Hyperfast Contextual Custom LLM with Agents, Multitokens, Explainable AI, and Distillation

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.GenAItechLab.com
Version 1.0, September 2024

Abstract

I discuss version 2.0 of my enterprise multi-LLM. Version 1.0 was presented in my recent article entitled "Custom Enterprise LLM/RAG with Real-Time Fine-Tuning", posted here. Since version 2.0 is backward-compatible and consists of several important additions, I included all the relevant material from the previous article, in this paper. New additions include multitoken distillation when processing prompts, agents to meet user intent, singularization, and several improvements such as enhanced command menu. Most importantly, I added several illustrations, featuring xLLM in action as well as important parts of the code.

Contents

1 Innovative architecture

This article features an application of xLLM to extract information from a corporate corpus, using prompts referred to as "queries". The goal is to serve the business user – typically an employee of the company or someone allowed access – with condensed, relevant pieces of information including links, examples, PDFs, tables, charts, definitions and so on, to professional queries. The original xLLM technology is described in this presentation. The main differences with standard LLMs are:

- No training, no neural network involved. Thus, very fast and easy to fine-tune with explainable parameters, and much fewer tokens. Yet, most tokens consist of multiple terms and are called multitokens. Also, I use variable-length embeddings. Cosine similarity and dot products are replaced by customized pmi (pointwise mutual information, [Wiki]).
- Parameters have a different meaning in my context. In standard architectures, they represent the weights connecting neurons. You have billions or even trillions of them. But there is no neural network involved here: instead, I use parametric weights governed by a few top-level parameters. The weights – explicitly specified rather than iteratively computed – are not the parameters. My architecture uses two parameter sets: frontend and backend. The former are for scoring and relevancy; they are fine-tuned in real time with no latency, by the user or with some algorithm. A relevancy score is shown to the user, for each retrieved item.

```python
def update_nestedHash(hash, key, value, count=1):

    # 'key' is a word here, value is tuple or single value
    if key in hash:
        local_hash = hash[key]
    else:
        local_hash = {}
    if type(value) is not tuple:
        value = (value,)
    for item in value:
        if item in local_hash:
            local_hash[item] += count
        else:
            local_hash[item] = count
    hash[key] = local_hash
    return(hash)
```

Figure 1: Nested hash database, lines 12–27 in the code

- I don't use vector or graph databases. Tables are stored as nested hashes, and fit in memory (no GPU needed). By nested hashes, I mean key-value tables, where the value may also be a key-value table. The format is similar to JSON objects, see Figures 1 and 3. In standard architectures, the central table stores the embeddings. Here, embeddings are one of many backend tables. In addition, there are many contextual tables (taxonomy, knowledge graph, URLs) built during the crawling. This is possible because input sources are well structured, and elements of structure are recovered thanks to smart crawling.
- The Python code does not use any library, nor any API call. Not even Pandas, Numpy, or NLTK. So you can run it in any environment without concern for library versioning. Yet it has fewer than 600 lines of code, including the fine-tuning part in real time. I plan to leverage some library functions in the future such as auto-correct, singularize, stem, stopwords and so on. However, home-made solutions offer more customization, such as ad-hoc stopwords lists specific to each sub-LLM, for increased performance. For instance, the one-letter word 'p' can not be eliminated if the sub-LLM deals with statistical concepts. The only exception to the "no library" rule is the Requests library, if you choose to download the test enterprise corpus from its GitHub location.
- This article focuses only on one part of an enterprise corpus: the internal documentation about how to implement or integrate AI and machine learning solutions. Other parts include marketing, IT, product, sales, legal and HR. A specific sub-LLM is built for each part, using the same architecture. The full LLM consists of these sub-LLMs, glued together with an LLM router to redirect user prompts to the specific parts, possibly spanning across multiple sub-LLMs. For instance, 'security' is found in multiple sub-LLMs.

1.1 From frontend prompts to backend tables

The prompt is first stripped of common words such as 'how to', 'example', or 'what is'. The result is called a shortened prompt. The stripped words may be treated separately to determine the user intent, called action. They are also stripped from the corpus (crawled data) but again, used to assign an action label to each text entity in the corpus. Then the shortened prompt is sorted in alphabetical order and broken down into sorted n-grams. A shortened prompt with n words gives rise to $2^n - 1$ sorted n-grams containing from one to n words. Without sorting, that number would be $1! + 2! + \cdots + n!$, too large for fast processing.

```
tableNames = (
  'dictionary',     # multitokens (key = multitoken)
  'hash_pairs',     # multitoken associations (key = pairs of multitokens)
  'ctokens',        # not adjacent pairs in hash_pairs (key = pairs of multitokens)
  'hash_context1',  # categories (key = multitoken)
  'hash_context2',  # tags (key = multitoken)
  'hash_context3',  # titles (key = multitoken)
  'hash_context4',  # descriptions (key = multitoken)
  'hash_context5',  # meta (key = multitoken)
  'hash_ID',        # text entity ID table (key = multitoken, value is list of IDs)
  'hash_agents',    # agents (key = multitoken)
  'full_content',   # full content (key = multitoken)
  'ID_to_content',  # full content attached to text entity ID (key = text entity ID)
  'ID_to_agents',   # map text entity ID to agents list (key = text entity ID)
  'ID_size',        # content size (key = text entity ID)
  'KW_map',         # for singularization, map kw to single-token dictionary entry
  'stopwords',      # stopword list
)
```

Figure 2: Primary backend tables, lines 193–210 in the code

```
extraWeights = backendParams['extraWeights']
word = word.lower()  # add stemming
weight = 1.0
if word in category:
    weight += extraWeights['category']
if word in tag_list:
    weight += extraWeights['tag_list']
if word in title:
    weight += extraWeights['title']
if word in meta:
    weight += extraWeights['meta']

update_hash(backendTables['dictionary'], word, weight)
update_nestedHash(backendTables['hash_context1'], word, category)
update_nestedHash(backendTables['hash_context2'], word, tag_list)
update_nestedHash(backendTables['hash_context3'], word, title)
update_nestedHash(backendTables['hash_context4'], word, description) # takes space, don't build?
update_nestedHash(backendTables['hash_context5'], word, meta)
update_nestedHash(backendTables['hash_ID'], word, ID)
update_nestedHash(backendTables['hash_agents'], word, agents)
for agent in agents:
    update_nestedHash(backendTables['ID_to_agents'], ID, agent)
update_nestedHash(backendTables['full_content'], word, full_content) # takes space, don't nuild?
update_nestedHash(backendTables['ID_to_content'], ID, full_content)
```

Figure 3: Updating primary backend tables, lines 61–72 in the code

Sorted n-grams detected in the prompt are then matched against the sorted n-grams found in the backend table `sorted_ngrams` based on the corpus. Each entry in that table is a key-value table. For instance, the entry for the key 'data mining' (a sorted n-gram) might be {'data mining':15, 'mining data': 3}. It means that 'data mining' is found 15 times in the corpus, while 'mining data' is found 3 times. Of course, n-grams not found in the corpus are not in that table either. The sorted n-grams table helps retrieve unsorted word combinations found in the corpus and match them back to unsorted n-grams in the prompt. This is in contrast to systems where word order is ignored, leading to problems.

From there, each backend table is queried to retrieve the value attached to a specific n-gram found in the prompt. The value in question is also a key-value table: for instance a list of URLs where the key is an URL and the value is the number of occurrences of the n-gram in question, on the landing page. In each section (titles, URLs, descriptions and so on) results shown to the user are displayed in relevancy order, with a higher weight assigned to n-grams (that is, multitokens) consisting of many words, as opposed to multitokens consisting of one or two words. Embeddings are derived from a backend table called `hash_pairs` consisting of pairs of multitokens found in the same sub-entity in the corpus. Finally, multitokens may or may not be adjacent. Pairs with non-adjacent multitokens are called contextual pairs. Occurrences of both multitokens, as well as joint occurrence (when both are simultaneously found in a same sub-entity) are used to compute pmi, the core relevancy metric. Embeddings are stored in the `embeddings` key-value backend table, also indexed by multitokens. Again, values are key-value tables, but this time the nested values are pmi scores.

1.2 What is not covered here

The goal was to create a MVP (minimum viable product) featuring the original architecture and the fine-tuning capability in real time. With compact and generic code, to help you easily add backend tables of your choice, for instance to retrieve images, PDFs, spreadsheets and so on when available in your corpus.

Some features are not yet implemented in this version, but available in the previous version discussed here and in my book "State of the Art in GenAI & LLMs – Creative Projects, with Solutions", available here. The following will be available in the next release: auto-correct, stemming, singularization and other text processing techniques, both applied to the corpus (crawled data) and the prompt. I will also add the ability to use pre-computed backend tables rather than building them from the crawl each time. Backend tables produced with the default backend parameters (see code lines 193–262 in section 5) are on GitHub, here.

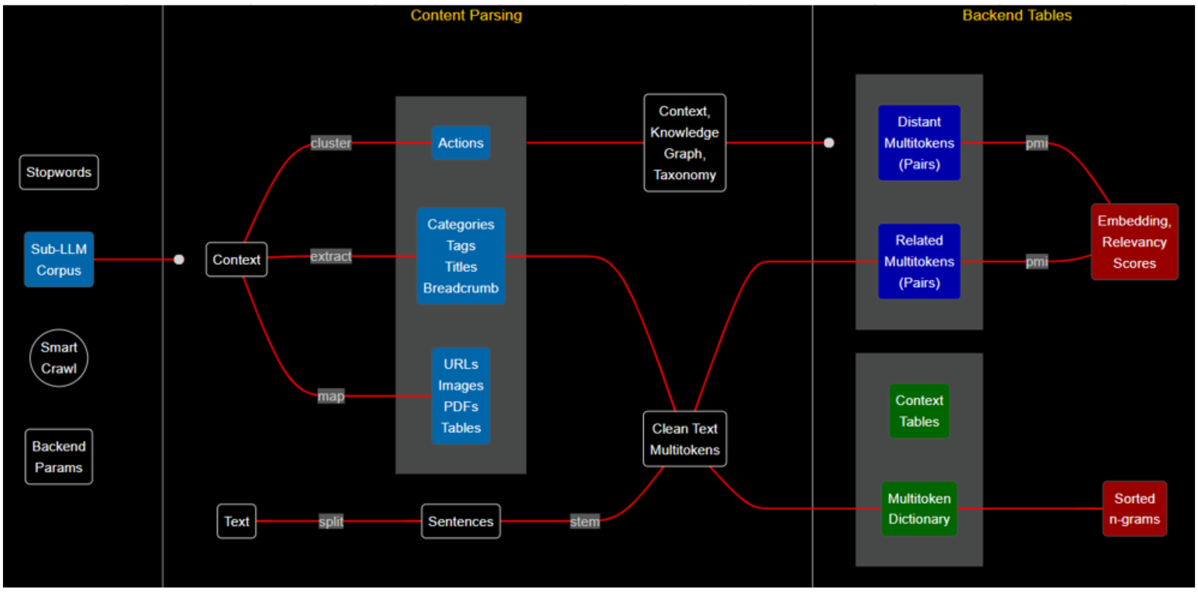

Figure 4: From crawl to backend tables (high resolution here)

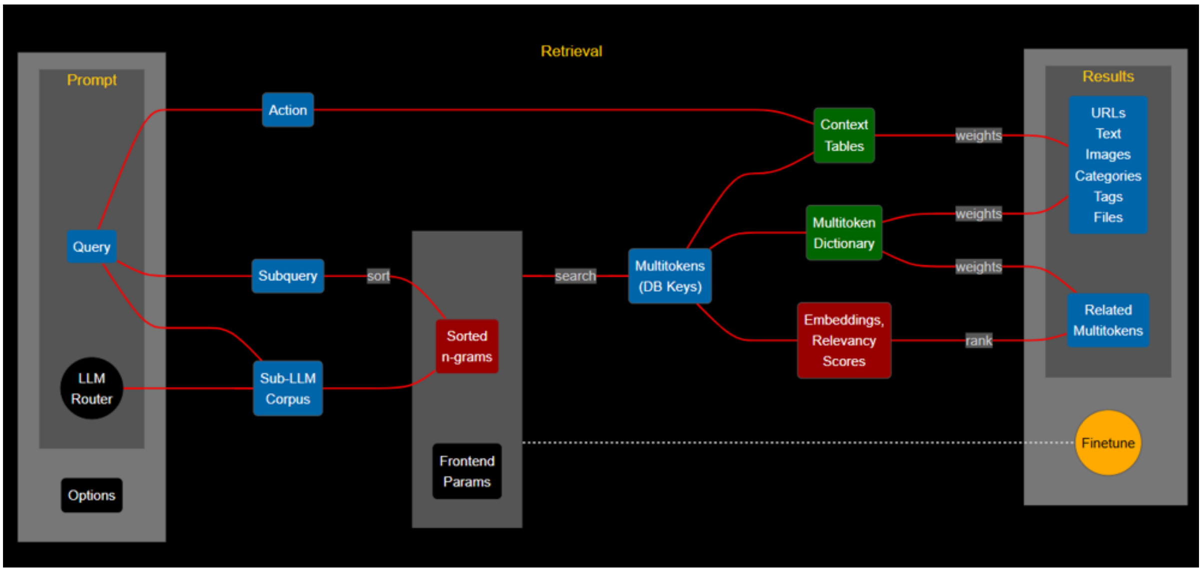

Figure 5: From prompt to query results, via backend tables (high resolution here)

Also to be included in the next release: corpus augmentation with synonyms and abbreviations dictionaries, as well as contextual multitokens. The latter is implemented in the previous version and discussed in section 8.3 in my book [1]. It consists of tokens containing non-adjacent words in the corpus. However, contextual pairs are included in the current release: it consists of pairs of non-adjacent multitokens, stored in a table called `ctokens` used to produce the embeddings. See lines 183–186 in the code. Then, words such as 'San Francisco' must be treated as single tokens.

