**Intro to Information Retrieval**

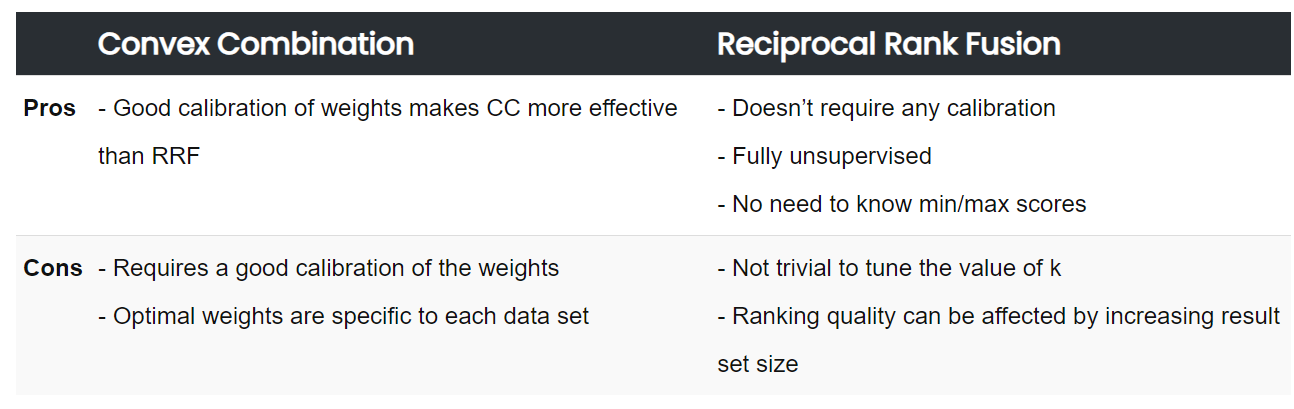
1. An Introduction to Information Retrieval <https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>
2. Search Engines Information Retrieval in Practice <https://ciir.cs.umass.edu/downloads/SEIRiP.pdf>
3. An Introduction to Neural Information Retrieval <https://www.microsoft.com/en-us/research/uploads/prod/2017/06/fntir2018-neuralir-mitra.pdf>
4. Pretrained Transformers for Text Ranking: BERT and Beyond <https://arxiv.org/pdf/2010.06467.pdf>

**Idea**

1. Hybrid Search Model

* Encoding (<https://www.pinecone.io/learn/series/nlp/dense-vector-embeddings-nlp/#dense-vs-sparse-vectors>) :
  + Sparse (TF\_IDF, **BM25 (first option)**, SPLADE (better, but second option)) <https://www.pinecone.io/learn/semantic-search/>
  + Dense (2vec – word2vec, sentence2vec, doc2vec, **SBERT,** Dense Passage Retriever)
* There are multiple ways to implement a hybrid search, including linear combination, giving a weight to each score and reciprocal rank fusion (RRF), where specifying a weight is not necessary (<https://www.elastic.co/search-labs/blog/articles/lexical-and-semantic-search-with-elasticsearch>). Weighting would be achievable using linear weighting scheme using a convex combination (<https://docs.pinecone.io/docs/weighting-sparse-and-dense-vectors>)
* RRF: <https://learn.microsoft.com/en-us/azure/search/hybrid-search-ranking> The score is calculated as 1/(rank + k), where rank is the position of the document in the list, and k is a constant, which was experimentally observed to perform best if it's set to a small value like 60
* Reciprocal Rank Fusion is sensitive to its parameter, instead Convex Combination is generally agnostic to the choice of score normalization. Convex Combination is sample efficient, requiring only a small set of training examples to tune its only parameter to a target domain (<https://arxiv.org/pdf/2210.11934.pdf> )
* Convex Combination:



* RRF:
* 

1. Semantic Re-ranking ( second target)
2. Query expansion (third target)

* History and technique: <https://arxiv.org/pdf/1708.00247.pdf>
* PRF : Cluster dari punya egen – konsep FAISS, get top k documents berdasarkan centroid sama vector querynya, keyword extraction seperti YAKE punya bu mar -> borda ranking buat nentuin kata” yang top -> query expansion dari kata” hasil borda ranking supaya dapet sinonim”nya kek yang semula batuk jadi batuk berdahak, batuk kering, dkk -> search- disini baru coba implement yang hybrid + semantic ranking)
* Query based:
  + **Query Expansion by Prompting Large Language Models** ( by Google): <https://arxiv.org/pdf/2305.03653.pdf>
  + **Query2doc: Query Expansion with Large Language Models** ( by Microsoft) <https://arxiv.org/pdf/2303.07678.pdf>
  + **Corpus-Steered Query Expansion with Large Language Models:** <https://openreview.net/pdf?id=n6V9An2Cse>. This journal compare BM25 + CSQE with other llm powered query expansion such as:
    1. **Unsupervised Dense Information Retrieval with Contrastive Learning** or also stated as Contriever+HyDE in the journal ( by Meta), a KEQE method that employs hypothetical documents generated by LLMs to enhance unsupervised Contriever <https://arxiv.org/pdf/2112.09118.pdf> & <https://github.com/facebookresearch/contriever>
    2. **Generative Relevance Feedback with Large Language Models** . BM25+GPR, a query expansion method that applies PRF upon LLM-knowledge empowered hypothetical texts. GPR is a strong baseline that out performs multiple SOTA PRF methods <https://arxiv.org/pdf/2304.13157.pdf>
    3. BM25+KEQE
  + Query transformation that also can be implemented using Langchain: <https://blog.langchain.dev/query-transformations/>
    1. **Query Rewriting for Retrieval-Augmented Large Language Models** (Rewrite-Retrieve-Read): <https://arxiv.org/pdf/2305.14283.pdf> & <https://github.com/langchain-ai/langchain/blob/master/cookbook/rewrite.ipynb?ref=blog.langchain.dev>
    2. **TAKE A STEP BACK: EVOKING REASONING VIA ABSTRACTION IN LARGE LANGUAGE MODELS** (Step back prompting): <https://arxiv.org/pdf/2310.06117.pdf> & <https://github.com/langchain-ai/langchain/blob/master/cookbook/stepback-qa.ipynb?ref=blog.langchain.dev>
    3. **Multi Query Retrieval:** <https://python.langchain.com/docs/modules/data_connection/retrievers/MultiQueryRetriever?ref=blog.langchain.dev>
    4. **RAG-Fusion**: <https://towardsdatascience.com/forget-rag-the-future-is-rag-fusion-1147298d8ad1> & <https://github.com/langchain-ai/langchain/blob/master/cookbook/rag_fusion.ipynb?ref=blog.langchain.dev>



* SPLADE:
  + Overview: <https://medium.com/@sowmiyajaganathan/hybrid-search-splade-sparse-encoder-neural-retrieval-models-d092e5f46913>
  + v1: <https://arxiv.org/pdf/2107.05720.pdf>
  + v2: <https://arxiv.org/pdf/2109.10086.pdf>

**Process**

1. Implement semantic and lexical search using BM25, Rank Reciprocal Fusion (8/12/2023)
   1. Dense using sbert is not able to solve the problem since our data using Bahasa. So change to indobert. I should create the encoder from scratch using the idea from <https://www.youtube.com/watch?v=jVPd7lEvjtg&t=40s>
   2. Problem: it will crash due to run out of memory and ram if we processed the whole data to the encoder
2. Prepro data & Implement batch concept for semantic search (9/12/2023)
   1. Just realized that in the description most likely have sentence that is not important such as ‘petunjuk dokter’, ‘biaya’. Cut that out so now I have more compact sentences. Which this process will reduce memory usage
   2. I’ve learnt about the max length and padding function, then I’ve just realized that in the previous code I hard code all of the max length to 128. In fact after checking the word length, the average is only around 30, the median is 40 and the mode is 60. Then, I thought how if the max length is dynamic not always 128 since 128 will take lots of ram. That’s why I sort the df based on the word length, and then classify to 25, 50, 75, 100, and 128. With that classifier I determined the max\_length. I also implement batch concept. If the corpora is more than one than do batching, and if in the corpus is more than one then split it again into smaller batch. In each process I implement time.sleep so it will give a chance for the ram to go back normal before continue the process. For 25 until 75 we can use batch limit 100, for 100 and 128, better to use 50 to tackle error. The concept turns out similar with uniform batching in the colab, but I decide to write my own since in the colab is little bit hard to understand. Batch concept and references:
      1. <https://colab.research.google.com/drive/1Er23iD96x_SzmRG8md1kVggbmz0su_Q5#scrollTo=MMVDieLwz-f5>
      2. <https://www.youtube.com/watch?v=ynOZUNnbEWU&lc=Ugyg8zDqCO7RkkrgTXJ4AaABAg>
      3. <https://stackoverflow.com/questions/68337487/what-is-the-correct-way-of-encoding-a-large-batch-of-documents-with-sentence-tra>
   3. Problem: the process of searching takes a bit long time around 1 – 1.5 minute, where I think it’s not acceptable if being implemented in real case like google. Even second is still too slow, and now it’s a minute
3. Learn Approximate Nearest Neighbor before semantic search, move the computation to GPU, and modify some code to process query input, renaming class name to more appropriate and correct naming convention which are DenseSearch to SemanticSearch and SparseSearch to LexicalSearch (10/12/2023)
   1. At first I thought this because I safe the index in an numpy file, then I though vector databases could be the solution. Then, I start learn about pinecone, chromadb, etc. But at some point I didn’t get why we should use vector database and how to choose it, until I watch this video: Building a Powerful Vector Search Engine with Pinecone, ChromaDB, and Faiss Using Langchain: <https://www.youtube.com/watch?v=GTvN-F1ExVA&t=43s> . Next, I learn how vector database work: <https://www.pinecone.io/learn/vector-database/> and <https://www.elastic.co/what-is/vector-database> . End up with this article <https://labelbox.com/blog/how-vector-similarity-search-works/> , I’ve just realized where clustering taking place in similarity search. So, the general idea behind ANN algorithms is to preprocess the dataset to create an index or data structure that allows for efficient querying. When a query point is provided, the algorithm uses the index to quickly identify a set of candidate points that are likely to be close to the query point. This way, when querying the vector database to find the nearest neighbors of a query point, instead of computing distances between the query point and all vectors in the database, we only compute distances between the query point and the small number of candidate points around it. The next question is how to implement this?
   2. Learn FAISS. The quickest answer for now is FAISS. Learn it from this youtube playlist (<https://youtube.com/playlist?list=PLIUOU7oqGTLhlWpTz4NnuT3FekouIVlqc&si=vqn83aOThTCHJ8FB> ) or from this article series <https://www.pinecone.io/learn/series/faiss/>. Here is why I decide that FAISS is the solution:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| no | query | standard | IndexFlatL2 | IVF HNSW |
| 1 | obat untuk meredakan sakit kepala | 2.43 | 2.02 | 1.64 |
| 2 | vitamin untuk pertumbuhan bayi dan anak | 2.49 | 1.55 | 1.71 |
| 3 | serum pencerah kulit | 2.45 | 1.73 | 1.6 |
| 4 | susu untuk ibu hamil | 2.35 | 2.29 | 1.53 |
| 5 | penurun tensi, darah tinggi, dan jantung | 2.38 | 1.68 | 1.76 |

standard: no index from FAISS, just using base code

IndexFlatL2: standard index from FAISS

IVF HNSW: composite index using FAISS ( IVF4096\_HNSW32,Flat)

I decide to use IVF HNSW to faster the process.

