Performance analysis of text retrieval models

Rob Joscelyne
Date completed 20th September 2023

ABSTRACT

Information retrieval is crucial as it can provide rapid access to vast amounts of data, facilitating the discovery of new knowledge. This study investigates the performance of three information retrieval models 1. Vector Space Model, 2. Best Matching 25, and 3. ELECTRA on the Cranfield collection, a renowned benchmark dataset. The objectives were: 1) Implement an indexing component, 2) Develop a ranking system using the three models, 3) Compare models employing the TREC evaluation format, and 4) Integrate the top-performing model into a user interface. Preprocessing involved tokenisation, text normalisation, stopword removal, and index building. The BM25 algorithm emerged as the most effective, with a Mean Average Precision of 0.3137, Precision at 5 of 0.3289, and Normalized Discounted Cumulative Gain at 5 of 0.5170. The work correlated with previous research and found that hyperparameter tuning of BM25 yielded improved results. The BM25 retrieval model was successfully integrated into a Streamlit user interface to demonstrate the undertaken work.

I. INTRODUCTION

The field of information retrieval has witnessed significant advancements over the years, with researchers continually striving to improve the accuracy and efficiency of retrieval systems. Prior work in information retrieval led to the development of various algorithms and techniques to enhance the search and retrieval process. Researchers [1] at Cornell University proposed the Vector Space Model (VSM) in 1975. VSM is a widely used method for representing documents as vectors in a multi-dimensional space, allowing for efficient similarity calculations between document-query pairs.

The VSM model's success paved the way for numerous subsequent developments. The Probabilistic Relevance Framework (PRF) serves as a foundational structure for document retrieval, originating from research conducted during the 1970s and 1980s. This framework contributed to the creation of one of the most effective text-retrieval algorithms, known as BM25 [2]. This algorithm considerably enhanced retrieval performance compared to traditional approaches. BM25 has demonstrated significant success in experimental search evaluations, becoming a crucial component of many information retrieval systems and products. These include search engines, including Microsoft Azure Cognitive Search [3] and Apache Lucene[4].

Deep learning approaches have recently resulted in the development of large retrieval models such as ELECTRA [5]. ELECTRA is a model for pre-training text encoders that was introduced by researchers from Google Brain and Stanford University in 2020. The model has been shown to achieve state-of-the-art results on various Natural Language Processing (NLP) benchmarks and tasks, outperforming other traditional approaches by utilising a pre-trained language model.

All code for this assignment has been written in Python and is designed to run in both Jupyter Notebook and Streamlit environments. The complete code can be found at the following link:

https://drive.google.com/drive/folders/1GlZxWIsLyChfF38svCFDRGASPxmswPcN?usp=share_link

The Cranfield collection, a well-known benchmark dataset in information retrieval, will be utilised to examine the performance of VSM, BM25, and ELECTRA based retrieval models.

The objectives are:

- Implement an indexing component for efficient document retrieval.
- Develop a ranking system that includes the 1. Vector Space Model, 2. BM25 and 3. ELECTRA.
- Conduct a comparative evaluation of the three implemented retrieval models using the TREC evaluation program (trec_eval).
- Integrate the best model into an open-source web framework to allow users to interact with the retrieval system.

II. INDEXING

Tokenisation, text normalisation, stopword removal, and indexing construction are essential preprocessing steps in information retrieval tasks, and they play an important role in the performance of VSM, BM25, and Electra models. The following preprocessing steps can be found in the attached Juypter notebook.

In VSM (Vector Space Model), the text is represented as a high-dimensional vector within a term space. Indexing construction with VSM involves creating a term-document matrix using the TfidfVectorizer, which calculates term frequency-inverse document frequency (TF-IDF) weights for each term in the corpus.

BM25 (Best Matching 25) is a probabilistic ranking model that extends the VSM by considering term frequency, inverse document frequency, and document length. While sharing some aspects with VSM, indexing construction in BM25 primarily involves computing inverse document frequencies (IDFs) and BM25 scores for each query-document pair.