Finally, prompts are not broken down into sub-prompts. But the concept of action is now implemented. An action determines the user intent: whether he/she is searching for 'how to', 'what is', 'examples', 'data', 'comparisons', and so on. It requires the addition of an extra backend table, corresponding to the 'action' field in the text entities, along with 'category', 'description', 'title' and so on. However, there is no 'action' field. It must be constructed with a clustering algorithm applied to the corpus as a pre-processing step, to add action labels to each text entity. My current approach is actually simpler and discussed in section 2

2 Parameters, features, and fine-tuning

In the case study discussed here, the input source consists of about 500 text elements stored as JSON entities, each with a number of fields: title, description, category, tags, URL, ID, and so on. It comes from a Bubble database that populates the website where the corpus is accessible to end-users. In the Python code, the list of entities covering the entire corpus is named `entities`, while a single entry is named `entity`. For each entity, the various fields are stored in a local key-value table called `hash_crawl`, where the key is a field name (for instance, category) and the value is the corresponding content. See lines 292–338 in the code in section 5. The full corpus (the anonymized input source) is available as a text file named `repository.txt`, here on GitHub.

2.1 Backend parameters

Multitokens contain up to 4 terms, as specified by the backend parameter `max_multitokens` in line 265 in the code. The `hash_pairs` table consists of multitokens pairs, each with up to 3 terms: see parameter `maxTerms` in line 267. The maximum gap allowed between two contextual multitokens is 3 terms: see parameter `maxDist` in line 266. These limitations are set to prevent the number of pairs and tokens from exploding. In the end, there are 12,575 multitokens, stored in the `dictionary` table, after removing stopwords. The total number of multitoken pairs is 223,154, while the size of the corpus is 427KB uncompressed.

Stopwords – the words to ignore when building the tables – are manually detected by looking at the most frequent tokens, both in the corpus and in prompt result: see the list in lines 216–222. Finally, when counting multitoken occurrences, appearances in categories, titles and tags get an extra boost, compared to regular text: see lines 268–275 and Figure 3. For the full list of backend parameters, see Figure 6.

```
backendParams = {
    'max_multitoken': 4, # max. consecutive terms per multi-token for inclusion in dictionary
    'maxDist' : 3,       # max. position delta between 2 multitokens to link them in hash_pairs
    'maxTerms': 3,       # maxTerms must be <= max_multitoken
    'extraWeights' :     # deafault weight is 1
       {
          'description': 0.0,
          'category':    0.3,
          'tag_list':    0.4,
          'title':       0.2,
          'meta':        0.1
       }
}
```

Figure 6: Backend parameters, lines 697–722 in the code

I did not include `embeddings` and `sorted_ngrams` in the `backendTables` structure in lines 193–214, because they are built on top of primary backend tables, more specifically `dictionary` and `hash_pairs`. The `pmi` values attached to the embeddings are computed as follows:

$$\mathrm{pmi}(t_A, t_B) = \frac{n_{AB}}{\sqrt{n_A \cdot n_B}}, \tag{1}$$

where n_A, n_B, n_{AB} are the counts (computed on the full corpus) respectively for multitokens t_A, t_B, and the joint occurrence of t_A, t_B within a same sub-entity (that is, a sentence identified by separators, within a text entity). The user can choose a different formula, or different separators. Primary backend tables are listed in Figure 2.

2.2 Frontend parameters

Given the small size of the corpus and backend tables, the backend parameters can be updated in real time. Currently, the code allows the user to easily update the frontend parameters while testing various prompts. The frontend parameters are found in lines 699–721 in the code, and in Figure 8. They control the results displayed, including the choice of a customized pmi function, and top keywords to exclude such as 'data' found in almost all text entities. Adding 'data' to the `ignore` list does not eliminate results based on multitokens containing 'data', as long as the multitokens in question consist of more than one word, such as 'data asset'.

```python
def default_frontendParams():

    frontendParams = {
                        'embeddingKeyMinSize': 1, # try 2
                        'embeddingValuesMinSize': 2,
                        'min_pmi': 0.00,
                        'nABmin': 1,
                        'Customized_pmi': True,
                        'ContextMultitokenMinSize': 1, # try 2
                        'minOutputListSize': 1,
                        'bypassIgnoreList': False,
                        'ignoreList': ('data',),
                        'maxTokenCount': 100,  # ignore generic tokens if large enough
                        'show': {
                                  # names of sections to display in output results
                                  'Embeddings': True,
                                  'Category'  : True,
                                  'Tags'      : True,
                                  'Titles'    : True,
                                  'Descr.'    : False, # do not built to save space
                                  'Whole'     : False, # do not build to save space
                                  'ID'        : True,
                                  'Agents'    : True,
                                 }
                      }
    return(frontendParams)
```

Figure 7: Default frontend parameters, lines 699–721 in the code

When entering a prompt, the end-user can choose pre-selected queries listed in lines 760-769, his/her own queries, or simple instructions to update or view the frontend parameters, using one of the options in lines 773–792. The catch-all parameter set (with all values set to zero) yields the largest potential output. Do not use it except for debugging, as the output may be very long. However, if you want to try it, choose the option `-f` for full results. This is accomplished by entering `-f` on the command prompt.

2.3 Agents

Agents determine the user intent to retrieve the appropriate content. For instance:, examples, data, definitions, best practices, standards, on-boarding, and so on. In Figure 5, they are represented by the action box. One way to create an agentic LLM is to add an agent field in each text entity when crawling the corpus. See sample text entity in Table 1. You can do it using clustering techniques, applied to the corpus. Text entities are relatively small pieces of content coming straight from the corpus, usually determined by the corpus structure: in this case, a bubble database, but it could also be a repository of PDF documents or web pages.

```python
agent_map = {
             'template':'Template',
             'policy':'Policy',
             'governance':'Governance',
             'documentation':'Documentation',
             'best practice':'Best Practices',
             'bestpractice':'Best Practices',
             'standard':'Standards',
             'naming':'Naming',
             'glossary':'Glossary',
             'historical data':'Data',
             'overview':'Overview',
             'training':'Training',
             'genai':'GenAI',
             'gen ai':'GenAI',
             'example':'Example',
             'example1':'Example',
             'example2':'Example',
            }
```

Figure 8: Agent map, lines 227–245 in the code

Getting a list of top multitokens helps your build your agent backend table. In our example, see the list in question Table 1, extracted from the `dictionary` backend table. Another option consists in analyzing dozens, thousands, or millions of user prompts to identify potential actions. The ideal solution is to combine all these options to create agents that correspond not only to user intent, but also to what is actually in the corpus.

The agent map for my case study, is pictured in Figure 8. I will improve the format in the next version, and use a many-to-many rather than many-to-one table. In the key-value pairs in the picture, the value on the right is an agent, while the key on the left is a multitoken. The structure thus maps words found in the corpus, to agents. Agents are then incorporated to backend tables for retrieval. In my current implementation, there are two agent backend tables, besides `agent_map` just described:

- `hash_agents` indexed by multitokens found in `dictionary`, to retrieve agents associated to multitokens.
- `ID_to_agents` indexed by text entity IDs (`ID` in the code) , to retrieve agents associated to entity IDs.

These two tables are used to produce the agent section in the query results, as shown in Figure 9. For details, see lines 679–686 in the code. For instance, the fourth line in the picture tells you that the multitoken 'data assets' is associated to agent 'Governance' (among others), and that four text entity IDs match this combination: 42, 48, 199, 259, with 259 having the most content with 1153 characters.

In Figure 9, the size of each entity ID is also displayed to help the user identify IDs with more content; they might be more valuable. With the command `-i ID` in the prompt box, the user can then retrieve the full content of entity `ID`, in a format similar to Table 1. Two extra backend tables are involved in the process: `hash_size` and `ID_to_content`.

```
('Data', 'detailed') --> (511, 513)
('Example', 'data assets') --> (90,)
('Example', 'detailed') --> (90,)
('Governance', 'data assets') --> (42, 48, 199, 259)
('Governance', 'detailed') --> (101, 107)
('Governance', 'information assets') --> (223,)
('Policy', 'data assets') --> (42, 48, 199)
('Policy', 'detailed') --> (101,)
('Policy', 'information assets') --> (223,)
('Template', 'detailed') --> (107,)

  ID  Size

 511   690
 513   692
  90   772
  42   948
  48   916
 199   980
 259  1153
 101   851
 107  1242
 223   978
```

Figure 9: Example of agent section shown in query results

Currently, the agent(s) are not automatically detected from the user prompt. I will add this feature in the next version. In the meanwhile, it is possible to display the full list of agents to the user, and let him make his selection. Finally, my agents do not perform actions such as writing messages or solving math problems. Their goal is to deliver more relevant results, based on what users are looking for by analyzing prompt data. A different version of my xLLM performs clustering, build taxonomies, and make predictions based on text: see here, and Figure 13.

2.4 Reproducibility

Most GenAI applications rely on deep neural networks (DNN) such as GANs (generative adversarial networks). This is the case for transformers, a component of many LLMs. These DNNs rely on random numbers to generate latent variables. The result can be very sensitive to the seed.

In many instances, particularly for synthetic data generation and GPU-based apps, the author does not specify seeds for the various PRNG (pseudo-random number generator) involved, be it from the Numpy, Random, Pandas, PyTorch libraries, base Python, or GPU. The result is lack of reproducibility. This is not the case with my algorithms, whether GAN or NoGAN. All of them lead to reproducible results, including the xLLM system described here, which does not rely on transformers or random numbers.

There have been some attempts to improve the situation recently, for instance with the `set_seed` function in some transformer libraries. However, it is not a full fix. Furthermore, the internal PRNGs found in Python libraries are subject to change without control on your side. To avoid these problems, I invite to check out my own PRNGs, some of them faster and better than any other one on the market. See my article "Fast Random Generators with Infinite Period for Large-Scale Reproducible AI and Cryptography", available here.

2.5 Singularization, stemming, auto-correct

The KW_map backend table built in lines 870–888 in the code (see Figure 10), is a first attempt at adding NLP functions without using Python libraries. The table is created and saved after running the full code for the first time. Python libraries have glitches that can result in hallucinations, for instance singularizing "hypothesis" to "hypothesi". They require exception lists such as do-not-singularize as a workaround. Thus the idea to avoid them.

The code featured in Figure 10 links the singular and plural version of single-tokens found in the dictionary (when both exist), so that a user looking for (say) "tests" also gets result coming from "test". See lines 822–823 in the code when processing frontend prompts, and lines 148–149 when building backend tables.

More NLP functions will be added in the next version, including from Python libraries, such as singularize, stemming and auto-correct. To minimize hallucinations, it is better to have a specific list for each sub-LLM. Even then, one must be careful to avoid singularizing (say) "timeliness" to "timelines" or "practices" (noun) to "practice" (verb or noun). In the next version, KW_map will also be used as a synonyms and abbreviation dictionary.

```
def create_KW_map(dictionary):
    # singularization
    # map key to KW_map[key], here key is a single token
    # need to map unseen prompt tokens to related dictionary entries
    #    example: ANOVA -> analysis~variance, ...

    OUT = open("KW_map.txt","w")

    for key in dictionary:
        if key.count('~') == 0:
            j = len(key)
            keyB = key[0:j-1]
            if keyB in dictionary and key[j-1] == 's':
                if dictionary[key] > dictionary[keyB]:
                    OUT.write(keyB + "\t" + key + "\n")
                else:
                    OUT.write(key + "\t" + keyB + "\n")
    OUT.close()
    return()
```

Figure 10: Building the KW_map backend table

2.6 Augmentation, distillation, and frontend tables

I build two frontend tables q_dictionary and q_embeddings each time a new prompt is generated, in order to retrieve the relevant content from the corpus. These tables are similar and linked to backend dictionary and embeddings, but far smaller and focusing on prompt content only. See lines 828–855 in the code.

```
def distill_frontendTables(q_dictionary, q_embeddings, frontendParams):
    # purge q_dictionary then q_embeddings (frontend tables)

    maxTokenCount = frontendParams['maxTokenCount']
    local_hash = {}
    for key in q_dictionary:
        if q_dictionary[key] > maxTokenCount:
            local_hash[key] = 1
    for keyA in q_dictionary:
        for keyB in q_dictionary:
            nA = q_dictionary[keyA]
            nB = q_dictionary[keyB]
            if keyA != keyB:
                if (keyA in keyB and nA == nB) or (keyA in keyB.split('~')):
                    local_hash[keyA] = 1
    for key in local_hash:
        del q_dictionary[key]

    local_hash = {}
    for key in q_embeddings:
        if key[0] not in q_dictionary:
            local_hash[key] = 1
    for key in local_hash:
        del q_embeddings[key]

    return(q_dictionary, q_embeddings)
```

Figure 11: Frontend token distillation before returning results

Then, I remove single tokens that are part of a multitoken when both have the same count in the dictionary. See line 862 in the code, calling the function pictured in Figure 11. It makes the output shown to the user, less cluttered. This step is called distillation. In standard LLMs, distillation is performed on backend tokens using a different mechanism, since multitokens are usually absent; it may result in hallucinations if not done properly. Also, in standard LLMs, the motivation is different: reducing a 500 billion token list, to (say) 50 billion. In xLLM, token lists are at least 1000 times smaller, so there is no real need for backend distillation.

