Ad-hoc Retrieval of Scientific Documents on the LIVIVO Search Portal

Anh Huy Matthias Tran[a], Andreas Kruff[a], Joshua Thos[a], Constantin Krah[a], Michelle Reiners[a], Fabian Ax[a], Saskia Brech[a], Sascha Gharib[a] and Verena Pawlas[a]

[a] *TH Köln – University of Applied Sciences, Cologne, Germany*

Abstract

This paper documents the participation in LiLAS @ CLEF2021 concerning the ad-hoc retrieval of scientific documents for LIVIVO and illustrates several approaches on tackling the problem of multi-language search on the scientific corpus found on LIVIVO using Elasticsearch and the STELLA framework with two different search systems: Ingestion Pipelines to offer multi-language indexing with half the indexing space and an elaborate preprocessing process in tandem with language libraries such as Google Translate and SpaCy. The results show, however, that while relevant results appear to be of higher quality in comparison to the base system found on LIVIVO, the general search and relevance performance of these two systems turn out to be demonstrably worse than the base system in terms of click metrics.

Keywords

Elasticsearch, Information Retrieval, Preprocessing, Tokenization, Multi-Language

1. Introduction

In LiLAS [1] one of the two tasks involved developing an ad-hoc retrieval system for scientific documents for the multi-source life science search portal LIVIVO over two separate rounds, where tackling the problems and complexity of supporting multi-language search queries in an effective manner quickly became an important aspect. Therefore, the aim of this paper lies in implementing and testing multiple measures ranging from multi-fields, ingestion processes and involving the use of language libraries like SpaCy [2] and Google Translate [3] in an elaborate preprocessing measure in order to improve the search system with respect to retrieving relevant queries for both English and German language. For more details on the different tasks and the setup of the lab, please see the official lab overview [4].

With the help of the STELLA Infrastructure for Living Labs [5], two different lines of search systems by two groups were developed over the two rounds and implemented into the working environment of the LIVIVO platform. While one line of systems(lemuren_elk and LEO) is completely relying on the built-in functions of Elasticsearch [6], the other line of systems(save_fami and LEPREP) outsources the preprocessing of the documents to a Python application with the aim to find out which approach performs better. The Team-Draft-Interleaving (TDI) [7] process allows us to deliver search results from both the experimental systems and the base system to a user, allowing us to directly compare the performance of the two systems with the current existing base system.

CLEF 2021 – Conference and Labs of the Evaluation Forum, September 21–24, 2021, Bucharest, Romania

Table 1
List of submitted Systems for each Round in CLEF2021

Round 1	Round 2
lemuren_elk	lemuren_elastic_only **(LEO)**
save_fami	lemuren_elastic_preprocessing **(LEPREP)**

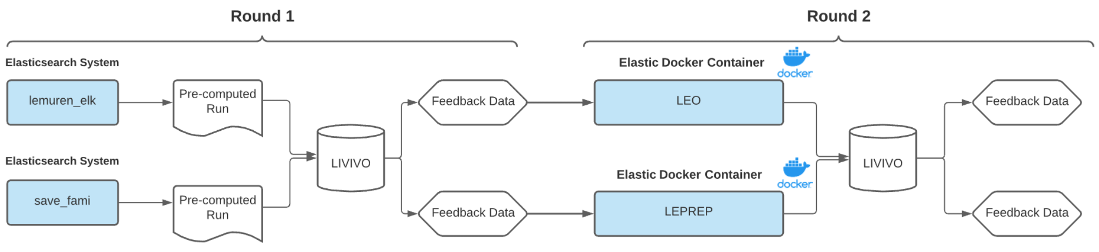

Figure 1: Overview of the provided Systems for each round

In this task, the retrieval approaches are evaluated in two separate rounds, where the first preliminary round enables us to first test out the first approaches with pre-computed runs in an easy and fast to implement manner and then act upon the received feedback to create an improved, fully dockerized version for the following second round. As such, the paper will describe the implemented retrieval approaches in the first round and then the two subsequent improved and dockerized implementations in the subsequent second round: lemuren_elastic_only **(LEO)** and lemuren_elastic_preprocessing **(LEPREP)** (See Table 1 and Figure 1).
Consequently, the paper as a whole is structured as follows: In Section [2], we give a short overview of the many tools and resources utilized in this paper. In Section [3], the implementation of the first round of this task is discussed. In Section [4] the frameworks, pipelines and implementation of our two dockerized final systems, LEO and LEPREP, are explained in closer detail. In Section [5], the performance of the systems from both the first and second round are analysed, evaluated and then compared with the existing base system on LIVIVO according to self-defined metrics. Finally, in Section [6], we discuss our results in a short conclusion and provide pointers for possible future improvements and research.

2. Tools & Resources

2.1. LIVIVO Data set

The basis of this research paper deals with the LIVIVO search portal, which contains a huge data set of more than 58 million references to various life science literature. It contains a various amount of different literature, crossing a multi-language domain between German, French, English and other languages [8]. For this project, a document data subset of over 30 million documents from various databases from LIVIVO were provided, which includes metadata such as titles, abstracts and controlled vocabulary such as MESH and CHEM tokens, of which only a partial subset was indexed on our own systems for the first round, because the given headqueries and the corresponding candidates were also generated on the reduced document corpus. For the ad-hoc retrieval system in round 2 the complete corpus was indexed so that individual user queries could be applied on the full corpus. While the document set is predominantly English, various other document languages such as French or German can be found within the data set. This document data set is complimented with a candidate list with various most common queries used by users of LIVIVO, alongside the specific query frequency and a search result list of candidates ordered by relevance.

2.2. spaCy

spaCy is an open-source library for the Python language and is used for natural language processing (NLP). The library supports over 64 languages and includes NLP-features such as tokenization, part-of-speech tagging (POS), lemmatization or named entity recognition (NER), which were used for the preprocessing approach to recognize specific tokens, and offers a plenty of prebuilt statistical models for different languages and purposes for usage such as SciSpacy, which is used for specifically scientific language [9]. In our project the library is used in the save_fami and the LEPREP system for preprocessing the documents.