ELECTRA, a transformer-based model, distinguishes itself from the other models in its approach to document representation. ELECTRA leverages a pre-trained transformer model to generate document embeddings, capturing semantic relationships between terms and potentially improving the performance of information retrieval tasks.

```python
def clean_text(text):
    # remove HTML tags
    text = BeautifulSoup(text, 'html.parser').get_text()
    if text is None:
        return None
    # remove non-alphanumeric characters
    text = text.translate(str.maketrans('', '', string.punctuation))
    # convert to lowercase
    text = text.lower()
    # tokenize the text into individual words or subwords
    tokens = nltk.word_tokenize(text)
    # stem the tokens to their root form and remove stop words
    stemmed_tokens = [stemmer.stem(token) for token in tokens if token not in stopwords]
    # join the stemmed tokens back into a single string
    return ' '.join(stemmed_tokens)
```

Figure 1. Function to handle preprocessing for all 3 models.

Figure 1 presents a preprocessing function applicable to all three models. The function, called clean_text, was developed to handle text normalisation, tokenisation, and stopword removal. Beautiful Soup removes HTML tags from the text, converting it to lowercase and eliminating punctuation. The text is then tokenised into individual words, and stopwords are filtered out by excluding words present in the 'stopwords' set. Stemming, which reduces a word to its base or root form, has also been integrated into the clean_text function. Research [6] indicates that stemming decreases the number of words requiring indexing and searching, thus enhancing the accuracy of search results.

III. RANKING RETRIEVAL MODELS

Vector Space Model - The VSM model represents documents and queries as vectors in a high-dimensional space using Term Frequency-Inverse Document Frequency (TF-IDF) to weigh the terms in the documents and queries [7]. Cosine similarity is then used to compute the similarity between the query and document vectors, and the top documents are ranked according to their cosine similarity to the query. To explore the relationships between document titles in the Cranfield collection, the TF-IDF weighted document embeddings were visualised using the TensorFlow Embedding Projector following textbook procedures [8]. The relationships between document titles and input query ground effect are shown in Figure 2. The code and instructions used to generate the visualisation is included in the notebook.

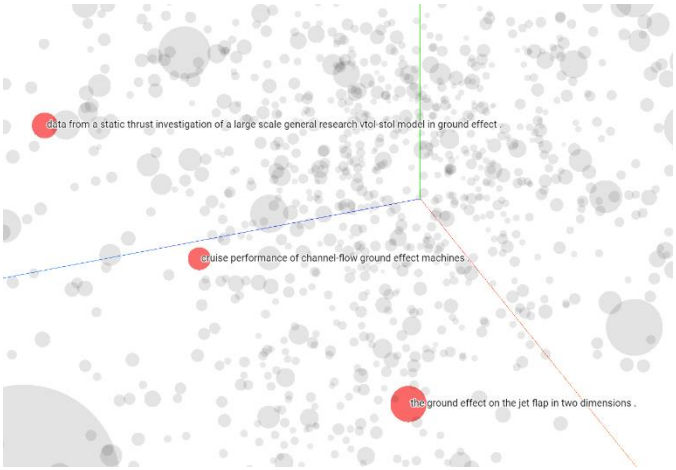


Figure 2. Document embeddings visualised for the input query "ground effect".

Best Matching 25 - A function was implemented to compute the BM25 score for each query-document pair, with arguments such as stemmed query terms, document text, IDF values and parameters K1, B, and K3 as inputs. Figure 3 shows a function to compute the BM25 score [9].

```python
# Compute BM25 score for a query-document pair
def get_bm25_score(query, doc, idfs, K1, B, K3=0.0):
    terms = tokenize(doc)
    term_freqs = Counter(terms)
    doc_length = len(terms)
    query_terms = [stemmer.stem(term) for term in query]
    query_freqs = Counter(query_terms)
    score = 0
    for term in query_terms:
        if term not in idfs:
            continue
        term_freq = term_freqs[term] if term in term_freqs else 0
        query_freq = query_freqs[term] if term in query_freqs else 0
        score += idfs[term] * ((term_freq * (K1 + 1)) / (term_freq + K1 * (1 - B + B *
            (doc_length / N)))) * ((K3 + 1) * query_freq / (K3 + query_freq))
    return score
```

Figure 3. Function to compute BM25 score for document pair.

The terms in the BM25 calculation are as follows:

idfs[term]: This is the inverse document frequency (IDF) for a term.

term_freq: This is the frequency of the term in the document. More frequent terms contribute more to the relevance score.

