Specific diseases studied: rheumatology, multiple sclerosis, Alzheimer

One big research area: transfer learning from large to small hospitals

My work at CELEHS

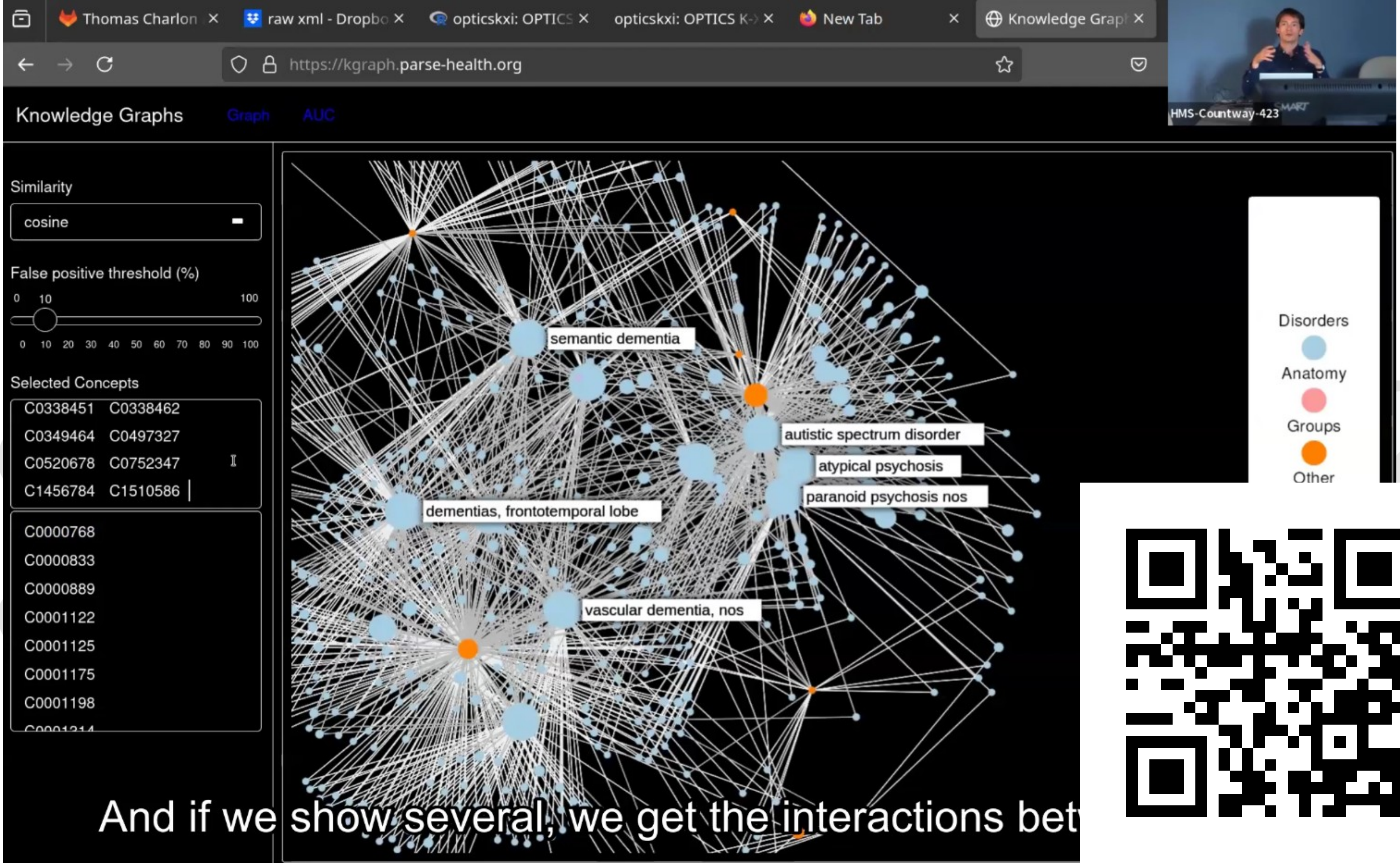
50% suicide prevention, 50% lab level, multiple projects

Suicide risk prediction, codified data + NLP on clinicians notes
csrp.mgh.harvard.edu

Dictionary codebook:
Map hospital variables to standardized classifications

Enhancing research apps by customizing RShiny with JS

Implementation quality, reproducibility (pharma quality processes)