2.3. Elasticsearch

Elasticsearch is an open source search and analytics engine for textual, numeric and geospatial data in structured and unstructured form. In this project Elasticsearch was used to create a search system and index of the provided LIVIVO data set. Using Elasticsearch, the technical foundation to preprocess data was built, also it was used to build indexes and enable subsequent search queries[6].

2.4. Docker

Docker is an application platform primarily used for distributing applications and services, the so-called deployment. It enables to deploy the different search systems in isolation without dependency on any other system in an easy and scalable manner. This means that they are significantly more efficient in terms of system resources than "hypervisors" for example. Instead of virtualizing the whole set of hardware, separate containers are run on a singular Linux instances. The biggest advantage is that less resources are needed on private systems and can run processes faster than on a "pure" virtual machine while significantly simplifying deployment in conjunction with the STELLA framework. Docker was used to to implement the search system with individual Elasticsearch and Kibana instances[10].

2.5. STELLA

As a multi-container-application, STELLA provides a user-focused evaluation approach that allows users to deploy and evaluate search and recommendation systems that they wish to implement in a live environment [5]. For this purpose, STELLA accepts the submission of either pre-computed runs or full docker images that follow the pre-defined REST-API, enabling full reproducibility of the system. In practice, following the integration of an experimental search system as a docker-image into STELLA, its indexing-endpoint is called when the system in question is called for the first time, where it builds the index with the provided data from LIVIVO. Then for the whole userbase using LIVIVO, their search query gets sent to the ranking-endpoint of both the docker system and the base system, which then return a list of relevant document-ids formed from a mix of results from both the implemented experimental docker system and the base system. Following this, the STELLA app will log user-feedback and activity data such as clicks, downloads, dwell times but also other relevant performance indicators such as wins, losses, the number of sessions and impressions and CTR metrics between the experimental search systems and the on-production base system.

3. Approach & Implementation: First Round

The first prototypes of the search systems(save_fami and lemuren_elk) involve a simple re-ranking of the search results of a given a set of the most common headqueries on LIVIVO, called a pre-computed run, which then transfer this specific re-ranking onto the live system on LIVIVO for just these headqueries. A user querying these specific headqueries on LIVIVO would then retrieve these altered search results. This approach however, would not yield enough raw click data for any kind of adequate analysis, which we solve with the docker container approach mentioned in Section [4] for the second round.

3.1. SYSTEM 1: lemuren_elk

The **lemuren_elk** system[1] is fully based on built-in functions from Elasticsearch, and the index settings were based on a previous Elasticsearch configuration from the TREC-COVID Challenge [11], a challenge to study methods on how quickly a reliable information retrieval system specialized on COVID-relevant documents can be built [11]. For long text fields like the abstract field the DFR similarity model [12]was determined as a good metric in this challenge, while for the title, the LMJ similarity performed best in comparison to a fully standard base system using BM25 similarity. Based on this observation, these two similarity models were applied on different fields. These exact settings were also used in the second round and will be described in detail later on. Furthermore, like the keyword field, multi-fields were applied for the title field, which was an attempt to implement an exact-match for the system by using the raw title as one keyword. Furthermore the synonym list adapted for common terms and medical terms especially in regard to COVID specific terms were re-used. In general, when compared to the systems implemented in the second round, this system is not language specific, which means that the synonyms, the acronyms, the stopword removal and the stemming procedure were specialized for English language only.

For the query settings, a different approach in comparison to the TREC-COVID settings was implemented, since the head queries that were retrieved from the supplied head queries list had a significant amount of boolean operators. As such, instead of the built-in "multi_match" function,

[1]https://github.com/AH-Tran/DIS17.1-Suchmaschinentechnologie

the "query_string" operator was implemented, because it allows the parsing of multiple boolean operators inside one query. In general three query requests for every query were created. First the query was applied to the three title multi-fields, then it was applied on the abstract and after that it was applied on the "MESH" field. As such, this version already explicitly reads and parses queries with boolean operator logic in mind and applies boosting to the various used fields as described in the later sections.

To roughly estimate a baseline quality, this result list was then compared with a fixed candidate list from the live base system for the query-list and then complemented with any missing candidate for every query with a fixed minimal relevance score of 0.01.

3.2. SYSTEM 2: save_fami

The **save_fami**[2] system is characterized by realizing the text preprocessing in Python with usage of spaCy. The choice to use spaCy is mostly based on the fact that great results were already achieved in the TREC-COVID Challenge in a previously finished group project using the CORD-19 data set [11]. With the help of the subject-specific models, subject terms such as the MESH terms should be better processed. In order to prepare the content of the index properly for queries, it is necessary to analyze and convert the content from the metadata into proper tokens that can be used for an accurate search.

For the implementation, Elasticsearch (Elastic) as the search engine combined with a Python application was used. Elastic was used for the indexing process and the actual retrieval application. With Python, the remaining tasks, such as creating an indexing pipeline or configuring the search, were realized.

Finally, the save_fami system appended missing candidates to the results list. However, this system ranked the remaining candidates in descending order according to their position in the candidate list. A so-called pseudo-score was created, which started at the value 1 and was decreased by the value 0.001 after each candidate.

Further details and improvements to this system can be found in the description of its continuation system LEPREP in Section [4.2].

[2]https://github.com/dis22-lilas/save_fami

4. Approach & Implementation: Second Round

In the second round, we implemented our systems mentioned in the first round as a fully dockerized version implemented in the live LIVIVO search engine environment and improved them further. The docker container approach allows our experimental systems to dynamically work with all kind of queries sent to LIVIVO instead of only handling the searches for a pre-defined set of headqueries, enabling us to collect click data on a much bigger scale than we did in the first round (see: Section [5]).