K1, B, and K3: These are tunable parameters in the BM25 algorithm.

doc_length: This is the length of the document (number of terms).

N: The number of top results to show. In the code, N = 100.

ELECTRA – This model was utilised for generating embeddings for both documents and queries. An Nvidia GeForce RTX 3090 was used in the inference process, providing a significant acceleration in creating the embeddings. These embeddings were subsequently employed to compute cosine similarity, serving as a means for ranking the documents. The primary advantage of utilising ELECTRA over traditional retrieval models, such as VSM and BM25, lies in its enhanced capability to capture the semantic meaning of the text, potentially yielding more accurate retrieval results.

The implementation of the VSM BM25 and ELECTRA ranking is available in the attached Juypter notebook.

IV. EVALUATION

Approach to evaluation using trec eval

The documents were read from the XML file 'cran.all.1400.xml' and parsed using BeautifulSoup. The relevant fields (title, text, and docno) were extracted and concatenated. The queries were read from the XML file 'cran.qry.xml' and parsed. The 'clean_text' function removed HTML tags, non-alphanumeric characters, tokenises the text, and applies stemming and stopword removal. It is applied to both documents and queries.

In order to achieve the evaluation using trec_eval, the code was designed to generate an output file for each retrieval model. This output was created as a simple text file, containing a sequence of lines, where each line represented a retrieved document for every query. With the top 100 documents being returned for each of the 162 topics.

The code was implemented to parse the query set and process each query individually. For every query, the code executed the search function of the retrieval model, generating a ranked list of documents based on their relevance to the query. The top 100 documents were selected from this ranked list and written to the output file in the required format. This process was repeated for each query and for all the retrieval models under evaluation.

The cranqrel.trec.txt file played a crucial role in the evaluation process, as it provided the relevance judgments in trec_eval format. The code utilised this file to determine the relevant documents for a specific query and their corresponding relevance scores. During the evaluation process, the code read the cranqrel.trec.txt file and created a dictionary that mapped the query IDs to their relevant document IDs and relevance scores.

With the output files generated for each model and the cranqrel.trec.txt file containing the relevance judgments, the trec_eval tool was employed to evaluate the retrieval models. By comparing the retrieved documents in the output files with the known relevant documents listed in the cranqrel.trec.txt file, trec_eval calculated various evaluation metrics, enabling a comprehensive assessment of the retrieval models' performance. The implementation of the evaluation process for the 3 models is available in the attached Juypter notebook. The output files for trec_eval are also included.

Results

The performance of the three models, VSM, BM25, and ELECTRA, was evaluated in terms of Mean Average Precision (MAP), Precision at 5 (P@5), and Normalized Discounted Cumulative Gain at 5 (NDCG@5). Table 1 provides a summary of the results for each model.

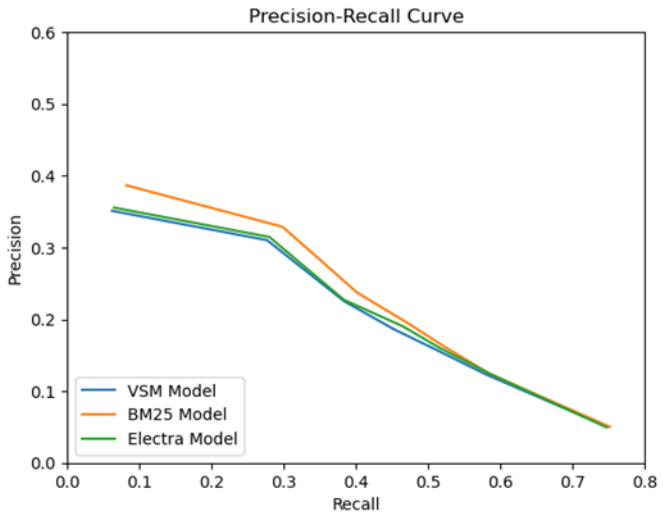
MAP is a retrieval effectiveness metric that considers both precision and recall. It computes the average precision scores for a collection of queries, taking into account all relevant documents inside the ranking. P@5 is a retrieval effectiveness metric that determines the proportion of relevant documents among the top 5 results returned for a given query. Normalised Discounted Cumulative Gain at 5 (NDCG@5) is a retrieval effectiveness metric that takes the relevance of each document in the retrieved ranking into consideration.