Also, I keep a single copy of duplicate entities, see section 2.7. In the next version, only a limited number selected items will be shown to the user, based on relevancy score, rather than a full list. Even now, it is possible to drastically reduce the size of the output by choosing frontend parameters accordingly.

Finally, you can extend the corpus with external input sources. This step is called augmentation in RAG (retrieval augmented generation) systems. The augmented data is split into standard text entities, processed as standard entities, possibly with the 'Augmented' tag to distinguish them from organic content, when displaying results. It is also possible to perform knowledge graph and taxonomy augmentation, as described in my article "Build and Evaluate High Performance Taxonomy-Based LLMs From Scratch", available here.

2.7 In-memory database, latency, and scalability

The whole corpus and the backend tables easily fit in memory even on an old laptop. Building the tables takes less than a second. Once the tables are created or loaded, there is no latency. This is due to the small size of the corpus, and because the implementation described here deals with only one sub-LLM; the full corpus requires about 15 sub-LLMs. However, for scalability, here are some recommendations:

- Pre-load the backend tables once they have been created on the first run; do not build them each time.
- Do not create the `hash_context4` and `full_content` tables; these are among the largest, and redundant with `ID_to_content`.
- Keep only one copy of identical text entities: ideally remove duplicates directly in the corpus, as opposed to using memory-consuming `entity_list` (see lines 296 and 305).
- Unless feasible, do not store `ID_to_content` that maps the entity IDs to their full content, in memory. Only store the list of IDs using small ID tables (`hash_ID`, `ID_size`, `ID_to agents`). The idea is to search for matching IDs in the backend tables when processing a prompt, and then retrieve the actual content from a database matching IDs to content.
- A distributed architecture can be useful, whereas separate sub-LLMs are stored on different clusters, if needed.

For the time being, my system is a full in-memory LLM with in-memory database. All the backend tables and text entities (see example in Table 1) are stored in memory.

Table 1: Sample text entity from corporate corpus

Field	Value
Entity ID	1682014217673x617007804545499100
Created Date	2023-04-20T18:10:18.215Z
Modified Date	2024-06-04T16:42:51.866Z
Created by	1681751874529x883105704081238400
Title	Business Metadata Template
Description	It outlines detailed instructions for completing the template accurately, covering various sections such as data dictionary, data source, sensitivity information, and roles. After filling out the template, users can interpret the entered data, ensuring clarity on sensitivity classifications, business details, and key roles. Once completed and reviewed, the metadata is uploaded to MLTxQuest, making it accessible through the MLTxQuest portal for all authorized users, thereby centralizing and simplifying access to critical information within the organization.
Tags	metadata, mltxquest, business
Categories	Governance
URLs	

3 Case study

I now show how xLLM (the name of my LLM) works on one part of a corporate corpus (fortune 100 company), dealing with documentation on internal AI systems and policies. Here, I implemented the sub-LLM dedicated to this content. The other parts – marketing, products, finance, sales, legal, HR, and so on – require separate overlapping sub-LLMs not covered here. The anonymized corpus consists of about 300 distinct text entities, and can be found here. Table 1 features a sample text entity. The full corpus would be processed with a multi-LLM and LLM router.

In addition to the original features described in section 2, xLLM comes with a command menu, shown in Figure 12. This menu allows you to enter a standard prompt, but also to change the front-end parameters for real-time fine-tuning. Figures 4 and 5 show the main components and workflow for a single sub-LLM. Zoom in for higher resolution. For best resolution, download the original here on Google Drive for the backend diagram, and here for the frontend. Finally, the home-made LLM discussed here can be used to create a new taxonomy of the crawled corpus, based on top multitokens. These are listed, from left to right and top to bottom by order of importance, in Table 2. Note that here, I did not give a higher weight to mutlitokens consisting of multiple words. The table was produced using lines 372–375 in the Python code.

Table 2: Top multitokens found in corpus, ordered by importance

adls	storage	azure	examples	adf
csa	pipeline	development	framework	architecture
design	mltxdat	process	extract	orc
overview	quality	databricks	data quality	table
guidelines	new	guide	best practices	performance
platform	metadata	solution	business	products
project	resources	create	request	mltxhub
case	zones	key	feature	governance
devops	github	naming	standards	ops
service	monitoring	glossary	global	policy
documentation	data governance	management	document	user
roles	team	onboarding	access	integration
infrastructure	responsibilities	security	engineering	bi
ci	cd	code	learning	support
foundation	admin	timbr	ai	metrics
index	mltxdoc	serving	semantic	layer
applications	environment	mltxquest	deployment	training
api	components	essential	fitness	score
model	genai	machine learning	governance framework	alpha
ai platform	genai platform	systems		

Now, let's try two prompts, starting with 'metadata template'. With the default frontend parameters, one text entity is found: the correct one entitled 'business metadata template', because the system tries to detect the joint presence of the two words 'data' and 'template' within a same text sub-entity, whether adjacent or not. A lot more would be displayed if using the catch-all parameter set. The interesting part is the embeddings, linking the prompt to other multitokens, especially 'instructions completing template','completing template accurately', 'filling out template' and 'completed reviewed metadata'. These multitokens, also linked to other text entities, are of precious help. They can be used to extent the search or build agents.

My second test prompt is 'data governance best practices'. It returns far more results, although few clearly stand out based on the relevancy scores. The most relevant category is 'governance', the most relevant tags are 'DQ' and 'data quality', with one text entity dominating the results. Its title is 'Data Quality Lifecycle'. The other titles listed in the results are 'Data Literacy and Training Policy', 'Audit and Compliance Policy', 'Data Governance Vision', and 'Data Steward Policy'. Related multitokens include 'robust data governance', 'best practices glossary', 'training policy', 'data informed decision making' and 'data governance practices'.

3.1 Real-time fine-tuning, prompts, and command menu

Here I illustrate a full xLLM session, using a more complex sample query. It also involves fine-tuning front-end parameters in real time. The full session with commands from the command menu, and output results, is listed in section 3.2. Figure 12 shows how the command prompt looks like, as well as the result after executing the −v command.

```
--------------------------------------------------------------------
Command menu:

  -q             : print last non-command prompt
  -x             : print sample queries
  -p key value   : set frontendParams[key] = value
  -f             : use catch-all parameter set for debugging
  -d             : use default parameter set
  -v             : view parameter set
  -a multitoken  : add multitoken to 'ignore' list
  -r multitoken  : remove multitoken from 'ignore' list
  -l             : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s             : print size of core backend tables
  -c F1 F2 ...   : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
--------------------------------------------------------------------

Query, command, or integer in [0, 7] for sample query: -v

Key Description               Value
  2 min_pmi                   0.0
  3 nABmin                    1
  4 Customized_pmi            True
  5 ContextMultitokenMinSize  1
  6 minOutputListSize         1
  7 bypassIgnoreList          False
  8 ignoreList                ('data',)
  9 maxTokenCount             100

Show sections:

    Embeddings True
    Category   True
    Tags       True
    Titles     True
    Descr.     False
    Whole      False
    ID         True
    Agents     True
```

Figure 12: Command options and frontend parameters

I started with sample query 6 (the first action in Table 3), then looked at the results, fine-tune parameters (actions 5 and 6) and removed some junk (action 3), then rerun the query (action 7) then focused on getting article titles only (action 8) and rerun the query a final time (action 9).

Action	Command	Log Line
1	6	23
2	-i 107 259	591
3	-a detailed	670
4	-v	697
5	-p 6 2	747
6	-p 2 0.50	774
7	6	801
8	-c Titles	961
9	6	988

Table 3: Sample xLLM session

The detailed log with executed commands and all the output is shown in section 3.2. In particular, the nine commands in Table 3 are found at the corresponding line numbers (rightmost column in Table 3), in the log file in section 3.2. Perhaps the most useful results consist of the IDs attached to agents and multitokens related to the prompt, in lines 542–567. Also pictured in Figure 8, along with interpretation details in section 2.3. The actual content corresponding to these IDs is shown in lines 593–641. The prompt itself is shown in line 24.

I was particularly interested in finding the articles (text entities) matching my prompt, especially the titles, to check out those that interest me most. This is accomplished with the `-c Titles` command, and the results are shown in lines 988–1001. In the next code release, the corresponding text entity IDs will also be displayed along with the titles, as in the Agents section (Figure 8). This way, it is very easy to retrieve the full content corresponding the the titles in question, with the `-i` command.

Since everything is already built for this functionality, adding a few lines of code to retrieve the IDs is straightforward. I encourage you to modify the code accordingly, on your own. This would be a good exercise to help you understand my architecture. The next step is to also add the corresponding IDs in the other sections (Categories, Tags, Descr., Whole, and so on).

3.2 Sample session

Here is the full log obtained by executing the commands in Table 3, including standard prompts. The executed program is called `xllm-enterprise-v2.py`, with source code in section 5 and on GitHub. The input data source, also on GitHub, is a fully anonymized version of one part of a corporate corpus. Keyword pairs (at the beginning) come from the embeddings backend table. Entries flagged with a star (*) mark contextual pairs. Also,

- Some original word from the prompt, is on the right ('word' column in line 26).
- The related multitoken from the embeddings backend table, associated to the prompt word in question, is in the middle (the 'token' column). The user may try some of these tokens in a subsequent prompt.
- The 'F' column indicates if the pair is contextual or not.
- The 'pmi' column represents the pointwise mutual information (PMI), a measure of association between a word and a token.
- The 'N' column on the left shows the number of joint occurrences of ('token', 'word') in the corpus.