4.1. SYSTEM 3: lemuren_elastic_only (LEO)

The lemuren_elastic_only **(LEO)** system[3] implemented for the **second round** is characterized by the fact that mainly only Elasticsearch built-in function were used to create the index and handle the querying process. It is a continuation of the previously mentioned lemuren_elk system (see: 3.1).

In general, it is a straight forward approach, where different languages are indexed in different fields of the same title multi-fields and different language-dependent analyzers for the title and abstract fields are used to tokenize them appropriately. In addition, two different similarity modules are used for longer and shorter texts found in the title and abstracts fields specifically (See Appendix Code-Block A.1).

An overview of LEO's general pipeline can be seen in Figure 2.

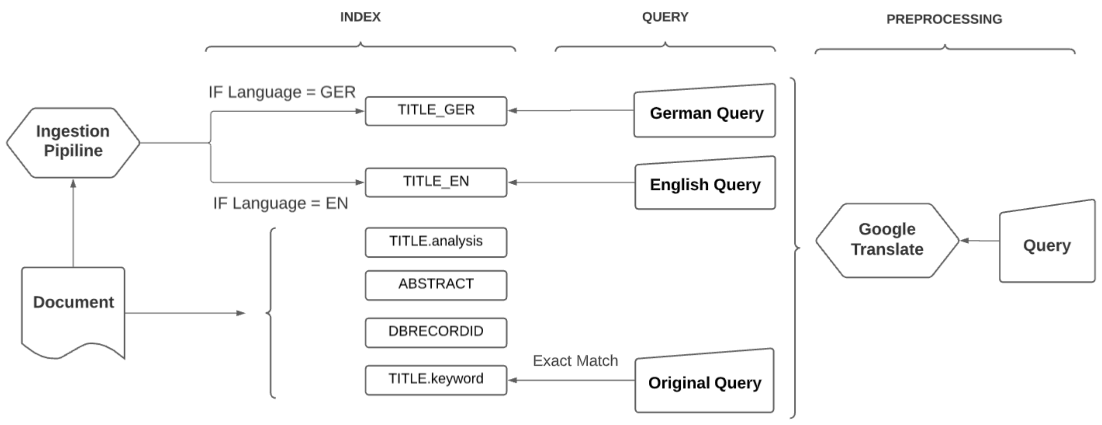

Figure 2: LEO overview detailing the Pipeline from Document to Ingestion Pipeline to Index and the Query

[3]https://github.com/dis22-lilas/lemuren_elastic_only

4.1.1. Multi-field Indexing

The title-field in the index is separated into four different multi-fields in order to enable a more specific searching of the titles of each documents, each with their own specific language-specific analyzer:

- **TITLE_EN:** Titles processed by English-specific language analyzers
- **TITLE_GER:** Titles processed by German-specific language analyzers
- **TITLE.analysis:** Titles processed by scientific language analyzers
- **TITLE.keyword:** The whole title as a single keyword

While TITLE_EN and TITLE_GER follow a language-specific logic defined by an implemented ingestion pipeline, TITLE.analysis contains added acronyms and TITLE.keywords contains the whole title of document as one singular token in order to allow exact-match keyword search for document titles(See Appendix Code-Block A.1).

4.1.2. Ingestion Pipeline

In order to apply language specific analyzers on the title field while avoiding the drawback of indexing all titles as separate German and English indexes, an ingestion pipelines was used to create new fields, based on specified conditions, given the content of the original title metadata(See Appendix Code-Block A.2). This allows us the utility of language-specific indexes while retaining the same space size of a regular index.

In this specific use case, the assigned language field of each document was parsed and used to roughly determine the document's language and subsequently either create a title field for German language (TITLE_GER) or English language (TITLE_EN). This process allows an analysis of the language and to tokenize the title in this language-specific manner without blowing up the index size by indiscriminately creating a German and English index for all documents [13].

4.1.3. Acronyms

Concerning the handling of acronyms, the built-in "word_delimiter" function was used. It allows the system to split words into subwords and performs optional transformations on subword groups before indexing.

The "catenate_all" option catenates all subwords parts into one token, like "wi-fi-4000" -> "wifi4000", if set on true. The generate_word_parts parameter was set to false, so that words with case transition (e.g. "PowerShot" -> "Power" "Shot") will not get split on that transition. The generate_number_part parameter was also set to false, to prevent number sequences not get split at special characters (e.g. "500-42" -> "500" "42"). At last the preserve_original parameter was set on true. With that parameter on set true, the original word will also be retained (e.g. "500-42" -> "500-42" "500" "42"). The split_on_numerics parameter was not changed, defaulting true. This parameter will split a word, if a number appears in it (e.g. "j2se" -> "j" "2" "se"). Finally, the stem_english_parameter was not changed which defaulting to true. This setting removes trailings from subwords (e.g. "O'Neil's" -> "O", "Neil").

The main motivation behind this implementation was to cover multiple options for acronyms so that semantically identical terms such as "USA" and "U.S.A" would both be recognized as the same term and thus would be matched be matched for the same document, preventing a bad user experience due to small spelling differences [14].

4.1.4. Querying

The querying approach assumed that most of the users would form their queries in either Enlish or German. Under this assumption, with the help of the Google Translator API, each query gets translated in both an English or German version for the purpose of achieving language independent results.

Furthermore, any boolean "UND" and "ODER" operators were translated into proper "AND" and "OR" operators, as Elasticsearch's query_string operator only supports english boolean operators within its logic, since the system received many queries using german boolean operators in the first few days of round two, prompting a system update enabling this feature after the fourth day of experiment.

As the query gets sent into Elasticsearch, it is processed further with various analyzers in the multi-fields to automatically delete any language-specific stop words, tokenize, stem and apply additional asciifolding to prevent encoding problems.