Thomas Charlon
raw xml - Dropbo
opticskxi: OPTICS
opticskxi: OPTICS K-
New Tab
Knowledge Grapl
https://kgraph.parse-health.org
HMS-Countway-423
Knowledge Graphs
Graph
AUC
Similarity
cosine
False positive threshold (%)
0 10 100
0 10 20 30 40 50 60 70 80 90 100
Selected Concepts
C0338451 C0338462
C0349464 C0497327
C0520678 C0752347
C1456784 C1510586
C0000768
C0000833
C0000889
C0001122
C0001125
C0001175
C0001198
semantic dementia
autistic spectrum disorder
atypical psychosis
paranoid psychosis nos
dementias, frontotemporal lobe
vascular dementia, nos
Disorders
Anatomy
Groups
Other
And if we show several, we get the interactions bet

https://phewascatalog.org/phewas/#phe12

PheWAS Resources Powered by VANDERBILT

PheWAS HOME | Phecode Map X (Extended) | Phecode Map 1.2 (Combined) | PheWAS Catalog | HLA | Telo

Phecode Map 1.2 with ICD9 and ICD-10cm Codes

Download Filtered Data

Show 20 entries

Search:

ICD	Flag	ICDString	Phecode	PhecodeString	PhecodeCategory
All	All	All	All	schizo	All
295	9	Schizophrenic disorders	295	Schizophrenia and other psychotic disorders	psychological disorders
295.0	9	Simple type schizophrenia	295.1	Schizophrenia	psychological disorders
295.00	9	Simple type schizophrenia, unspecified state	295.1	Schizophrenia	psychological disorders
295.01	9	Simple type schizophrenia, subchronic state	295.1	Schizophrenia	psychological disorders
295.02	9	Simple type schizophrenia, chronic state	295.1	Schizophrenia	psychological disorders
295.03	9	Simple type schizophrenia, subchronic state with acute exacerbation	295.1	Schizophrenia	psychological disorders
295.04	9	Simple type schizophrenia, chronic state with acute exacerbation	295.1	Schizophrenia	psychological disorders
295.05	9	Simple type schizophrenia, in	295.1	Schizophrenia	psychological disorders

Building R / Python pipelines for biomedical semantic search

<u>Part 1</u>

The best of both worlds

Once upon a time

- “Oh Thomas you could do us an Elasticsearch embedding based app right ?”

- “A what ?”

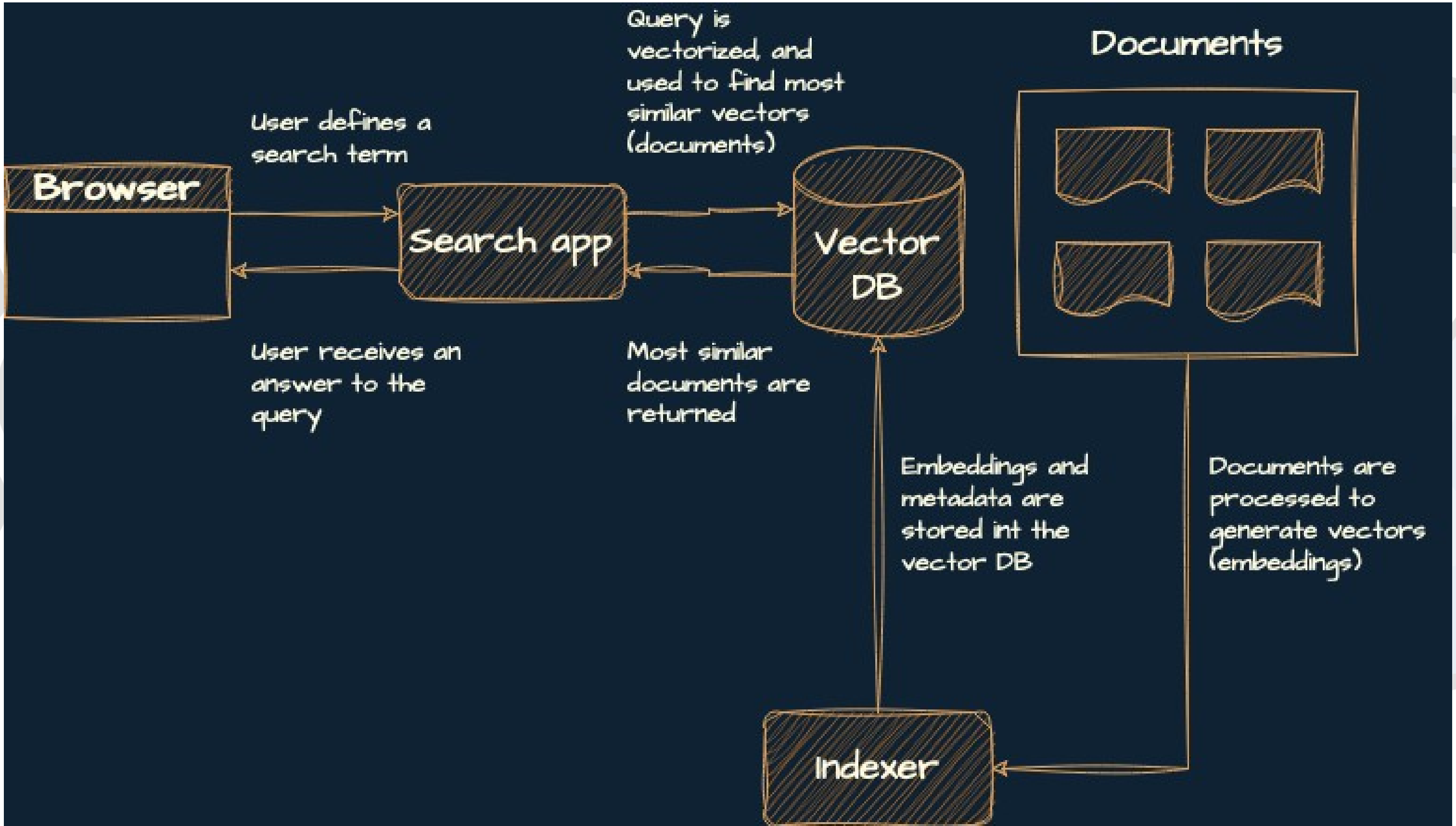
- “You previously did an app with Elasticsearch right ?”

