

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	1
1.1	From frontend prompts to backend tables	2
1.2	What is not covered here	3
2	Parameters, features, and fine-tuning	5
2.1	Backend parameters	5
2.2	Frontend parameters	5
2.3	Agents	6
2.4	Reproducibility	7
2.5	Singularization, stemming, auto-correct	8
2.6	Augmentation, distillation, and frontend tables	8
2.7	In-memory database, latency, and scalability	9
3	Case study	10
3.1	Real-time fine-tuning, prompts, and command menu	11
3.2	Sample session	12
4	Conclusions	26
5	Python code	26
6	Appendix: 10 features to dramatically improve performance	38
6.1	Fast search	38
6.2	Sparse databases	39
6.3	Contextual tokens	39
6.4	Adaptive loss function	39
6.5	Contextual tables	39
6.6	Smart crawling	39
6.7	LLM router	39
6.8	From one trillion parameters down to two	40
6.9	Agentic LLMs	40
6.10	Data augmentation via dictionaries	41
	References	41
	Index	42

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.

```
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 $1! + 2! + \dots + n!$, 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 build?
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](#).

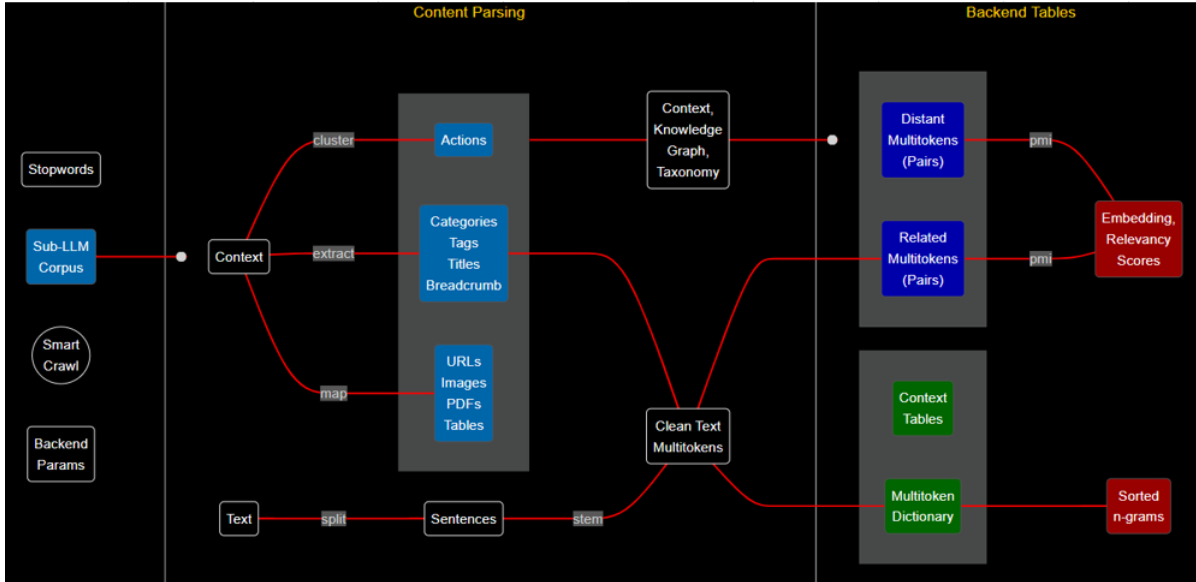


Figure 4: From crawl to backend tables (high resolution [here](#))

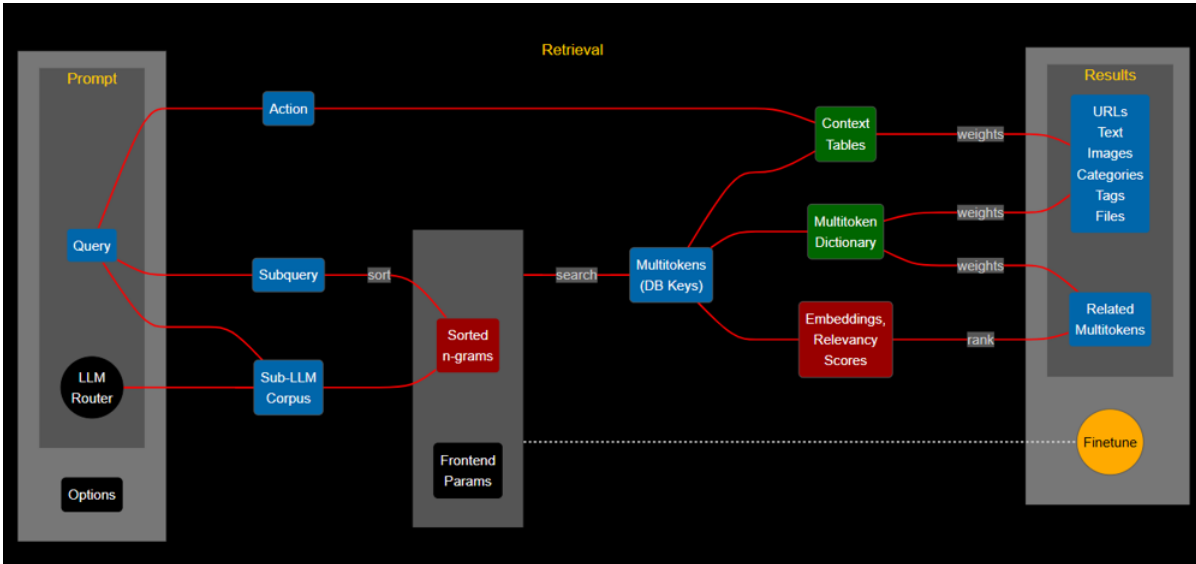


Figure 5: From prompt to query results, via backend tables (high resolution [here](#))

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':    # deaefault 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:

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

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’.

```
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 build 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.

```
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`.

<pre>('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,)</pre>	
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 to 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

```

```

35 1 1.00 * systems|managed information|assets
36 1 1.00 * managed|mltxdat information|assets
37 1 1.00 * systems|managed|mltxdat information|assets
38 1 1.00 managed|mltxdat|csa information|assets
39 1 1.00 platform|against information|assets
40 1 1.00 * platform|against|threats information|assets
41 1 1.00 * threats|such information|assets
42 1 1.00 * data|systems|managed information|assets
43 1 1.00 csa|platform|against information|assets
44 1 1.00 * against|threats information|assets
45 1 1.00 * against|threats|such information|assets
46 1 0.71 * navigating|data page|mltxquest
47 1 0.71 * efficiently|navigating|data page|mltxquest
48 1 0.71 * navigating|data|assets page|mltxquest
49 1 0.71 assets|page page|mltxquest
50 1 0.71 data|assets|page page|mltxquest
51 1 0.71 page|mltxquest|while page|mltxquest
52 1 0.71 * while|facilitating page|mltxquest
53 1 0.71 * while|facilitating|comprehensive page|mltxquest
54 1 0.71 assets|page|mltxquest page|mltxquest
55 1 0.71 mltxquest|while page|mltxquest
56 1 0.71 * mltxquest|while|facilitating page|mltxquest
57 1 0.71 * facilitating|comprehensive page|mltxquest
58 1 0.71 assets|deta page|mltxquest
59 1 0.71 information|page page|mltxquest
60 1 0.71 page|mltxquest|data page|mltxquest
61 1 0.71 information|page|mltxquest page|mltxquest
62 1 0.71 mltxquest|data page|mltxquest
63 1 0.71 * mltxquest|data|assets page|mltxquest
64 1 0.71 * assets|users page|mltxquest
65 1 0.71 * data|assets|users page|mltxquest
66 1 0.71 mltxdat|csa|platform information|assets
67 1 0.71 csa|platform information|assets
68 2 0.67 * users|efficiently data|assets
69 2 0.67 * efficiently|navigating data|assets
70 2 0.67 * users|efficiently|navigating data|assets
71 2 0.67 * aid|users|efficiently data|assets
72 2 0.50 * global|search detailed
73 2 0.50 detailed|process detailed
74 2 0.50 * process|migrating detailed
75 2 0.50 * detailed|process|migrating detailed
76 2 0.50 * migrating|historical detailed
77 2 0.50 * process|migrating|historical detailed
78 2 0.50 describes|detailed detailed
79 2 0.50 describes|detailed|process detailed
80 2 0.47 * data|assets page|mltxquest
81 2 0.47 * page|mltxquest data|assets
82 1 0.45 mltxdat|csa information|assets
83 2 0.41 data|migration detailed
84 1 0.35 * guide|global detailed
85 1 0.35 * guide|global|search detailed
86 1 0.35 * information|search detailed
87 1 0.35 * search|data detailed
88 1 0.35 * information|search|data detailed
89 1 0.35 * roles|raci detailed
90 1 0.35 * responsibilities|policy detailed
91 1 0.35 * zones|roles detailed
92 1 0.35 zones|roles|responsibilities detailed
93 1 0.35 responsibilities|detailed detailed
94 1 0.35 roles|responsibilities|detailed detailed
95 1 0.35 detailed|along detailed
96 1 0.35 responsibilities|detailed|along detailed
97 1 0.35 * detailed|along|raci detailed
98 1 0.35 * raci|matrix detailed
99 1 0.35 * along|raci detailed
100 1 0.35 * along|raci|matrix detailed
101 1 0.35 mltxquest|business detailed
102 1 0.35 metadata|templates detailed
103 1 0.35 detailed|instructions detailed
104 1 0.35 * instructions|completing detailed
105 1 0.35 * detailed|instructions|completing detailed
106 1 0.35 * instructions|completing|templates detailed
107 1 0.35 filling|out detailed
108 1 0.35 * out|templates detailed
109 1 0.35 * filling|out|templates detailed
110 1 0.35 * templates|users detailed