The query logic in Elasticsearch is implemented specifically in a three-way process, where the English, German and original query are matched against the index in their specific aforementioned multifields:

- **query_tokenized_eng** for TITLE_EN
- **query_tokenized_ger** for TITLE_GER
- **query_raw** for TITLE.analysis

In detail, for every query language (original, English, German), the system searches in every available multi-field in the title and only account for the score retrieved from the best match. While the abstract was also taken into account, a boost on the language-specific title fields by a factor of 2 was applied. Also a match for the original raw query with the TITLE.keywords fields and a boosting factor of 10 on the condition of a match of 75% was applied in order to implement an exact match search where an exact match with a title will always be ranked first in the resulting search list. To fulfill the need of the user without forcing him to match the title of the document exactly we used a parameter called "minimum-should-match", an built-in function from Elasticsearch, that allows a deviation of the exact match by a certain amount.

4.1.5. Fuzzyness

In order to further prevent poor search results for mistyped queries, the LEO system is complemented by a fuzzymatch and a fuzzy query-expansion. The parameters resulted from experiments with the "fuzzy_match_expansions" parameter, which limits the maximum amount of terms to which the query expands for fuzzy matching.

Additionally the "fuzzy_prefix_length" was set to the default of 0 which defines the number of beginning characters left unchanged for fuzzy matching. The parameter "fuzziness" was set to "AUTO" to dynamically define the maximum edit distance allowed for matching. At last "fuzzy_transpositions" is to true so that edits for fuzzy matching include transpositions of two adjacent characters ($ab \rightarrow ba$) [15].

4.1.6. Similarity Models

Concerning the Ranking Models, two different similarity modules and scoring methods have been implemented:

- **DFR:** Divergence from randomness.
- **LMJelinekMercer:** A Language model based on the Jelinek-Mercer smoothing method.

These ranking models were quickly tested and evaluated for their search performance in the title and abstract fields respectively in a previous TREC-Covid Challenge, in order to determine the most optimal one for similar scientific corpus specified for documents relevant to COVID-19 [11]. Both data sets from the TREC-Covid Challenge and LIVIVO are based on scientific publications in the medical field and thus it was assumed that the approaches from the TREC-Covid Challenge are also generally applicable to for this project. Under these assumption, two different similarity modules were used for the fields; LMJelinekMercer for the title multi-fields and the DFR algorithm for the longer texts found in the abstracts.

4.1.7. Analyzer

For this implementation, four different analyzers were applied to process different kinds of given data and the incoming queries.

German Analyzer This analyzer processes German words. It sets all words to lowercase and normalized them, and then stems the words and removes any German stopwords and is applied for the German title field TITLE_GER. Additionally, it uses the German normalizer in order to normalize special characters like "ß", "ä", "ö", "ü", "ae", "oe", "ue" and replaces these by more easily parsed letters such as "ss", "a", "o", "u", "a", "o", "u". "Ue" is only replaced if it is not followed by a q [16].

English Analyzer This analyzer is basically the same as the German analyzer with the difference that it removes English stopwords, uses an English stemmer and that it does not normalize the words, because there is no need to do it for English language.

Query Analyzer The query analyzer has a few more parameter than the English or German analyzer. It processes the incoming queries and sets them to lowercase, compares the query to a given keyword list so that these specific words wont be separated from each other like "Vitamin D" will not be seperated into "Vitamin" + "D". This analyzer also removes any English stopwords and includes an asciifolding filter. This filter converts tokens like "á" into "a".

LIVIVO Analyzer This analyzer has the same parameters as the query analyzer, but it also has a filter for acronyms. With this filter acronyms like "U.S.A" will be taken as one token and not as single digits.

4.2. SYSTEM 4: lemuren_elastic_preprocessing (LEPREP)

The lemuren_elastic_preprocessing **(LEPREP)** system[4] implemented for the **second round** completely takes over the preprocessing from the save_fami system from the first round, utilizing spaCy to realize text preprocessing and creating a custom indexing pipeline(see: 3.2).

The main aspect of improvement over the iteration in the first round lies in the correct implementation of boolean queries as they were removed in the first round due to the translation logic, significant performance improvements through the use of parallelization with the use of the pandarallel module[17] and the dockerization enabling us to integrate the LEPREP search system into the live LIVIVO system for significantly more click data.

An overview of the general pipeline of the system can be seen in Figure 3.

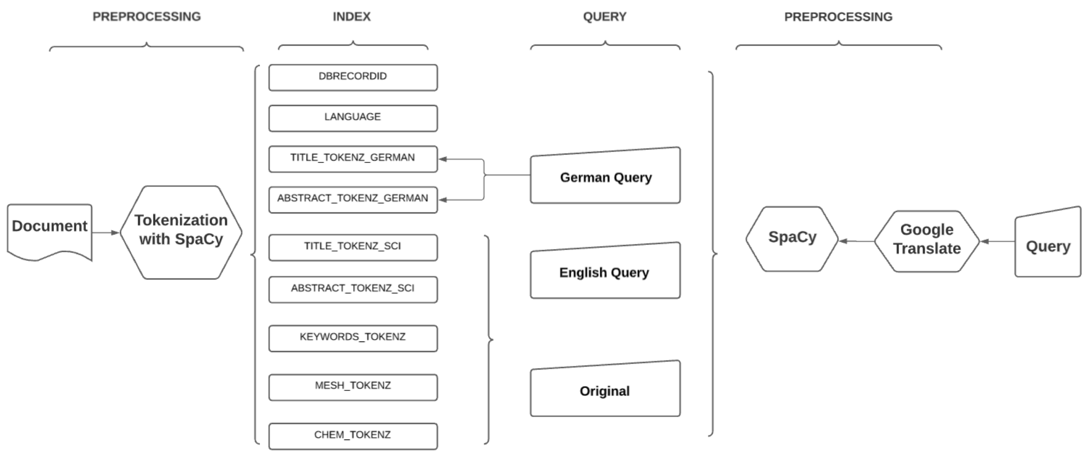

Figure 3: LEPREP Overview detailing the Pipeline between the raw Data Set, the Index Creation and the Query Process.