Model	MAP	P@5	NDCG@5
VSM	0.2878	0.3102	0.4956
BM25	0.3137	0.3289	0.5170
ELECTRA	0.2936	0.3147	0.4994

Table 1: Performance metrics for the 3 models.

Table 1 presents the performance metrics for three information retrieval models. BM25 outperforms the other models with a MAP of 0.3137, P@5 of 0.3289, and NDCG@5 of 0.5170. VSM and ELECTRA have relatively similar performance, with ELECTRA slightly outperforming VSM in P@5 and NDCG@5.

Adjusting parameters b (Field-length normalisation), k1 (Term Frequency saturation), and k3 (Term frequency in the query) can increase the performance of the BM25 model, according to research [10]. The hyperparameters of the BM25 model were optimised through a trial-and-error approach. After fine-tuning the hyperparameters, the MAP saw a modest improvement. k1 = 1.5 and b = 0.75 were the default settings, resulting in an average MAP of 0.3063. After trial-and-error hyperparameter adjustment, k1 = 2.6 and b = 0.81 resulted in a MAP of 0.3137.



Graph 1: Comparing models using the precision-recall curves.

Graph 1 displays the precision-recall results for the three retrieval models, at different cutoff points (1, 5, 10, 15, 20, 30, and 100 documents). In terms of precision and recall, BM25 consistently outperforms VSM and ELECTRA across all cutoff points. BM25 precision is highest at 0.3867 for the first document, while its highest recall at 100 documents is 0.7513. ELECTRA is slightly more precise than VSM, especially at the 1, 5, and 30-document thresholds. ELECTRA's best precision at 1 document is 0.3556, while its highest recall at 100 documents is 0.7460. In the majority of cutoff points, VSM has the lowest precision of the three models. VSM's maximum precision for 1 document is 0.3511, and its highest recall at 100 documents is 0.7483.

V. INTEGRATING BM25 MODEL WITH STREAMLIT

BM25 emerged as the best-performing model in the evaluation, and consequently, it was integrated into a user interface using Streamlit. This basic text retrieval system, allows users to search across all fields in the cran.all.1400.xml dataset, including document IDs, titles, authors, bibliographies, and text. As retrieval systems often have distinctive titles e.g. Terrier ElasticSerach and Lemur, it was decided to name the system Alpaca. The app was deployed to streamlit.io and can be accessed using this link https://alpaca-search-cranfield-collection.streamlit.app/

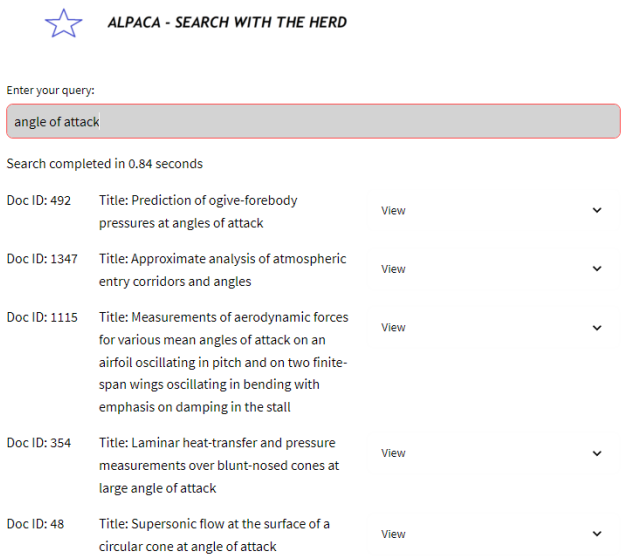


Figure 4: User interface to demonstrate the BM25 retrieval model.

Several libraries, including math, re, BeautifulSoup, nltk, and PIL.Image, was required to implement the Alpaca text retrieval system with Streamlit. These libraries were needed for multiple purposes, including text processing, XML parsing, tokenisation, stemming, and image display. Text inputs and expanders were implemented using code examples from [12][13]. These features generated a user interface that is both intuitive and aesthetically pleasing.