```

111	1 0.35	* out templates users	detailed
112	1 0.35	* reviewed metadata	detailed
113	1 0.35	* metadata uploaded	detailed
114	1 0.35	* reviewed metadata uploaded	detailed
115	1 0.35	* completing templates	detailed
116	1 0.35	completed reviewed	detailed
117	1 0.35	* completed reviewed metadata	detailed
118	1 0.35	* offers essential	detailed
119	1 0.35	* essential visual	detailed
120	1 0.35	* offers essential visual	detailed
121	1 0.35	essential visual representations	detailed
122	1 0.35	representations detailed	detailed
123	1 0.35	visual representations detailed	detailed
124	1 0.35	detailed table	detailed
125	1 0.35	representations detailed table	detailed
126	1 0.35	* table showcasing	detailed
127	1 0.35	* detailed table showcasing	detailed
128	1 0.35	* showcasing project	detailed
129	1 0.35	* table showcasing project	detailed
130	1 0.35	* set defined	detailed
131	1 0.35	* set defined rules	detailed
132	1 0.35	rules tab	detailed
133	1 0.35	tab ensures	detailed
134	1 0.35	rules tab ensures	detailed
135	1 0.35	ensures consistent	detailed
136	1 0.35	tab ensures consistent	detailed
137	1 0.35	* ensures consistent standardized	detailed
138	1 0.35	* standardized approach	detailed
139	1 0.35	* defined rules	detailed
140	1 0.35	defined rules tab	detailed
141	1 0.35	* consistent standardized	detailed
142	1 0.35	* consistent standardized approach	detailed
143	1 0.35	* batch process execution	detailed
144	1 0.35	databricks metrics	detailed
145	1 0.35	* applications performance	detailed
146	1 0.35	* datadog applications performance	detailed
147	1 0.35	* applications performance monitoring	detailed
148	1 0.35	* process execution	detailed
149	1 0.35	* execution including	detailed
150	1 0.35	* process execution including	detailed
151	1 0.35	including databricks	detailed
152	1 0.35	execution including databricks	detailed
153	1 0.35	including databricks metrics	detailed
154	1 0.35	monitoring apm	detailed
155	1 0.35	performance monitoring apm	detailed
156	1 0.35	monitoring apm detailed	detailed
157	1 0.35	detailed tracing	detailed
158	1 0.35	* tracing request	detailed
159	1 0.35	* detailed tracing request	detailed
160	1 0.35	* request log	detailed
161	1 0.35	* tracing request log	detailed
162	1 0.35	apm detailed	detailed
163	1 0.35	apm detailed tracing	detailed
164	1 0.33	* effectively manage	data assets
165	1 0.33	* regulations effectively manage	data assets
166	1 0.33	* manage protect	data assets
167	1 0.33	* effectively manage protect	data assets
168	1 0.33	manage protect data	data assets
169	1 0.33	protect data assets	data assets
170	1 0.33	* clarify data	data assets
171	1 0.33	* clarify data governance	data assets
172	1 0.33	data administration	data assets
173	1 0.33	* administration zones	data assets
174	1 0.33	* data administration zones	data assets
175	1 0.33	* steward policy	data assets
176	1 0.33	governance focused	data assets
177	1 0.33	data governance focused	data assets
178	1 0.33	focused data	data assets
179	1 0.33	governance focused data	data assets
180	1 0.33	focused data administration	data assets
181	1 0.33	* data steward	data assets
182	1 0.33	* steward governing	data assets
183	1 0.33	* data steward governing	data assets
184	1 0.33	governing data	data assets
185	1 0.33	steward governing data	data assets
186	1 0.33	governing data assets	data assets

```

187 1 0.33 data|assets|respective      data|assets
188 1 0.33 * respective|zones          data|assets
189 1 0.33 * zones|outlined             data|assets
190 1 0.33 * respective|zones|outlined data|assets
191 1 0.33 assets|respective             data|assets
192 1 0.33 * assets|respective|zones    data|assets
193 1 0.33 search|mltxquest              data|assets
194 1 0.33 global|search|mltxquest          data|assets
195 1 0.33 search|mltxquest|landing      data|assets
196 1 0.33 * landing|summary              data|assets
197 1 0.33 * mltxquest|landing|summary    data|assets
198 1 0.33 * summary|page                 data|assets
199 1 0.33 * landing|summary|page          data|assets
200 1 0.33 search|data|assets              data|assets
201 1 0.33 data|assets|filters              data|assets
202 1 0.33 * filters|better                data|assets
203 1 0.33 * filters|better|search          data|assets
204 1 0.33 assets|filters                  data|assets
205 1 0.33 * assets|filters|better          data|assets
206 1 0.33 * better|search                  data|assets
207 1 0.33 designed|aid                     data|assets
208 1 0.33 designed|aid|users               data|assets
209 1 0.33 aid|users                         data|assets
210 1 0.33 * users|access|both                data|assets
211 1 0.33 * assets|users|access              data|assets
212 1 0.33 * access|both                     data|assets
213 1 0.33 * both|technical|business      data|assets
214 1 0.33 business|metadata|data            data|assets
215 1 0.33 metadata|data                    data|assets
216 1 0.33 metadata|data|assets              data|assets
217 1 0.33 data|assets|available             data|assets
218 1 0.33 assets|available                 data|assets
219 1 0.33 * available|mltxdat               data|assets
220 1 0.33 * assets|available|mltxdat       data|assets
221 1 0.33 * accountability|individuals     data|assets
222 1 0.33 * clear|framework|managing        data|assets
223 1 0.33 * fundamental|components|data    data|assets
224 1 0.33 governance|defines                 data|assets
225 1 0.33 data|governance|defines            data|assets
226 1 0.33 defines|roles                     data|assets
227 1 0.33 governance|defines|roles           data|assets
228 1 0.33 defines|roles|responsibilities    data|assets
229 1 0.33 * responsibilities|accountability   data|assets
230 1 0.33 * roles|responsibilities|accountability data|assets
231 1 0.33 * responsibilities|accountability|individuals data|assets
232 1 0.33 * framework|managing              data|assets
233 1 0.33 * managing|stewarding              data|assets
234 1 0.33 * framework|managing|stewarding   data|assets
235 1 0.33 stewarding|data                   data|assets
236 1 0.33 managing|stewarding|data          data|assets
237 1 0.33 stewarding|data|assets            data|assets
238 1 0.33 data|assets|quality                data|assets
239 1 0.33 * security|proper                  data|assets
240 1 0.33 * quality|security|proper          data|assets
241 1 0.33 assets|quality                     data|assets
242 1 0.33 * assets|quality|security          data|assets
243 1 0.33 * badge|mltxquest                  data|assets
244 1 0.33 * badge|mltxquest|awarded          data|assets
245 1 0.33 awarded|data                       data|assets
246 1 0.33 * governance|metadata              data|assets
247 1 0.33 * governance|badge                  data|assets
248 1 0.33 * mltxquest|awarded                 data|assets
249 1 0.33 mltxquest|awarded|data             data|assets
250 1 0.33 data|assets|table                   data|assets
251 1 0.33 * table|demonstrate                 data|assets
252 1 0.33 * demonstrate|exceptional          data|assets
253 1 0.33 * table|demonstrate|exceptional    data|assets
254 1 0.33 * table|meets                       data|assets
255 1 0.33 meets|stringent                     data|assets
256 1 0.33 table|meets|stringent               data|assets
257 1 0.33 stringent|criteria                  data|assets
258 1 0.33 meets|stringent|criteria            data|assets
259 1 0.33 stringent|criteria|including        data|assets
260 1 0.33 * including|robust                  data|assets
261 1 0.33 * robust|technical                  data|assets
262 1 0.33 * including|robust|technical        data|assets

```



```

263 1 0.33 * signifies|commitment      data|assets
264 1 0.33 * signifies|commitment|high data|assets
265 1 0.33 high|data                   data|assets
266 1 0.33 high|data|governance        data|assets
267 1 0.33 governance|standards        data|assets
268 1 0.33 data|governance|standards   data|assets
269 1 0.33 * standards|providing       data|assets
270 1 0.33 * governance|standards|providing data|assets
271 1 0.33 * providing|users           data|assets
272 1 0.33 * standards|providing|users data|assets
273 1 0.33 awarded|data|assets         data|assets
274 1 0.33 assets|table                data|assets
275 1 0.33 * assets|table|demonstrate data|assets
276 1 0.33 * badge|table               data|assets
277 1 0.33 * badge|table|meets         data|assets
278 1 0.33 criteria|including          data|assets
279 1 0.33 * criteria|including|robust data|assets
280 1 0.33 * commitment|high          data|assets
281 1 0.33 commitment|high|data       data|assets
282 1 0.25 visual|representations      detailed
283 1 0.25 * performance|monitoring   detailed
284 1 0.24 protect|data                data|assets
285 1 0.24 * business|metadata         data|assets
286 1 0.24 * technical|business        data|assets
287 1 0.24 * technical|business|metadata data|assets
288 1 0.24 * quality|security          data|assets
289 3 0.23 * data|governance           data|assets
290 1 0.19 mltxquest|landing           data|assets
291 1 0.19 * users|access              data|assets
292 1 0.16 roles|responsibilities      detailed
293 1 0.15 * components|data           data|assets
294 1 0.15 * components|data|governance data|assets
295 1 0.10 * data|products             detailed
296
297 N = occurrences of (token, word) in corpus. F = * if contextual pair.
298 If no result, try option '-p f'.
299
300 >>> RESULTS - SECTION: Category
301
302 Category: 'Products' [6 entries]
303 Linked to: page|mltxquest (2)
304 Linked to: detailed (8)
305 Linked to: information|page|mltxquest|data (1)
306 Linked to: data|assets (9)
307 Linked to: data|assets|page|mltxquest (1)
308 Linked to: page|mltxquest|data|assets (1)
309
310 Category: 'Governance' [3 entries]
311 Linked to: detailed (8)
312 Linked to: information|assets (1)
313 Linked to: data|assets (9)
314
315 Category: 'BI Solution' [1 entries]
316 Linked to: detailed (8)
317
318 Category: 'Observability & Monitoring' [1 entries]
319 Linked to: detailed (8)
320
321 Category: 'One Platform' [1 entries]
322 Linked to: detailed (8)
323
324
325 >>> RESULTS - SECTION: Tags
326
327 Tags: MLTxQuest [6 entries]
328 Linked to: page|mltxquest (2)
329 Linked to: detailed (8)
330 Linked to: information|page|mltxquest|data (1)
331 Linked to: data|assets (9)
332 Linked to: data|assets|page|mltxquest (1)
333 Linked to: page|mltxquest|data|assets (1)
334
335 Tags: Guideline [3 entries]
336 Linked to: page|mltxquest (2)
337 Linked to: data|assets (9)
338 Linked to: data|assets|page|mltxquest (1)