Below is the session log.

```
1  ______________________________________________________________________
2  Command menu:
3
4    -q          : print last non-command prompt
5    -x          : print sample queries
6    -p key value : set frontendParams[key] = value
7    -f          : use catch-all parameter set for debugging
8    -d          : use default parameter set
9    -v          : view parameter set
10   -a multitoken : add multitoken to 'ignore' list
11   -r multitoken : remove multitoken from 'ignore' list
12   -l          : view 'ignore' list
13   -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
14   -s          : print size of core backend tables
15   -c F1 F2 ... : show sections F1 F2 ... in output results
16
17 To view available sections for -c command, enter -v command.
18 To view available keys for -p command, enter -v command.
19 For -i command, choose IDs from list shown in prompt results.
20 For standard prompts, enter text not starting with '-' or digit.
21 ______________________________________________________________________
22
23 Query, command, or integer in [0, 7] for sample query: 6
24 query: MLTxQuest Data Assets Detailed Information page
25
26   N pmi F token [from embeddings]   word [from prompt]
27
28   1 1.00 * confidentiality|availability information|assets
29   1 1.00 * availability|organization information|assets
30   1 1.00 * confidentiality|availability|organization information|assets
31   1 1.00 availability|organization|information information|assets
32   1 1.00 * integrity|confidentiality|availability information|assets
33   1 1.00 organization|information   information|assets
34   1 1.00 organization|information|assets information|assets
```

```
1 1.00 * systems|managed          information|assets
1 1.00 * managed|mltxdat          information|assets
1 1.00 * systems|managed|mltxdat  information|assets
1 1.00 managed|mltxdat|csa        information|assets
1 1.00 platform|against           information|assets
1 1.00 * platform|against|threats information|assets
1 1.00 * threats|such             information|assets
1 1.00 * data|systems|managed     information|assets
1 1.00 csa|platform|against       information|assets
1 1.00 * against|threats          information|assets
1 1.00 * against|threats|such     information|assets
1 0.71 * navigating|data          page|mltxquest
1 0.71 * efficiently|navigating|data page|mltxquest
1 0.71 * navigating|data|assets   page|mltxquest
1 0.71 assets|page                page|mltxquest
1 0.71 data|assets|page           page|mltxquest
1 0.71 page|mltxquest|while       page|mltxquest
1 0.71 * while|facilitating       page|mltxquest
1 0.71 * while|facilitating|comprehensive page|mltxquest
1 0.71 assets|page|mltxquest      page|mltxquest
1 0.71 mltxquest|while            page|mltxquest
1 0.71 * mltxquest|while|facilitating page|mltxquest
1 0.71 * facilitating|comprehensive page|mltxquest
1 0.71 assets|deta                page|mltxquest
1 0.71 information|page           page|mltxquest
1 0.71 page|mltxquest|data        page|mltxquest
1 0.71 information|page|mltxquest page|mltxquest
1 0.71 mltxquest|data             page|mltxquest
1 0.71 * mltxquest|data|assets    page|mltxquest
1 0.71 * assets|users             page|mltxquest
1 0.71 * data|assets|users        page|mltxquest
1 0.71 mltxdat|csa|platform       information|assets
1 0.71 csa|platform               information|assets
2 0.67 * users|efficiently        data|assets
2 0.67 * efficiently|navigating   data|assets
2 0.67 * users|efficiently|navigating data|assets
2 0.67 * aid|users|efficiently    data|assets
2 0.50 * global|search            detailed
2 0.50 detailed|process           detailed
2 0.50 * process|migrating        detailed
2 0.50 * detailed|process|migrating detailed
2 0.50 * migrating|historical     detailed
2 0.50 * process|migrating|historical detailed
2 0.50 describes|detailed         detailed
2 0.50 describes|detailed|process detailed
2 0.47 * data|assets              page|mltxquest
2 0.47 * page|mltxquest           data|assets
1 0.45 mltxdat|csa                information|assets
2 0.41 data|migration             detailed
1 0.35 * guide|global             detailed
1 0.35 * guide|global|search      detailed
1 0.35 * information|search       detailed
1 0.35 * search|data              detailed
1 0.35 * information|search|data  detailed
1 0.35 * roles|raci               detailed
1 0.35 * responsibilities|policy  detailed
1 0.35 * zones|roles              detailed
1 0.35 zones|roles|responsibilities detailed
1 0.35 responsibilities|detailed  detailed
1 0.35 roles|responsibilities|detailed detailed
1 0.35 detailed|along             detailed
1 0.35 responsibilities|detailed|along detailed
1 0.35 * detailed|along|raci      detailed
1 0.35 * raci|matrix              detailed
1 0.35 * along|raci               detailed
1 0.35 * along|raci|matrix        detailed
1 0.35 mltxquest|business         detailed
1 0.35 metadata|templates         detailed
1 0.35 detailed|instructions      detailed
1 0.35 * instructions|completing  detailed
1 0.35 * detailed|instructions|completing detailed
1 0.35 * instructions|completing|templates detailed
1 0.35 filling|out                detailed
1 0.35 * out|templates            detailed
1 0.35 * filling|out|templates    detailed
1 0.35 * templates|users          detailed
```

```
1 0.35 * out|templates|users      detailed
1 0.35 * reviewed|metadata        detailed
1 0.35 * metadata|uploaded        detailed
1 0.35 * reviewed|metadata|uploaded detailed
1 0.35 * completing|templates     detailed
1 0.35 completed|reviewed         detailed
1 0.35 * completed|reviewed|metadata detailed
1 0.35 * offers|essential         detailed
1 0.35 * essential|visual         detailed
1 0.35 * offers|essential|visual  detailed
1 0.35 essential|visual|representations detailed
1 0.35 representations|detailed   detailed
1 0.35 visual|representations|detailed detailed
1 0.35 detailed|table             detailed
1 0.35 representations|detailed|table detailed
1 0.35 * table|showcasing         detailed
1 0.35 * detailed|table|showcasing detailed
1 0.35 * showcasing|project       detailed
1 0.35 * table|showcasing|project detailed
1 0.35 * set|defined              detailed
1 0.35 * set|defined|rules        detailed
1 0.35 rules|tab                  detailed
1 0.35 tab|ensures                detailed
1 0.35 rules|tab|ensures          detailed
1 0.35 ensures|consistent         detailed
1 0.35 tab|ensures|consistent     detailed
1 0.35 * ensures|consistent|standardized detailed
1 0.35 * standardized|approach    detailed
1 0.35 * defined|rules            detailed
1 0.35 defined|rules|tab          detailed
1 0.35 * consistent|standardized  detailed
1 0.35 * consistent|standardized|approach detailed
1 0.35 * batch|process|execution  detailed
1 0.35 databricks|metrics         detailed
1 0.35 * applications|performance detailed
1 0.35 * datadog|applications|performance detailed
1 0.35 * applications|performance|monitoring detailed
1 0.35 * process|execution        detailed
1 0.35 * execution|including      detailed
1 0.35 * process|execution|including detailed
1 0.35 including|databricks       detailed
1 0.35 execution|including|databricks detailed
1 0.35 including|databricks|metrics detailed
1 0.35 monitoring|apm             detailed
1 0.35 performance|monitoring|apm detailed
1 0.35 monitoring|apm|detailed    detailed
1 0.35 detailed|tracing           detailed
1 0.35 * tracing|request          detailed
1 0.35 * detailed|tracing|request detailed
1 0.35 * request|log              detailed
1 0.35 * tracing|request|log      detailed
1 0.35 apm|detailed               detailed
1 0.35 apm|detailed|tracing       detailed
1 0.33 * effectively|manage       data|assets
1 0.33 * regulations|effectively|manage data|assets
1 0.33 * manage|protect           data|assets
1 0.33 * effectively|manage|protect data|assets
1 0.33 manage|protect|data        data|assets
1 0.33 protect|data|assets        data|assets
1 0.33 * clarify|data             data|assets
1 0.33 * clarify|data|governance  data|assets
1 0.33 data|administration        data|assets
1 0.33 * administration|zones     data|assets
1 0.33 * data|administration|zones data|assets
1 0.33 * steward|policy           data|assets
1 0.33 governance|focused         data|assets
1 0.33 data|governance|focused    data|assets
1 0.33 focused|data               data|assets
1 0.33 governance|focused|data    data|assets
1 0.33 focused|data|administration data|assets
1 0.33 * data|steward             data|assets
1 0.33 * steward|governing        data|assets
1 0.33 * data|steward|governing   data|assets
1 0.33 governing|data             data|assets
1 0.33 steward|governing|data     data|assets
1 0.33 governing|data|assets      data|assets
```

```
1 0.33 data|assets|respective     data|assets
1 0.33 * respective|zones         data|assets
1 0.33 * zones|outlined           data|assets
1 0.33 * respective|zones|outlined data|assets
1 0.33 assets|respective          data|assets
1 0.33 * assets|respective|zones  data|assets
1 0.33 search|mltxquest           data|assets
1 0.33 global|search|mltxquest    data|assets
1 0.33 search|mltxquest|landing   data|assets
1 0.33 * landing|summary          data|assets
1 0.33 * mltxquest|landing|summary data|assets
1 0.33 * summary|page             data|assets
1 0.33 * landing|summary|page     data|assets
1 0.33 search|data|assets         data|assets
1 0.33 data|assets|filters        data|assets
1 0.33 * filters|better           data|assets
1 0.33 * filters|better|search    data|assets
1 0.33 assets|filters             data|assets
1 0.33 * assets|filters|better    data|assets
1 0.33 * better|search            data|assets
1 0.33 designed|aid               data|assets
1 0.33 designed|aid|users         data|assets
1 0.33 aid|users                  data|assets
1 0.33 * users|access|both        data|assets
1 0.33 * assets|users|access      data|assets
1 0.33 * access|both              data|assets
1 0.33 * both|technical|business  data|assets
1 0.33 business|metadata|data     data|assets
1 0.33 metadata|data              data|assets
1 0.33 metadata|data|assets       data|assets
1 0.33 data|assets|available      data|assets
1 0.33 assets|available           data|assets
1 0.33 * available|mltxdat        data|assets
1 0.33 * assets|available|mltxdat data|assets
1 0.33 * accountability|individuals data|assets
1 0.33 * clear|framework|managing data|assets
1 0.33 * fundamental|components|data data|assets
1 0.33 governance|defines         data|assets
1 0.33 data|governance|defines    data|assets
1 0.33 defines|roles              data|assets
1 0.33 governance|defines|roles   data|assets
1 0.33 defines|roles|responsibilities data|assets
1 0.33 * responsibilities|accountability data|assets
1 0.33 * roles|responsibilities|accountability data|assets
1 0.33 * responsibilities|accountability|individuals data|assets
1 0.33 * framework|managing       data|assets
1 0.33 * managing|stewarding      data|assets
1 0.33 * framework|managing|stewarding data|assets
1 0.33 stewarding|data            data|assets
1 0.33 managing|stewarding|data   data|assets
1 0.33 stewarding|data|assets     data|assets
1 0.33 data|assets|quality        data|assets
1 0.33 * security|proper          data|assets
1 0.33 * quality|security|proper  data|assets
1 0.33 assets|quality             data|assets
1 0.33 * assets|quality|security  data|assets
1 0.33 * badge|mltxquest          data|assets
1 0.33 * badge|mltxquest|awarded  data|assets
1 0.33 awarded|data               data|assets
1 0.33 * governance|metadata      data|assets
1 0.33 * governance|badge         data|assets
1 0.33 * mltxquest|awarded        data|assets
1 0.33 mltxquest|awarded|data     data|assets
1 0.33 data|assets|table          data|assets
1 0.33 * table|demonstrate        data|assets
1 0.33 * demonstrate|exceptional  data|assets
1 0.33 * table|demonstrate|exceptional data|assets
1 0.33 * table|meets              data|assets
1 0.33 meets|stringent            data|assets
1 0.33 table|meets|stringent      data|assets
1 0.33 stringent|criteria         data|assets
1 0.33 meets|stringent|criteria   data|assets
1 0.33 stringent|criteria|including data|assets
1 0.33 * including|robust         data|assets
1 0.33 * robust|technical         data|assets
1 0.33 * including|robust|technical data|assets
```

```
 1 0.33 * signifies|commitment     data|assets
 1 0.33 * signifies|commitment|high data|assets
 1 0.33 high|data                  data|assets
 1 0.33 high|data|governance       data|assets
 1 0.33 governance|standards       data|assets
 1 0.33 data|governance|standards  data|assets
 1 0.33 * standards|providing      data|assets
 1 0.33 * governance|standards|providing data|assets
 1 0.33 * providing|users          data|assets
 1 0.33 * standards|providing|users data|assets
 1 0.33 awarded|data|assets        data|assets
 1 0.33 assets|table               data|assets
 1 0.33 * assets|table|demonstrate data|assets
 1 0.33 * badge|table              data|assets
 1 0.33 * badge|table|meets        data|assets
 1 0.33 criteria|including         data|assets
 1 0.33 * criteria|including|robust data|assets
 1 0.33 * commitment|high          data|assets
 1 0.33 commitment|high|data       data|assets
 1 0.25 visual|representations     detailed
 1 0.25 * performance|monitoring   detailed
 1 0.24 protect|data               data|assets
 1 0.24 * business|metadata        data|assets
 1 0.24 * technical|business       data|assets
 1 0.24 * technical|business|metadata data|assets
 1 0.24 * quality|security         data|assets
 3 0.23 * data|governance          data|assets
 1 0.19 mltxquest|landing          data|assets
 1 0.19 * users|access             data|assets
 1 0.16 roles|responsibilities     detailed
 1 0.15 * components|data          data|assets
 1 0.15 * components|data|governance data|assets
 1 0.10 * data|products            detailed

N = occurrences of (token, word) in corpus. F = * if contextual pair.
If no result, try option '-p f'.

>>> RESULTS - SECTION: Category

  Category: 'Products' [6 entries]
  Linked to: page|mltxquest (2)
  Linked to: detailed (8)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)
  Linked to: page|mltxquest|data|assets (1)

  Category: 'Governance' [3 entries]
  Linked to: detailed (8)
  Linked to: information|assets (1)
  Linked to: data|assets (9)

  Category: 'BI Solution' [1 entries]
  Linked to: detailed (8)

  Category: 'Observability & Monitoring' [1 entries]
  Linked to: detailed (8)

  Category: 'One Platform' [1 entries]
  Linked to: detailed (8)


>>> RESULTS - SECTION: Tags

  Tags: MLTxQuest [6 entries]
  Linked to: page|mltxquest (2)
  Linked to: detailed (8)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)
  Linked to: page|mltxquest|data|assets (1)

  Tags: Guideline [3 entries]
  Linked to: page|mltxquest (2)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)
```

```

  Tags: Guidelines [5 entries]
  Linked to: page|mltxquest (2)
  Linked to: detailed (8)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: page|mltxquest|data|assets (1)

  Tags: example1 [2 entries]
  Linked to: detailed (8)
  Linked to: data|assets (9)

  Tags: example2 [2 entries]
  Linked to: detailed (8)
  Linked to: data|assets (9)

  Tags: governance [2 entries]
  Linked to: detailed (8)
  Linked to: data|assets (9)

  Tags: roles [1 entries]
  Linked to: detailed (8)

  Tags: raci [1 entries]
  Linked to: detailed (8)

  Tags: metadata [2 entries]