4.2.1. Tokenization

The indexing pipeline was built with Python and the spaCy module. For this purpose a reference that introduced the approaches of different ranking methods in Elastic was quite helpful [18]. Irrelevant characters and stop words - even words that are considered stop words in the specific scientific context - are removed. The remaining words are put into their base form by using lemmatization.

By means of a comprehensive NLP model of SpaCy with the designation en_core_sci_lg, which is suitable for the treated scientific topics, the subject-related tokens MESH and CHEM were generated which are of primary relevance for the further course [19]. Additionally, the NLP model de_core_news_lg containing a preprocessing pipeline optimized for German language was used for processing German vocabulary [20]. But since Elastic itself also uses a tokenizer, it was necessary to create a new token only with the use of each blank character [21].

[4]https://github.com/dis22-lilas/elastic_lemuren_preprocessing

4.2.2. Language Handling

Googletrans is a free and publicly available Python library that calls language detection and translation methods using the Google Translate Ajax API. Due to changes in the Google Translation API, the original module "googletrans 3.0.0" [22] may no longer work. For this reason the alternative "google-trans-new 1.1.9" [3] was implemented, which has fixed the problems of the previous module.

Since the documents in the corpus are available in different languages, different language queries were used. Due to the fact that most of the queries are formulated in German or English and the majority of the documents are available in German or English, the queries were translated into the languages mentioned above. However, some queries contain technical terms, which is why the translation generated rather less meaningful search terms in some places. Therefore, not only the English and German queries, but also the original queries were sent.

It is also important to note that Google has restrictions and usage limits[23]. The request limit for the Google Translate API is limited to 600 queries per minute, i.e. 300 queries with 2 translations each. Therefore, in the first round a sleeptimer which provided a pause of 1.5 seconds after each topic was installed. Since the second round worked without the candidate, the sleeptimer was replaced with a try/except statement.

The three different queries were processed with the same tokenizers, which was also used for the documents (see: Tokenization 4.2.1). In Elastic, the Boolean Query "should" was used to ensure that one of the three tokenized queries is used at a time.

4.2.3. Elastic Settings after Preprocessing

A boost on the with SciSpacy processed MESH and CHEM tokens by the factor 1.7 and KEYWORDS by the factor 0.75 was applied, because it was assumed to be logical that these keywords have a high significance in a scientific database. For the ranking method, we majorly used the DFR similarity model as explained in Section [4.1.6].

4.2.4. Improving Performance with Parallelization

The performance of the preprocessing pipeline has been a great issue throughout the development. The first version used for the precomputed first run took 40 minutes to process the 100k documents. While this time was acceptable for the first run, it would have taken multiple weeks to run on the entire data set for the second run.

After trying different ways to address this problem, the pandarallel module emerged as the best approach. It is a helper module for pandas and is capable of spreading the normal .apply() method to multiple threads instead of just one. Therefore the processing time is roughly divided by the amount of available threads. A relatively powerful machine with a 16 Core 32-thread processor and 128 GB of RAM was available for testing purposes, of which 90 GB could be used freely. The LIVIVO Machine in comparison had 1 TB RAM and 72 Cores. Using pandarallel on this machine meant a reduction of computing time by a factor of 32: the time needed per 100k documents dropped to just 1 minute and 15 seconds and the performance went up from **0.217 GB/h** to **7 GB/h**. This enabled us to process the entire data set in a more reasonable time[17].

5. Experiments & Evaluation

With the help of logged feedback data containing information about received queries, clicks and further details beyond the clicks, an analysis and comparison of the experimental systems and the on-production base system can be made. For the evaluation, an inspection of the received feedback from the first round and second round while inspecting general metrics that indicate the general performance of the experimental search systems LEO and LEPREP is performed, with a closer look on user behaviour after a click. Following that, the success rate of the experimental systems and the base system concerning the different categories of queries such as English queries, German queries or exact matches were compared in order to measure the success of the additional multi-language components.

5.1. First Round: save_fami & lemuren_elk

Unfortunately, the approach to re-rank the head queries of LIVIVO by only providing a pre-computed run did not work out as well as expected, as only a very small subset of users in the first round explicitly used the pre-defined headqueries and as such, only minimal click data was retrieved and made subsequent analysis negligible (see Table 2). This, however, gave more urgency to implement the next systems in round two as fully functional dockerized systems in order to capture a bigger amount of feedback.

Table 2
Results Round 1 with pre-computed Runs

	# of Days active	# of Queries	Clicks	Impressions	Win	Lose	Outcome%
Base	18	41	14	66	8	4	66.7
lemuren_elk	18	41	9	41	4	8	33.3
Base	23	52	14	54	12	10	54.55
save_fami	23	52	15	54	10	12	45.45

5.2. Second Round

Thanks the docker approach utilized in the second round, the submitted systems were integrated into the live LIVIVO system and thus were able to process any kind of query rather than just pre-defined headqueries. As such, the systems were able to gather substantially more click data in comparison to the first round, making further analysis possible.

In the framework, a search engine is considered improved, if it manages to achieve an overall higher win-ratio than the opposing base system. However, the different metrics are weighted differently, as orders and bookmarks are worth more than just clicks by themselves.

As such, in order to properly evaluate the search engines, we need to introduce the basic metrics that determine the performance of a search engine in the LIVIVO environment [7].

At first glance, both submitted systems were outperformed by the base system in terms of wins and the win outcome percentage, with both systems having substantially less amount of clicks compared to the base system (see Table 3).