- “Er yes… I was using a software that made use of ES…”

- “Ok great you’ll be fine”

One duckduckgo search later

https://dylancastillo.co/posts/semantic-search-elasticsearch-openai-langchain.html



“Oh I think I get it, oh wow” (aka Eureka)

Remember those people who were saying ES would be useful when doing search on elements with many numbers of characteristics / indexes ?

You know how we’re using vector embeddings to represent words…

You should be able to search for words based on the embeddings directly !

Crash course on embeddings

Word2vec and the latest natural language processing methods are not that different.

Let’s say you have a paragraph with several sentences…
Say each time two words appear in the same sentence,
you give +1 similarity to that pair of words.

You get a symmetric matrix with all your words pairs, and their similarities.

You can then use that matrix to ask:
“What are the 5 words most similar to word X ?”

Crash course on embeddings (2)

Methods like ChatGPT are slightly more sophisticated,
they train “neural networks” instead of just a similarity matrix.

But they do know that a lot of people like those similarity matrices,
so even if it’s not at the core of the method,
they still provide them because (some) people prefer to use that.

Ever heard about hallucinations ? My first scientific encounter:

Give 10 variables related to obesity (e.g. overweight, anorexia, bariatric surgery)
Ask to regroup them by class (synonyms, eating disorders, consequences)
After ~10 passes of prompt engineering, LLM includes a variable not given in list (bulimia)
Even with a RAG system prompt (“use only the list provided”)
My guess: the more prompt complexity, the higher risk of hallucinations