```

```

339
340 Tags: Guidelines [5 entries]
341 Linked to: page|mltxquest (2)
342 Linked to: detailed (8)
343 Linked to: information|page|mltxquest|data (1)
344 Linked to: data|assets (9)
345 Linked to: page|mltxquest|data|assets (1)
346
347 Tags: example1 [2 entries]
348 Linked to: detailed (8)
349 Linked to: data|assets (9)
350
351 Tags: example2 [2 entries]
352 Linked to: detailed (8)
353 Linked to: data|assets (9)
354
355 Tags: governance [2 entries]
356 Linked to: detailed (8)
357 Linked to: data|assets (9)
358
359 Tags: roles [1 entries]
360 Linked to: detailed (8)
361
362 Tags: raci [1 entries]
363 Linked to: detailed (8)
364
365 Tags: metadata [2 entries]
366 Linked to: detailed (8)
367 Linked to: data|assets (9)
368
369 Tags: mltxquest [1 entries]
370 Linked to: detailed (8)
371
372 Tags: business [1 entries]
373 Linked to: detailed (8)
374
375 Tags: products [1 entries]
376 Linked to: detailed (8)
377
378 Tags: metrics [1 entries]
379 Linked to: detailed (8)
380
381 Tags: Historical data [1 entries]
382 Linked to: detailed (8)
383
384 Tags: Security [1 entries]
385 Linked to: information|assets (1)
386
387 Tags: privacy [1 entries]
388 Linked to: data|assets (9)
389
390 Tags: Steward [1 entries]
391 Linked to: data|assets (9)
392
393 Tags: policy [1 entries]
394 Linked to: data|assets (9)
395
396 Tags: owner [1 entries]
397 Linked to: data|assets (9)
398
399 Tags: badge [1 entries]
400 Linked to: data|assets (9)
401
402
403 >>> RESULTS - SECTION: Titles
404
405 Titles: 'MLTxQuest - Data Assets' [3 entries]
406 Linked to: page|mltxquest (2)
407 Linked to: data|assets (9)
408 Linked to: data|assets|page|mltxquest (1)
409
410 Titles: 'MLTxQuest-Data Asset Deta' [5 entries]
411 Linked to: page|mltxquest (2)
412 Linked to: detailed (8)
413 Linked to: information|page|mltxquest|data (1)
414 Linked to: data|assets (9)

```

```

415 Linked to: page|mltxquest|data|assets (1)
416
417 Titles: 'MLTxQuest - Global Search' [2 entries]
418 Linked to: detailed (8)
419 Linked to: data|assets (9)
420
421 Titles: 'Roles and Responsibilities Policy' [1 entries]
422 Linked to: detailed (8)
423
424 Titles: 'Business Metadata Template' [1 entries]
425 Linked to: detailed (8)
426
427 Titles: '[METRICS] Data Products' [1 entries]
428 Linked to: detailed (8)
429
430 Titles: 'Exploration - Monitoring' [1 entries]
431 Linked to: detailed (8)
432
433 Titles: 'Historical data migration' [1 entries]
434 Linked to: detailed (8)
435
436 Titles: 'Data Security Policy ' [1 entries]
437 Linked to: information|assets (1)
438
439 Titles: 'Data Privacy Policy' [1 entries]
440 Linked to: data|assets (9)
441
442 Titles: 'Data Steward Policy' [1 entries]
443 Linked to: data|assets (9)
444
445 Titles: 'Data Owner Policy' [1 entries]
446 Linked to: data|assets (9)
447
448 Titles: 'MLTxQuest - Governance Badge' [1 entries]
449 Linked to: data|assets (9)
450
451
452 >>> RESULTS - SECTION: ID
453
454 ID: 91 [3 entries]
455 Linked to: page|mltxquest (2)
456 Linked to: data|assets (9)
457 Linked to: data|assets|page|mltxquest (1)
458
459 ID: 92 [5 entries]
460 Linked to: page|mltxquest (2)
461 Linked to: detailed (8)
462 Linked to: information|page|mltxquest|data (1)
463 Linked to: data|assets (9)
464 Linked to: page|mltxquest|data|assets (1)
465
466 ID: 90 [2 entries]
467 Linked to: detailed (8)
468 Agents: ('Example',)
469 Linked to: data|assets (9)
470 Agents: ('Example',)
471
472 ID: 101 [1 entries]
473 Linked to: detailed (8)
474 Agents: ('Policy', 'Governance')
475
476 ID: 107 [1 entries]
477 Linked to: detailed (8)
478 Agents: ('Template', 'Governance')
479
480 ID: 139 [1 entries]
481 Linked to: detailed (8)
482
483 ID: 381 [1 entries]
484 Linked to: detailed (8)
485
486 ID: 511 [1 entries]
487 Linked to: detailed (8)
488 Agents: ('Data',)
489
490 ID: 513 [1 entries]

```

```

491 Linked to: detailed (8)
492 Agents: ('Data',)
493
494 ID: 223 [1 entries]
495 Linked to: information|assets (1)
496 Agents: ('Policy', 'Governance')
497
498 ID: 42 [1 entries]
499 Linked to: data|assets (9)
500 Agents: ('Policy', 'Governance')
501
502 ID: 48 [1 entries]
503 Linked to: data|assets (9)
504 Agents: ('Policy', 'Governance')
505
506 ID: 199 [1 entries]
507 Linked to: data|assets (9)
508 Agents: ('Policy', 'Governance')
509
510 ID: 259 [1 entries]
511 Linked to: data|assets (9)
512 Agents: ('Governance',)
513
514
515 >>> RESULTS - SECTION: Agents
516
517 Agents: Example [2 entries]
518 Linked to: detailed (8)
519 Linked to: data|assets (9)
520
521 Agents: Policy [3 entries]
522 Linked to: detailed (8)
523 Linked to: information|assets (1)
524 Linked to: data|assets (9)
525
526 Agents: Governance [3 entries]
527 Linked to: detailed (8)
528 Linked to: information|assets (1)
529 Linked to: data|assets (9)
530
531 Agents: Template [1 entries]
532 Linked to: detailed (8)
533
534 Agents: Data [1 entries]
535 Linked to: detailed (8)
536
537
538 Above results based on words found in prompt, matched back to backend tables.
539 Numbers in parentheses are occurrences of word in corpus.
540
541
542 >>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)
543 empty unless labels 'ID' and 'Agents' are in 'show'.
544
545 ('Data', 'detailed') --> (511, 513)
546 ('Example', 'data|assets') --> (90,)
547 ('Example', 'detailed') --> (90,)
548 ('Governance', 'data|assets') --> (42, 48, 199, 259)
549 ('Governance', 'detailed') --> (101, 107)
550 ('Governance', 'information|assets') --> (223,)
551 ('Policy', 'data|assets') --> (42, 48, 199)
552 ('Policy', 'detailed') --> (101,)
553 ('Policy', 'information|assets') --> (223,)
554 ('Template', 'detailed') --> (107,)
555
556 ID Size
557
558 511 690
559 513 692
560 90 772
561 42 948
562 48 916
563 199 980
564 259 1153
565 101 851
566 107 1242

```

```

567 223 978
568
569
570 Command menu:
571
572 -q      : print last non-command prompt
573 -x      : print sample queries
574 -p key value : set frontendParams[key] = value
575 -f      : use catch-all parameter set for debugging
576 -d      : use default parameter set
577 -v      : view parameter set
578 -a multitoken : add multitoken to 'ignore' list
579 -r multitoken : remove multitoken from 'ignore' list
580 -l      : view 'ignore' list
581 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
582 -s      : print size of core backend tables
583 -c F1 F2 ... : show sections F1 F2 ... in output results
584
585 To view available sections for -c command, enter -v command.
586 To view available keys for -p command, enter -v command.
587 For -i command, choose IDs from list shown in prompt results.
588 For standard prompts, enter text not starting with '-' or digit.
589
590
591 Query, command, or integer in [0, 7] for sample query: -i 107 259
592
593 --- Entity 107 ---
594
595 > Modified Date :
596 2024-07-02T12:51:31.993Z
597
598 > title_text :
599 Business Metadata Template
600
601 > description_text :
602 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 MLTQuest, making it accessible through the MLTQuest portal for all authorized
    users, thereby centralizing and simplifying access to critical information within the
    organization.
603
604 > tags_list_text :
605 metadata
606 mltxquest
607 business
608
609 > link_list_text :
610
611
612 > likes_list_text :
613 luiz.lagatosm@abc-mixa.com
614
615 > category_text :
616 Governance
617
618 --- Entity 259 ---
619
620 > Modified Date :
621 2024-06-27T11:36:39.594Z
622
623 > title_text :
624 MLTQuest - Governance Badge
625
626 > description_text :
627 The Governance Badge in MLTQuest 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.
628
629 > tags_list_text :
630 badge
631 governance
632 metadata