* 1. Problem ( solved in the same day):
     1. after this whole process, turns out the problem is mainly located in the query embedding process. The search is only take millisecond, even using FAISS it could be microsecond. But the embedding process it self took around 1 minute
     2. FAISS is using l2 ( Euclidean distance), I would say cosine is better to capture semantic, so I still search for other option maybe like FAISS but using cosine or anything else (<https://github.com/facebookresearch/faiss/wiki/MetricType-and-distances> )
  2. Add max\_length = len(corpus[0].split(" ")) to encode query, not 25. From 2 second to 1 second, so this method is works
  3. Just realized that 1 minute additional time is because i add time.sleep(60). So replace it above, so if the input is only one corpus like query no need to add time.sleep(60)
  4. Moving the process to GPU also reduce the time needed to encode roughly from 2 second to 1 second
  5. Rename DenseSearch to SemanticSearch, SparseSearch to LexicalSearch
  6. Problem: still not sure using FAISS or not, if time is the key then use FAISS, however if accuracy in getting semantic meaning is the key I think using own code is still better. Need further test

1. Add borda ranking as other options (14/12/2023)
   1. Problem: Has just found out that is a bug in max\_length in query process, the vector result is different ( need further checking)
2. Add LLM query expansion using langchain (17/12/2023)
   1. I tried to find LLM model that suits Bahasa Indonesia. Then found: Gemini by Google. Example of query expansion using Gemini <https://makersuite.google.com/app/prompts/new_freeform> :

Query: lakukan query expansion dan transformation dengan domain di bidang medis, farmasi dan kesehatan. Buat lima query dengan makna yang sama berdasarkan sinonim dan hiponim dari tiap kata yang ada pada query original ini.

Temperature: 0.9

* + 1. obat untuk maag dan asam lambung dengan kandungan cimetidine

1. Obat untuk maag dan refluks asam lambung yang mengandung zat aktif cimetidine.
2. Obat yang meredakan tukak lambung dan asam lambung yang mengandung cimetidine.
3. Obat untuk mengobati gastritis dan asam lambung yang mengandung cimetidine.
4. Obat yang digunakan untuk mengobati dispepsia dan asam lambung dengan kandungan cimetidine.
5. Obat untuk mengurangi gejala sakit maag dan refluks gastroesofageal yang mengandung cimetidine.
   * 1. obat cacing yang mengandung pirantel pamoat
6. Obat antihelmintik dengan komposisi pirantel pamoat
7. Obat pembasmi cacing mengandung pirantel pamoat
8. Obat cacing yang memiliki kandungan pirantel pamoat
9. Obat vermifuge yang mengandung pirantel pamoat
10. Antiparasitik dengan kandungan pirantel pamoat\_
    * 1. obat migrain dengan komposisi paracetamol
11. Obat nyeri kepala dengan kandungan asetaminofen
12. Pereda sakit kepala yang mengandung parasetamol
13. Obat untuk meredakan sakit kepala yang mengandung paracetamol
14. Pengobatan migrain dengan bahan aktif asetaminofen
15. Obat sakit kepala yang mengandung parasetamol untuk meredakan migrain
    * 1. obat pereda mual setelah kemoterapi dengan komposisi ondansetron
16. Obat antiemetik berbasis Ondansetron untuk pasien pasca-kemoterapi
17. Obat penangkal mual dengan kandungan Ondansetron untuk pasien yang menjalani kemoterapi
18. Ondansetron guna meredakan rasa mual setelah menjalani kemoterapi
19. Terapi Ondansetron sebagai pencegah mual pasca-kemoterapi
20. Obat mual jenis Ondansetron untuk pasien kemoterapi
    * 1. obat pereda gejala arthritis dengan komposisi meloxicam
21. ﻿ ﻿NSAID untuk mengurangi keparahan osteoarthritis dan artritis reumatoid yang mengandung meloxicam
22. Anti-inflamasi non-steroid yang meredakan nyeri dan kekakuan karena radang sendi yang mengandung meloxicam
23. Obat antirematik yang meredakan peradangan, nyeri dan kekakuan yang disebabkan oleh osteoarthritis dan artritis reumatoid yang mengandung meloxicam
24. Pereda nyeri, obat komposisi meloxicam, untuk mengatasi nyeri dan kekakuan akibat osteoarthritis dan artritis reumatoid
25. Pengobatan untuk radang sendi yang mengandung meloxicam
    * 1. obat sirup batuk tidak berdahak
    1. ﻿ Obat sirup untuk batuk non-produktif
    2. Obat sirup penekan batuk kering
    3. Sirup penurun batuk tanpa dahak
    4. Ekspektoran untuk batuk tidak berdahak
    5. Obat-obatan untuk pengobatan batuk kering
    6. Implement it using Langchain and LCEL:
       1. LCEL: <https://github.com/pinecone-io/examples/blob/master/learn/generation/langchain/handbook/11-langchain-expression-language.ipynb> & <https://www.youtube.com/watch?v=O0dUOtOIrfs&t=178s>
       2. How to use Gemini in Langchain: <https://colab.research.google.com/drive/1sVWzo_7iKAyXl1lLTXYE18bIWOkAcJiC?usp=sharing#scrollTo=DOLgH7rkcKVF> & <https://www.youtube.com/watch?v=G3-YOEVg-xc>
    7. Modify the process code, if len corpus is bigger than 100, then separate into smaller chunk, if not just run it so no need time.sleep. This problem raise because embedding process of the query expansion
    8. Thought about keyword extraction. Have tested and the result using extracted keyword has higher similarity score compare to directly compare with the original long sentences.
    9. Problem: since query alternatives are five, so I need to update RRF and borda ranking code to accept more than two list or rank, and fix redundancy code. Also thought to change the name to Ranking not Hybrid Search. So create Hybrid Search separately. Ranking section is focus on the ranking algoritm only like RRF and borda
26. 18/12/2023
    1. Update ranking algorithm
    2. Decide to remove dynamic max\_length based on len of query since the embedding result is different. Better to use the first idea which is 25. Time consumed is also not different that much
    3. Try to implement threading using threading package for ranking, but it’s slower than without threading. Threading is better for function without lots of computation (known as I/O bound task like downloading task), if we are using CPU ( ex: calculating, image enhancing) use **multiprocessing** (<https://www.youtube.com/watch?v=fKl2JW_qrso>)
       1. (Threading) It will return based on the fastest running code

with concurrent.futures.ThreadPoolExecutor() as executor:

query\_alternatives = [query1, query2, query3]

lexical\_future = [executor.submit(lexical\_rank\_task, corpus, query) for query in query\_alternatives]

for f in concurrent.futures.as\_completed(lexical\_future):

print(list(f.result().keys()))

* + 1. (Threading) It will return based on the order from the original list without time difference with the first method. Prefer to use this instead of the first one

with concurrent.futures.ThreadPoolExecutor() as executor:

query\_alternatives = [query1, query2, query3] results = executor.map(lexical\_rank\_task, corpus, query\_alternatives) for i in results:

print(list(i.keys()))

* + 1. Multiprocessing

with concurrent.futures.ProcessPoolExecutor() as executor:

query\_alternatives = [query1, query2, query3]

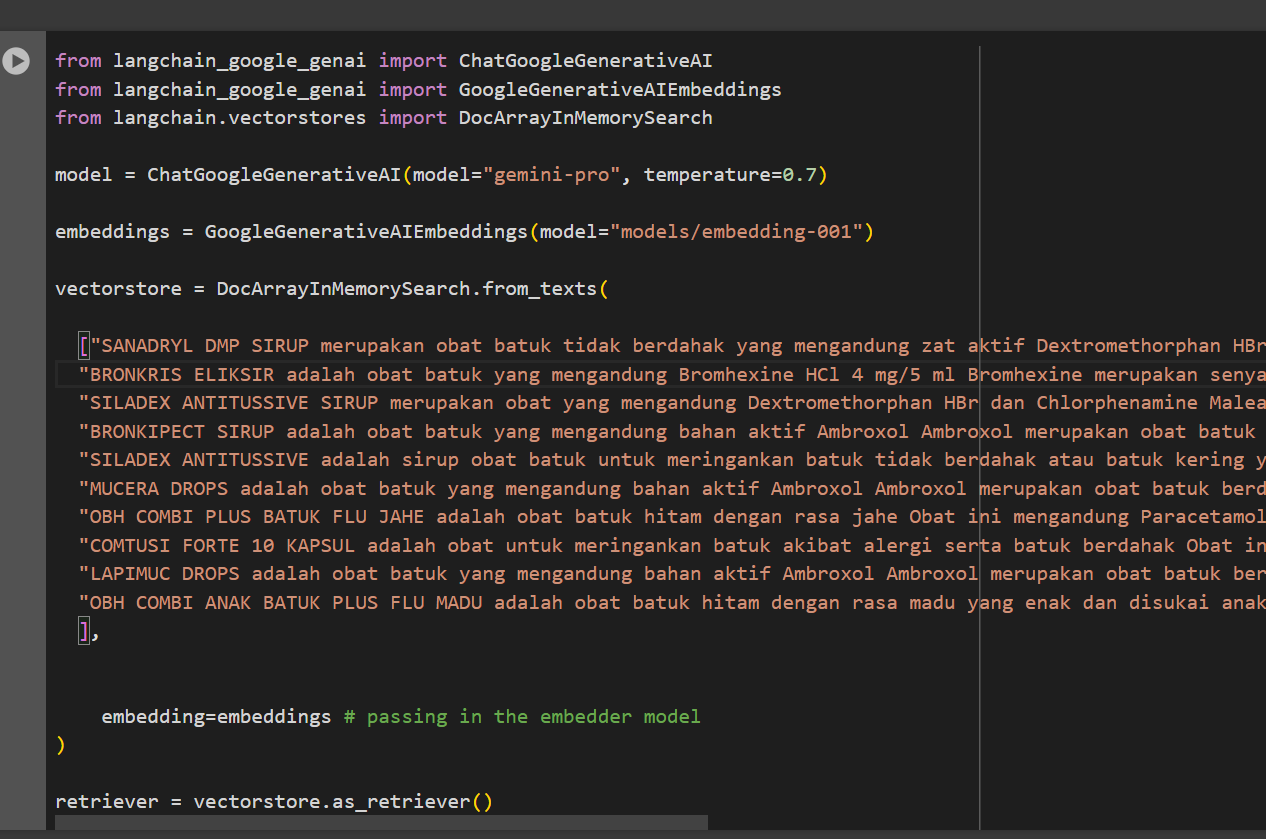
results = executor.map(lexical\_rank\_task, corpus, query\_alternatives)

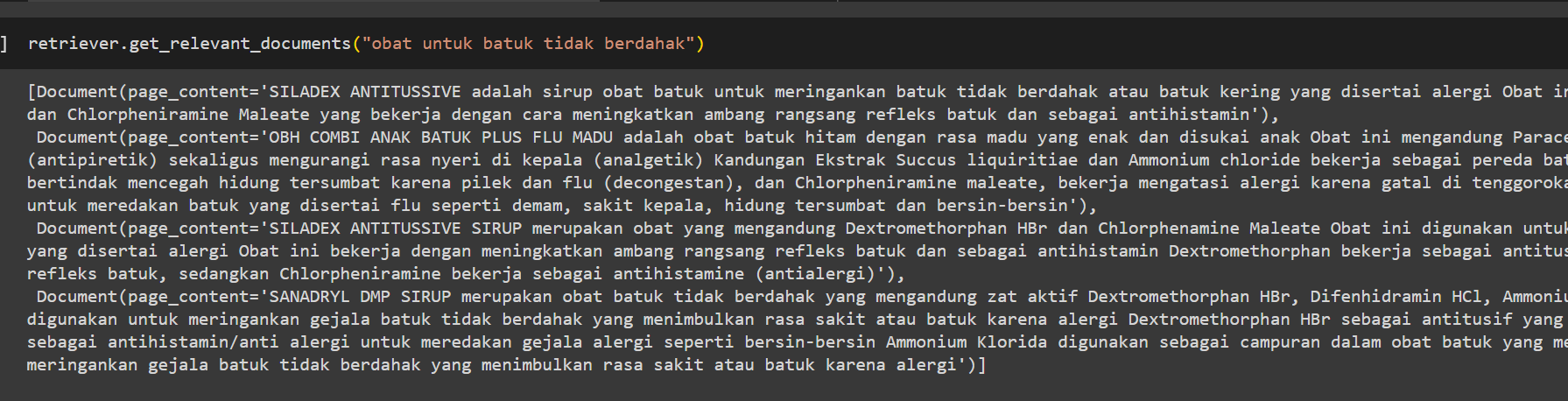
for i in results:

print(list(i.keys()))

* 1. Problem: Turns out after testing with multiprocessing just for LexicalSearch it’s still slower if compared to code without Threading or Multiprocessing. I think it would be useful for the SemanticSearch. However, since I’m using CUDA or GPU I got this error: RuntimeError: Cannot re-initialize CUDA in forked subprocess. To use CUDA with multiprocessing, you must use the 'spawn' start method. It has also discusses in this github: <https://github.com/pytorch/pytorch/issues/40403>

1. 19/12/2023
   1. Add - {'tidak'} in stopword for lexical search. We just found out that “obat batuk tidak berdahak” will become “obat batuk berdahak’ because of the stopword “tidak”
   2. I have tested using Langchain that embedding-001 by Google is better to solve “obat batuk tidak berdahak” query





Then, I think how if I implement this manually following the google’s original tutorial ini here: <https://ai.google.dev/examples/doc_search_emb>. Turns out the result is bad. Both using langchain or not, if the document is larger the result is becoming bad. So I have idea how if the top ranked document is re rank again using this model, because it proofs that “batuk tidak berdahak” become the top using it. It also only return the related things which is good. So the idea is for Semantic Re- Ranker task ( originally target 2)

* 1. Learn about MMR <https://medium.com/tech-that-works/maximal-marginal-relevance-to-rerank-results-in-unsupervised-keyphrase-extraction-22d95015c7c5>. I think this would be great for query expansion ( sometimes LLM produce several sentence in similar structure and word, using MMR they able to rank but also make sure that duplication is minimize).