Hallucinating is bad

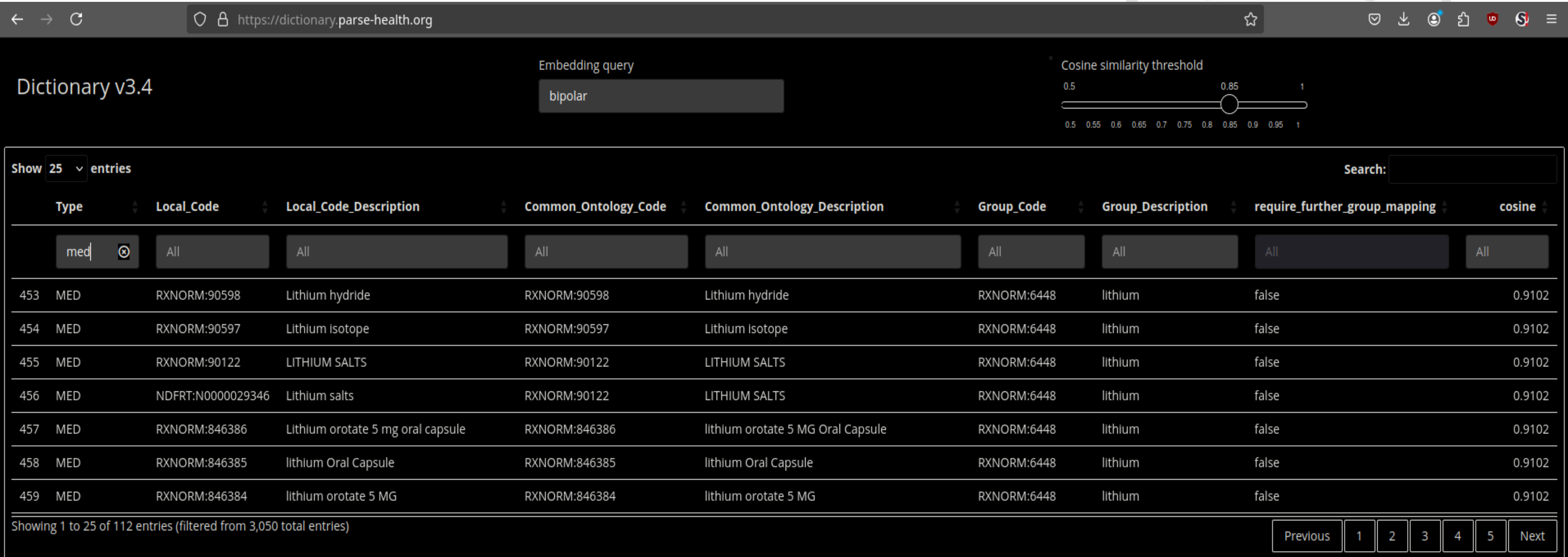
Consider you have a biomedical dataset…
You have, say, a few variables for 1000 patients over a few weeks
Something like body mass index, average heart rate etc.

You ask ChatGPT,
“I’m gonna give you a list of variables,
which ones should I study to best predict cardio-vascular diseases ?”
He will tell you maybe,
“Out of the 300 you gave me I think these 3 are the most relevant”
And then you’ll notice,
“Ah, but 2 out of 3 are not in the list I provided”

This is why we prefer to use embeddings:
less natural language understanding (e.g. negations)
but more control on output (hallucination free guarantee)