Table 3
General Metrics including Run Time, Number of Queries, Impressions and Win/Lose Ratio

	# of Days active	# of Queries	Impressions	Win	Lose	Outcome%
Base	24	4083	4083	669	444	60.01
LEO	24	4083	3985	444	669	39.9
Base	20	3763	3763	791	172	82.14
LEPREP	20	3763	3674	172	791	17.9

Upon further inspection of the user behaviour after a click has been received, users seemed more interested in the results retrieved by the submitted LEO system than the results from the base system. This can be seen when inspecting the segmentation of click behaviour.

Although the LEO system has received 288 less clicks than the base system, the ratio between the bookmark and order metrics in comparison to the raw number of clicks received is substantially higher for the LEO system than the base system. This is also reflected in the "Click Conversion Rate", where the ratio between clicks and subsequent clicks on the document's detail field was calculated.

Overall, this inspection could be interpreted as a strong positive sign that the user had a high interest in the shown result, rather than just clicking the document's title and leaving. For this aspect and metric alone, the LEO system was able to outperform the base system (see Table 4).

Table 4
Segmentation of Click Behaviour

	Clicks	Details	Fulltext	Title	Bookmarks	Order	Click Conversion Rate
Base	1075	656	386	539	27	38	0.61
LEO	787	509	274	400	31	30	0.64
Base	1164	692	438	542	47	44	0.59
LEPREP	346	201	159	162	9	11	0.58

Additionally, some other metrics to determine the user experience of the submitted systems were calculated. The "Clicks per Query" metric signifies the amount of click per retrieved result list where at least on result was clicked.
On one hand, a low "Clicks per Query" metric could be interpreted as a user already being satisfied with the first clicked result, having no further need to inspect the rest of the result list. Under this assumption, the LEPREP system was able to outperform the other system based on this metric, followed by the LEO system. On the other hand, the user ideally wants to investigate multiple relevant results rather than just one, under which the LEPREP system performed the worst and the base system the best.
The metrics for the "Mean Reciprocal Rank" are very distorted and hard to interpret because of the high "Abandonment Rate" of all systems, meaning most of the sent result for both the

submitted systems and the base system did not lead to any clicks, leaving room for a lot of potential future improvements (see Table 5).

Table 5
Additional Metrics

	Clicks per Query	Mean Reciprocal Rank	Abandonment Rate
Base	1.27	0.16	0.79
LEO	1.23	0.12	0.84
Base	1.3	0.18	0.76
LEPREP	1.17	0.055	0.92

5.2.1. Query Analysis

For further analysis, it is also important to inspect how the respective search systems performed for different kind of search queries:

- **[Long and short Queries]**
- **[German and English Queries]**
- **[Queries with boolean Operators]**
- **[Queries with MeSH terms]**

At first, the long and the short German and English queries are compared between the submitted systems and the base system, filtered by length. The short queries are defined by a length shorter than 20 characters and the long queries are defined by the length between 20 to 50 characters, while exact match searches on the title of a document longer than 50 characters should not be a part of this analysis. For the LEO system, the document titles that were filtered for their usage for the "Keyword & Query" (see: 5.2.2) analysis were also excluded from the long queries.

Table 6
Metrics for long Queries

	Query Language	Wins	Lose	Ties	Outcome%
Base	English	100	89	35	52.9
LEO	English	89	100	35	47.1
Base	German	224	165	46	57.58
LEO	German	165	224	46	42.42
Base	English	127	37	14	78.44
LEPREP	English	37	127	14	22.56
Base	German	261	43	22	85.86
LEPREP	German	43	261	22	14.14

Inspecting the metrics of both submitted systems shows, that they performed generally better for English queries in comparison to German queries(see Table 6). This can be observed for

Table 7
Metrics for short Queries

	Language	Wins	Lose	Ties	Outcome%
Base	English	87	56	19	60.85
LEO	English	56	87	19	39.16
Base	German	120	60	22	66.67
LEO	German	60	120	22	33.33
Base	English	85	27	17	76.89
LEPREP	English	27	85	17	24.11
Base	German	151	28	21	84.36
LEPREP	German	28	151	21	15.64

both longer and shorter queries. Looking at the performance of the LEO system, it is notable that long queries for both German and English queries perform substantially better than for short queries, which is interesting when these results are compared with the following results from the section "Keyword & Queries"(see: 5.2.2). For long queries ranging between 20 and 50 characters, the results are pretty lower than the base system's performance with a performance difference of 5.8 % for the English and 15.16 % for the German queries.

For the LEPREP system however, the gap between the long and short queries is pretty small and varies for the language, so a general hypothesis cannot be made (see Table 7). Unfortunately for both submitted systems, approximately 40 % of the received queries were in English, which leads to a poor overall performance if you do not differentiate in terms of the language.

Table 8
Metrics for Queries with MeSH Terms

	Query Language	Wins	Lose	Ties	Outcome%
Base	English	181	140	52	56.39
LEO	English	140	181	52	43.61
Base	German	328	213	66	60.63
LEO	German	213	328	66	39.37
Base	English	208	63	30	76.75
LEPREP	English	63	208	30	23.25
Base	German	398	71	42	84.76
LEPREP	German	71	396	42	15.24

For a scientific database like LIVIVO, MeSH terms are important keywords to consider. MeSH (Medical Subject Headings) present a hierarchical thesaurus that is controlled and edited by the National Library of Medicine [24].
For the MeSH term analysis, if a query contains at least one known MeSH term, the queries were then filtered with the scispacy module and the "en_core_sci_lg", which contains 785.000 scientific words and 600.000 word vectors via the "EntityLinker" function[19]. For this analysis the queries were also separated between German and English queries. At first it is worth mentioning that almost every query contained at least one or more MeSH terms, so only a few

queries ended up being excluded, but increasing the minimum required amount of MeSH terms would have led to the problem that terms like "clown therapy", which are not MeSH terms but might present a medical term, would have been filtered out. Since MeSH terms are always in English, both the German and the English queries are parsed by the biomedical pipeline model "en_core_sci_lg". Once again it can be observed that both submitted systems perform better with English Queries than with German Queries (see Table 8).
While the LEO system just used the title and abstract field for querying, the LEPREP system utilized scientific MeSH Term tokins in the "Mesh_tokens" field with an additional boosting of 1.7 for its query process. However the LEPREP system was outperformed by both the base system and the LEO system. It might be that similar to the "clown therapy" problem, very general MeSH terms were present in the MeSH term fields, leading to a high intersection of many documents from the corpus with a variety of actually semantically different "therapies". This is also a reason why the criteria of minimum MeSH terms for the tested queries were kept at one, because the phenomenon and source of this discrepancy in the two systems should be detected and avoided (see Table 8).