```

```

633
634 > link_list_text :
635
636
637 > likes_list_text :
638 luiz.lagatosm@abc-mixa.com
639
640 > category_text :
641 Governance
642
643
644 2 text entities found.
645
646 Completed task: -i 107 259
647
648
649 Command menu:
650
651 -q          : print last non-command prompt
652 -x          : print sample queries
653 -p key value : set frontendParams[key] = value
654 -f          : use catch-all parameter set for debugging
655 -d          : use default parameter set
656 -v          : view parameter set
657 -a multitoken : add multitoken to 'ignore' list
658 -r multitoken : remove multitoken from 'ignore' list
659 -l          : view 'ignore' list
660 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
661 -s          : print size of core backend tables
662 -c F1 F2 ... : show sections F1 F2 ... in output results
663
664 To view available sections for -c command, enter -v command.
665 To view available keys for -p command, enter -v command.
666 For -i command, choose IDs from list shown in prompt results.
667 For standard prompts, enter text not starting with '-' or digit.
668
669
670 Query, command, or integer in [0, 7] for sample query: -a detailed
671
672
673 Completed task: -a detailed
674
675
676 Command menu:
677
678 -q          : print last non-command prompt
679 -x          : print sample queries
680 -p key value : set frontendParams[key] = value
681 -f          : use catch-all parameter set for debugging
682 -d          : use default parameter set
683 -v          : view parameter set
684 -a multitoken : add multitoken to 'ignore' list
685 -r multitoken : remove multitoken from 'ignore' list
686 -l          : view 'ignore' list
687 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
688 -s          : print size of core backend tables
689 -c F1 F2 ... : show sections F1 F2 ... in output results
690
691 To view available sections for -c command, enter -v command.
692 To view available keys for -p command, enter -v command.
693 For -i command, choose IDs from list shown in prompt results.
694 For standard prompts, enter text not starting with '-' or digit.
695
696
697 Query, command, or integer in [0, 7] for sample query: -v
698
699 Key Description      Value
700
701 0 embeddingKeyMinSize 1
702 1 embeddingValuesMinSize 2
703 2 min_pmi             0.0
704 3 nABmin              1
705 4 Customized_pmi      True
706 5 ContextMultitokenMinSize 1
707 6 minOutputListSize   1
708 7 bypassIgnoreList    False

```

```

709     8 ignoreList          ('data', 'detailed')
710     9 maxTokenCount      100
711
712 Show sections:
713
714     Embeddings True
715     Category True
716     Tags      True
717     Titles    True
718     Descr.    False
719     Whole     False
720     ID        True
721     Agents    True
722
723 Completed task: -v
724
725
726 Command menu:
727
728     -q          : print last non-command prompt
729     -x          : print sample queries
730     -p key value : set frontendParams[key] = value
731     -f          : use catch-all parameter set for debugging
732     -d          : use default parameter set
733     -v          : view parameter set
734     -a multitoken : add multitoken to 'ignore' list
735     -r multitoken : remove multitoken from 'ignore' list
736     -l          : view 'ignore' list
737     -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
738     -s          : print size of core backend tables
739     -c F1 F2 ... : show sections F1 F2 ... in output results
740
741 To view available sections for -c command, enter -v command.
742 To view available keys for -p command, enter -v command.
743 For -i command, choose IDs from list shown in prompt results.
744 For standard prompts, enter text not starting with '-' or digit.
745
746
747 Query, command, or integer in [0, 7] for sample query: -p 6 2
748
749
750 Completed task: -p 6 2
751
752
753 Command menu:
754
755     -q          : print last non-command prompt
756     -x          : print sample queries
757     -p key value : set frontendParams[key] = value
758     -f          : use catch-all parameter set for debugging
759     -d          : use default parameter set
760     -v          : view parameter set
761     -a multitoken : add multitoken to 'ignore' list
762     -r multitoken : remove multitoken from 'ignore' list
763     -l          : view 'ignore' list
764     -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
765     -s          : print size of core backend tables
766     -c F1 F2 ... : show sections F1 F2 ... in output results
767
768 To view available sections for -c command, enter -v command.
769 To view available keys for -p command, enter -v command.
770 For -i command, choose IDs from list shown in prompt results.
771 For standard prompts, enter text not starting with '-' or digit.
772
773
774 Query, command, or integer in [0, 7] for sample query: -p 2 0.50
775
776
777 Completed task: -p 2 0.50
778
779
780 Command menu:
781
782     -q          : print last non-command prompt
783     -x          : print sample queries
784     -p key value : set frontendParams[key] = value

```



```

785 -f          : use catch-all parameter set for debugging
786 -d          : use default parameter set
787 -v          : view parameter set
788 -a multitoken : add multitoken to 'ignore' list
789 -r multitoken : remove multitoken from 'ignore' list
790 -l          : view 'ignore' list
791 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
792 -s          : print size of core backend tables
793 -c F1 F2 ... : show sections F1 F2 ... in output results
794
795 To view available sections for -c command, enter -v command.
796 To view available keys for -p command, enter -v command.
797 For -i command, choose IDs from list shown in prompt results.
798 For standard prompts, enter text not starting with '-' or digit.
799
800
801 Query, command, or integer in [0, 7] for sample query: 6
802 query: MLTxQuest Data Assets Detailed Information page
803
804 N pmi F token [from embeddings] word [from prompt]
805
806 1 1.00 * confidentiality|availability information|assets
807 1 1.00 * availability|organization information|assets
808 1 1.00 * confidentiality|availability|organization information|assets
809 1 1.00 availability|organization|information information|assets
810 1 1.00 * integrity|confidentiality|availability information|assets
811 1 1.00 organization|information information|assets
812 1 1.00 organization|information|assets information|assets
813 1 1.00 * systems|managed information|assets
814 1 1.00 * managed|mltxdat information|assets
815 1 1.00 * systems|managed|mltxdat information|assets
816 1 1.00 managed|mltxdat|csa information|assets
817 1 1.00 platform|against information|assets
818 1 1.00 * platform|against|threats information|assets
819 1 1.00 * threats|such information|assets
820 1 1.00 * data|systems|managed information|assets
821 1 1.00 csa|platform|against information|assets
822 1 1.00 * against|threats information|assets
823 1 1.00 * against|threats|such information|assets
824 1 0.71 * navigating|data page|mltxquest
825 1 0.71 * efficiently|navigating|data page|mltxquest
826 1 0.71 * navigating|data|assets page|mltxquest
827 1 0.71 assets|page page|mltxquest
828 1 0.71 data|assets|page page|mltxquest
829 1 0.71 page|mltxquest|while page|mltxquest
830 1 0.71 * while|facilitating page|mltxquest
831 1 0.71 * while|facilitating|comprehensive page|mltxquest
832 1 0.71 assets|page|mltxquest page|mltxquest
833 1 0.71 mltxquest|while page|mltxquest
834 1 0.71 * mltxquest|while|facilitating page|mltxquest
835 1 0.71 * facilitating|comprehensive page|mltxquest
836 1 0.71 assets|deta page|mltxquest
837 1 0.71 information|page page|mltxquest
838 1 0.71 page|mltxquest|data page|mltxquest
839 1 0.71 information|page|mltxquest page|mltxquest
840 1 0.71 mltxquest|data page|mltxquest
841 1 0.71 * mltxquest|data|assets page|mltxquest
842 1 0.71 * assets|users page|mltxquest
843 1 0.71 * data|assets|users page|mltxquest
844 1 0.71 mltxdat|csa|platform information|assets
845 1 0.71 csa|platform information|assets
846 2 0.67 * users|efficiently data|assets
847 2 0.67 * efficiently|navigating data|assets
848 2 0.67 * users|efficiently|navigating data|assets
849 2 0.67 * aid|users|efficiently data|assets
850
851 N = occurrences of (token, word) in corpus. F = * if contextual pair.
852 If no result, try option '-p f'.
853
854 >>> RESULTS - SECTION: Category
855
856 Category: 'Products' [5 entries]
857 Linked to: page|mltxquest (2)
858 Linked to: information|page|mltxquest|data (1)
859 Linked to: data|assets (9)
860 Linked to: data|assets|page|mltxquest (1)