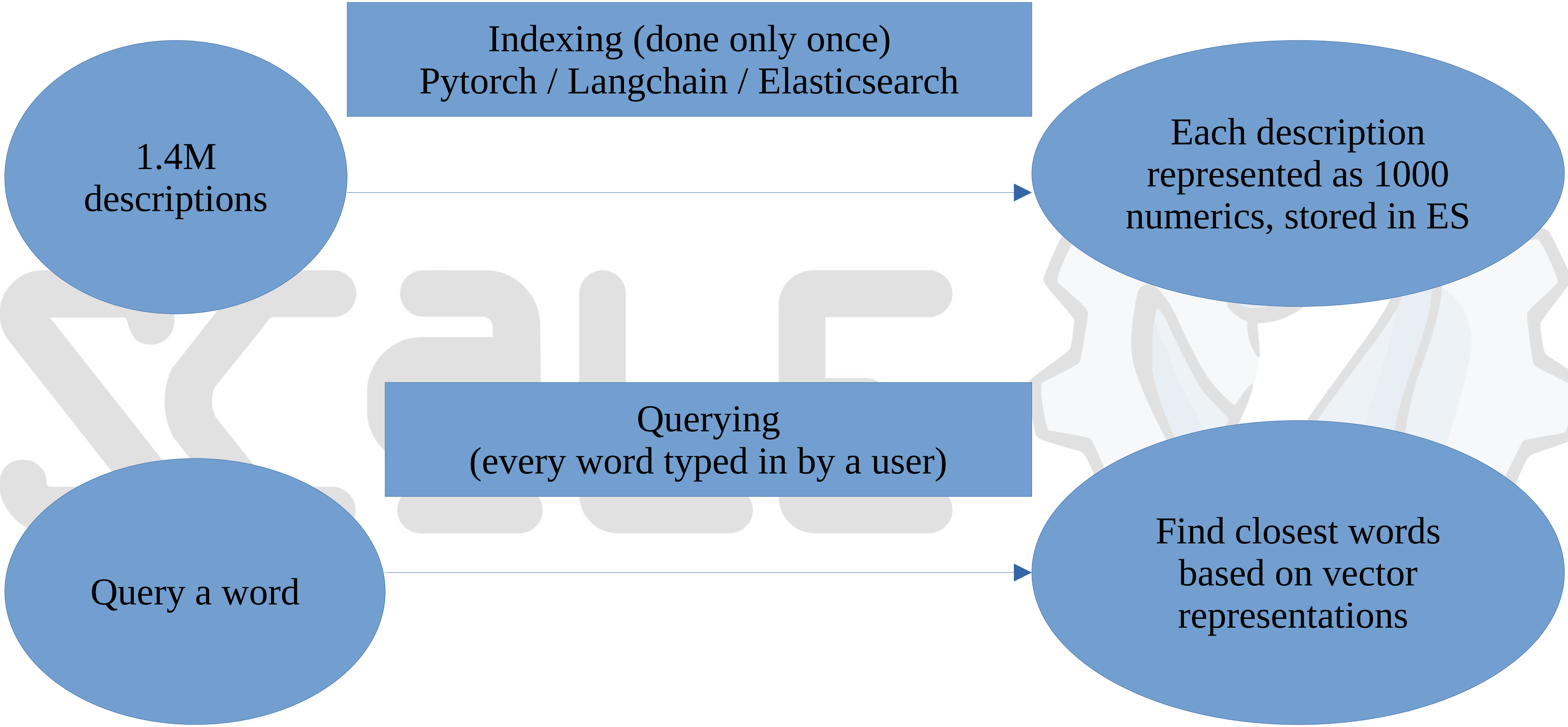
Our first embedding based app

We have 1.4 M variables with text descriptions,
we build the similarity matrix between descriptions (aka embedding matrix),
then we can query new words and get the most similar ones (BGE based)



	Type	Local_Code	Local_Code_Description	Common_Ontology_Code	Common_Ontology_Description	Group_Code	Group_Description	require_further_group_mapping	cosine
	med	All	All	All	All	All	All	All	All
453	MED	RXNORM:90598	Lithium hydride	RXNORM:90598	Lithium hydride	RXNORM:6448	lithium	false	0.9102
454	MED	RXNORM:90597	Lithium isotope	RXNORM:90597	Lithium isotope	RXNORM:6448	lithium	false	0.9102
455	MED	RXNORM:90122	LITHIUM SALTS	RXNORM:90122	LITHIUM SALTS	RXNORM:6448	lithium	false	0.9102
456	MED	NDFRT:N0000029346	Lithium salts	RXNORM:90122	LITHIUM SALTS	RXNORM:6448	lithium	false	0.9102
457	MED	RXNORM:846386	Lithium orotate 5 mg oral capsule	RXNORM:846386	lithium orotate 5 MG Oral Capsule	RXNORM:6448	lithium	false	0.9102
458	MED	RXNORM:846385	lithium Oral Capsule	RXNORM:846385	lithium Oral Capsule	RXNORM:6448	lithium	false	0.9102
459	MED	RXNORM:846384	lithium orotate 5 MG	RXNORM:846384	lithium orotate 5 MG	RXNORM:6448	lithium	false	0.9102

How it's built



How it's built (2)

<u>Pytorch enables GPU indexing, Langchain enables vectorization</u>

Computing a word's representation is fast, computing 1.4M representations can take a while.
If you do a naive for loop your GPU will be at 1% capacity
Langchain batches words together, gets your GPU at 100%.
→ 1.4M descriptions in 25min on a low grade GPU (RTX 4060, $300)
(I do recommend however getting everything right with a naive for loop for starters)

<u>Elasticsearch enables easy querying</u>

Native functions to perform cosine similarity search

How it’s built (Python)

The Python indexer

Done with Langchain

```
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import ElasticVectorSearch
from langchain_community.document_loaders import JSONLoader
from langchain_elasticsearch.vectorstores import ElasticsearchStore

embeddings = HuggingFaceEmbeddings(model_name="BAAI/bge-large-en-v1.5")

file_paths = ["app/mgb_dict_batch1.json", "app/mgb_dict_batch2.json", "app/

#file_paths = ["app/mgb_dict_batch_test1.json", "app/mgb_dict_batch_test2.j

for file_path in file_paths:

  loader = JSONLoader(file_path, jq_schema=".[].desc", text_content=False)
  docs = loader.load()

  ElasticsearchStore.from_documents(
    docs,
    embeddings,
    es_url="http://es:9200",
    index_name="elastic-index",
  )
```

How it’s built (Docker / ES)

```
services:
  es:
    image: elasticsearch:8.15.2
    environment:
      - xpack.security.enabled=false
      - discovery.type=single-node
    volumes:
      - esdata:/usr/share/elasticsearch/data
      - ./data_backup:/usr/share/elasticsearch/data_backup
      - ./heap_size.options:/usr/share/elasticsearch/config/jvm.options.d/heap_size.options
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s http://es:9200 >/dev/null || exit 1",
        ]
      interval: 20s
      timeout: 10s
      retries: 50
  index:
    depends_on:
      es:
        condition: service_healthy
    build: ./
    command: python app/indexer.py
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]
```