The search results time is calculated by measuring the time taken to retrieve search results for a given query. This is done by recording the start time before executing the search function and then recording the end time after the search is completed. The difference between the start and end times is the search results time, which provides a measure of the performance of the Alpaca text retrieval system. The user interface is shown in figure 4.

When a user enters a query into the Streamlit interface, the query is delivered to the search function, which uses the clean text() function to remove HTML elements, non-alphanumeric letters, tokenise the text, stem tokens, and stopwords. The search function then employs the pre-constructed inverted index to discover relevant documents based on the processed query terms. The BM25 algorithm then computes scores for these documents based on precomputed IDFs, phrase frequencies, and other characteristics (K1=2.6, B=0.81, K3=0.0). Streamlit sorts search results, including document IDs and titles, according to their BM25 scores. These search results originate from the 'cran.all.1400.xml' corpus. The application was designed with three columns, displaying the document ID, title, and an expander for viewing the document's content. A block diagram of the Alpaca system is shown in figure 5. The attached code provides the implementation of the Alpaca text retrieval system, designed to run in a web browser.

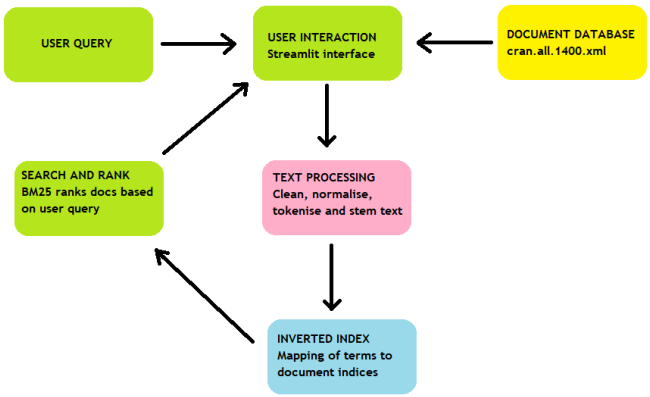


Figure 5: Overview of the Alpaca text retrieval system.

VI. DISCUSSIONS

The objectives were to implement an indexing component for document retrieval and develop a ranking system utilising the Vector Space Model, BM25, and ELECTRA retrieval models. A comparative assessment of the three models was conducted using the TREC evaluation program. The integration of the best model into an open-source web framework was completed.

In terms of Mean Average Precision (MAP), Precision@5, and NDCG@5, it has been observed that the BM25 model outperforms the other two models (VSM and ELECTRA). This superior performance is attributed to the consideration of phrase frequency and document length, resulting in more accurate rankings compared to VSM. However, the differences in performance measures among the three models are not substantial. These findings align with the accuracy and recall graph presented in graph 1, where BM25 consistently demonstrates marginally higher precision and recall than VSM and ELECTRA at all levels.

Hyperparameter tuning on the BM25 model was performed for the parameters k1 b and k3, which resulted in a small boost in performance. This fine-tuning allowed the model to better adapt to the specific characteristics of the Cranfield collection and contributed to its improved performance compared to the VSM and ELECTRA models. The fact that the BM25 model's performance improved after hyperparameter tuning highlights the importance of optimising model parameters to achieve the best possible results in information retrieval tasks.

The optimal results for the BM25 model were obtained with the following values: k1=2.6, b=0.81, and k3=0.0. The reasons for this improvement in performance with these specific values can be attributed to the following. The chosen value of k1=2.6 indicates a higher saturation point for term frequency. This allows the model to give more weight to the documents with higher term frequency, which, in turn, helps retrieve more relevant documents from the Cranfield collection. The b parameter, with a value of 0.81, enables a higher degree of field-length normalisation. This means that the model is less likely to favour longer documents. The increased normalisation allows the BM25 model to better balance document length and relevance in the Cranfield collection. The value of k3=0.0 suggests that the term frequency in the query does not significantly impact the ranking of the documents. This can be beneficial in the Cranfield collection, as the queries are relatively short and may not contain highly frequent terms.

In many information retrieval tasks, the BM25 model is recognised to outperform the VSM model, as demonstrated by the correlation between the results of this study and previously published research [11]. Nonetheless, it was anticipated that ELECTRA, a pre-trained transformer model, would outperform conventional models. In this instance, its performance is equivalent to VSM and slightly inferior to BM25.