```

```

861 Linked to: page|mltxquest|data|assets (1)
862
863 Category: 'Governance' [2 entries]
864 Linked to: information|assets (1)
865 Linked to: data|assets (9)
866
867
868 >>> RESULTS - SECTION: Tags
869
870 Tags: MLTxQuest [5 entries]
871 Linked to: page|mltxquest (2)
872 Linked to: information|page|mltxquest|data (1)
873 Linked to: data|assets (9)
874 Linked to: data|assets|page|mltxquest (1)
875 Linked to: page|mltxquest|data|assets (1)
876
877 Tags: Guideline [3 entries]
878 Linked to: page|mltxquest (2)
879 Linked to: data|assets (9)
880 Linked to: data|assets|page|mltxquest (1)
881
882 Tags: Guidelines [4 entries]
883 Linked to: page|mltxquest (2)
884 Linked to: information|page|mltxquest|data (1)
885 Linked to: data|assets (9)
886 Linked to: page|mltxquest|data|assets (1)
887
888
889 >>> RESULTS - SECTION: Titles
890
891 Titles: 'MLTxQuest - Data Assets' [3 entries]
892 Linked to: page|mltxquest (2)
893 Linked to: data|assets (9)
894 Linked to: data|assets|page|mltxquest (1)
895
896 Titles: 'MLTxQuest-Data Asset Deta' [4 entries]
897 Linked to: page|mltxquest (2)
898 Linked to: information|page|mltxquest|data (1)
899 Linked to: data|assets (9)
900 Linked to: page|mltxquest|data|assets (1)
901
902
903 >>> RESULTS - SECTION: ID
904
905 ID: 91 [3 entries]
906 Linked to: page|mltxquest (2)
907 Linked to: data|assets (9)
908 Linked to: data|assets|page|mltxquest (1)
909
910 ID: 92 [4 entries]
911 Linked to: page|mltxquest (2)
912 Linked to: information|page|mltxquest|data (1)
913 Linked to: data|assets (9)
914 Linked to: page|mltxquest|data|assets (1)
915
916
917 >>> RESULTS - SECTION: Agents
918
919 Agents: Policy [2 entries]
920 Linked to: information|assets (1)
921 Linked to: data|assets (9)
922
923 Agents: Governance [2 entries]
924 Linked to: information|assets (1)
925 Linked to: data|assets (9)
926
927
928 Above results based on words found in prompt, matched back to backend tables.
929 Numbers in parentheses are occurrences of word in corpus.
930
931
932 >>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)
933 empty unless labels 'ID' and 'Agents' are in 'show'.
934
935
936 ID Size

```

```

937
938
939
940 Command menu:
941
942 -q      : print last non-command prompt
943 -x      : print sample queries
944 -p key value : set frontendParams[key] = value
945 -f      : use catch-all parameter set for debugging
946 -d      : use default parameter set
947 -v      : view parameter set
948 -a multitoken : add multitoken to 'ignore' list
949 -r multitoken : remove multitoken from 'ignore' list
950 -l      : view 'ignore' list
951 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
952 -s      : print size of core backend tables
953 -c F1 F2 ... : show sections F1 F2 ... in output results
954
955 To view available sections for -c command, enter -v command.
956 To view available keys for -p command, enter -v command.
957 For -i command, choose IDs from list shown in prompt results.
958 For standard prompts, enter text not starting with '-' or digit.
959
960
961 Query, command, or integer in [0, 7] for sample query: -c Titles
962
963
964 Completed task: -c Titles
965
966
967 Command menu:
968
969 -q      : print last non-command prompt
970 -x      : print sample queries
971 -p key value : set frontendParams[key] = value
972 -f      : use catch-all parameter set for debugging
973 -d      : use default parameter set
974 -v      : view parameter set
975 -a multitoken : add multitoken to 'ignore' list
976 -r multitoken : remove multitoken from 'ignore' list
977 -l      : view 'ignore' list
978 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
979 -s      : print size of core backend tables
980 -c F1 F2 ... : show sections F1 F2 ... in output results
981
982 To view available sections for -c command, enter -v command.
983 To view available keys for -p command, enter -v command.
984 For -i command, choose IDs from list shown in prompt results.
985 For standard prompts, enter text not starting with '-' or digit.
986
987
988 Query, command, or integer in [0, 7] for sample query: 6
989 query: MLTxQuest Data Assets Detailed Information page
990 >>> RESULTS - SECTION: Titles
991
992 Titles: 'MLTxQuest - Data Assets' [3 entries]
993 Linked to: page|mltxquest (2)
994 Linked to: data|assets (9)
995 Linked to: data|assets|page|mltxquest (1)
996
997 Titles: 'MLTxQuest-Data Asset Deta' [4 entries]
998 Linked to: page|mltxquest (2)
999 Linked to: information|page|mltxquest|data (1)
1000 Linked to: data|assets (9)
1001 Linked to: page|mltxquest|data|assets (1)
1002
1003
1004 Above results based on words found in prompt, matched back to backend tables.
1005 Numbers in parentheses are occurrences of word in corpus.
1006
1007
1008 >>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)
1009 empty unless labels 'ID' and 'Agents' are in 'show'.
1010
1011
1012 ID Size

```

```

1013
1014
1015
1016 Command menu:
1017
1018 -q          : print last non-command prompt
1019 -x          : print sample queries
1020 -p key value : set frontendParams[key] = value
1021 -f          : use catch-all parameter set for debugging
1022 -d          : use default parameter set
1023 -v          : view parameter set
1024 -a multitoken : add multitoken to 'ignore' list
1025 -r multitoken : remove multitoken from 'ignore' list
1026 -l          : view 'ignore' list
1027 -i ID1 ID2 ... : print content of text entities ID1 ID2 ...
1028 -s          : print size of core backend tables
1029 -c F1 F2 ... : show sections F1 F2 ... in output results
1030
1031 To view available sections for -c command, enter -v command.
1032 To view available keys for -p command, enter -v command.
1033 For -i command, choose IDs from list shown in prompt results.
1034 For standard prompts, enter text not starting with '-' or digit.
1035
1036
1037 Query, command, or integer in [0, 7] for sample query:
1038

```

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.

```

1  #--- [1] Backend: functions
2
3  def update_hash(hash, key, count=1):
4
5      if key in hash:
6          hash[key] += count
7      else:
8          hash[key] = count
9      return(hash)
10
11
12  def update_nestedHash(hash, key, value, count=1):
13

```

```

14 # 'key' is a word here, value is tuple or single value
15 if key in hash:
16     local_hash = hash[key]
17 else:
18     local_hash = {}
19 if type(value) is not tuple:
20     value = (value,)
21 for item in value:
22     if item in local_hash:
23         local_hash[item] += count
24     else:
25         local_hash[item] = count
26 hash[key] = local_hash
27 return(hash)
28
29
30 def get_value(key, hash):
31     if key in hash:
32         value = hash[key]
33     else:
34         value = ''
35     return(value)
36
37
38 def update_tables(backendTables, word, hash_crawl, backendParams):
39
40     category = get_value('category', hash_crawl)
41     tag_list = get_value('tag_list', hash_crawl)
42     title = get_value('title', hash_crawl)
43     description = get_value('description', hash_crawl) #
44     meta = get_value('meta', hash_crawl)
45     ID = get_value('ID', hash_crawl)
46     agents = get_value('agents', hash_crawl)
47     full_content = get_value('full_content', hash_crawl) #
48
49     extraWeights = backendParams['extraWeights']
50     word = word.lower() # add stemming
51     weight = 1.0
52     if word in category:
53         weight += extraWeights['category']
54     if word in tag_list:
55         weight += extraWeights['tag_list']
56     if word in title:
57         weight += extraWeights['title']
58     if word in meta:
59         weight += extraWeights['meta']
60
61     update_hash(backendTables['dictionary'], word, weight)
62     update_nestedHash(backendTables['hash_context1'], word, category)
63     update_nestedHash(backendTables['hash_context2'], word, tag_list)
64     update_nestedHash(backendTables['hash_context3'], word, title)
65     update_nestedHash(backendTables['hash_context4'], word, description) # takes space, don't build?
66     update_nestedHash(backendTables['hash_context5'], word, meta)
67     update_nestedHash(backendTables['hash_ID'], word, ID)
68     update_nestedHash(backendTables['hash_agents'], word, agents)
69     for agent in agents:
70         update_nestedHash(backendTables['ID_to_agents'], ID, agent)
71     update_nestedHash(backendTables['full_content'], word, full_content) # takes space, don't build?
72     update_nestedHash(backendTables['ID_to_content'], ID, full_content)
73
74     return(backendTables)
75
76
77 def clean_list(value):
78
79     # change string "[ 'a', 'b', ...]" to ('a', 'b', ...)
80     value = value.replace("[", "(").replace("]", ")")
81     aux = value.split("~")
82     value_list = ()
83     for val in aux:
84         val = val.replace("'", "").replace('"', "").rstrip()
85         if val != '':
86             value_list = (*value_list, val)
87     return(value_list)
88
89

```

```

90 def get_key_value_pairs(entity):
91
92     # extract key-value pairs from 'entity' (a string)
93     entity = entity[1].replace(")", " ", "'")
94     flag = False
95     entity2 = ""
96
97     for idx in range(len(entity)):
98         if entity[idx] == '[':
99             flag = True
100         elif entity[idx] == ']':
101             flag = False
102         if flag and entity[idx] == ",":
103             entity2 += "~"
104         else:
105             entity2 += entity[idx]
106
107     entity = entity2
108     key_value_pairs = entity.split(", ' ")
109     return(key_value_pairs)
110
111
112 def update_dict(backendTables, hash_crawl, backendParams):
113
114     max_multitoken = backendParams['max_multitoken']
115     maxDist = backendParams['maxDist']
116     maxTerms = backendParams['maxTerms']
117
118     category = get_value('category', hash_crawl)
119     tag_list = get_value('tag_list', hash_crawl)
120     title = get_value('title', hash_crawl)
121     description = get_value('description', hash_crawl)
122     meta = get_value('meta', hash_crawl)
123
124     text = category + "." + str(tag_list) + "." + title + "." + description + "." + meta
125     text = text.replace('/', " ").replace('(', ' ').replace(')', ' ').replace('?', '')
126     text = text.replace('"', "").replace("'", "").replace('\n', '').replace('!', '')
127     text = text.replace("\s", '').replace("\t", '').replace(", ", " ").replace(":", " ")
128     text = text.lower()
129     sentence_separators = ('.',)
130     for sep in sentence_separators:
131         text = text.replace(sep, '_~')
132     text = text.split('_~')
133
134     hash_pairs = backendTables['hash_pairs']
135     ctokens = backendTables['ctokens']
136     KW_map = backendTables['KW_map']
137     stopwords = backendTables['stopwords']
138     hwords = {} # local word hash with word position, to update hash_pairs
139
140     for sentence in text:
141
142         words = sentence.split(" ")
143         position = 0
144         buffer = []
145
146         for word in words:
147
148             if word in KW_map:
149                 word = KW_map[word]
150
151             if word not in stopwords:
152                 # word is single token
153                 buffer.append(word)
154                 key = (word, position)
155                 update_hash(hwords, key) # for word correlation table (hash_pairs)
156                 update_tables(backendTables, word, hash_crawl, backendParams)
157
158             for k in range(1, max_multitoken):
159                 if position > k:
160                     # word is now multi-token with k+1 tokens
161                     word = buffer[position-k] + "~" + word
162                     key = (word, position)
163                     update_hash(hwords, key) # for word correlation table (hash_pairs)
164                     update_tables(backendTables, word, hash_crawl, backendParams)
165