1. 20/12/2023

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| query | lexical | L + QE | Semantic | S+QE | Hybrid | H + QE |
| Obat batuk tidak berdahak | 2 | 3 | 1 | 0 | 1 | 3 |
| Obat untuk mengobati memar | 0 | 1 | 1 | 1 | 1 | 2 |
| Obat migrain dengan komposisi paracetamol | 7, 1 migrain bkn paracetamol | 9, 1 migrain bkn paracetamol | 9, 1 migrain bkn paracetamol | 9, 1 migrain bkn paracetamol | 10 | 10 |
| Obat Pereda gejala arthritis dengan komposisi meloxicam | 6, 1 arthritis bkn meloxicam | 6, 4 arthritis bkn meloxicam | 5, 4 arthritis bkn meloxicam | 4, 4 arthritis bkn meloxicam | 6, 4 arthritis bkn meloxicam | 5, 5 arthritis bkn meloxicam |
| obat untuk maag dan asam lambung dengan kandungan cimetidine | 10 obat maag tapi bkn cimetidine | 10 obat maag tapi bkn cimetidine | 10 obat maag tapi bkn cimetidine | 10 obat maag tapi bkn cimetidine | 1 cimetidine, 9 obat maag tapi bkn cimetidine | 1 cimetidine, 9 obat maag tapi bkn cimetidine |

* 1. I want to implement semantic re- ranking. Do a fast check to decide which model is better. Seems like Lexical + QE and Hybrid + QE are the best choice. Need further evaluation using appropriate metrics like precision and recall. For the fast check result see below
  2. Problem: From this fast check, I just found out new problem that sometimes if we query specific composition, it won’t return as what we want although the function is similar, such as “obat untuk maag dan asam lambung dengan kandungan cimetidine”. Most of the IR return good result in terms of the drugs functionality which is maag dan asam lambung, but most of them return a drug with different composition like simethicone, ranitidine, and pantoprazole. I think it can be solved by this: <https://python.langchain.com/docs/modules/data_connection/retrievers/self_query> . Simply filter the metadata to, in this case, I can fill the metadata with the composition of the drug, drug’s name.
  3. Berhubungan sama problem diatas punya ide:
     1. pas query masuk pakek LLM “ekstrak komposisi obat dari pernyataaan ini: obat migrain yang mengandung paracetamol”.
     2. Kalo dia mengeluarkan hasil seperti paracetamol, maka lanjut ke chain berikutnya yaitu mencari nama lain dari paracetamol. “apa nama lain dari komposisi obat berikut dalam dunia farmasi jangan berikan nama obat dagang. Jika tidak ada maka return None: paracetamol”.
     3. Selain ekstrak komposisi obat ekstrak juga apa yang mau disembuhkan dari query tersebut “penyakit apa yang hendak disembuhkan oleh user dari pernyataan ini, jika tidak ada return None: obat untuk menyembuhkan migrain dengan kandungan paracetamol﻿”
     4. Cari nama lain dari penyakit tersebut
     5. Generate query yang lengkap contoh:
        1. obat maag -> obat maag, tukak lambung, gastritis, dyspepsia
        2. Obat maag yang mengandung A -> obat maag, tukak lambung, gastritis, dyspepsia yang mengandung A,B,C
        3. Obat dengan komposisi paracetamol -> obat dengan komposisi paracetamol asetaminofen

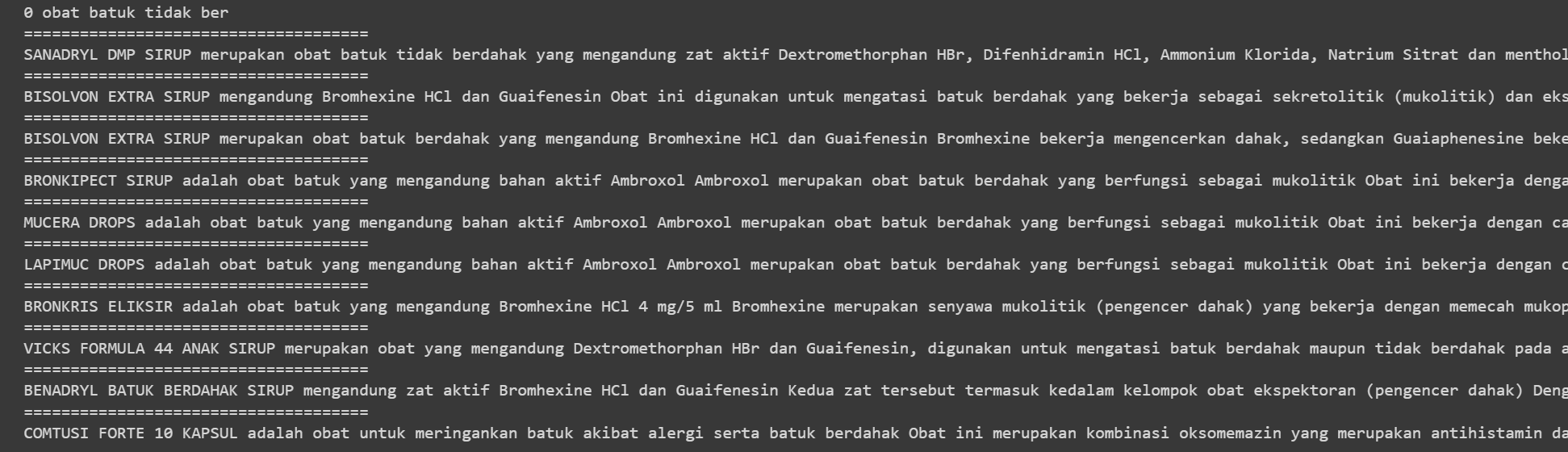
Target:

1. Semantic ranking pakek langchain yang DocArrayInMemorySearch pakek embedding gugel
2. Coba multiquery datanya masukin chroma ikutin tutorial, liat apaka lebi baik daripada yang udah dibuat manual ( kalo mau coba aja dulu pakek yang DocArrayInMemorySearch)
3. Implement MMR
4. Implement self querying
5. Klasifikasi query, jadi diekstrak pakek llm kalo itu tentang komposisi maka pakek lexical, kalo fungsional maka pakek semantic, kalo both pakek hybrid. Atau buat system Dimana bobot untuk lexical itu lebih besar daripada semantic mirip CC.

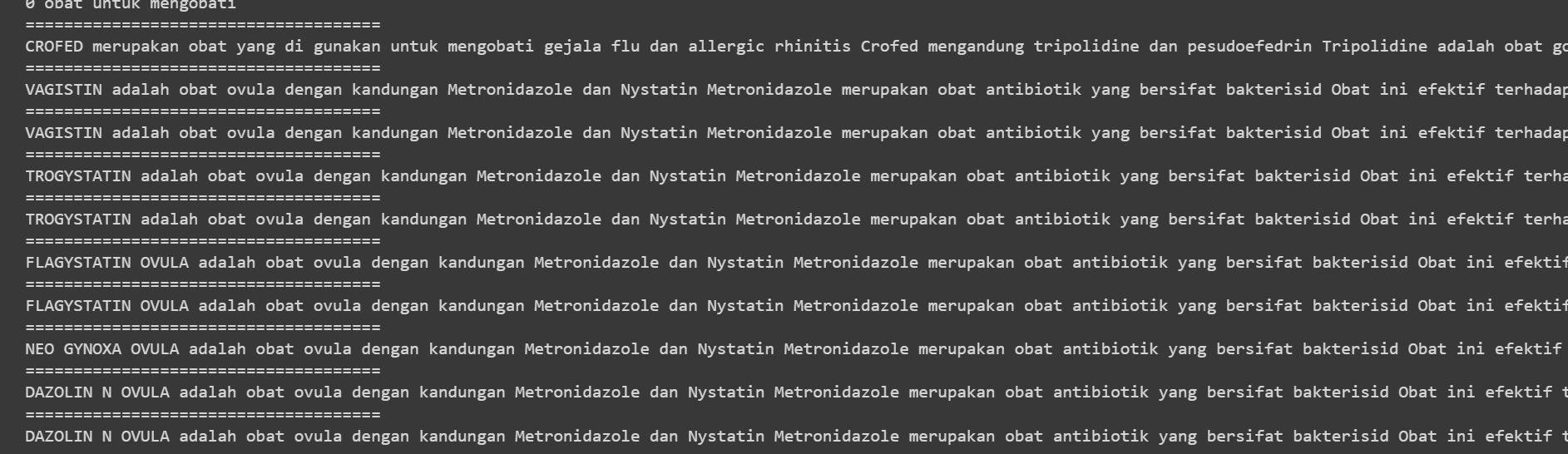
Fast Check Result:

Lexical:

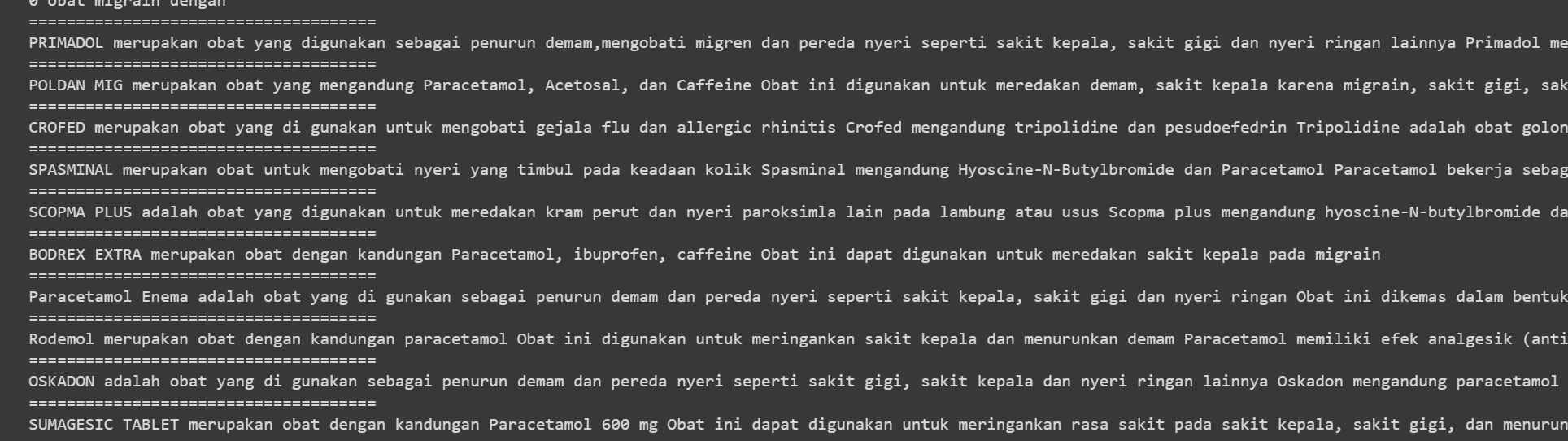
* Obat batuk tidak berdahak



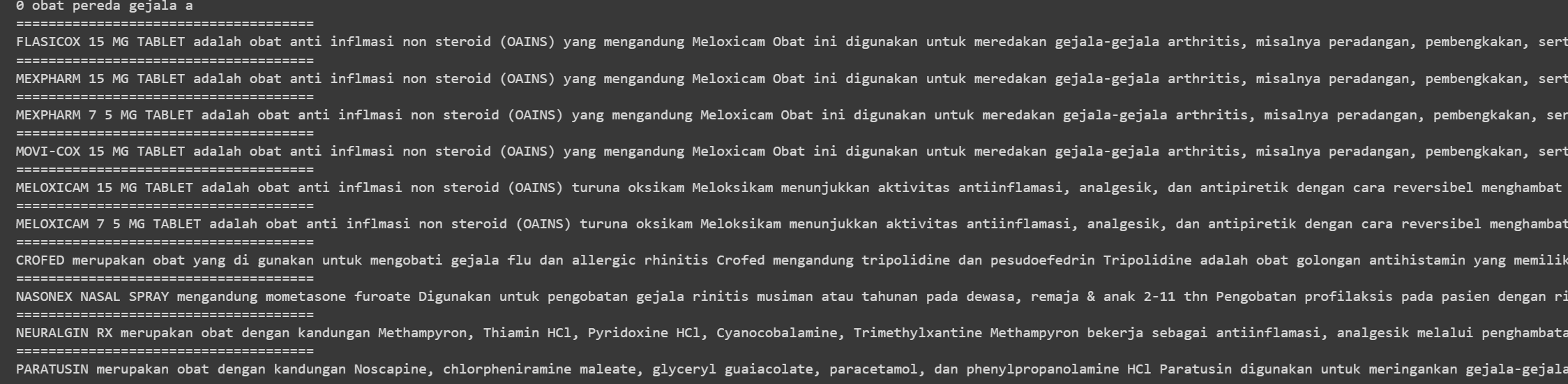
* Obat untuk mengobati memar



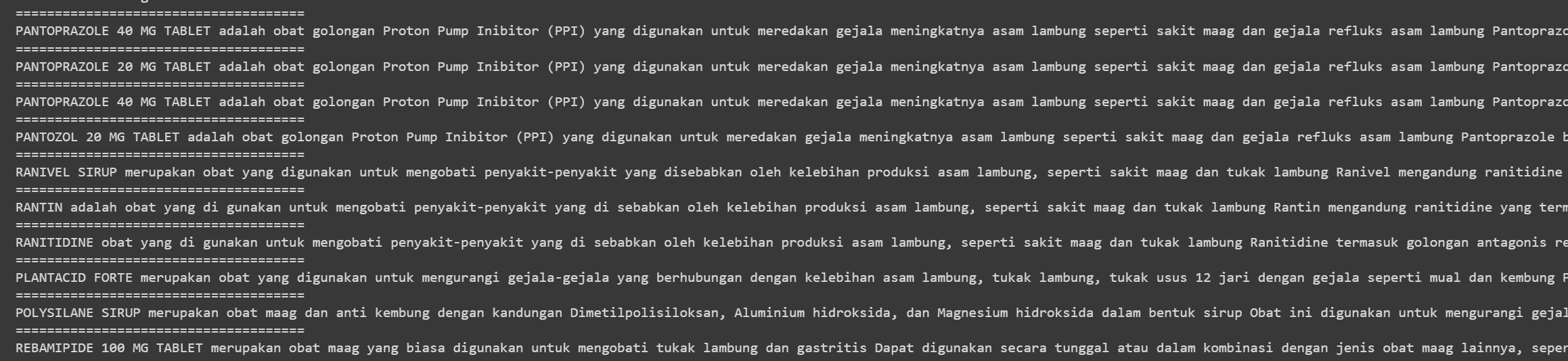
* Obat migrain dengan komposisi paracetamol



* obat pereda gejala arthritis dengan komposisi meloxicam



* obat untuk maag dan asam lambung dengan kandungan cimetidine

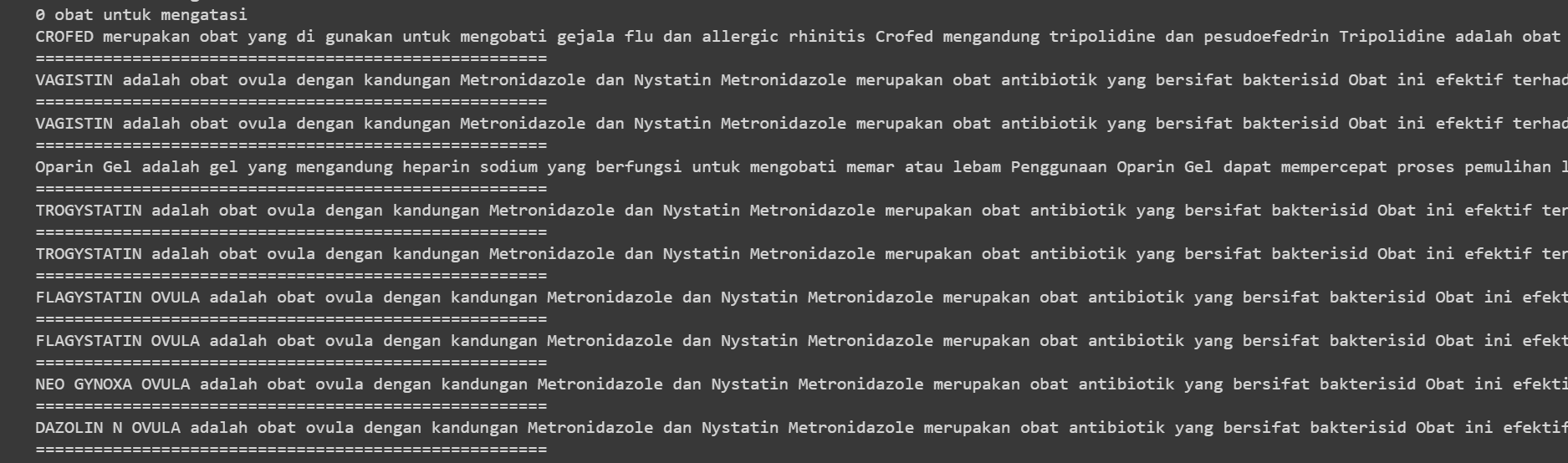


Lexical + QE:

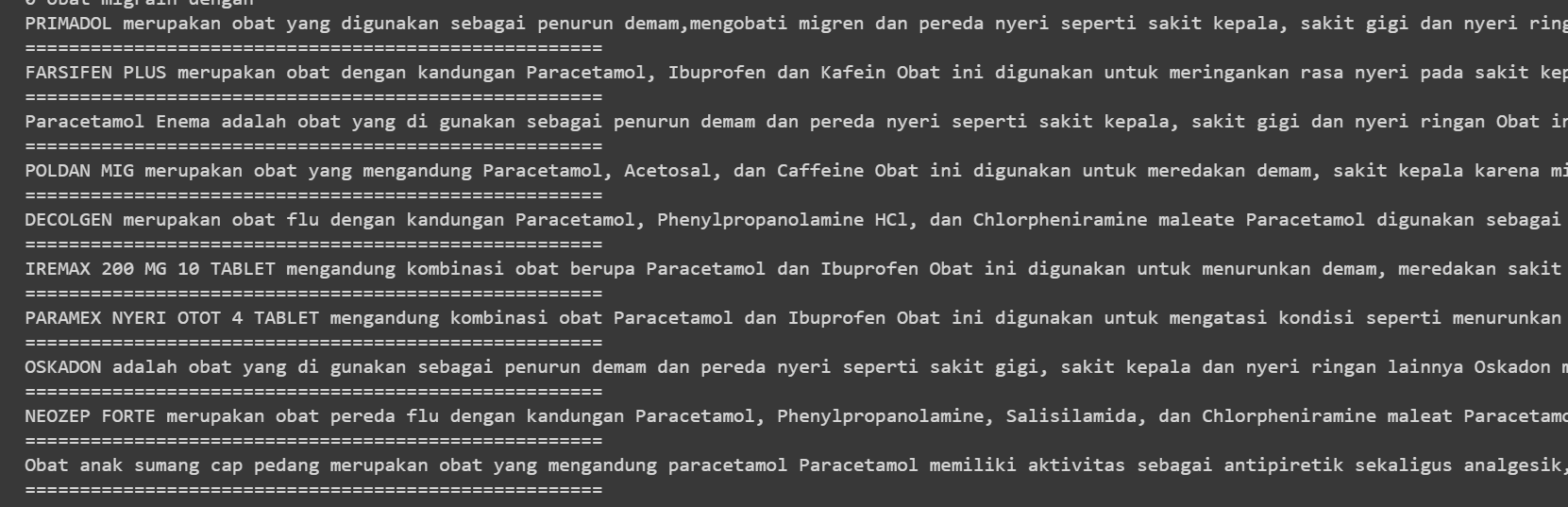
* Obat batuk tidak berdahak



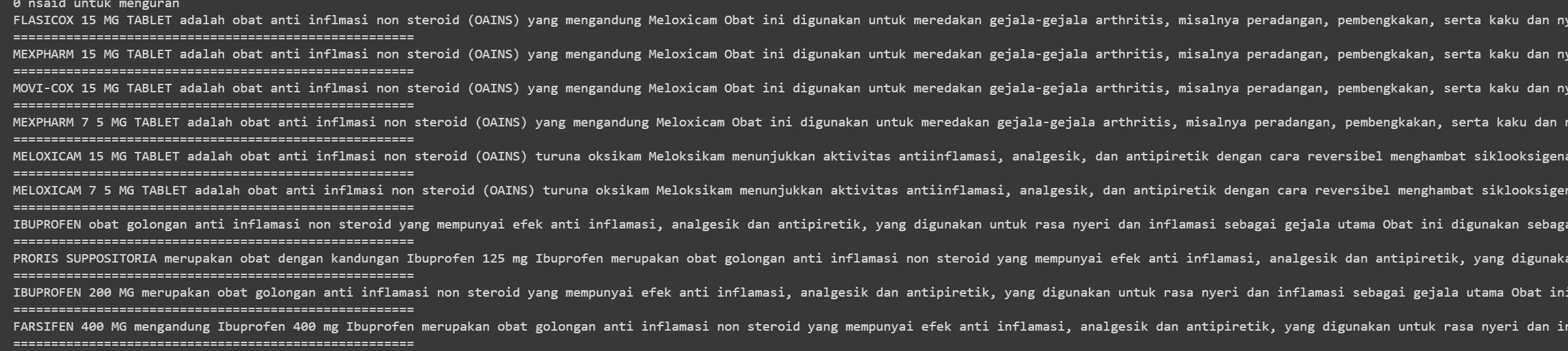
* Obat memar



* Obat migrain dengan komposisi paracetamol



* obat pereda gejala arthritis dengan komposisi meloxicam

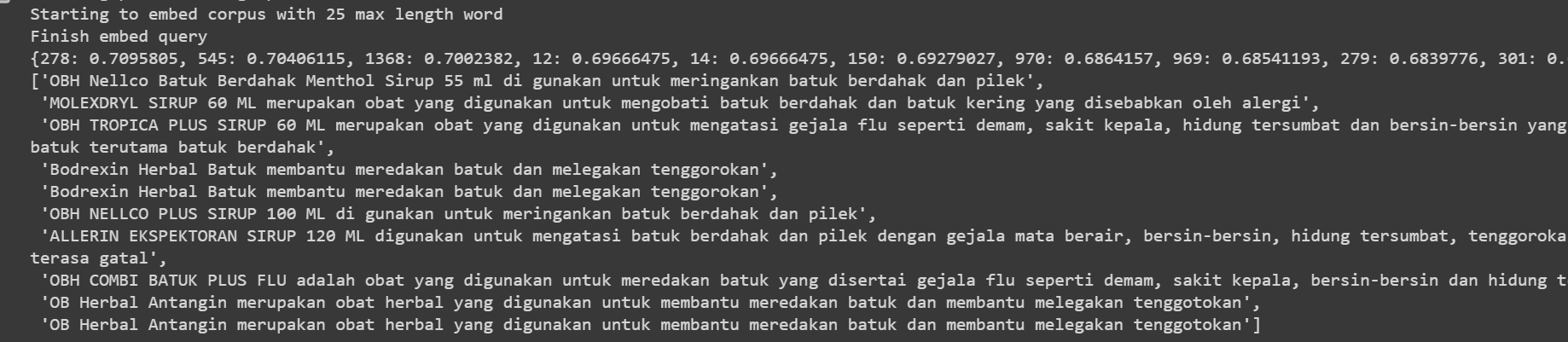


* obat untuk maag dan asam lambung dengan kandungan cimetidine

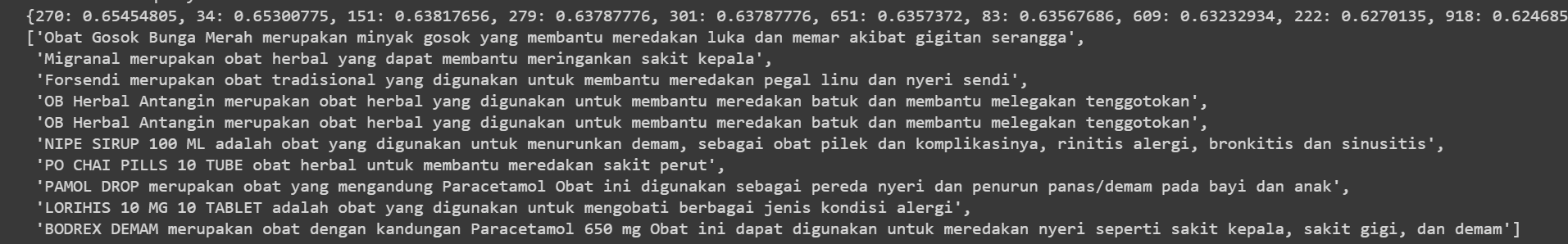


Semantic:

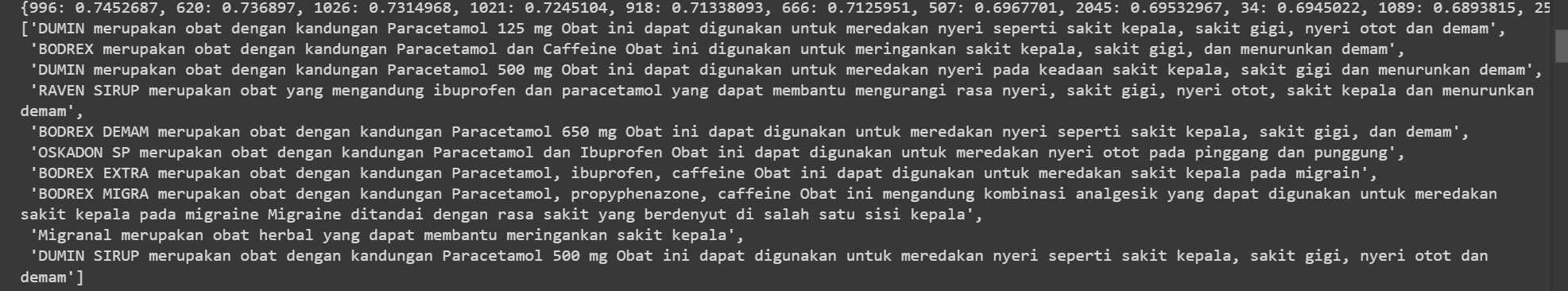
* Obat batuk tidak berdahak



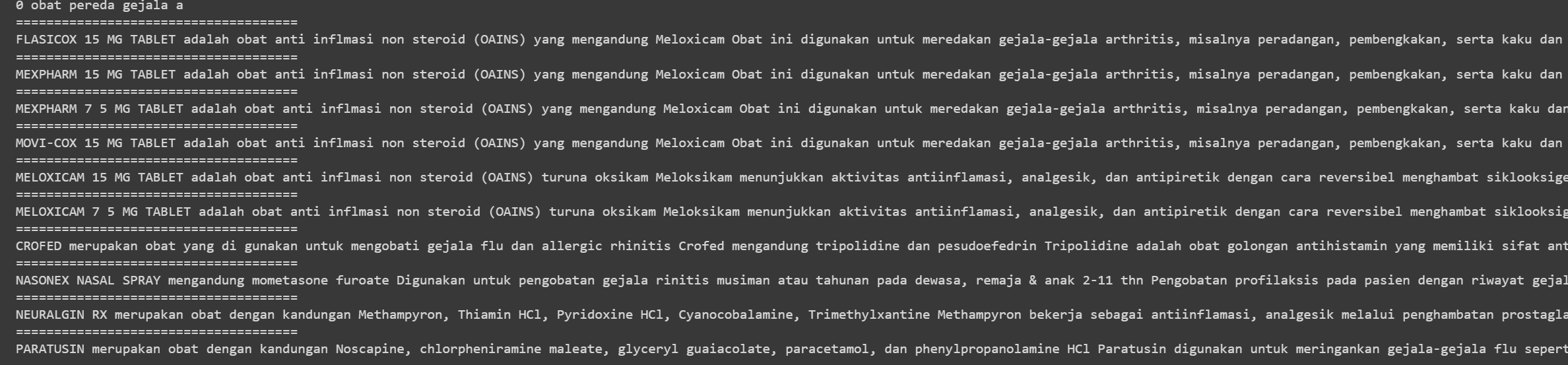
* Obat memar

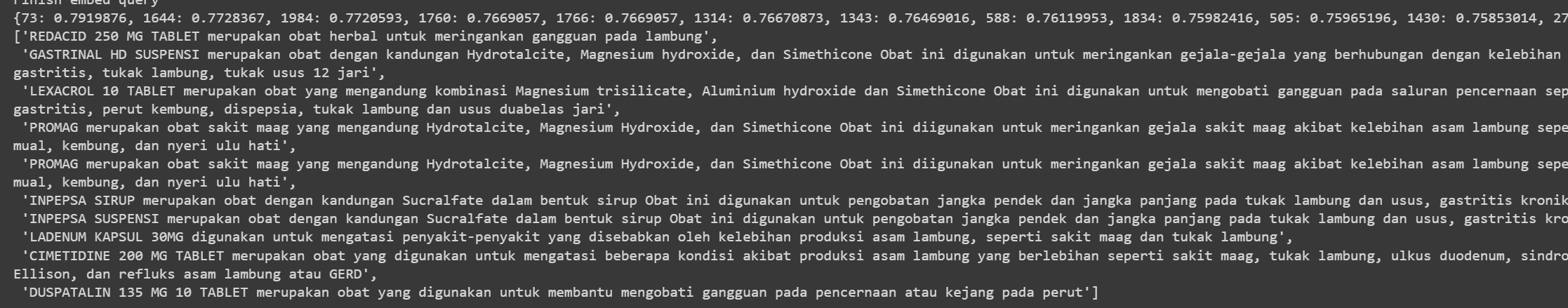


* Obat migrain dengan komposisi paracetamol



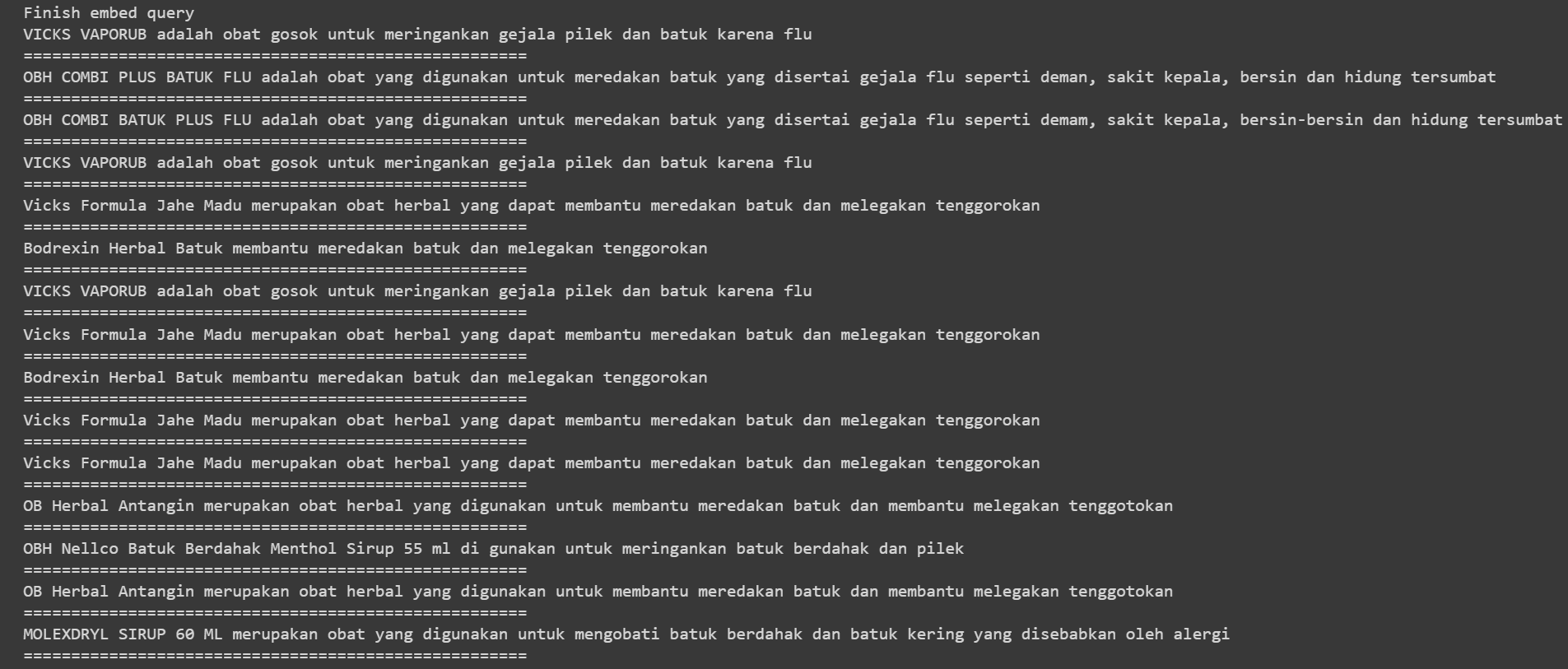
* obat pereda gejala arthritis dengan komposisi meloxicam