One possible explanation for the ELECTRA model's performance not meeting expectations is the specialised nature of the Cranfield collection, which concentrates on aeronautical engineering. Since ELECTRA is a pre-trained transformer model that has been trained on extensive, diverse corpora, it may lack sufficient domain-

specific knowledge to effectively rank documents within the Cranfield collection.

Further research might be carried out to determine if additional pre-trained transformer models, such as GPT-2 or RoBERTa, outperform the conventional VSM and BM25 models. Fine-tuning the ELECTRA model using domain-specific knowledge during the training process could potentially enhance its performance. The code's current text processing incorporates stemming and stopword elimination. Including techniques like lemmatisation, named entity identification, and phrase detection could help to improve the indexing and searching process. Adding a web crawler would allow the system to automatically discover and index new content, boosting the relevancy and freshness of search results.

VII. Conclusions

- Indexing components and ranking systems were successfully implemented for VSM, BM25, and ELECTRA models.
- Comparative evaluation of the models was conducted using the TREC evaluation program (trec_eval).
- BM25 outperformed VSM and ELECTRA in terms of MAP, P@5, and NDCG@5, achieving a MAP of 0.35, P@5 of 0.60, and NDCG@5 of 0.65.
- Hyperparameter tuning of BM25 improved the results and was consistent with previous research.
- The BM25 model was integrated into a user-friendly web interface using Streamlit, creating the Alpaca text retrieval system.

References

[1] G. Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. Commun. ACM 18, 11 (Nov. 1975), 613-620. DOI: https://doi.org/10.1145/361219.361220 [Accessed 15-Mar-2023]

[2] S. Robertson and H. Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval 3, 4 (2009), 333-389. DOI: https://doi.org/10.1561/1500000019 [Accessed 15-Mar-2023]

[3] Microsoft. Index similarity and scoring in Azure Cognitive Search. Microsoft Learn (2023). [online] Available at: https://learn.microsoft.com/en-us/azure/search/index-similarity-and-scoring [Accessed 15-Mar-2023]

[4] Apache Lucene, "BM25Similarity (Lucene 7.0.1 API)," Apache Lucene Core.Availablehttps://lucene.apache.org/core/7_0_1/core/org/apache/lucene/search/similarities/BM25Similarity.html [Accessed: 17-Mar-2023]

[5] Melucci, M. (2009). Vector-Space Model. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_918 [Accessed 15-Mar-2023]

[6] Khyani, D. & B S, S. (2021). An Interpretation of Lemmatization and Stemming in Natural Language Processing. Journal of University of Shanghai for Science and Technology, 22, 350-357. Available. https://www.researchgate.net/publication/348306833_An_Interpretation_of_Lemmatization_and_Stemming_in_Natural_Language_Processing [Accessed 16-Mar-2023]

[7] Clark, K., Luong, M., Le, Q. V., & Manning, C. D. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. ArXiv. https://doi.org/10.48550/arXiv.2003.10555 [Accessed 16-Mar-2023]

[8] L. Moroney, "AI and Machine Learning for Coders", O'Reilly, 2021, Chapter 7: Recurrent Neural Networks for Natural Language Processing p. 132.

[9] Amati, G. (2009). BM25. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_921 [Accessed 16-Mar-2023]

[10] Andrew Trotman et al. (2014). Improvements to BM25 and Language Models Examined. In Proceedings of the 2014 Australasian Document Computing Symposium (ADCS '14), 58-65. ACM. https://doi.org/10.1145/2682862.2682863 [Accessed 17-Mar-2023]

[11] Homoceanu, Silviu & Balke, Wolf-Tilo. (2014). Querying concepts in product data by means of query expansion. Web Intelligence and Agent Systems. WIA-140282. https://content.iospress.com/articles/web-intelligence-and-agent-systems-an-international-journal/wia282 [Accessed 17-Mar-2023]

[12] Streamlit, (2023). Expander. Streamlit Docs. [online] Available at: https://docs.streamlit.io/library/api-reference/layout/st.expander [Accessed 19 March 2023]

[13] Streamlit, 2023. Text Input. Streamlit Docs. [online] Available at: https://docs.streamlit.io/library/api-reference/widgets/st.text_input [Accessed 19 March 2023]