  Linked to: detailed (8)
  Linked to: data|assets (9)

  Tags: mltxquest [1 entries]
  Linked to: detailed (8)

  Tags: business [1 entries]
  Linked to: detailed (8)

  Tags: products [1 entries]
  Linked to: detailed (8)

  Tags: metrics [1 entries]
  Linked to: detailed (8)

  Tags: Historical data [1 entries]
  Linked to: detailed (8)

  Tags: Security [1 entries]
  Linked to: information|assets (1)

  Tags: privacy [1 entries]
  Linked to: data|assets (9)

  Tags: Steward [1 entries]
  Linked to: data|assets (9)

  Tags: policy [1 entries]
  Linked to: data|assets (9)

  Tags: owner [1 entries]
  Linked to: data|assets (9)

  Tags: badge [1 entries]
  Linked to: data|assets (9)


>>> RESULTS - SECTION: Titles

  Titles: 'MLTxQuest - Data Assets' [3 entries]
  Linked to: page|mltxquest (2)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)

  Titles: 'MLTxQuest-Data Asset Deta' [5 entries]
  Linked to: page|mltxquest (2)
  Linked to: detailed (8)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
```

```
  Linked to: page|mltxquest|data|assets (1)

  Titles: 'MLTxQuest - Global Search' [2 entries]
  Linked to: detailed (8)
  Linked to: data|assets (9)

  Titles: 'Roles and Responsibilities Policy' [1 entries]
  Linked to: detailed (8)

  Titles: 'Business Metadata Template' [1 entries]
  Linked to: detailed (8)

  Titles: '[METRICS] Data Products' [1 entries]
  Linked to: detailed (8)

  Titles: 'Exploration - Monitoring' [1 entries]
  Linked to: detailed (8)

  Titles: 'Historical data migration' [1 entries]
  Linked to: detailed (8)

  Titles: 'Data Security Policy ' [1 entries]
  Linked to: information|assets (1)

  Titles: 'Data Privacy Policy' [1 entries]
  Linked to: data|assets (9)

  Titles: 'Data Steward Policy' [1 entries]
  Linked to: data|assets (9)

  Titles: 'Data Owner Policy' [1 entries]
  Linked to: data|assets (9)

  Titles: 'MLTxQuest - Governance Badge' [1 entries]
  Linked to: data|assets (9)


>>> RESULTS - SECTION: ID

  ID: 91 [3 entries]
  Linked to: page|mltxquest (2)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)

  ID: 92 [5 entries]
  Linked to: page|mltxquest (2)
  Linked to: detailed (8)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: page|mltxquest|data|assets (1)

  ID: 90 [2 entries]
  Linked to: detailed (8)
  Agents: ('Example',)
  Linked to: data|assets (9)
  Agents: ('Example',)

  ID: 101 [1 entries]
  Linked to: detailed (8)
  Agents: ('Policy', 'Governance')

  ID: 107 [1 entries]
  Linked to: detailed (8)
  Agents: ('Template', 'Governance')

  ID: 139 [1 entries]
  Linked to: detailed (8)

  ID: 381 [1 entries]
  Linked to: detailed (8)

  ID: 511 [1 entries]
  Linked to: detailed (8)
  Agents: ('Data',)

  ID: 513 [1 entries]
```

```
  Linked to: detailed (8)
  Agents: ('Data',)

  ID: 223 [1 entries]
  Linked to: information|assets (1)
  Agents: ('Policy', 'Governance')

  ID: 42 [1 entries]
  Linked to: data|assets (9)
  Agents: ('Policy', 'Governance')

  ID: 48 [1 entries]
  Linked to: data|assets (9)
  Agents: ('Policy', 'Governance')

  ID: 199 [1 entries]
  Linked to: data|assets (9)
  Agents: ('Policy', 'Governance')

  ID: 259 [1 entries]
  Linked to: data|assets (9)
  Agents: ('Governance',)


>>> RESULTS - SECTION: Agents

  Agents: Example [2 entries]
  Linked to: detailed (8)
  Linked to: data|assets (9)

  Agents: Policy [3 entries]
  Linked to: detailed (8)
  Linked to: information|assets (1)
  Linked to: data|assets (9)

  Agents: Governance [3 entries]
  Linked to: detailed (8)
  Linked to: information|assets (1)
  Linked to: data|assets (9)

  Agents: Template [1 entries]
  Linked to: detailed (8)

  Agents: Data [1 entries]
  Linked to: detailed (8)


Above results based on words found in prompt, matched back to backend tables.
Numbers in parentheses are occurrences of word in corpus.

_____________________________________________________________________
>>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)
   empty unless labels 'ID' and 'Agents' are in 'show'.

('Data', 'detailed') --> (511, 513)
('Example', 'data|assets') --> (90,)
('Example', 'detailed') --> (90,)
('Governance', 'data|assets') --> (42, 48, 199, 259)
('Governance', 'detailed') --> (101, 107)
('Governance', 'information|assets') --> (223,)
('Policy', 'data|assets') --> (42, 48, 199)
('Policy', 'detailed') --> (101,)
('Policy', 'information|assets') --> (223,)
('Template', 'detailed') --> (107,)

  ID Size

 511  690
 513  692
  90  772
  42  948
  48  916
 199  980
 259 1153
 101  851
 107 1242
```

```
 223  978

______________________________________________________________________
Command menu:

  -q           : print last non-command prompt
  -x           : print sample queries
  -p key value : set frontendParams[key] = value
  -f           : use catch-all parameter set for debugging
  -d           : use default parameter set
  -v           : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l           : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s           : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
______________________________________________________________________

Query, command, or integer in [0, 7] for sample query: -i 107 259

--- Entity 107 ---

> Modified Date :
2024-07-02T12:51:31.993Z

> title_text :
Business Metadata Template

> description_text :
It outlines detailed instructions for completing the template accurately, covering various sections
    such as data dictionary, data source, sensitivity information, and roles. After filling out the
    template, users can interpret the entered data, ensuring clarity on sensitivity
    classifications, business details, and key roles. Once completed and reviewed, the metadata is
    uploaded to MLTxQuest, making it accessible through the MLTxQuest portal for all authorized
    users, thereby centralizing and simplifying access to critical information within the
    organization.

> tags_list_text :
metadata
mltxquest
business

> link_list_text :


> likes_list_text :
luiz.lagatosm@abc-mixa.com

> category_text :
Governance

--- Entity 259 ---

> Modified Date :
2024-06-27T11:36:39.594Z

> title_text :
MLTxQuest - Governance Badge

> description_text :
The Governance Badge in MLTxQuest is awarded to data assets (tables) that demonstrate exceptional
    metadata management and data quality. To earn this badge, tables must meet stringent criteria,
    including robust technical and business metadata descriptions, alongside maintaining a Fitness
    Index score above 90 consistently. This badge signifies a commitment to high data governance
    standards, providing users with confidence in data accuracy and transparency in its usage.

> tags_list_text :
badge
governance
metadata
```

```
> link_list_text :


> likes_list_text :
luiz.lagatosm@abc-mixa.com

> category_text :
Governance


 2 text entities found.

Completed task: -i 107 259

______________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
______________________________________________________________________

Query, command, or integer in [0, 7] for sample query: -a detailed


Completed task: -a detailed

______________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
______________________________________________________________________

Query, command, or integer in [0, 7] for sample query: -v

Key Description        Value

  0 embeddingKeyMinSize 1
  1 embeddingValuesMinSize 2
  2 min_pmi            0.0
  3 nABmin             1
  4 Customized_pmi     True
  5 ContextMultitokenMinSize 1
  6 minOutputListSize  1
  7 bypassIgnoreList   False
```

```
  8 ignoreList         ('data', 'detailed')
  9 maxTokenCount      100

Show sections:

   Embeddings True
   Category True
   Tags     True
   Titles   True
   Descr.   False
   Whole    False
   ID       True
   Agents   True

Completed task: -v

____________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
____________________________________________________________________

Query, command, or integer in [0, 7] for sample query: -p 6 2


Completed task: -p 6 2

____________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
____________________________________________________________________

Query, command, or integer in [0, 7] for sample query: -p 2 0.50


Completed task: -p 2 0.50

____________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
```

```
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
_____________________________________________________________________

Query, command, or integer in [0, 7] for sample query: 6
query: MLTxQuest Data Assets Detailed Information page

  N pmi F token [from embeddings]   word [from prompt]

  1 1.00 * confidentiality|availability information|assets
  1 1.00 * availability|organization information|assets
  1 1.00 * confidentiality|availability|organization information|assets
  1 1.00 availability|organization|information information|assets
  1 1.00 * integrity|confidentiality|availability information|assets
  1 1.00 organization|information   information|assets
  1 1.00 organization|information|assets information|assets
  1 1.00 * systems|managed          information|assets
  1 1.00 * managed|mltxdat          information|assets
  1 1.00 * systems|managed|mltxdat  information|assets
  1 1.00 managed|mltxdat|csa        information|assets
  1 1.00 platform|against           information|assets
  1 1.00 * platform|against|threats information|assets
  1 1.00 * threats|such             information|assets
  1 1.00 * data|systems|managed     information|assets
  1 1.00 csa|platform|against       information|assets
  1 1.00 * against|threats          information|assets
  1 1.00 * against|threats|such     information|assets
  1 0.71 * navigating|data          page|mltxquest
  1 0.71 * efficiently|navigating|data page|mltxquest
  1 0.71 * navigating|data|assets   page|mltxquest
  1 0.71 assets|page                page|mltxquest
  1 0.71 data|assets|page           page|mltxquest
  1 0.71 page|mltxquest|while       page|mltxquest
  1 0.71 * while|facilitating       page|mltxquest
  1 0.71 * while|facilitating|comprehensive page|mltxquest
  1 0.71 assets|page|mltxquest      page|mltxquest
  1 0.71 mltxquest|while            page|mltxquest
  1 0.71 * mltxquest|while|facilitating page|mltxquest
  1 0.71 * facilitating|comprehensive page|mltxquest
  1 0.71 assets|deta                page|mltxquest
  1 0.71 information|page           page|mltxquest
  1 0.71 page|mltxquest|data        page|mltxquest
  1 0.71 information|page|mltxquest page|mltxquest
  1 0.71 mltxquest|data             page|mltxquest
  1 0.71 * mltxquest|data|assets    page|mltxquest
  1 0.71 * assets|users             page|mltxquest
  1 0.71 * data|assets|users        page|mltxquest
  1 0.71 mltxdat|csa|platform       information|assets
  1 0.71 csa|platform               information|assets
  2 0.67 * users|efficiently        data|assets
  2 0.67 * efficiently|navigating   data|assets
  2 0.67 * users|efficiently|navigating data|assets
  2 0.67 * aid|users|efficiently    data|assets

N = occurrences of (token, word) in corpus. F = * if contextual pair.
If no result, try option '-p f'.

>>> RESULTS - SECTION: Category

   Category: 'Products' [5 entries]
   Linked to: page|mltxquest (2)
   Linked to: information|page|mltxquest|data (1)
   Linked to: data|assets (9)
   Linked to: data|assets|page|mltxquest (1)
```

```
  Linked to: page|mltxquest|data|assets (1)

  Category: 'Governance' [2 entries]
  Linked to: information|assets (1)
  Linked to: data|assets (9)


>>> RESULTS - SECTION: Tags

  Tags: MLTxQuest [5 entries]
  Linked to: page|mltxquest (2)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)
  Linked to: page|mltxquest|data|assets (1)

  Tags: Guideline [3 entries]
  Linked to: page|mltxquest (2)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)

  Tags: Guidelines [4 entries]
  Linked to: page|mltxquest (2)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: page|mltxquest|data|assets (1)


>>> RESULTS - SECTION: Titles

  Titles: 'MLTxQuest - Data Assets' [3 entries]
  Linked to: page|mltxquest (2)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)

  Titles: 'MLTxQuest-Data Asset Deta' [4 entries]
  Linked to: page|mltxquest (2)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: page|mltxquest|data|assets (1)


>>> RESULTS - SECTION: ID

  ID: 91 [3 entries]
  Linked to: page|mltxquest (2)
  Linked to: data|assets (9)
  Linked to: data|assets|page|mltxquest (1)

  ID: 92 [4 entries]
  Linked to: page|mltxquest (2)
  Linked to: information|page|mltxquest|data (1)
  Linked to: data|assets (9)
  Linked to: page|mltxquest|data|assets (1)


>>> RESULTS - SECTION: Agents

  Agents: Policy [2 entries]
  Linked to: information|assets (1)
  Linked to: data|assets (9)

  Agents: Governance [2 entries]
  Linked to: information|assets (1)
  Linked to: data|assets (9)


Above results based on words found in prompt, matched back to backend tables.
Numbers in parentheses are occurrences of word in corpus.

_____________________________________________________________________
>>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)
   empty unless labels 'ID' and 'Agents' are in 'show'.


 ID Size
```

```
______________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
______________________________________________________________________

Query, command, or integer in [0, 7] for sample query: -c Titles


Completed task: -c Titles

______________________________________________________________________
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
______________________________________________________________________

Query, command, or integer in [0, 7] for sample query: 6
query: MLTxQuest Data Assets Detailed Information page
>>> RESULTS - SECTION: Titles

   Titles: 'MLTxQuest - Data Assets' [3 entries]
   Linked to: page|mltxquest (2)
   Linked to: data|assets (9)
   Linked to: data|assets|page|mltxquest (1)

   Titles: 'MLTxQuest-Data Asset Deta' [4 entries]
   Linked to: page|mltxquest (2)
   Linked to: information|page|mltxquest|data (1)
   Linked to: data|assets (9)
   Linked to: page|mltxquest|data|assets (1)


Above results based on words found in prompt, matched back to backend tables.
Numbers in parentheses are occurrences of word in corpus.

______________________________________________________________________
>>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)
    empty unless labels 'ID' and 'Agents' are in 'show'.


  ID Size
```

```
Command menu:

  -q          : print last non-command prompt
  -x          : print sample queries
  -p key value : set frontendParams[key] = value
  -f          : use catch-all parameter set for debugging
  -d          : use default parameter set
  -v          : view parameter set
  -a multitoken : add multitoken to 'ignore' list
  -r multitoken : remove multitoken from 'ignore' list
  -l          : view 'ignore' list
  -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
  -s          : print size of core backend tables
  -c F1 F2 ... : show sections F1 F2 ... in output results

To view available sections for -c command, enter -v command.
To view available keys for -p command, enter -v command.
For -i command, choose IDs from list shown in prompt results.
For standard prompts, enter text not starting with '-' or digit.
```

```
Query, command, or integer in [0, 7] for sample query:
```