Table 9
Queries with logical operators. The amount of won queries per system is shown: E.g. the base system during won 576 queries while the experimental LEO system won 376 queries. None applies when no system received clicks and tie applies when the base and experimental system received the same amount of clicks.

	Wins	Lose	Ties	Outcome%
Base	576	376	131	60.5
LEO	376	576	131	39.5
Base	676	142	21	82.64
LEPREP	142	676	21	17.37

Looking at the win/loss rate of the experimental systems compared to the base system, it is apparent that the LEO system performs almost 3 times better than the LEPREP system in terms of queries with boolean operators (see Table 9). Seeing how the second system with an elaborate preprocessing and tokenizing process with an overall focus using on scientific language such as MeSH Terms and CHEM tokens to determine matches lead to worse performance an almost aspects in comparison to the first system and even more to the base system, it can be concluded that this custom implementation of this process does not work in terms of satisfying a user's information need in a scientific environment like LIVIVO.

5.2.2. Keyword & Queries

For the analysis of our approach in handling titles as keywords within queries, the possibly affected keywords needed to be extracted from the rest of the queries 4.1.1. Several patterns were used to filter "normal" queries from the keyword queries. The most effective one was to filter all queries by a certain string length, although this might exclude certain short titles.
In the end the queries containing titles as keyword were identified and reduced to a total of 285 queries which were then again inspected manually. This reduced the amount of queries

to a final product of 238 queries. For this particular set, the win and loss rate was calculated for the LEO system in comparison to the base system. It turned out that 126 of these queries got results from both systems. While the LEO system won 40 of these queries, the base system outperformed this system with 75 wins, so the win ratio of 34.78% turns out to be worse than the overall performance comparison of these two systems. Furthermore 11 ties were calculated for this set of keyword queries.
As such, the approach with a "minimum_should_match" of 75% did not pay off as expected. Another issue might have been that the score was calculated in a should statement with four different queries, so the scores were accumulated, making it trigger on a semantically false match. However, the significant boosting of ten for the LEO system should effect the score enough to fulfill the set goal.

5.2.3. Fuzzy Matches

As described earlier in the paper one approach for the submitted LEO system was to extend the queries with fuzzy matching in order to improve query results containing typos from the user input (See 4.1.5). During the evaluation period, it turned out that it was hard to evaluate the performance of this approach for the LEO system, since most names and medical terms were detected as typos from the used pyspellchecker. After some filtering, 454 terms in German and English were left, which had to be filtered manually, resulting in 50 terms with some kind of typos. With just a small set of 50 terms, it is not possible to form an adequate analysis about the performance of the fuzzy match. However when calculating the outcomes for wins and losses, it turns out that, because of the high abandonment rate of the two systems, just six of the 50 queries actually collected a click at all. Five of these queries were won by the base system, which means that the submitted LEO system had an win outcome of 16.6%. Although the amount of usable test data is very small, it raises the question whether the "fuzzy match" built-in function of Elasticsearch actually worsened the search performance for the correctly spelled queries as well.

6. Conclusion & Future Research

Overall, the individual approaches towards improving query search for multi-language domains did not manage to win over the existing base system on LIVIVO when considering the base metrics such as wins and loses. The more sophisticated approach in handling the preprocessing and tokenization of the documents primarily externally outside of the in-built Elasticsearch functions with SpaCy and an elevated importance on controlled vocabulary such as MESH and CHEM tokens even worsened the performance of the search engine considerably.

Furthermore, it became very apparent in the experiments section that the more approaches were implemented on the index side of Elasticsearch and the more custom made the tokenization process became, the worse the overall resulting system became. This could be a case of oversaturation of tokens in the LEPREP system, as the inclusion of MeSH and CHEM tokens towards its index might have worsened the search results in terms of apparent relevancy, as a lot of documents contain many rather unspecific MeSH terms such as "disorder", thus polluting the search result list. Therefore, it might be beneficial to rely on in-built Elastic functions for the index side and lessen the associated boosting of MeSH Terms, while looking into implementing more elaborate ways to handle the multi-language problem more on the query side.

For additional further research, other approaches could be inspected closer such as: Implementing a German compound word list to improve the handling of compound words that can be commonly found in the German language, further expanding the existing ingestion pipeline with more conditional fields in order to finetune search depending on the query and its language received and expanding the query process of the experimental systems with a more comprehensive synonym word list adapted to the knowledge needs of the LIVIVO platform.

Acknowledgments

Thanks to the developers of STELLA [25] and to the developers of LIVIVO [8] for the cooperation and development with their respective platforms.

References

[1] P. Schaer, J. Schaible, L. J. G. Castro, Living lab evaluation for life and social sciences search platforms - lilas at CLEF 2021, in: D. Hiemstra, M. Moens, J. Mothe, R. Perego, M. Potthast, F. Sebastiani (Eds.), Advances in Information Retrieval - 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 - April 1, 2021, Proceedings, Part II, volume 12657 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 657–664. URL: https://doi.org/10.1007/978-3-030-72240-1_77. doi:10.1007/978-3-030-72240-1_77.