```

```

166         position +=1
167
168     for keyA in hwords:
169         for keyB in hwords:
170
171             wordA = keyA[0]
172             positionA = keyA[1]
173             n_termsA = len(wordA.split("~"))
174
175             wordB = keyB[0]
176             positionB = keyB[1]
177             n_termsB = len(wordB.split("~"))
178
179             key = (wordA, wordB)
180             n_termsAB = max(n_termsA, n_termsB)
181             distanceAB = abs(positionA - positionB)
182
183             if wordA < wordB and distanceAB <= maxDist and n_termsAB <= maxTerms:
184                 hash_pairs = update_hash(hash_pairs, key)
185                 if distanceAB > 1:
186                     ctokens = update_hash(ctokens, key)
187
188     return(backendTables)
189
190
191 #--- [2] Backend: main (create backend tables based on crawled corpus)
192
193 tableNames = (
194     'dictionary', # multitokens (key = multitoken)
195     'hash_pairs', # multitoken associations (key = pairs of multitokens)
196     'ctokens', # not adjacent pairs in hash_pairs (key = pairs of multitokens)
197     'hash_context1', # categories (key = multitoken)
198     'hash_context2', # tags (key = multitoken)
199     'hash_context3', # titles (key = multitoken)
200     'hash_context4', # descriptions (key = multitoken)
201     'hash_context5', # meta (key = multitoken)
202     'hash_ID', # text entity ID table (key = multitoken, value is list of IDs)
203     'hash_agents', # agents (key = multitoken)
204     'full_content', # full content (key = multitoken)
205     'ID_to_content', # full content attached to text entity ID (key = text entity ID)
206     'ID_to_agents', # map text entity ID to agents list (key = text entity ID)
207     'ID_size', # content size (key = text entity ID)
208     'KW_map', # for singularization, map kw to single-token dictionary entry
209     'stopwords', # stopword list
210 )
211
212 backendTables = {}
213 for name in tableNames:
214     backendTables[name] = {}
215
216 stopwords = ('', '-', 'in', 'the', 'and', 'to', 'of', 'a', 'this', 'for', 'is', 'with', 'from',
217             'as', 'on', 'an', 'that', 'it', 'are', 'within', 'will', 'by', 'or', 'its', 'can',
218             'your', 'be', 'about', 'used', 'our', 'their', 'you', 'into', 'using', 'these',
219             'which', 'we', 'how', 'see', 'below', 'all', 'use', 'across', 'provide', 'provides',
220             'aims', 'one', '&', 'ensuring', 'crucial', 'at', 'various', 'through', 'find', 'ensure',
221             'more', 'another', 'but', 'should', 'considered', 'provided', 'must', 'whether',
222             'located', 'where', 'begins', 'any')
223 backendTables['stopwords'] = stopwords
224
225 # agent_map works, but hash structure should be improved
226 # key is word, value is agent (many-to-one). Allow for many-to-many
227 agent_map = {
228     'template': 'Template',
229     'policy': 'Policy',
230     'governance': 'Governance',
231     'documentation': 'Documentation',
232     'best practice': 'Best Practices',
233     'bestpractice': 'Best Practices',
234     'standard': 'Standards',
235     'naming': 'Naming',
236     'glossary': 'Glossary',
237     'historical data': 'Data',
238     'overview': 'Overview',
239     'training': 'Training',
240     'genai': 'GenAI',
241     'gen ai': 'GenAI',

```



```

242         'example': 'Example',
243         'example1': 'Example',
244         'example2': 'Example',
245     }
246
247 KW_map = {}
248 try:
249     IN = open("KW_map.txt", "r")
250 except:
251     print("KW_map.txt not found on first run: working with empty KW_map.")
252     print("KW_map.txt will be created after exiting if save = True.")
253 else:
254     content = IN.read()
255     pairs = content.split('\n')
256     for pair in pairs:
257         pair = pair.split('\t')
258         key = pair[0]
259         if len(pair) > 1:
260             KW_map[key] = pair[1]
261     IN.close()
262 backendTables['KW_map'] = KW_map
263
264 backendParams = {
265     'max_multitoken': 4, # max. consecutive terms per multi-token for inclusion in dictionary
266     'maxDist': 3, # max. position delta between 2 multitokens to link them in hash_pairs
267     'maxTerms': 3, # maxTerms must be <= max_multitoken
268     'extraWeights': # default weight is 1
269     {
270         'description': 0.0,
271         'category': 0.3,
272         'tag_list': 0.4,
273         'title': 0.2,
274         'meta': 0.1
275     }
276 }
277
278
279 local = True # first time run, set to False
280 if local:
281     # get repository from local file
282     IN = open("repository.txt", "r")
283     data = IN.read()
284     IN.close()
285 else:
286     # get anonymized repository from GitHub url
287     import requests
288     url = "https://mltblog.com/3y8MXq5"
289     response = requests.get(url)
290     data = response.text
291
292 entities = data.split("\n")
293 ID_size = backendTables['ID_size']
294
295 # to avoid duplicate entities (takes space, better to remove them in the corpus)
296 entity_list = ()
297
298 for entity_raw in entities:
299
300     entity = entity_raw.split("~")
301     agent_list = ()
302
303     if len(entity) > 1 and entity[1] not in entity_list:
304
305         entity_list = (*entity_list, entity[1])
306         entity_ID = int(entity[0])
307         entity = entity[1].split("{")
308         hash_crawl = {}
309         hash_crawl['ID'] = entity_ID
310         ID_size[entity_ID] = len(entity[1])
311         hash_crawl['full_content'] = entity_raw # do not build to save space
312
313         key_value_pairs = get_key_value_pairs(entity)
314
315         for pair in key_value_pairs:
316
317             if ": " in pair:

```

```

318         key, value = pair.split(": ", 1)
319         key = key.replace("'", "")
320         if key == 'category_text':
321             hash_crawl['category'] = value
322         elif key == 'tags_list_text':
323             hash_crawl['tag_list'] = clean_list(value)
324         elif key == 'title_text':
325             hash_crawl['title'] = value
326         elif key == 'description_text':
327             hash_crawl['description'] = value # do not build to save space
328         elif key == 'tower_option_tower':
329             hash_crawl['meta'] = value
330         if key in ('category_text', 'tags_list_text', 'title_text'):
331             for word in agent_map:
332                 if word in value.lower():
333                     agent = agent_map[word]
334                     if agent not in agent_list:
335                         agent_list = (*agent_list, agent)
336
337         hash_crawl['agents'] = agent_list
338         update_dict(backendTables, hash_crawl, backendParams)
339
340     # [2.1] Create embeddings
341
342     embeddings = {} # multitoken embeddings based on hash_pairs
343
344     hash_pairs = backendTables['hash_pairs']
345     dictionary = backendTables['dictionary']
346
347     for key in hash_pairs:
348         wordA = key[0]
349         wordB = key[1]
350         nA = dictionary[wordA]
351         nB = dictionary[wordB]
352         nAB = hash_pairs[key]
353         pmi = nAB/(nA*nB)**0.5 # try: nAB/(nA + nB - nAB)
354         # if nA + nB <= nAB:
355         #     print(key, nA, nB, nAB)
356         update_nestedHash(embeddings, wordA, wordB, pmi)
357         update_nestedHash(embeddings, wordB, wordA, pmi)
358
359     # [2.2] Create sorted n-grams
360
361     sorted_ngrams = {} # to match ngram prompts with embeddings entries
362
363     for word in dictionary:
364         tokens = word.split('~')
365         tokens.sort()
366         sorted_ngram = tokens[0]
367         for token in tokens[1:len(tokens)]:
368             sorted_ngram += "~" + token
369         update_nestedHash(sorted_ngrams, sorted_ngram, word)
370
371     # print top multitokens: useful to build agents, along with sample prompts
372     # for key in dictionary:
373     #     if dictionary[key] > 20:
374     #         print(key, dictionary[key])
375
376
377     #--- [3] Frontend: functions
378
379     # [3.1] custom pmi
380
381     def custom_pmi(word, token, backendTables):
382
383         dictionary = backendTables['dictionary']
384         hash_pairs = backendTables['hash_pairs']
385
386         nAB = 0
387         pmi = 0.00
388         keyAB = (word, token)
389         if word > token:
390             keyAB = (token, word)
391         if keyAB in hash_pairs:
392             nAB = hash_pairs[keyAB]