4 Conclusions

My custom sub-LLM designed from scratch does not rely on any Python library or API, and performs better than search tools available on the market, in terms of speed and results relevancy. It offers the user the ability to fine-tune parameters in real time, and can detect user intent to deliver appropriate output. The good performance comes from the quality of the well structured input sources, combined with smart crawling to retrieve the embedded knowledge graph and integrate it in the backend tables. Traditional tools rely mostly on tokens, embeddings, billions of parameters and frontend tricks such as prompt engineering to fix backend issues.

To the contrary, my approach focuses on building a solid backend foundational architecture from the ground up. Tokens and embeddings are not the most important components, by a long shot. Cosine similarity and dot products are replaced by pointwise mutual information. There is no neural network, no training, and a small number of explainable parameters, easy to fine-tune. When you think about it, the average human being has a vocabulary of 30,000 words. Even if you added variations and other pieces of information (typos, plural, grammatical tenses, product IDs, street names, and so on), you end up with a few millions at most, not trillions. Indeed, in expensive multi-billion systems, most tokens and weights are just noise: most are rarely fetched to serve an answer. This noise is a source of hallucinations.

Finally, gather a large number of user queries even before your start designing your architecture, and add prompt elements into your backend tables, as a source of data augmentation. It contributes to enhancing the quality of your system.

5 Python code

The Python code is also on GitHub, here, along with the crawled input source and backend tables. The enterprise corpus shared on GitHub – actually, a small portion corresponding to the AI section – is fully anonymized.

```python
#--- [1] Backend: functions

def update_hash(hash, key, count=1):

   if key in hash:
      hash[key] += count
   else:
      hash[key] = count
   return(hash)


def update_nestedHash(hash, key, value, count=1):

```

```
   # 'key' is a word here, value is tuple or single value
   if key in hash:
      local_hash = hash[key]
   else:
      local_hash = {}
   if type(value) is not tuple:
      value = (value,)
   for item in value:
      if item in local_hash:
         local_hash[item] += count
      else:
         local_hash[item] = count
   hash[key] = local_hash
   return(hash)


def get_value(key, hash):
   if key in hash:
      value = hash[key]
   else:
      value = ''
   return(value)


def update_tables(backendTables, word, hash_crawl, backendParams):

   category  = get_value('category', hash_crawl)
   tag_list  = get_value('tag_list', hash_crawl)
   title     = get_value('title', hash_crawl)
   description = get_value('description', hash_crawl) #
   meta      = get_value('meta', hash_crawl)
   ID        = get_value('ID', hash_crawl)
   agents    = get_value('agents', hash_crawl)
   full_content = get_value('full_content', hash_crawl) #

   extraWeights = backendParams['extraWeights']
   word = word.lower() # add stemming
   weight = 1.0
   if word in category:
      weight += extraWeights['category']
   if word in tag_list:
      weight += extraWeights['tag_list']
   if word in title:
      weight += extraWeights['title']
   if word in meta:
      weight += extraWeights['meta']

   update_hash(backendTables['dictionary'], word, weight)
   update_nestedHash(backendTables['hash_context1'], word, category)
   update_nestedHash(backendTables['hash_context2'], word, tag_list)
   update_nestedHash(backendTables['hash_context3'], word, title)
   update_nestedHash(backendTables['hash_context4'], word, description) # takes space, don't build?
   update_nestedHash(backendTables['hash_context5'], word, meta)
   update_nestedHash(backendTables['hash_ID'], word, ID)
   update_nestedHash(backendTables['hash_agents'], word, agents)
   for agent in agents:
       update_nestedHash(backendTables['ID_to_agents'], ID, agent)
   update_nestedHash(backendTables['full_content'], word, full_content) # takes space, don't nuild?
   update_nestedHash(backendTables['ID_to_content'], ID, full_content)

   return(backendTables)


def clean_list(value):

   # change string "['a', 'b', ...]" to ('a', 'b', ...)
   value = value.replace("[", "").replace("]","")
   aux = value.split("~")
   value_list = ()
   for val in aux:
     val = val.replace("'","").replace('"',"").lstrip()
     if val != '':
        value_list = (*value_list, val)
   return(value_list)
```

```
def get_key_value_pairs(entity):

   # extract key-value pairs from 'entity' (a string)
   entity = entity[1].replace("}",", '")
   flag = False
   entity2 = ""

   for idx in range(len(entity)):
      if entity[idx] == '[':
         flag = True
      elif entity[idx] == ']':
         flag = False
      if flag and entity[idx] == ",":
         entity2 += "~"
      else:
         entity2 += entity[idx]

   entity = entity2
   key_value_pairs = entity.split(", '")
   return(key_value_pairs)


def update_dict(backendTables, hash_crawl, backendParams):

   max_multitoken = backendParams['max_multitoken']
   maxDist = backendParams['maxDist']
   maxTerms = backendParams['maxTerms']

   category = get_value('category', hash_crawl)
   tag_list = get_value('tag_list', hash_crawl)
   title = get_value('title', hash_crawl)
   description = get_value('description', hash_crawl)
   meta = get_value('meta', hash_crawl)

   text = category + "." + str(tag_list) + "." + title + "." + description + "." + meta
   text = text.replace('/'," ").replace('(',' ').replace(')',' ').replace('?','')
   text = text.replace("'","").replace('"',"").replace('\\n','').replace('!','')
   text = text.replace("\\s",'').replace("\\t",'').replace(","," ").replace(":"," ")
   text = text.lower()
   sentence_separators = ('.',)
   for sep in sentence_separators:
      text = text.replace(sep, '_~')
   text = text.split('_~')

   hash_pairs = backendTables['hash_pairs']
   ctokens = backendTables['ctokens']
   KW_map = backendTables['KW_map']
   stopwords = backendTables['stopwords']
   hwords = {} # local word hash with word position, to update hash_pairs

   for sentence in text:

      words = sentence.split(" ")
      position = 0
      buffer = []

      for word in words:

         if word in KW_map:
            word = KW_map[word]

         if word not in stopwords:
            # word is single token
            buffer.append(word)
            key = (word, position)
            update_hash(hwords, key) # for word correlation table (hash_pairs)
            update_tables(backendTables, word, hash_crawl, backendParams)

            for k in range(1, max_multitoken):
               if position > k:
                  # word is now multi-token with k+1 tokens
                  word = buffer[position-k] + "~" + word
                  key = (word, position)
                  update_hash(hwords, key) # for word correlation table (hash_pairs)
                  update_tables(backendTables, word, hash_crawl, backendParams)