Use Docker Compose to connect to an independent ElasticSearch instance easily

```
from pytorch/pytorch:2.4.1-cuda11.8-cudnn9-runtime

run pip install elasticsearch fastapi sentence_transformers uvicorn langchain .

workdir /code

add ./app/install.py /install.py
run python /install.py

copy ./app /code/app
```

How it's built (RShiny JS)

A second "deployment" Docker Compose file uses the pre-indexed ES data and connects to R/Shiny

(Index locally with your GPU, then copy your ES data to your AWS Shiny server)

```
---
title: "Dictionary"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
    theme: spacelab
runtime: shiny
---

<style>

body {
  padding-top:20px !important
}

.navbar{
  visibility: hidden
}

</style>


```{r}
  library(magrittr)
  # con = elastic::connect(host = 'es')

  # test index exists
  # readLines('http://es:9200/_aliases') == "{\"elastic-index\":{\"aliases\":{}}}"
```
```

And this is where the fun begins

- “Can you use rather PubMedBert instead of BGE ?
Since it’s trained specifically on biomedical data it should be better”

- “Well I explored a few cases manually,
BGE seems better, look at these examples: (…)

[and a while later]

- “You really need to try SAPBert, literature says it’s the best”

[“is a reprex provided ?”]

Automated evaluation !

<u>We often use clinician-curated known pairs to measure our prediction models' performances</u>

E.g. in our known pairs we have “Schizophrenia is related to Psychosis”
We query “schizophrenia”, if the top matches include “Psychosis”, +1 (or AUC)
This also helps us to automatically find good similarity thresholds,
we can use thresholds that correspond to 5 or 10% false positives, instead of raw similarity values
Out of the 1.4M descriptions, we have 20k pairs between 5k concepts

<u>I would really like to avoid the previous two-step indexing + deployment,</u>
(you know, for reproducibility)
But, the thing is, I kinda hate mandatory indentation
Plus I already did an R package to do these models' performances (check out kgraph)
I mean, Python is great and all, but I just want to do as much as possible in R

Automated evaluation (??)

But…

I already have a Docker Compose, called by a Makefile…

Should I rather do a Docker Compose calling a Makefile…

Or should I install R in the Pytorch Docker image…
(Last time it failed after 2 hours of installation)

Or should I install Pytorch in a R Docker image…
(It’s gonna take 6 hours and the GPU will probably not work)

Automated evaluation, yes indeed

"Ah yes, I'll just do a Makefile,
calling a Docker Compose,
calling a Makefile (or two)"

1. Build your evaluation dataset in R:
Subset your 1.4 M descriptions to the 20k
included in your known pairs

2. Move the dataset to your Python folder

3. Index with Python,
store in ES with a Docker volume

4. From R, connect to ES,
call your evaluation scripts

```
| 0 | 0 | ~/gits/llmeval
| $ tree
.
├── docker-compose.yml
├── eslang
│   ├── app
│   │   ├── eval_db.json
│   │   ├── indexer.py
│   │   ├── indexer_sapbert.py
│   │   ├── install_p2.py
│   │   └── install.py
│   ├── Dockerfile
│   ├── heap_size.options
│   └── Makefile
├── Makefile
├── README.md
└── relastic
    ├── auc_bgebase.txt
    ├── auc_bgem3.txt
    ├── auc_bge.txt
    ├── auc_pubmedbert.txt
    ├── auc_sapbert.txt
    ├── auc_sentencebert.txt
    ├── dictionary_mapping_v3.4.tsv.gz
    ├── Dockerfile
    ├── eval_llms.R
    ├── Makefile
    └── pairs_arranged.Rdata
```

Automated evaluation, yes indeed

```
services:
  write_eval_db:
    build: ./relastic/
    command: make write_eval_db
    working_dir: /relastic
    volumes:
      - ./relastic:/relastic
  es:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.15.2
    environment:
      - xpack.security.enabled=false
      - discovery.type=single-node
    volumes:
      - esdata:/usr/share/elasticsearch/data
      - ./eslang/heap_size.options:/usr/share/elasticsearch/config/jvm
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "curl -s http://es:9200 >/dev/null || exit 1",
        ]
      interval: 20s
      timeout: 10s
      retries: 50
```

```
      timeout: 10s
      retries: 50
  index:
    build: ./eslang/
    depends_on:
      es:
        condition: service_healthy
    command: make
    working_dir: /eslang
    volumes:
      - ./eslang:/eslang
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: 1
              capabilities: [gpu]
  write_embeds_auc:
    build: ./relastic/
    depends_on:
      - es
    command: make write_embeds_auc
    working_dir: /relastic
    volumes:
      - ./relastic:/relastic

volumes:
  esdata:
```