```

```

394     nA = dictionary[word]
395     nB = dictionary[token]
396     pmi = nAB / (nA*nB) ** 0.5
397     return(pmi)
398
399 # [3.2] update frontend params
400
401 def cprint(ID, entity):
402     # print text_entity (a JSON text string) nicely
403
404     print("--- Entity %d ---\n" % (ID))
405     keys = (
406         'title_text',
407         'description_text',
408         'tags_list_text',
409         'category_text',
410         'likes_list_text',
411         'link_list_text',
412         'Modified Date',
413     )
414     entity = str(entity).split("~~")
415     entity = entity[1].split("{")
416     key_value_pairs = get_key_value_pairs(entity)
417
418     for pair in key_value_pairs:
419         if ":" in pair:
420             key, value = pair.split(": ", 1)
421             key = key.replace("'", "")
422             if key in keys:
423                 print("> ", key, ":")
424                 value = value.replace("'", '').split("~~")
425                 for item in value:
426                     item = item.lstrip().replace("[", "").replace("]", "")
427                     print(item)
428                 print()
429     return()
430
431 def update_params(option, saved_query, sample_queries, frontendParams, backendTables):
432
433     arr = []
434     ID_to_content = backendTables['ID_to_content']
435     for param in frontendParams:
436         arr.append(param)
437     task = option
438     print()
439
440     if option == '-l':
441         print("Multitoken ignore list:\n", frontendParams['ignoreList'])
442
443     elif option == '-v':
444         print("%3s %s %s\n" % ('Key', 'Description'.ljust(25), 'Value'))
445         for key in range(len(arr)):
446             param = arr[key]
447             value = frontendParams[param]
448             if param != 'show':
449                 print("%3d %s %s" % (key, param.ljust(25), value))
450             else:
451                 print("\nShow sections:\n")
452                 for section in value:
453                     print(" %s %s" % (section.ljust(10), value[section]))
454
455     elif option == '-f':
456         # use parameter set to show as much as possible
457         for param in frontendParams:
458             if param == 'ignoreList':
459                 frontendParams[param] = ()
460             elif param == 'Customized_pmi':
461                 # use customized pmi
462                 frontendParams[param] = True
463             elif param == 'show':
464                 showHash = frontendParams[param]
465                 for section in showHash:
466                     # show all sections in output results
467                     showHash[section] = True
468             elif param == 'maxTokenCount':
469                 frontendParams[param] = 999999999

```

```

470         else:
471             frontendParams[param] = 0
472
473     elif option == '-d':
474         frontendParams = default_frontendParams()
475
476     elif '-p' in option:
477         option = option.split(' ')
478         if len(option) == 3:
479             paramID = int(option[1])
480             if paramID < len(arr):
481                 param = arr[paramID]
482                 value = option[2]
483                 if value == 'True':
484                     value = True
485                 elif value == 'False':
486                     value = False
487                 else:
488                     value = float(option[2])
489                 frontendParams[param] = value
490             else:
491                 print("Error 101: key outside range")
492         else:
493             print("Error 102: wrong number of arguments")
494
495     elif '-a' in option:
496         option = option.split(' ')
497         if len(option) == 2:
498             ignore = frontendParams['ignoreList']
499             ignore = (*ignore, option[1])
500             frontendParams['ignoreList'] = ignore
501         else:
502             print("Error 103: wrong number of arguments")
503
504     elif '-r' in option:
505         option = option.split(' ')
506         if len(option) == 2:
507             ignore2 = ()
508             ignore = frontendParams['ignoreList']
509             for item in ignore:
510                 if item != option[1]:
511                     ignore2 = (*ignore2, item)
512             frontendParams['ignoreList'] = ignore2
513         else:
514             print("Error 104: wrong number of arguments")
515
516     elif '-i' in option:
517         option = option.split(' ')
518         nIDs = 0
519         for ID in option:
520             if ID.isdigit():
521                 ID = int(ID)
522                 # print content of text entity ID
523                 if ID in ID_to_content:
524                     cprint(ID, ID_to_content[ID])
525                 nIDs += 1
526         print("\n %d text entities found." % (nIDs))
527
528     elif option == '-s':
529         print("Size of some backend tables:")
530         print(" dictionary:", len(backendTables['dictionary']))
531         print(" pairs   :", len(backendTables['hash_pairs']))
532         print(" ctokens  :", len(backendTables['ctokens']))
533         print(" ID_size  :", len(backendTables['ID_size']))
534
535     elif '-c' in option:
536         show = frontendParams['show']
537         option = option.split(' ')
538         for section in show:
539             if section in option or '*' in option:
540                 show[section] = True
541             else:
542                 show[section] = False
543
544     elif option == '-q':
545         print("Saved query:", saved_query)

```

```

546
547 elif option == '-x':
548     print("Index Query\n")
549     for k in range(len(sample_queries)):
550         print(" %3d %s" %(k, sample_queries[k]))
551
552 print("\nCompleted task: %s" %(task))
553 return(frontendParams)
554
555 # [3.3] retrieve info and print results
556
557 def print_results(q_dictionary, q_embeddings, backendTables, frontendParams):
558
559     dictionary = backendTables['dictionary']
560     hash_pairs = backendTables['hash_pairs']
561     ctokens = backendTables['ctokens']
562     ID_to_agents = backendTables['ID_to_agents']
563     ID_size = backendTables['ID_size']
564     show = frontendParams['show']
565
566     if frontendParams['bypassIgnoreList'] == True:
567         # bypass 'ignore' list
568         ignore = ()
569     else:
570         # ignore multitokens specified in 'ignoreList'
571         ignore = frontendParams['ignoreList']
572
573     if show['Embeddings']:
574         # show results from embedding table
575
576         local_hash = {} # used to not show same token 2x (linked to 2 different words)
577         q_embeddings = dict(sorted(q_embeddings.items(),key=lambda item: item[1],reverse=True))
578         print()
579         print("%3s %s %1s %s %s"
580               %('N','pmi'.ljust(4),'F','token [from embeddings]'.ljust(35),
581                 'word [from prompt]'.ljust(35)))
582         print()
583
584         for key in q_embeddings:
585
586             word = key[0]
587             token = key[1]
588             pmi = q_embeddings[key]
589             ntk1 = len(word.split('~'))
590             ntk2 = len(token.split('~'))
591             flag = " "
592             nAB = 0
593             keyAB = (word, token)
594
595             if word > token:
596                 keyAB = (token, word)
597             if keyAB in hash_pairs:
598                 nAB = hash_pairs[keyAB]
599             if keyAB in ctokens:
600                 flag = '*'
601             if ( ntk1 >= frontendParams['embeddingKeyMinSize'] and
602                 ntk2 >= frontendParams['embeddingValuesMinSize'] and
603                 pmi >= frontendParams['min_pmi'] and
604                 nAB >= frontendParams['nABmin'] and
605                 token not in local_hash and word not in ignore
606             ):
607                 print("%3d %4.2f %1s %s %s"
608                       %(nAB,pmi,flag,token.ljust(35),word.ljust(35)))
609                 local_hash[token] = 1 # token marked as displayed, won't be shown again
610
611         print()
612         print("N = occurrences of (token, word) in corpus. F = * if contextual pair.")
613         print("If no result, try option '-p f'.")
614         print()
615
616     sectionLabels = {
617         # map section label to corresponding backend table name
618         'Dict' : 'dictionary',
619         'Pairs' : 'hash_pairs',
620         'Category' : 'hash_context1',
621         'Tags' : 'hash_context2',

```

```

622     'Titles': 'hash_context3',
623     'Descr.': 'hash_context4',
624     'Meta': 'hash_context5',
625     'ID': 'hash_ID',
626     'Agents': 'hash_agents',
627     'Whole': 'full_content',
628 }
629 local_hash = {}
630 agentAndWord_to_IDs = {}
631
632 for label in show:
633     # labels: 'Category', 'Tags', 'Titles', 'Descr.', 'ID', 'Whole', 'Agents', 'Embeddings'
634
635     if show[label] and label in sectionLabels:
636         # show results for section corresponding to label
637
638         tableName = sectionLabels[label]
639         table = backendTables[tableName]
640         local_hash = {}
641         print(">>> RESULTS - SECTION: %s\n" % (label))
642
643         for word in q_dictionary:
644
645             ntk3 = len(word.split('~'))
646             if word not in ignore and ntk3 >= frontendParams['ContextMultitokenMinSize']:
647                 content = table[word] # content is a hash
648                 count = int(dictionary[word])
649                 for item in content:
650                     update_nestedHash(local_hash, item, word, count)
651
652         for item in local_hash:
653
654             hash2 = local_hash[item]
655             if len(hash2) >= frontendParams['minOutputListSize']:
656                 print(" %s: %s [%d entries]" % (label, item, len(hash2)))
657                 for key in hash2:
658                     print(" Linked to: %s (%s)" % (key, hash2[key]))
659                     if label == 'ID' and item in ID_to_agents:
660                         # here item is a text entity ID
661                         LocalAgentHash = ID_to_agents[item]
662                         local_ID_list = ()
663                         for ID in LocalAgentHash:
664                             local_ID_list = (*local_ID_list, ID)
665                         print(" Agents:", local_ID_list)
666                         for agent in local_ID_list:
667                             key3 = (agent, key) # key is a multitoken
668                             update_nestedHash(agentAndWord_to_IDs, key3, item)
669
670                 print()
671         print()
672
673 print("Above results based on words found in prompt, matched back to backend tables.")
674 print("Numbers in parentheses are occurrences of word in corpus.\n")
675
676 print("-----")
677 print(">>> RESULTS - SECTION: (Agent, Multitoken) --> (ID list)")
678 print(" empty unless labels 'ID' and 'Agents' are in 'show'.\n")
679 hash_size = {}
680 for key in sorted(agentAndWord_to_IDs):
681     ID_list = ()
682     for ID in agentAndWord_to_IDs[key]:
683         ID_list = (*ID_list, ID)
684         hash_size[ID] = ID_size[ID]
685     print(key, "-->", ID_list)
686 print("\n ID Size\n")
687 for ID in hash_size:
688     print("%4d %5d" % (ID, hash_size[ID]))
689
690 return()
691
692
693 #--- [4] Frontend: main (process prompt)
694
695 # [4.1] Set default parameters
696
697 def default_frontendParams():