```

```
            position +=1

   for keyA in hwords:
      for keyB in hwords:

         wordA = keyA[0]
         positionA = keyA[1]
         n_termsA = len(wordA.split("~"))

         wordB = keyB[0]
         positionB = keyB[1]
         n_termsB = len(wordB.split("~"))

         key = (wordA, wordB)
         n_termsAB = max(n_termsA, n_termsB)
         distanceAB = abs(positionA - positionB)

         if wordA < wordB and distanceAB <= maxDist and n_termsAB <= maxTerms:
             hash_pairs = update_hash(hash_pairs, key)
             if distanceAB > 1:
                ctokens = update_hash(ctokens, key)

   return(backendTables)


#--- [2] Backend: main (create backend tables based on crawled corpus)

tableNames = (
 'dictionary', # multitokens (key = multitoken)
 'hash_pairs', # multitoken associations (key = pairs of multitokens)
 'ctokens',    # not adjacent pairs in hash_pairs (key = pairs of multitokens)
 'hash_context1', # categories (key = multitoken)
 'hash_context2', # tags (key = multitoken)
 'hash_context3', # titles (key = multitoken)
 'hash_context4', # descriptions (key = multitoken)
 'hash_context5', # meta (key = multitoken)
 'hash_ID',    # text entity ID table (key = multitoken, value is list of IDs)
 'hash_agents', # agents (key = multitoken)
 'full_content', # full content (key = multitoken)
 'ID_to_content', # full content attached to text entity ID (key = text entity ID)
 'ID_to_agents', # map text entity ID to agents list (key = text entity ID)
 'ID_size',    # content size (key = text entity ID)
 'KW_map',     # for singularization, map kw to single-token dictionary entry
 'stopwords',  # stopword list
)

backendTables = {}
for name in tableNames:
   backendTables[name] = {}

stopwords = ('', '-', 'in', 'the', 'and', 'to', 'of', 'a', 'this', 'for', 'is', 'with', 'from',
          'as', 'on', 'an', 'that', 'it', 'are', 'within', 'will', 'by', 'or', 'its', 'can',
          'your', 'be','about', 'used', 'our', 'their', 'you', 'into', 'using', 'these',
          'which', 'we', 'how', 'see', 'below', 'all', 'use', 'across', 'provide', 'provides',
          'aims', 'one', '&', 'ensuring', 'crucial', 'at', 'various', 'through', 'find', 'ensure',
          'more', 'another', 'but', 'should', 'considered', 'provided', 'must', 'whether',
          'located', 'where', 'begins', 'any')
backendTables['stopwords'] = stopwords

# agent_map works, but hash structure should be improved
# key is word, value is agent (many-to-one). Allow for many-to-many
agent_map = {
          'template':'Template',
          'policy':'Policy',
          'governance':'Governance',
          'documentation':'Documentation',
          'best practice':'Best Practices',
          'bestpractice':'Best Practices',
          'standard':'Standards',
          'naming':'Naming',
          'glossary':'Glossary',
          'historical data':'Data',
          'overview':'Overview',
          'training':'Training',
          'genai':'GenAI',
          'gen ai':'GenAI',
```

```
            'example':'Example',
            'example1':'Example',
            'example2':'Example',
           }

KW_map = {}
try:
   IN = open("KW_map.txt","r")
except:
   print("KW_map.txt not found on first run: working with empty KW_map.")
   print("KW_map.txt will be created after exiting if save = True.")
else:
   content = IN.read()
   pairs = content.split('\n')
   for pair in pairs:
      pair = pair.split('\t')
      key = pair[0]
      if len(pair) > 1:
         KW_map[key] = pair[1]
   IN.close()
backendTables['KW_map'] = KW_map

backendParams = {
   'max_multitoken': 4, # max. consecutive terms per multi-token for inclusion in dictionary
   'maxDist' : 3,  # max. position delta between 2 multitokens to link them in hash_pairs
   'maxTerms': 3,  # maxTerms must be <= max_multitoken
   'extraWeights' : # deafault weight is 1
     {
       'description': 0.0,
       'category': 0.3,
       'tag_list': 0.4,
       'title':   0.2,
       'meta':    0.1
     }
}


local = True # first time run, set to False
if local:
   # get repository from local file
   IN = open("repository.txt","r")
   data = IN.read()
   IN.close()
else:
   # get anonymized repository from GitHub url
   import requests
   url = "https://mltblog.com/3y8MXq5"
   response = requests.get(url)
   data = response.text

entities = data.split("\n")
ID_size = backendTables['ID_size']

# to avoid duplicate entities (takes space, better to remove them in the corpus)
entity_list = ()

for entity_raw in entities:

   entity = entity_raw.split("~~")
   agent_list = ()

   if len(entity) > 1 and entity[1] not in entity_list:

      entity_list = (*entity_list, entity[1])
      entity_ID = int(entity[0])
      entity = entity[1].split("{")
      hash_crawl = {}
      hash_crawl['ID'] = entity_ID
      ID_size[entity_ID] = len(entity[1])
      hash_crawl['full_content'] = entity_raw # do not build to save space

      key_value_pairs = get_key_value_pairs(entity)

      for pair in key_value_pairs:

         if ": " in pair:
```

```
            key, value = pair.split(": ", 1)
            key = key.replace("'","")
            if key == 'category_text':
               hash_crawl['category'] = value
            elif key == 'tags_list_text':
               hash_crawl['tag_list'] = clean_list(value)
            elif key == 'title_text':
               hash_crawl['title'] = value
            elif key == 'description_text':
               hash_crawl['description'] = value # do not build to save space
            elif key == 'tower_option_tower':
               hash_crawl['meta'] = value
            if key in ('category_text','tags_list_text','title_text'):
               for word in agent_map:
                  if word in value.lower():
                     agent = agent_map[word]
                     if agent not in agent_list:
                        agent_list =(*agent_list, agent)

      hash_crawl['agents'] = agent_list
      update_dict(backendTables, hash_crawl, backendParams)

# [2.1] Create embeddings

embeddings = {} # multitoken embeddings based on hash_pairs

hash_pairs = backendTables['hash_pairs']
dictionary = backendTables['dictionary']

for key in hash_pairs:
   wordA = key[0]
   wordB = key[1]
   nA = dictionary[wordA]
   nB = dictionary[wordB]
   nAB = hash_pairs[key]
   pmi = nAB/(nA*nB)**0.5 # try: nAB/(nA + nB - nAB)
   # if nA + nB <= nAB:
   #   print(key, nA, nB, nAB)
   update_nestedHash(embeddings, wordA, wordB, pmi)
   update_nestedHash(embeddings, wordB, wordA, pmi)


# [2.2] Create sorted n-grams

sorted_ngrams = {} # to match ngram prompts with embeddings entries

for word in dictionary:
   tokens = word.split('~')
   tokens.sort()
   sorted_ngram = tokens[0]
   for token in tokens[1:len(tokens)]:
      sorted_ngram += "~" + token
   update_nestedHash(sorted_ngrams, sorted_ngram, word)

# print top multitokens: useful to build agents, along with sample prompts
# for key in dictionary:
#    if dictionary[key] > 20:
#       print(key, dictionary[key])


#--- [3] Frontend: functions

# [3.1] custom pmi

def custom_pmi(word, token, backendTables):

   dictionary = backendTables['dictionary']
   hash_pairs = backendTables['hash_pairs']

   nAB = 0
   pmi = 0.00
   keyAB = (word, token)
   if word > token:
      keyAB = (token, word)
   if keyAB in hash_pairs:
      nAB = hash_pairs[keyAB]
```

```
        nA = dictionary[word]
        nB = dictionary[token]
        pmi = nAB/(nA*nB)**0.5
    return(pmi)

# [3.2] update frontend params

def cprint(ID, entity):
    # print text_entity (a JSON text string) nicely

    print("--- Entity %d ---\n" %(ID))
    keys = (
            'title_text',
            'description_text',
            'tags_list_text',
            'category_text',
            'likes_list_text',
            'link_list_text',
            'Modified Date',
           )
    entity = str(entity).split("~~")
    entity = entity[1].split("{")
    key_value_pairs = get_key_value_pairs(entity)

    for pair in key_value_pairs:
        if ": " in pair:
            key, value = pair.split(": ", 1)
            key = key.replace("'","")
            if key in keys:
                print("> ",key,":")
                value = value.replace("'",'').split("~")
                for item in value:
                    item = item.lstrip().replace("[","").replace("]","")
                    print(item)
                print()
    return()

def update_params(option, saved_query, sample_queries, frontendParams, backendTables):

    arr = []
    ID_to_content = backendTables['ID_to_content']
    for param in frontendParams:
        arr.append(param)
    task = option
    print()

    if option == '-l':
        print("Multitoken ignore list:\n", frontendParams['ignoreList'])

    elif option == '-v':
        print("%3s %s %s\n" %('Key', 'Description'.ljust(25), 'Value'))
        for key in range(len(arr)):
            param = arr[key]
            value = frontendParams[param]
            if param != 'show':
                print("%3d %s %s" %(key, param.ljust(25), value))
            else:
                print("\nShow sections:\n")
                for section in value:
                    print("  %s %s" %(section.ljust(10),value[section]))

    elif option == '-f':
        # use parameter set to show as much as possible
        for param in frontendParams:
            if param == 'ignoreList':
                frontendParams[param] = ()
            elif param == 'Customized_pmi':
                # use customized pmi
                frontendParams[param] = True
            elif param == 'show':
                showHash = frontendParams[param]
                for section in showHash:
                    # show all sections in output results
                    showHash[section] = True
            elif param == 'maxTokenCount':
                frontendParams[param] = 999999999
```

```
         else:
            frontendParams[param] = 0

   elif option == '-d':
      frontendParams = default_frontendParams()

   elif '-p' in option:
      option = option.split(' ')
      if len(option) == 3:
         paramID = int(option[1])
         if paramID < len(arr):
            param = arr[paramID]
            value = option[2]
            if value == 'True':
               value = True
            elif value == 'False':
               value = False
            else:
               value = float(option[2])
            frontendParams[param] = value
         else:
            print("Error 101: key outside range")
      else:
         print("Error 102: wrong number of arguments")

   elif '-a' in option:
      option = option.split(' ')
      if len(option) == 2:
         ignore = frontendParams['ignoreList']
         ignore =(*ignore, option[1])
         frontendParams['ignoreList'] = ignore
      else:
         print("Error 103: wrong number of arguments")

   elif '-r' in option:
      option = option.split(' ')
      if len(option) == 2:
         ignore2 = ()
         ignore = frontendParams['ignoreList']
         for item in ignore:
            if item != option[1]:
               ignore2 = (*ignore2, item)
         frontendParams['ignoreList'] = ignore2
      else:
         print("Error 104: wrong number of arguments")

   elif '-i' in option:
      option = option.split(' ')
      nIDs = 0
      for ID in option:
         if ID.isdigit():
            ID = int(ID)
            # print content of text entity ID
            if ID in ID_to_content:
               cprint(ID, ID_to_content[ID])
               nIDs += 1
      print("\n %d text entities found." % (nIDs))

   elif option == '-s':
      print("Size of some backend tables:")
      print(" dictionary:", len(backendTables['dictionary']))
      print(" pairs    :", len(backendTables['hash_pairs']))
      print(" ctokens :", len(backendTables['ctokens']))
      print(" ID_size :", len(backendTables['ID_size']))

   elif '-c' in option:
      show = frontendParams['show']
      option = option.split(' ')
      for section in show:
         if section in option or '*' in option:
            show[section] = True
         else:
            show[section] = False

   elif option == '-q':
      print("Saved query:", saved_query)
```

```
    elif option == '-x':
        print("Index Query\n")
        for k in range(len(sample_queries)):
            print(" %3d %s" %(k, sample_queries[k]))

    print("\nCompleted task: %s" %(task))
    return(frontendParams)

# [3.3] retrieve info and print results

def print_results(q_dictionary, q_embeddings, backendTables, frontendParams):

    dictionary = backendTables['dictionary']
    hash_pairs = backendTables['hash_pairs']
    ctokens   = backendTables['ctokens']
    ID_to_agents = backendTables['ID_to_agents']
    ID_size   = backendTables['ID_size']
    show      = frontendParams['show']

    if frontendParams['bypassIgnoreList'] == True:
        # bypass 'ignore' list
        ignore = ()
    else:
        # ignore multitokens specified in 'ignoreList'
        ignore = frontendParams['ignoreList']

    if show['Embeddings']:
        # show results from embedding table

        local_hash = {} # used to not show same token 2x (linked to 2 different words)
        q_embeddings = dict(sorted(q_embeddings.items(),key=lambda item: item[1],reverse=True))
        print()
        print("%3s %s %1s %s %s"
            %('N','pmi'.ljust(4),'F','token [from embeddings]'.ljust(35),
              'word [from prompt]'.ljust(35)))
        print()

        for key in q_embeddings:

            word = key[0]
            token = key[1]
            pmi = q_embeddings[key]
            ntk1 = len(word.split('~'))
            ntk2 = len(token.split('~'))
            flag = " "
            nAB = 0
            keyAB = (word, token)

            if word > token:
                keyAB = (token, word)
            if keyAB in hash_pairs:
                nAB = hash_pairs[keyAB]
            if keyAB in ctokens:
                flag = '*'
            if ( ntk1 >= frontendParams['embeddingKeyMinSize'] and
                 ntk2 >= frontendParams['embeddingValuesMinSize'] and
                 pmi >= frontendParams['min_pmi'] and
                 nAB >= frontendParams['nABmin'] and
                 token not in local_hash and word not in ignore
                ):
                print("%3d %4.2f %1s %s %s"
                     %(nAB,pmi,flag,token.ljust(35),word.ljust(35)))
                local_hash[token] = 1 # token marked as displayed, won't be shown again

        print()
        print("N = occurrences of (token, word) in corpus. F = * if contextual pair.")
        print("If no result, try option '-p f'.")
        print()

    sectionLabels = {
       # map section label to corresponding backend table name
       'Dict'  :'dictionary',
       'Pairs':'hash_pairs',
       'Category':'hash_context1',
       'Tags'  :'hash_context2',
```

```
     'Titles':'hash_context3',
     'Descr.':'hash_context4',
     'Meta'  :'hash_context5',
     'ID'    :'hash_ID',
     'Agents':'hash_agents',
     'Whole' :'full_content',
   }
   local_hash = {}
   agentAndWord_to_IDs = {}

   for label in show:
      # labels: 'Category','Tags','Titles','Descr.','ID','Whole','Agents','Embeddings'

      if show[label] and label in sectionLabels:
         # show results for section corresponding to label

         tableName = sectionLabels[label]
         table = backendTables[tableName]
         local_hash = {}
         print(">>> RESULTS - SECTION: %s\n" % (label))

         for word in q_dictionary:

            ntk3 = len(word.split('~'))
            if word not in ignore and ntk3 >= frontendParams['ContextMultitokenMinSize']:
               content = table[word] # content is a hash
               count = int(dictionary[word])
               for item in content:
                  update_nestedHash(local_hash, item, word, count)

         for item in local_hash:

            hash2 = local_hash[item]
            if len(hash2) >= frontendParams['minOutputListSize']:
               print(" %s: %s [%d entries]" % (label, item, len(hash2)))
               for key in hash2:
                  print(" Linked to: %s (%s)" %(key, hash2[key]))
                  if label == 'ID' and item in ID_to_agents:
                     # here item is a text entity ID
                     LocalAgentHash = ID_to_agents[item]
                     local_ID_list = ()
                     for ID in LocalAgentHash:
                        local_ID_list = (*local_ID_list, ID)
                     print(" Agents:", local_ID_list)
                     for agent in local_ID_list:
                        key3 = (agent, key) # key is a multitoken
                        update_nestedHash(agentAndWord_to_IDs, key3, item)

               print()
         print()

   print("Above results based on words found in prompt, matched back to backend tables.")
   print("Numbers in parentheses are occurrences of word in corpus.\n")

   print("--------------------------------------")
   print(">>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)")
   print(" empty unless labels 'ID' and 'Agents' are in 'show'.\n")
   hash_size = {}
   for key in sorted(agentAndWord_to_IDs):
      ID_list = ()
      for ID in agentAndWord_to_IDs[key]:
         ID_list = (*ID_list, ID)
         hash_size[ID] = ID_size[ID]
      print(key,"-->",ID_list)
   print("\n ID Size\n")
   for ID in hash_size:
      print("%4d %5d" %(ID, hash_size[ID]))

   return()


#--- [4] Frontend: main (process prompt)

# [4.1] Set default parameters

def default_frontendParams():
```

```
   frontendParams = {
                   'embeddingKeyMinSize': 1, # try 2
                   'embeddingValuesMinSize': 2,
                   'min_pmi': 0.00,
                   'nABmin': 1,
                   'Customized_pmi': True,
                   'ContextMultitokenMinSize': 1, # try 2
                   'minOutputListSize': 1,
                   'bypassIgnoreList': False,
                   'ignoreList': ('data',),
                   'maxTokenCount': 100, # ignore generic tokens if large enough
                   'show': {
                           # names of sections to display in output results
                           'Embeddings': True,
                           'Category' : True,
                           'Tags'    : True,
                           'Titles' : True,
                           'Descr.' : False, # do not built to save space
                           'Whole'  : False, # do not build to save space
                           'ID'     : True,
                           'Agents' : True,
                          }
                  }
   return(frontendParams)

# [4.2] Purge function

def distill_frontendTables(q_dictionary, q_embeddings, frontendParams):
   # purge q_dictionary then q_embeddings (frontend tables)

   maxTokenCount = frontendParams['maxTokenCount']
   local_hash = {}
   for key in q_dictionary:
      if q_dictionary[key] > maxTokenCount:
         local_hash[key] = 1
   for keyA in q_dictionary:
      for keyB in q_dictionary:
         nA = q_dictionary[keyA]
         nB = q_dictionary[keyB]
         if keyA != keyB:
            if (keyA in keyB and nA == nB) or (keyA in keyB.split('~')):
               local_hash[keyA] = 1
   for key in local_hash:
      del q_dictionary[key]

   local_hash = {}
   for key in q_embeddings:
      if key[0] not in q_dictionary:
         local_hash[key] = 1
   for key in local_hash:
      del q_embeddings[key]

   return(q_dictionary, q_embeddings)

# [4.3] Main

print("\n") #
input_ = " "
saved_query = ""
get_bin = lambda x, n: format(x, 'b').zfill(n)
frontendParams = default_frontendParams()
sample_queries = (
               'parameterized datasets map tables sql server',
               'data load templates importing data database data warehouse',
               'pipeline extract data eventhub files',
               'blob storage single parquet file adls gen2',
               'eventhub files blob storage single parquet',
               'parquet blob eventhub more files less storage single table',
               'MLTxQuest Data Assets Detailed Information page',
               'table asset',
             )

while len(input_) > 0:

   print()
```

```
    print("------------------------------------")
    print("Command menu:\n")
    print(" -q         : print last non-command prompt")
    print(" -x         : print sample queries")
    print(" -p key value : set frontendParams[key] = value")
    print(" -f         : use catch-all parameter set for debugging")
    print(" -d         : use default parameter set")
    print(" -v         : view parameter set")
    print(" -a multitoken : add multitoken to 'ignore' list")
    print(" -r multitoken : remove multitoken from 'ignore' list")
    print(" -l         : view 'ignore' list")
    print(" -i ID1 ID2 ... : print content of text entities ID1 ID2 ...")
    print(" -s         : print size of core backend tables")
    print(" -c F1 F2 ... : show sections F1 F2 ... in output results")
    print("\nTo view available sections for -c command, enter -v command.")
    print("To view available keys for -p command, enter -v command.")
    print("For -i command, choose IDs from list shown in prompt results.")
    print("For standard prompts, enter text not starting with '-' or digit.")
    print("------------------------------------\n")

    input_ = input("Query, command, or integer in [0, %d] for sample query: "
                %(len(sample_queries)-1))
    flag = True # False --> query to change params, True --> real query
    if input_ != "" and input_[0] == '-':
        # query to modify options
        frontendParams = update_params(input_, saved_query,
                                       sample_queries, frontendParams,
                                       backendTables)
        query = ""
        flag = False
    elif input_.isdigit():
       # actual query (prompt)
       if int(input_) < len(sample_queries):
         query = sample_queries[int(input_)]
         saved_query = query
         print("query:",query)
       else:
         print("Value must be <", len(sample_queries))
         query = ""
    else:
       # actual query (prompt)
       query = input_
       saved_query = query

    query = query.split(' ')
    new_query = []
    for k in range(len(query)):
       token = query[k].lower()
       if token in KW_map:
          token = KW_map[token]
       if token in dictionary:
          new_query.append(token)
    query = new_query.copy()
    query.sort()
    q_embeddings = {}
    q_dictionary = {}

    for k in range(1, 2**len(query)):

       binary = get_bin(k, len(query))
       sorted_word = ""
       for k in range(0, len(binary)):
          if binary[k] == '1':
             if sorted_word == "":
                sorted_word = query[k]
             else:
                sorted_word += "~" + query[k]

       if sorted_word in sorted_ngrams:
          ngrams = sorted_ngrams[sorted_word]
          for word in ngrams:
             if word in dictionary:
                q_dictionary[word] = dictionary[word]
                if word in embeddings:
                   embedding = embeddings[word]
                   for token in embedding:
```

```
                     if not frontendParams['Customized_pmi']:
                        pmi = embedding[token]
                     else:
                        # customized pmi
                        pmi = custom_pmi(word, token, backendTables)
                     q_embeddings[(word, token)] = pmi

   # if len(query) == 1:
   #    # single-token query
   #    frontendParams['embeddingKeyMinSize'] = 1
   #    frontendParams['ContextMultitokenMinSize'] = 1

   distill_frontendTables(q_dictionary,q_embeddings,frontendParams)

   if len(input_) > 0 and flag:
      print_results(q_dictionary, q_embeddings, backendTables, frontendParams)


#--- [5] Save backend tables

def create_KW_map(dictionary):
   # singularization
   # map key to KW_map[key], here key is a single token
   # need to map unseen prompt tokens to related dictionary entries
   #   example: ANOVA -> analysis~variance, ...

   OUT = open("KW_map.txt","w")

   for key in dictionary:
      if key.count('~') == 0:
         j = len(key)
         keyB = key[0:j-1]
         if keyB in dictionary and key[j-1] == 's':
            if dictionary[key] > dictionary[keyB]:
               OUT.write(keyB + "\t" + key + "\n")
            else:
               OUT.write(key + "\t" + keyB + "\n")
   OUT.close()
   return()


save = True
if save:
   create_KW_map(dictionary)
   for tableName in backendTables:
      table = backendTables[tableName]
      OUT = open('backend_' + tableName + '.txt', "w")
      OUT.write(str(table))
      OUT.close()

   OUT = open('backend_embeddings.txt', "w")
   OUT.write(str(embeddings))
   OUT.close()

   OUT = open('backend_sorted_ngrams.txt', "w")
   OUT.write(str(sorted_ngrams))
   OUT.close()
```