`make && cat relastic/*.txt`

And a few minutes later

```
| 130 | 0 | ~/gits/llmeval
| $ cat relastic/*.txt
Model: bgebase, All pairs: 0.8157,      No Dx-Dx: 0.6151,       Only Dx-Dx: 0.8492
Model: bgem3,   All pairs: 0.7324,      No Dx-Dx: 0.4712,       Only Dx-Dx: 0.7715
Model: bge,     All pairs: 0.8156,      No Dx-Dx: 0.5859,       Only Dx-Dx: 0.8597
Model: pubmedbert,      All pairs: 0.7802,      No Dx-Dx: 0.5568,       Only Dx-Dx: 0.836
Model: sapbert, All pairs: 0.758,       No Dx-Dx: 0.5325,       Only Dx-Dx: 0.8312
Model: sentencebert,    All pairs: 0.6223,      No Dx-Dx: 0.3912,       Only Dx-Dx: 0.7088
```

“- Mhhh, no. BGE is still better.
The fine-tuned model we developed with the intern however…”

Fine-tuned BGE, in a nutshell

I observed that in the top matches for “schizophrenia”,
Most are good but, we also get “bacteria”, “leukemia”, “pneumonia”

The culprit is called tokenization.
The similarity is based on the suffixes,
It happens especially for words the original model encountered rarely in the training set.

The thing is, when you fine-tune a model,
it’s quite hard to get with a few GPUs something better
than what the original team has trained with thousands of GPUs.
Even if you input billions of biomedical pairs.

However, if you focus your fine-tuning on words with identical suffixes…
And have a devoted intern that knows Python…

Clinical Trials and PubMed Article Search

A comprehensive platform to search for clinical trials and retrieve related PubMed articles.

Clinical Trials Search

Select search method:

Search by Condition

obesity

Search Trials

#	NCT ID	Title	Condition(s)	Link
4	NCT01692327	Study About High Fat Meal and Postprandial Lipemia	Obesity	View Trial
5	NCT01119976	Association Between the Menstrual Cycle and Weight Loss	Overweight, Obesity	View Trial

Brief Summary:
This is a research study to look at the association between weight loss and the menstrual cycle in healthy, overweight, prem
women. Participants will be asked to follow a reduced-calorie diet and exercise plan for 3 months.

Interventions:
BEHAVIORAL: Reduced calorie diet and exercise plan; BEHAVIORAL: Different reduced calorie diet and exercise plan

PubMed Article Retrieval

Enter NTC ID

NCT05013879

Search Articles

Retrieval Mode

- Fast (Threshold-based)
- Precise (Top 10 Reranked)
- Provided by Authors

False Positive Rate (%)

0 1.2 10

Cosine Similarity Threshold: 0.7258

Showing 200 of 200 results with similarity ≥ 0.7258

Filter by title:

Enter text to filter titles...

#	Article ID	Title	Similarity	URL
1	32459670	Effects of combining manual lymphatic drainage and Kinesiotaping on pain, edema, and range of motion in patients with total knee replacement: a randomized clinical trial.	0.8776	Link
2	24819349	The effectiveness of Kinesio Taping® after total knee replacement in early postoperative rehabilitation period. A randomized controlled trial.	0.8749	Link
3	23841976	Effects of kinesio tape to reduce hand edema in acute stroke.	0.8527	Link

BLAVATNIK INSTITUTE
HARVARD MEDICAL SCHOOL

TRANSLATIONAL DATA SCIENCE
CELEHS
CENTER FOR A LEARNING HEALTH SYSTEM

Building R / Python pipelines for biomedical semantic search

Part 2

The hidden agenda

Using Llama for UMLS → Phecode

Unified Medical Language System (UMLS):

→ Dictionary of ~5M codes / descriptions (CUIs) of diagnoses, medications etc.
→ We use it for NLP of clinicians notes, contains synonyms, plurals, acronyms…

→ Codes have relationships:
synonyms, parent-child, relatedness (e.g. medication "may treat" disease)

→ We want to perform roll-up
(regroup rare codes in larger statistically useful groups)

Llama: Facebook's "open-source" version of ChatGPT

(icymi)

Using Llama for UMLS → Phecode

We want to use the parent-child relationships
to replace rare children concepts by their parents

→ e.g. “suicide by hanging” child of “suicide” child of “self-harm”