```

```

698
699     frontendParams = {
700         'embeddingKeyMinSize': 1, # try 2
701         'embeddingValuesMinSize': 2,
702         'min_pmi': 0.00,
703         'nABmin': 1,
704         'Customized_pmi': True,
705         'ContextMultitokenMinSize': 1, # try 2
706         'minOutputListSize': 1,
707         'bypassIgnoreList': False,
708         'ignoreList': ('data',),
709         'maxTokenCount': 100, # ignore generic tokens if large enough
710         'show': {
711             # names of sections to display in output results
712             'Embeddings': True,
713             'Category' : True,
714             'Tags'      : True,
715             'Titles'    : True,
716             'Descr.'    : False, # do not built to save space
717             'Whole'     : False, # do not built to save space
718             'ID'        : True,
719             'Agents'    : True,
720         }
721     }
722     return(frontendParams)
723
724 # [4.2] Purge function
725
726 def distill_frontendTables(q_dictionary, q_embeddings, frontendParams):
727     # purge q_dictionary then q_embeddings (frontend tables)
728
729     maxTokenCount = frontendParams['maxTokenCount']
730     local_hash = {}
731     for key in q_dictionary:
732         if q_dictionary[key] > maxTokenCount:
733             local_hash[key] = 1
734     for keyA in q_dictionary:
735         for keyB in q_dictionary:
736             nA = q_dictionary[keyA]
737             nB = q_dictionary[keyB]
738             if keyA != keyB:
739                 if (keyA in keyB and nA == nB) or (keyA in keyB.split('~')):
740                     local_hash[keyA] = 1
741     for key in local_hash:
742         del q_dictionary[key]
743
744     local_hash = {}
745     for key in q_embeddings:
746         if key[0] not in q_dictionary:
747             local_hash[key] = 1
748     for key in local_hash:
749         del q_embeddings[key]
750
751     return(q_dictionary, q_embeddings)
752
753 # [4.3] Main
754
755 print("\n") #
756 input_ = " "
757 saved_query = ""
758 get_bin = lambda x, n: format(x, 'b').zfill(n)
759 frontendParams = default_frontendParams()
760 sample_queries = (
761     'parameterized datasets map tables sql server',
762     'data load templates importing data database data warehouse',
763     'pipeline extract data eventhub files',
764     'blob storage single parquet file adls gen2',
765     'eventhub files blob storage single parquet',
766     'parquet blob eventhub more files less storage single table',
767     'MLTxQuest Data Assets Detailed Information page',
768     'table asset',
769 )
770
771 while len(input_) > 0:
772
773     print()

```



```

774 print("-----")
775 print("Command menu:\n")
776 print(" -q          : print last non-command prompt")
777 print(" -x          : print sample queries")
778 print(" -p key value : set frontendParams[key] = value")
779 print(" -f          : use catch-all parameter set for debugging")
780 print(" -d          : use default parameter set")
781 print(" -v          : view parameter set")
782 print(" -a multitoken : add multitoken to 'ignore' list")
783 print(" -r multitoken : remove multitoken from 'ignore' list")
784 print(" -l          : view 'ignore' list")
785 print(" -i ID1 ID2 ... : print content of text entities ID1 ID2 ...")
786 print(" -s          : print size of core backend tables")
787 print(" -c F1 F2 ... : show sections F1 F2 ... in output results")
788 print("\nTo view available sections for -c command, enter -v command.")
789 print("To view available keys for -p command, enter -v command.")
790 print("For -i command, choose IDs from list shown in prompt results.")
791 print("For standard prompts, enter text not starting with '-' or digit.")
792 print("-----\n")
793
794 input_ = input("Query, command, or integer in [0, %d] for sample query: "
795               %(len(sample_queries)-1))
796 flag = True # False --> query to change params, True --> real query
797 if input_ != "" and input_[0] == '-':
798     # query to modify options
799     frontendParams = update_params(input_, saved_query,
800                                   sample_queries, frontendParams,
801                                   backendTables)
802     query = ""
803     flag = False
804 elif input_.isdigit():
805     # actual query (prompt)
806     if int(input_) < len(sample_queries):
807         query = sample_queries[int(input_)]
808         saved_query = query
809         print("query:", query)
810     else:
811         print("Value must be <", len(sample_queries))
812         query = ""
813 else:
814     # actual query (prompt)
815     query = input_
816     saved_query = query
817
818 query = query.split(' ')
819 new_query = []
820 for k in range(len(query)):
821     token = query[k].lower()
822     if token in KW_map:
823         token = KW_map[token]
824     if token in dictionary:
825         new_query.append(token)
826 query = new_query.copy()
827 query.sort()
828 q_embeddings = {}
829 q_dictionary = {}
830
831 for k in range(1, 2*len(query)):
832
833     binary = get_bin(k, len(query))
834     sorted_word = ""
835     for k in range(0, len(binary)):
836         if binary[k] == '1':
837             if sorted_word == "":
838                 sorted_word = query[k]
839             else:
840                 sorted_word += "~" + query[k]
841
842     if sorted_word in sorted_ngrams:
843         ngrams = sorted_ngrams[sorted_word]
844         for word in ngrams:
845             if word in dictionary:
846                 q_dictionary[word] = dictionary[word]
847             if word in embeddings:
848                 embedding = embeddings[word]
849                 for token in embedding:

```

```

850         if not frontendParams['Customized_pmi']:
851             pmi = embedding[token]
852         else:
853             # customized pmi
854             pmi = custom_pmi(word, token, backendTables)
855         q_embeddings[(word, token)] = pmi
856
857     # if len(query) == 1:
858     #     # single-token query
859     #     frontendParams['embeddingKeyMinSize'] = 1
860     #     frontendParams['ContextMultitokenMinSize'] = 1
861
862     distill_frontendTables(q_dictionary, q_embeddings, frontendParams)
863
864     if len(input_) > 0 and flag:
865         print_results(q_dictionary, q_embeddings, backendTables, frontendParams)
866
867
868     #--- [5] Save backend tables
869
870     def create_KW_map(dictionary):
871         # singularization
872         # map key to KW_map[key], here key is a single token
873         # need to map unseen prompt tokens to related dictionary entries
874         # example: ANOVA -> analysis~variance, ...
875
876         OUT = open("KW_map.txt", "w")
877
878         for key in dictionary:
879             if key.count('~') == 0:
880                 j = len(key)
881                 keyB = key[0:j-1]
882                 if keyB in dictionary and key[j-1] == 's':
883                     if dictionary[key] > dictionary[keyB]:
884                         OUT.write(keyB + "\t" + key + "\n")
885                 else:
886                     OUT.write(key + "\t" + keyB + "\n")
887         OUT.close()
888         return()
889
890     save = True
891     if save:
892         create_KW_map(dictionary)
893         for tableName in backendTables:
894             table = backendTables[tableName]
895             OUT = open('backend_' + tableName + '.txt', "w")
896             OUT.write(str(table))
897             OUT.close()
898
899     OUT = open('backend_embeddings.txt', "w")
900     OUT.write(str(embeddings))
901     OUT.close()
902
903     OUT = open('backend_sorted_ngrams.txt', "w")
904     OUT.write(str(sorted_ngrams))
905     OUT.close()
906

```

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 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 has 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 p_v . At the basic level, $p_v(A)$ 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 p_v because they have not accumulated much traffic yet. To correct for this, see lines 127–136 in the code. From now on, p_v refers to normalized pageview counts also adjusted for time. The pageview for a multi-token t is then defined as

$$p_v(t) = \frac{1}{|S(t)|} \cdot \sum_{A \in S(t)} p_v(A), \quad (8.2)$$

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 $p_v(C)$, the pageview count attached to a category C , by averaging p_v '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 $p_{v_0}(A)$ for an article A inside or outside the training set. The formula is

$$p_{v_0}(A) = \frac{1}{W_A} \cdot \sum_{t \in T(A)} w_t \cdot p_v(t), \quad (8.3)$$

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 p_{v_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

- n*-gram, 2
- abbreviation dictionary, 8
- action, 2, 6
- agent, 6, 10, 40
- ANN (approximate nearest neighbor search), 38
- augmentation, 9
 - augmented knowledge graph, 9
 - taxonomy augmentation, 9
- auto-correct, 8
- context
 - contextual pairs, 39
 - contextual tables, 39
 - contextual tokens, 39
- crawling
 - smart crawling, 2, 39
- distillation, 9
- DNN (deep neural network), 39
- embedding, 12, 39
 - variable length embedding, 2
- fine-tuning
 - in real time, 10
 - LLM parameters, 40
- GAN (generative adversarial network), 7
 - NoGAN, 7
- hash table
 - nested hash, 2, 39, 40
- Hellinger distance
 - multivariate, 39
- in-memory database, 9
- JSON, 39
- key-value table, 2
- Kolmogorov-Smirnov distance
 - multivariate, 39
- latency, 9
- LLama, 39
- LLM (large language model)
 - agentic LLM, 6
 - for clustering, 7
 - for predictive analytics, 7
 - in-memory, 9
 - LLM router, 2, 40
 - multi-LLM, 10
 - sub-LLM, 10
 - xLLM, 10
- loss function
 - adaptive, 39
- mixture of experts, 41
- NLP (natural language processing), 8
- PMI (pointwise mutual information), 2, 3, 12
- PRNG (pseudo-random number generator), 7
- RAG (retrieval augmented generation), 9
- reproducibility, 7
- scalability, 9
- seed (random number generator), 7
- singularization, 8
- stemming, 8
- stopwords, 2
- synonyms dictionary, 8
- text entity, 6, 40
 - sub-entity, 10
- token
 - contextual, 39
 - multitoken, 2, 12
- transformer, 7, 39
- xLLM, 10