6 Appendix: 10 features to dramatically improve performance

Many of these features are ground-breaking innovations that make LLMs much faster and not prone to hallucinations. They reduce the cost, latency, and amount of computer resources (GPU, training) by several orders of magnitude. Some of them improve security, making your LLM more attractive to corporate clients. For a larger list, see here.

6.1 Fast search

In order to match prompt components (say, embeddings) to the corresponding entities in the backend tables based on the corpus, you need good search technology. In general, you won't find an exact match. The solution consists in using approximate nearest neighbor search (ANN), together with smart encoding of embedding vectors. See how it works, here. Then, use a caching mechanism to handle common prompts, to further speed up the processing in real time.

6.2 Sparse databases

While vector and graph databases are popular in this context, they may not be the best solution. If you have two million tokens, you may have as many as one trillion pairs of tokens. In practice, most tokens are connected to a small number of related tokens, typically less than 1000. Thus, the network or graph structure is very sparse, with less than a billion active connections. This is a far cry from a trillion! Hash tables are very good at handling this type of structure.

In my case, I use nested hash tables, a format similar to JSON, that is, similar to the way the input source (HTML pages) is typically encoded. A nested hash is a key-value table, where the value is itself a key-value table. The key in the root hash is typically a word, possibly consisting of multiple tokens. The keys in the child hash may be categories, agents, or URLs associated to the parent key, while values are weights indicating the association strength between a category and the parent key.

6.3 Contextual tokens

In standard LLMs, tokens are tiny elements of text, part of a word. In my multi-LLM system, they are full words and even combination of multiple words. This is also the case in other architectures, such as LLama. They are referred to as multi-tokens. When it consists of non-adjacent words found in in a same text entity (paragraph and so on), I call them contextual tokens. Likewise, pairs of tokens consisting of non-adjacent tokens are called contextual pairs. When dealing with contextual pairs and tokens, you need to be careful to avoid generating a very large number of mostly irrelevant combinations. Otherwise, you face token implosion.

Note that a word such as “San Francisco” is a single token. It may exist along with other single tokens such as “San” and “Francisco”.

6.4 Adaptive loss function

The goal of many deep neural networks (DNN) is to minimize a loss function, usually via stochastic gradient descent. This is also true for LLM systems based on transformers. The loss function is a proxy to the evaluation metric that measures the quality of your output. In supervised learning LLMs (for instance, those performing supervised classification), you may use the evaluation metric as the loss function, to get better results. One of the best evaluation metrics is the full multivariate Kolmogorov-Smirnov distance (KS), see here, with Python library here.

But it is extremely hard to design an algorithm that makes billions of atomic changes to KS extremely fast, a requirement in all DNNs as it happens each time you update a weight. A workaround is to use an adaptive loss function that slowly converges to the KS distance over many epochs. I did not succeed at that, but I was able to build one that converges to the multivariate Hellinger distance, the discrete alternative that is asymptotically equivalent to the continuous KS.

6.5 Contextual tables

In most LLMs, the core table is the embeddings. Not in our systems: in addition to embeddings, we have category, tags, related items and various contextual backend tables. They play a more critical role than the embeddings. It is more efficient to have them as backend tables, built during smart crawling, as opposed to reconstructed post-creation as frontend elements.

6.6 Smart crawling

Libraries such as BeautifulSoup allow you to easily crawl and parse content such as JSON entities. However, they may not be useful to retrieve the embedded structure present in any good repository. The purpose of smart crawling is to extract structure elements (categories and so on) while crawling, to add them to your contextual backend tables. It requires just a few lines of ad-hoc Python code depending in your input source, and the result is dramatic. You end up with a well-structured system from the ground up, eliminating the need for prompt engineering.

6.7 LLM router

Good input sources usually have their own taxonomy, with categories and multiple levels of subcategories, sometimes with subcategories having multiple parent categories. You can replicate the same structure in your LLM, having multiple sub-LLMs, one per top category. It is possible to cover the entire human knowledge with 2000 sub-LLMs, each with less than 200,000 multi-tokens. The benefit is much faster processing and more relevant results served to the user.

To achieve this, you need an LLM router. It identifies prompt elements and retrieve the relevant information in the most appropriate sub-LLMs. Each one hast its set of backend tables, hyperparameters, stopword list, and so on. There may be overlap between different sub-LLMs. Fine-tuning can be done locally, initially for each sub-LLM separately, or globally. You may also allow the user to choose a sub-LLM, by having a sub-LLM prompt box, in addition to the standard agent and query prompt boxes.

6.8 From one trillion parameters down to two

By parameter, here I mean the weight between two connected neurons in a deep neural network. How can you possibly replace one trillion parameters by less than 5, and yet get better results, faster? The idea is to use parametric weights. In this case, you update the many weights with a simple formula relying on a handful of explainable parameters, as opposed to neural network activation functions updating (over time) billions of Blackbox parameters — the weights themselves — over and over. I illustrate this in Figure 13, featuring material from my coursebook, available here.

- The pageview function is denoted as pv. At the basic level, $\text{pv}(A)$ is the pageview number of article A, based on its title and categorization. It must be normalized, taking the logarithm: see lines 122–123 in the code. Then, the most recent articles have a lower pv because they have not accumulated much traffic yet. To correct for this, see lines 127–136 in the code. From now on, pv refers to normalized pageview counts also adjusted for time. The pageview for a multi-token t is then defined as

$$\text{pv}(t) = \frac{1}{|S(t)|} \cdot \sum_{A \in S(t)} \text{pv}(A), \tag{8.2}$$

  where $S(t)$ is the set of all article titles containing t, and $|\cdot|$ is the function that counts the number of elements in a set. Sometimes, two different tokens t_1, t_2 have $S(t_1) = S(t_2)$. In this case, to reduce the number of tokens, I only keep the longest one. This is done in lines 193–206 in the code.

- Likewise, you can define $\text{pv}(C)$, the pageview count attached to a category C, by averaging pv's over all articles assigned to that category. Finally, $T(A)$ denotes the set of multi-tokens attached to an article A.

With the notations and terminology introduced so far, it is very easy to explain how to predict the pageview count $\text{pv}_0(A)$ for an article A inside or outside the training set. The formula is

$$\text{pv}_0(A) = \frac{1}{W_A} \cdot \sum_{t \in T(A)} w_t \cdot \text{pv}(t), \tag{8.3}$$

with:

$$W_A = \sum_{t \in T(A)} w_t, \quad w_t = 0 \text{ if } |S(t)| \leq \alpha, \quad w_t = \frac{1}{|S(t)|^\beta} \text{ if } |S(t)| > \alpha.$$

Here $\alpha, \beta > 0$ are parameters. I use $\alpha = 1$ and $\beta = 2$. The algorithm puts more weights on rare tokens, but a large value of β or a small value of α leads to overfitting. Also, I use the notation pv_0 for an estimated value or

169

Figure 13: LLM for classification, with only 2 parameters

6.9 Agentic LLMs

An agent detects the intent of a user within a prompt and helps deliver results that meet the intent in question. For instance, a user may be looking for definitions, case studies, sample code, solution to a problem, examples, datasets, images, or PDFs related to a specific topic, or links and references. The task of the agent is to automatically detect the intent and guide the search accordingly. Alternatively, the LLM may feature two prompt boxes: one for the standard query, and one to allow the user to choose an agent within a pre-built list.

Either way, you need a mechanism to retrieve the most relevant information in the backend tables. Our approach is as follows. We first classify each text entity (say, a web page, PDF document or paragraph) prior to building the backend tables. More specifically, we assign one or multiple agent labels to each text entity, each with its own score or probability to indicate relevancy. Then, in addition to our standard backend tables (categories, URLs, tags, embeddings, and so on), we build an agent table with the same structure: a nested hash. The parent key is a multi-token as usual, and the value is also a hash table, where each daughter key is an agent label. The value attached to an agent label is the list of text entities matching the agent in question, each with its own relevancy score!relevancy score.

6.10 Data augmentation via dictionaries

When designing an LLM system serving professional users, it is critical to use top quality input sources. Not only to get high quality content, but also to leverage its embedded structure (breadcrumbs, taxonomy, knowledge graph). This allows you to create contextual backend tables, as opposed to adding knowledge graph as a top, frontend layer. However, some input sources may be too small, if specialized or if your LLM consists of multiple sub-LLMs, like a mixture of experts.

To augment your corpus, you can use dictionaries (synonyms, abbreviations), indexes, glossaries, or even books. You can also leverage user prompts. They help you identify what is missing in your corpus, leading to corpus improvement or alternate taxonomies. Augmentation is not limited to text. Taxonomy and knowledge graph augmentation can be done by importing external taxonomies. All this is eventually added to your backend tables. When returning results to a user prompt, you can mark each item either as internal (coming from the original corpus) or external (coming from augmentation). This feature will increase the security of your system, especially for enterprise LLMs.

References

[1] Vincent Granville. *State of the Art in GenAI & LLMs – Creative Projects, with Solutions*. MLTechniques.com, 2024. [Link]. 4

Index