But how do we know which level to roll-up to ?
Can we avoid using the frequencies we observe in a specific study ?
Can we have a study-independent dictionary, useful for several projects ?
(rheumatoid arthritis, suicide prevention, multiple sclerosis)

First experiments with CUI rollup:
→ Use graph properties (subcomponents, degrees)
→ Decent but many very small groups

Using Llama for UMLS → Phecode

The revelation:

→ Use the codified dictionary to guide the CUI roll-up
→ Start by finding exact string matches, map to corresponding group code (Phecode, ingredient)

Then use synonyms and parent-child relationships to map those with no string matches
→ 10% mapped globally, 1200 of the 1875 Phecodes, promising but not yet sufficient

One of the limitations:

"Suicide <u>or</u> self-inflicted…" (Phecode), "Suicide <u>and</u> self-inflicted…" (CUI)

We need to be careful with partial string matching
e.g. Type 1 Diabetes and Type 2 Diabetes

Using Llama for UMLS → Phecode

One of the core problematic:

Phecode:297 Suicide Ideation or Suicide Attempt
Phecode:297.1 Suicide Ideation Phecode:297.2 Suicide Attempt

→ We want the CUI mapping to follow this hierarchy,
we don’t want to map the same CUI to two different Phecodes

Using graph properties, good individual mappings,
but could not follow structure, same CUI for 297 and 297.2

I finally gave up and started taking the Llama road
→ Despite the risk of hallucination, but that’s actually not the worst part

Using Llama for UMLS → Phecode

Trying it out on Llama and ChatGPT:

“Here is a list of 30 variables,
[perform first CUI filter with BGE embeddings]
Which one is best match for ‘Suicidal Ideation or Suicide Attempt’ ?”

Llama: Suicidal behavior
ChatGPT: Suicidal ideation

“Oh wow it works ! And Llama (3.1 7B) is better than ChatGPT (4o) ! FOSS !”
But…

Using Llama for UMLS → Phecode

But…

Once integrated in Docker Compose with API call:

“Oh no so sorry you are having those thoughts.
Here is the hotline: 1-800-TalkToMe”

(or “I cannot help you hurt yourself”, or “Suicidal ideation”, or…)

Even with a temperature parameter of 0 ??
(should be “more reproducible”)

Using Llama for UMLS → Phecode

Annoying, but still the best we got
→ One big step was to include the variables we didn't want in the prompt,
to take care of the cases where it was answering 'suicidal ideation'
"Here is a list … You cannot use these variables: [297.1], [297.2]…"

After that, the safety answers were managed with a few loops in a while(TRUE)
→ Only one other case gave such "safety errors": gynecology related

The decimal level Phecodes (297.1, 297.2) are mapped by string distance,
Then the integer level Phecodes with BGE + Llama

Out of ~300 integer Phecodes, using ~10 loops, only one stuck on a hallucination
→ The variable makes sense, but it's not in the list I provided, and I'm unable to map it
→ But, it's not one of my disease of interest, so…

Using Llama for UMLS → Phecode

So…
Works for me !

The word of the end:

“UMLS → Llama → Phecode,
it’s actually more about the safety than the hallucinations.”

Summary

Semantic search = Embeddings based = Vector search
→ First part of a RAG, hallucination free guarantee

To use LLMs in science, need to constrain and control
→ I did as much as possible with classic methods, before going to Llama
→ Even then, another mapping takes place after, filters out hallucinations

Known pairs and benchmarks are highly valuable
→ Takes domain experts a lot of time
→ Relevance to real-world needs, quality, difficulty

Summary

LLM parameters are like climate science and computer systems,
There’s so much components, not one expert knows them from top to bottom

→ Deep learning experts play on learning rate, LLM experts keep it fixed

→ I think in Llama, ~5 parameters that influence reproducibility,
still a long way to go before submission to FDA

Need to identify which of your application can make use of LLMs,
and which questions are useless since most likely will hallucinate

→ Asking questions about books is really bad in my experience
→ ChatGPT including references in outputs is really cool

Summary

Since we need to constrain and control, not every application will be useful

Young devs, please don’t think it is smarter than you
You are the one who is making use of it

It’s kinda like Wikipedia,
You’re not supposed to cite it if you haven’t gone back to the reference

Like great storytellers say,
You have to know the rules to know when you can break the rules.

BLAVATNIK INSTITUTE
HARVARD MEDICAL SCHOOL

TRANSLATIONAL DATA SCIENCE
CELEHS
CENTER FOR A LEARNING HEALTH SYSTEM

Building R / Python pipelines for biomedical semantic search