* obat untuk maag dan asam lambung dengan kandungan cimetidine
* 

Semantic + QE

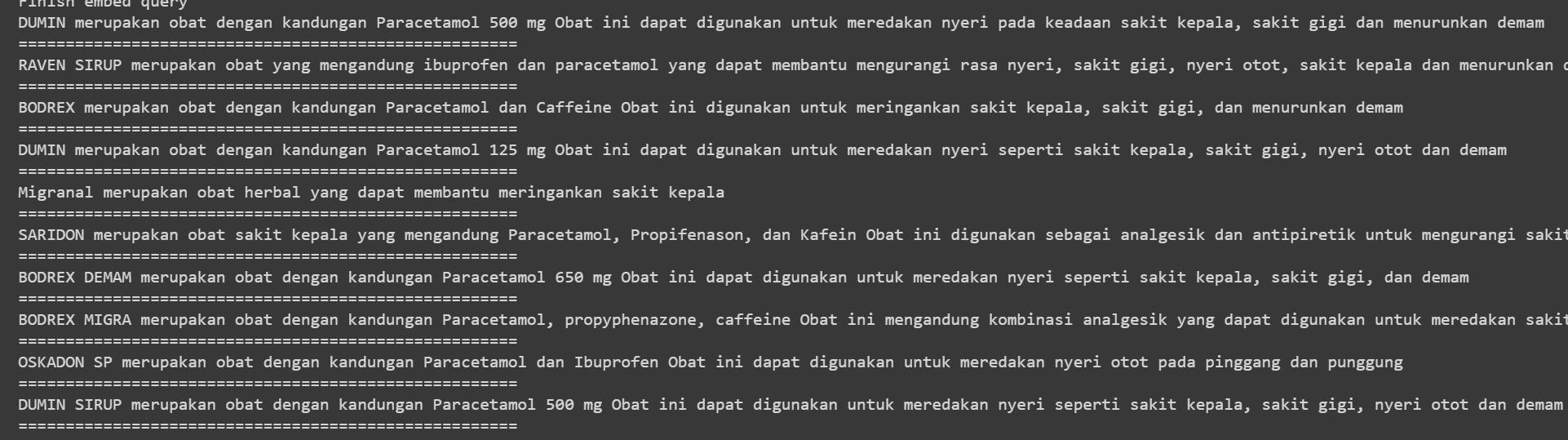
* Obat batuk tidak berdahak



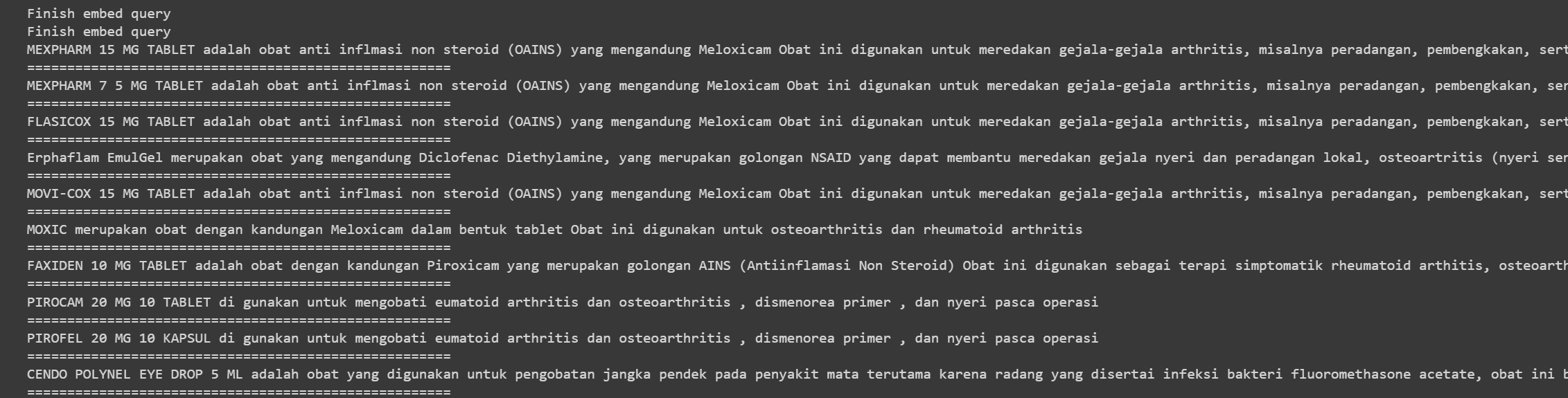
* Obat memar



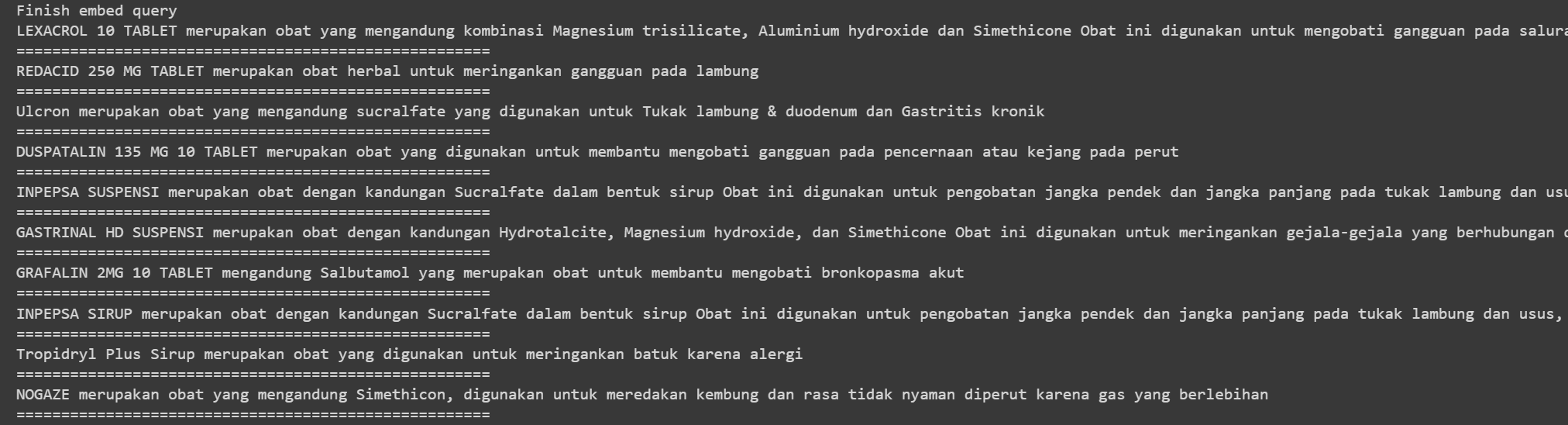
* Obat migrain dengan komposisi paracetamol



* obat pereda gejala arthritis dengan komposisi meloxicam

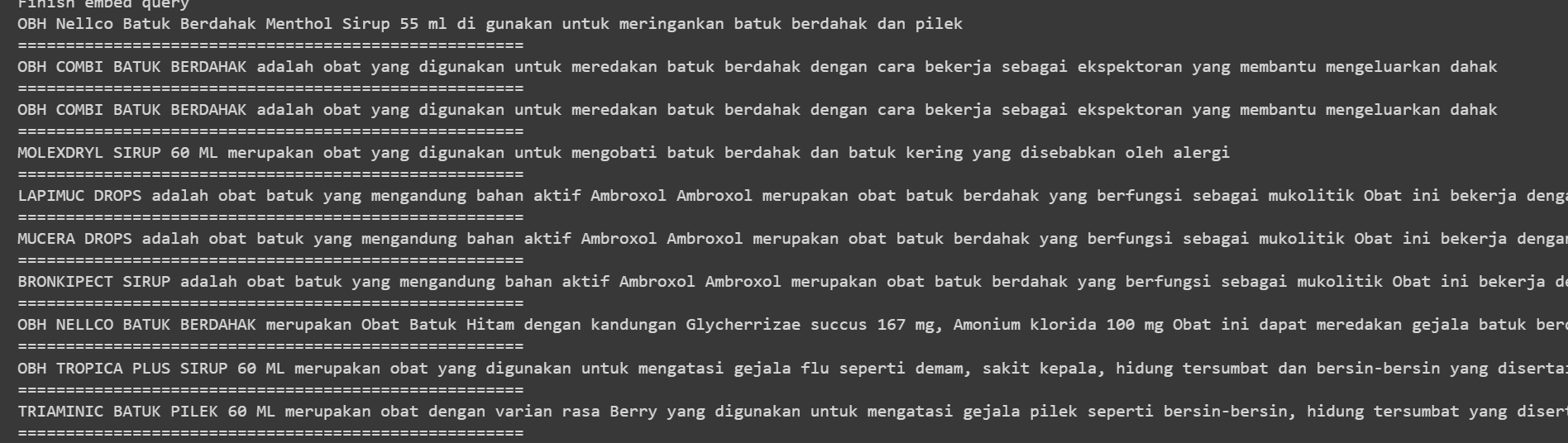


* obat untuk maag dan asam lambung dengan kandungan cimetidine

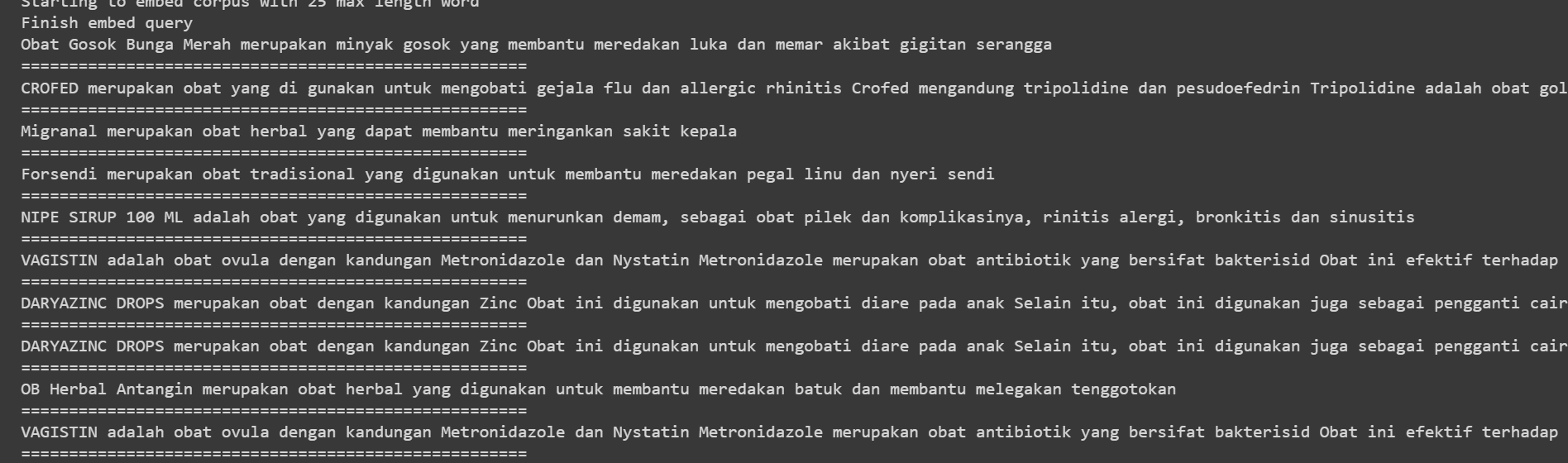


Hybrid:

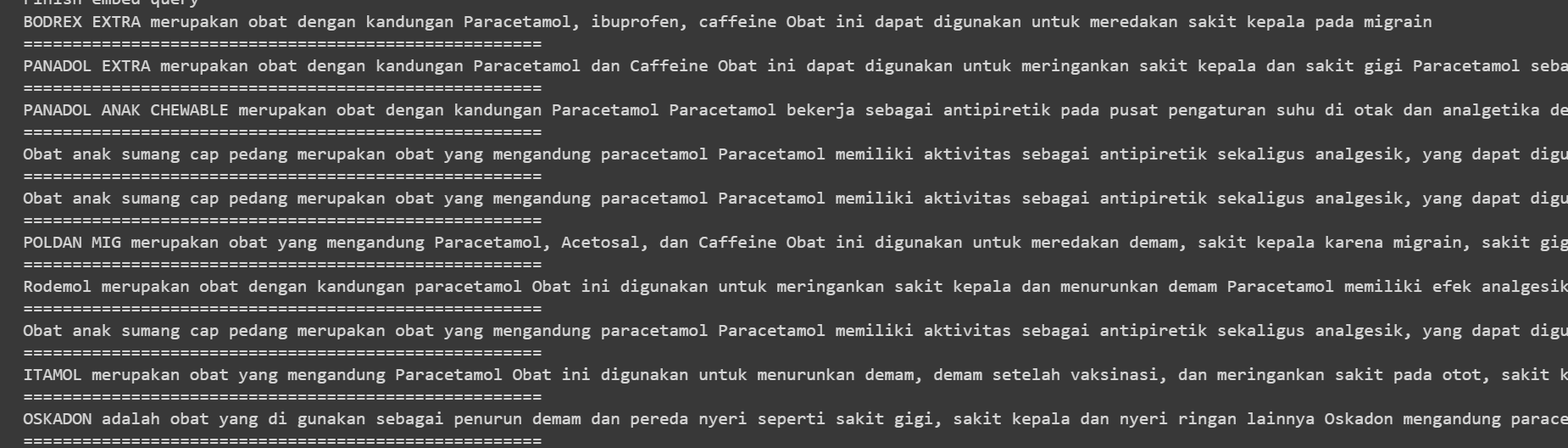
* Obat batuk tidak berdahak



* Obat memar



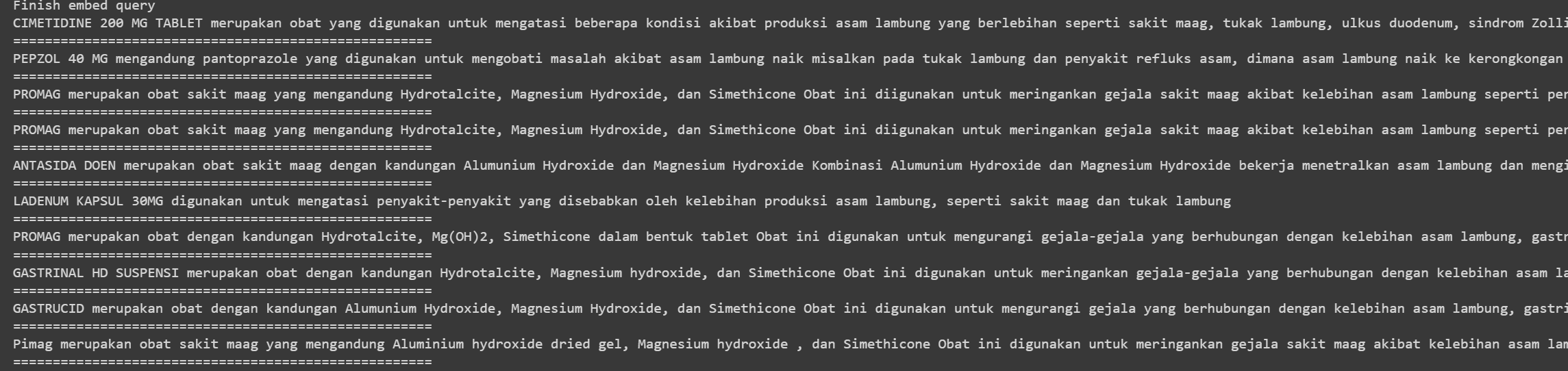
* Obat migrain dengan komposisi paracetamol



* obat pereda gejala arthritis dengan komposisi meloxicam

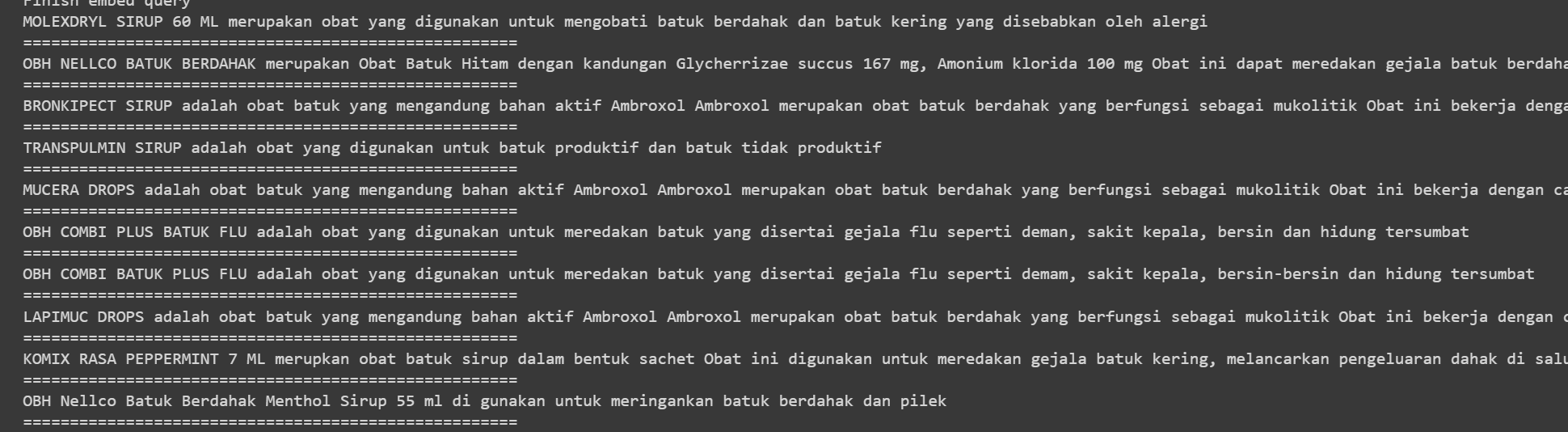


* obat untuk maag dan asam lambung dengan kandungan cimetidine

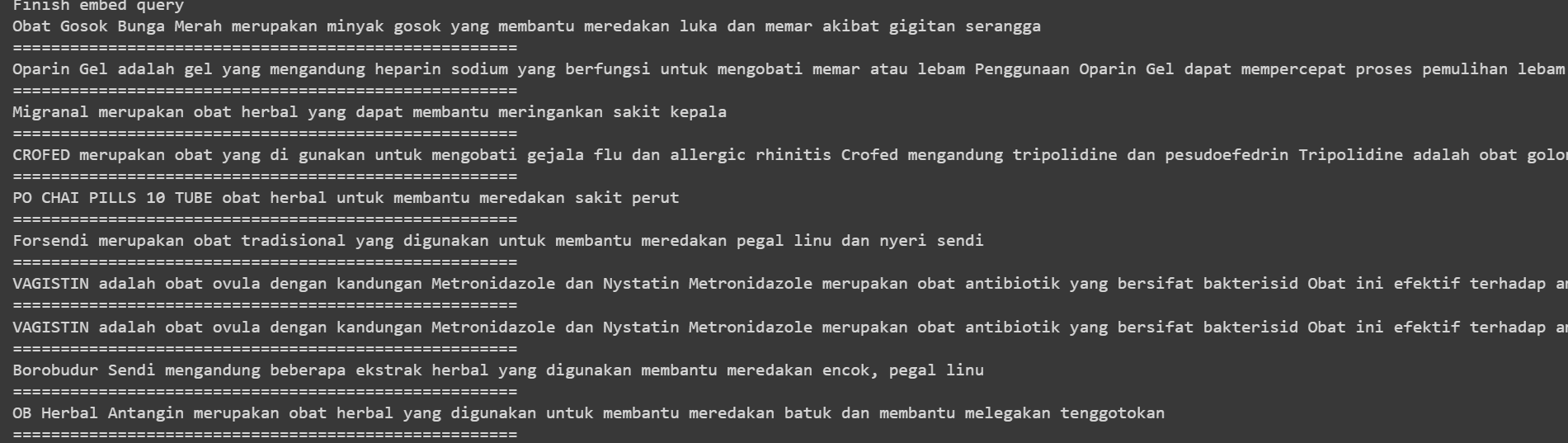


Hybrid + QE:

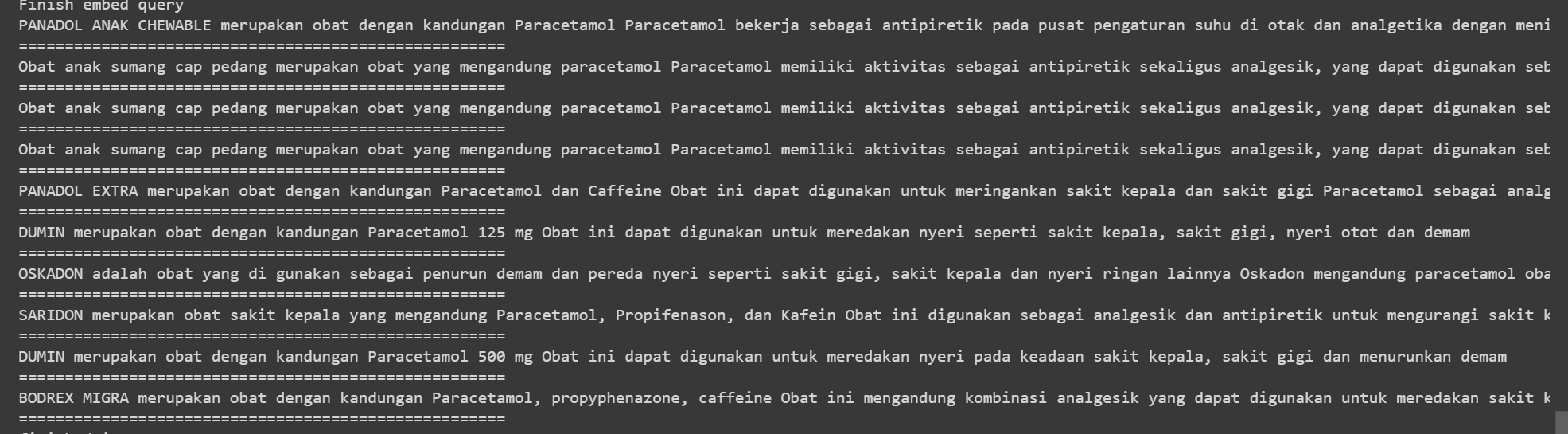
* Obat batuk tidak berdahak



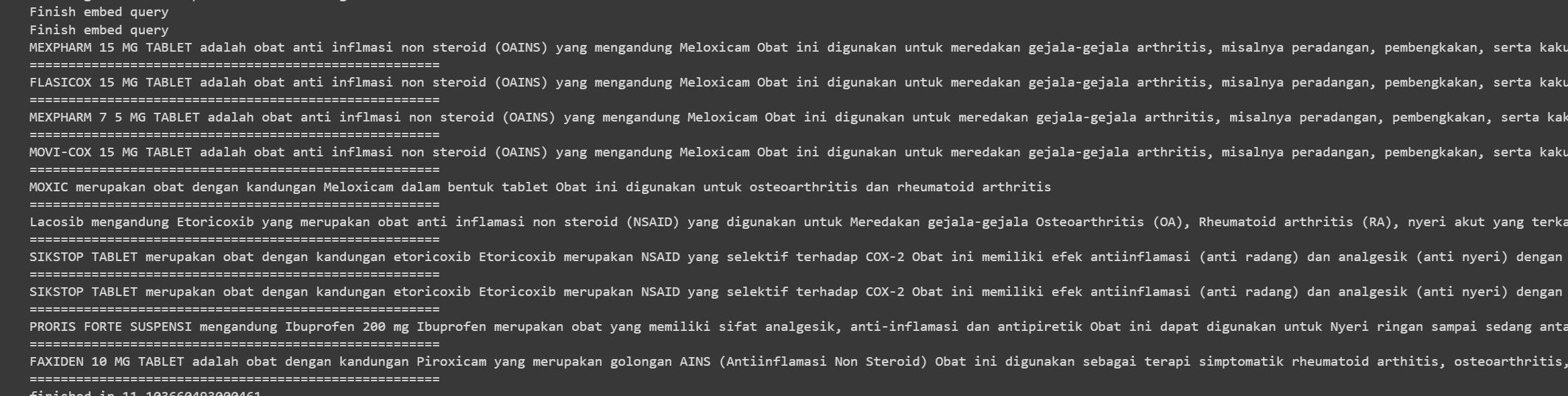
* Obat memar



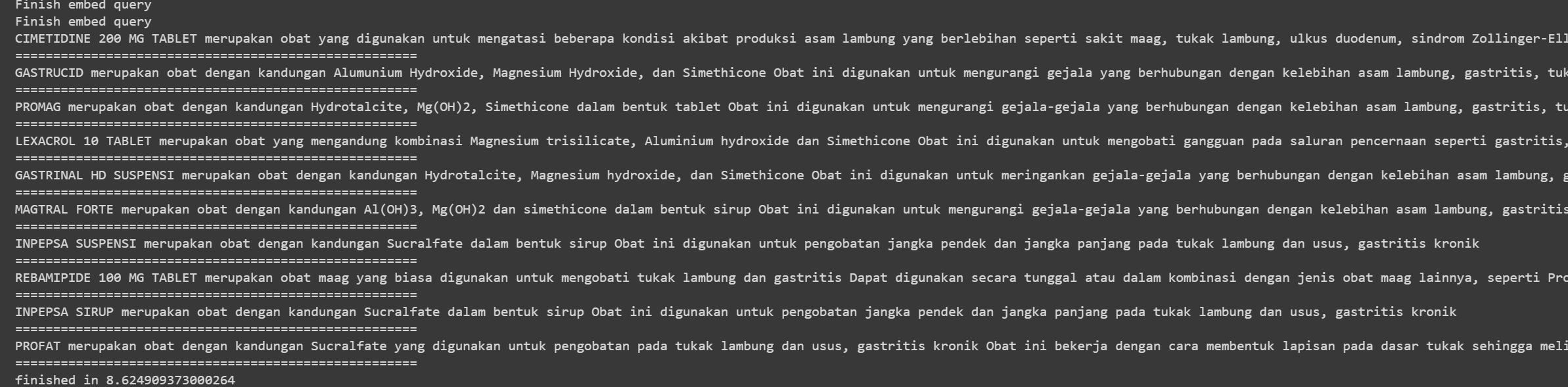
* Obat migrain dengan komposisi paracetamol



* obat pereda gejala arthritis dengan komposisi meloxicam



* obat untuk maag dan asam lambung dengan kandungan cimetidine



Buat clustering datmin: <https://github.com/PradipNichite/Youtube-Tutorials/blob/main/Youtube_Course_Sentence_Transformers.ipynb>

<https://github.com/dreji18/Clustering-with-Bert-Embeddings/blob/main/Clustering%20with%20Bert%20Embeddings.ipynb>

Hybrid search & Semantic Re-ranking Journal:

<https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview>

<https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/azure-cognitive-search-outperforming-vector-search-with-hybrid/ba-p/3929167>

Semantic Re-Ranking: <https://learn.microsoft.com/en-us/azure/search/semantic-search-overview>

Sparse Dense Index ( include BM25 vs SPLADE): <https://www.youtube.com/watch?v=EZfONAne55M>

Using Langchain for Hybrid Searh: <https://www.youtube.com/watch?v=lYxGYXjfrNI>

Langchain, RAG, FAISS: <https://medium.com/international-school-of-ai-data-science/implementing-rag-with-langchain-and-hugging-face-28e3ea66c5f7>

Code Inspiration:

<https://github.com/pinecone-io/examples/blob/master/learn/search/hybrid-search/ecommerce-search/ecommerce-search.ipynb>

query **=** "dark blue french connection jeans for men"

*# create sparse and dense vectors*

sparse **=** bm25**.**encode\_queries(query)

dense **=** model**.**encode(query)**.**tolist()

*# search*

result **=** index**.**query(

top\_k**=**14,

vector**=**dense,

sparse\_vector**=**sparse,

include\_metadata**=True**

)

*# used returned product ids to get images*

imgs **=** [images[int(r["id"])] **for** r **in** result["matches"]]

imgs

disini tu yang bagian index.query buat function yang secara otomatis cari cosine similarity trus diranking pakek RRF. Kalo di contoh dia bikin function hybrid scale ( pakek CC bukan RRF),

**def** hybrid\_scale(dense, sparse, alpha: float):

"""Hybrid vector scaling using a convex combination

alpha \* dense + (1 - alpha) \* sparse

Args:

dense: Array of floats representing

sparse: a dict of `indices` and `values`

alpha: float between 0 and 1 where 0 == sparse only

and 1 == dense only

"""

**if** alpha **<** 0 **or** alpha **>** 1:

**raise** ValueError("Alpha must be between 0 and 1")

*# scale sparse and dense vectors to create hybrid search vecs*

hsparse **=** {

'indices': sparse['indices'],

'values': [v **\*** (1 **-** alpha) **for** v **in** sparse['values']]

}

hdense **=** [v **\*** alpha **for** v **in** dense]

**return** hdense, hsparse

trus inputnya nanti jadi pakek vector = hdense, sparse\_vector = hsparse

question **=** "dark blue french connection jeans for men"

*# scale sparse and dense vectors*

hdense, hsparse **=** hybrid\_scale(dense, sparse, alpha**=**0)

*# search*

result **=** index**.**query(

top\_k**=**14,

vector**=**hdense,

sparse\_vector**=**hsparse,

include\_metadata**=True**

)

*# used returned product ids to get images*

imgs **=** [images[int(r["id"])] **for** r **in** result["matches"]]

*# display the images*

display\_result(imgs)

ges mungkin aku share juga gimana cara kerja query expansion yang di llm, biar kalian ada gambaran dan mungkin punya ide gimana ngembanginnya atau kalo mau di hybrid alurnya yang paling pas gimana

jadi untuk llm sendiri ada 2 masalah utama:

* pertama adalah hallucination karena terbatas pada data training llm itu sendiri
* keduaa adalah informasi yang udaah ngga up to date. Contoh: kita tanya berapa skor bola kemarin malem? LLM akan salah jawab karena dia gapunya datanya.

Gimana cara atasinnya? Ada konsep yang namanya RAG atau Retrieval Augmented Generation. Dimana dia akan menggabungkan antara domain dengan informasi yang ada di luar domain kita. Lebih jauh lagi, ada yang namanya RAG fusion. Dimana dia akan generate beberapa pertanyaan yang berhubungan sama pertanyaan asli, sehingga jawaban dari LLM itu bener” tepat. Contoh,

query asli: “jelaskan tentang panadol”, dari sini akan degenerate pertanyaan seperti:

* “pabrik farmasi apa yang mendistribusikan Panadol”
* “apa komposisi Panadol”
* “apa kegunaan Panadol”

Dari ketiga 4 query ini, akan dilakukan searching kemudian di ranking dengan RRF.

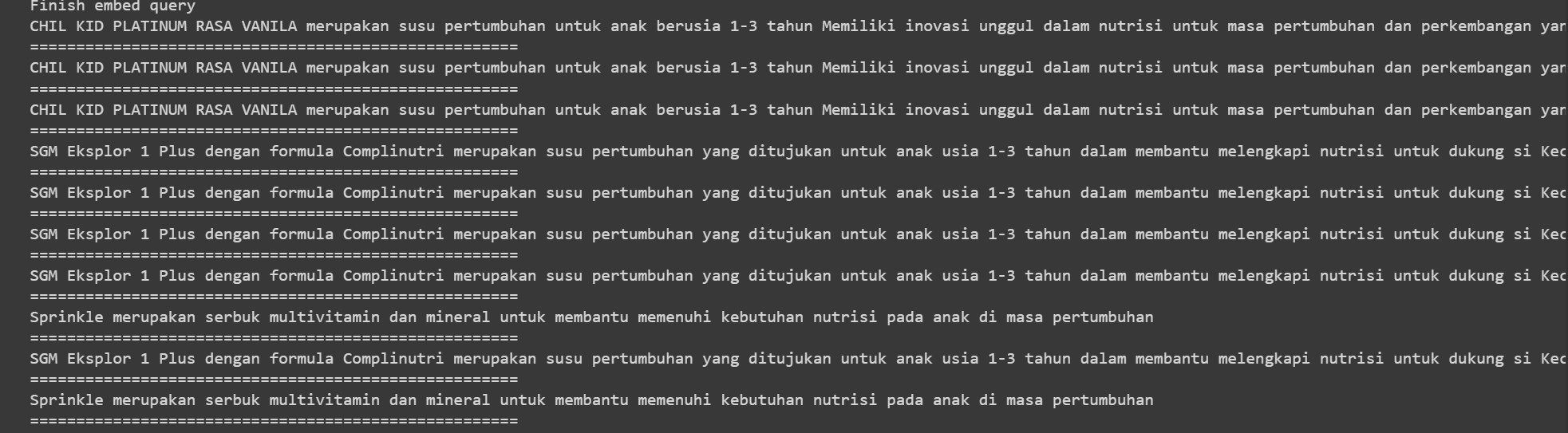
Ide ini sebenernya didesain untuk masalah chatbot dan query” di dokumen yang panjang kek artikel, text hukum, dkk. Intinya, jawaban yang bersifat generative. Nah berhubung sebenernya yang kita butuhkan bukan untuk generate paragraph atau jawaban baru, tapi lebih ke mencari jawaban yang paling tepat. Jadi yang kita butuhkan itu pertanyaan yang mirip (punya kesamaan makna dengan query asli. Perintah yang aku inputkan ke llm adalah: lakukan query expansion dan transformation dengan domain di bidang medis, farmasi dan kesehatan. Buat lima query dengan makna yang sama berdasarkan sinonim dan hiponim dari tiap kata yang ada pada query original ini.

Dari sini akan muncul banyak query alternatif. Setelah itu, aku cek satu” dengan cosine similarity. Kenapa? Untuk menghindari hallucinating LLM, karena mungkin aja llm generate kalimat yang sebenernya kalo kita piker masih berhubungan, tapi secara makna kalimat beda. Aku set thresholdnya antara 0.6 sampai 0.9. Kenapa kok ngga sampai 1? Karena 0.9 ke atas mayoritas adalah alternatif query yang struktur kalimatnya sangat mirip, sedangkan yang kita butuhkan adalah kombinasi yang berbeda”.

Dari proses ini maka kita punya total 6 query. 1 query asli + 5 query alternatif. Dari sini aku lakuin RRF lagi untuk lexical dan semantic. Kemudian hasil rrf dari lexical dan semantic aku rrf lagi sampai jadi 1 ranking final.

Nah ini aku barusan coba, 2 query yang rafik coba diatas.

Percobaan pertama untuk susu untuk tumbuh kembang anak. Dia generate beberapa query ini: ['nutrisi cair untuk pertumbuhan dan perkembangan anak', 'cairan bergizi untuk membantu anak tumbuh dan berkembang', 'minuman bergizi yang diformulasikan untuk mendukung tumbuh kembang anak', 'susu yang diformulasikan khusus untuk mendukung pertumbuhan dan perkembangan anak', 'susu yang mengandung nutrisi penting untuk membantu anak tumbuh dan berkembang secara optimal']. Disiniii yang aku bilang hasil LLM kadang hasilnya agak kurang pas. Yang harusnya susu, disini jadi minuman bergizi dan cairan bergizi. Mungkin bagi kita make sense, tapi hasilnya jadi kurang tepat. Contoh disini muncul Sprinkle (serbuk multivitamin dan mineral), mungkin hasil ini muncul karena keyword cairan bergizi dan minuman bergizi. Jadi dalam case ini, metode PRF yang rafik bikin jauh lebih optimal.



Nah di percobaan kedua, mungkin bisa dibilang hasil query expansion dari LLM tepat dan tidak mengganti konteks. Dari ‘obat untuk mengobati memar' jadi 'obat untuk memar yang bengkak', 'obat untuk memar di wajah', 'pengobatan untuk luka memar yang bengkak', 'pengobatan untuk luka memar di wajah', 'obat untuk mengatasi lebam'. Hasilnya pun menurutku better dari metode rafik yang prf kali ini, karena top 2 dokumen berkaitan sama memar. Walaupun sisanya ngaco (kemungkinan karena memang ngga ada lagii)