[2] spaCy, German spacy models documentation, 2021. URL: https://spacy.io/, [last accessed: 19.05.2021].

[3] LuShan, google-trans-new 1.1.9, 2020. URL: https://pypi.org/project/google-trans-new/, [last accessed: 19.05.2021].

[4] P. Schaer, T. Breuer, L. J. Castro, B. Wolff, J. Schaible, N. Tavakolpoursaleh, Overview of lilas 2021 - living labs for academic search, in: K. S. Candan, B. Ionescu, L. Goeuriot, B. Larsen, H. Müller, A. Joly, M. Maistro, F. Piroi, G. Faggioli, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Twelfth International Conference of the CLEF Association (CLEF 2021), volume 12880 of *Lecture Notes in Computer Science*, 2021.

[5] Stella - infrastructures for living labs, 2021. URL: https://stella-project.org/, [last accessed: 21.05.2021].

[6] Elastic, Elasticsearch: Die offizielle engine für verteilte suche und analytics, 2021. URL: https://www.elastic.co/de/elasticsearch/, [last accessed: 27.05.2021].

[7] F. Radlinski, M. Kurup, T. Joachims, How does clickthrough data reflect retrieval quality?, in: J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K. Choi, A. Chowdhury (Eds.), Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008, ACM, 2008, pp. 43–52. URL: https://doi.org/10.1145/1458082.1458092. doi:10.1145/1458082.1458092, [last accessed: 16.06.2021].

[8] Z. MED, Livivo - the search portal for life sciences, 2021. URL: https://www.livivo.de/, [last accessed: 28.05.2021].

[9] spaCy, Facts & figures, 2021. URL: https://spacy.io/usage/facts-figures, [last accessed: 19.05.2021].

[10] I. Docker, Docker @ elastic, 2021. URL: https://www.docker.elastic.co/, [last accessed: 19.05.2021].

[11] Kaggle, Covid-19 open research dataset challenge (cord-19), 2020. URL: https://ir.nist.gov/covidSubmit/index.html, [last accessed: 20.05.2021].

[12] G. Amati, C. J. Van Rijsbergen, Probabilistic models of information retrieval based on measuring the divergence from randomness, ACM Trans. Inf. Syst. 20 (2002) 357–389. URL: https://doi.org/10.1145/582415.582416. doi:10.1145/582415.582416.

[13] A. Bugara, Designing an optimal multi-language search engine with elasticsearch, 2020. URL: https://codarium.substack.com/p/designing-an-optimal-multi-language, [last accessed: 20.05.2021].

[14] Fossies, Analysis - anatomy of an analyzer, 2021. URL: https://fossies.org/linux/elasticsearch/docs/reference/analysis.asciidoc, [last accessed: 28.05.2021].

[15] Elastic, Query string query, 2021. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/release-notes-7.13.0.html, [last accessed: 28.05.2021].

[16] A. S. Foundation, Class german normalization filter, 2021. URL: https://lucene.apache.org/core/8_8_0/analyzers-common/org/apache/lucene/analysis/de/GermanNormalizationFilter.html, [last accessed: 28.05.2021].

[17] M. NALEPA, pandarallel, 2021. URL: https://github.com/nalepae/pandarallel, [last accessed: 19.05.2021].

[18] T. Singh, Natural Language Processing With spaCy in Python, 2021. URL: https://realpython.com/natural-language-processing-spacy-python/, [last accessed: 19.05.2021].

[19] allenai, scispaCy, 2021. URL: https://allenai.github.io/scispacy/, [last accessed: 19.05.2021].

[20] spaCy, German spacy models documentation, 2021. URL: https://spacy.io/models/de/, [last accessed: 19.05.2021].

[21] Elastic, Whitespace Tokenizer, 2021. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-whitespace-tokenizer.html, [last accessed: 19.05.2021].

[22] S. Han, googletrans 3.0.0, 2020. URL: https://pypi.org/project/googletrans/, [last accessed: 19.05.2021].

[23] G. LLC., Quotas and limits, 2021. URL: https://cloud.google.com/translate/quotas, [last accessed: 19.05.2021].

[24] N. L. of Medicine, Welcome to medical subject headings, 2021. URL: https://www.nlm.nih.gov/mesh/meshhome.html, [last accessed: 28.05.2021].

[25] T. Breuer, P. Schaer, A living lab architecture for reproducible shared task experimentation, in: Information between Data and Knowledge, volume 74 of *Schriften zur Informationswissenschaft*, Werner Hülsbusch, Glückstadt, 2021, pp. 348–362. doi:10.5283/epub.44953.

A. Appendix

A.1. Multi-fields

```
"TITLE": {
        "type": "text",
        "fields": {
            "analysis": {
                "type": "text",
                "analyzer": "general_analyzer",
                "similarity": "LMJelinekMercer_short"
            },
            "en": {
                "type": "text",
                "analyzer": "english_analyzer",
                "similarity": "LMJelinekMercer_short"
            },
            "ger": {
                "type": "text",
                "analyzer": "german_analyzer",
                "similarity": "LMJelinekMercer_short"
            },
            "keyword": {
                "type": "keyword"
            }
        }
```

A.2. Ingestion Pipeline

```
"processors": [
                {
      "set": {
        "if": "ctx.LANGUAGE == 'eng'",
        "field": "TITLE_EN",
        "value": "{{TITLE}}"
      }
    },
        {
      "set": {
        "if": "ctx.LANGUAGE == 'ger'",
        "field": "TITLE_GER",
        "value": "{{TITLE}}"
      }
    }
   ]
```

A.3. Acronym handling

```
"filter": {
        "acronym": {
            "type": "word_delimiter",
            "catenate_all": true,
            "generate_word_parts": false,
            "generate_number_parts": false,
            "preserve_original": true,
            "split_on_case_change": true,
            "split_on_numerics": true,
            "stem_english_possessive": true
        }
```