INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Neural File Search Engine

CS59001

DESIGN LABORATORY REPORT

Abhinav Bohra

18CS30049

Under the guidance of

Prof. Palash Dey

Department of Computer Science and Engineering

1. Introduction

Semantic Search[1] is a data-searching technique that goes beyond simply looking for keywords to understand the user's intent and the context of the words they are using.

Windows Search[2] is integrated into all Windows Explorer windows for instant access to search. This enables users to quickly search for files and items by file name, properties, and full-text contents. The search technique is based on fuzzy search and text matching with different file properties. However, it does not consider the language semantics of file's name and its content.

At its core, Search is about understanding language. With introduction of Bidirectional Encoder Representations from Transformers (BERT) [3], a model that process words in relation to all the other words in a sentence, rather than one-by-one in order, it is now possible to consider the full context of a word by looking at the words that come before and after it which is particularly useful for understanding the intent behind search queries.

2. Problem Statement

The objective is to design and develop an NLP based intelligent local-file search engine that searches for relevant text documents in a specific folder, considering the semantics of the file's name & its content and returns the most relevant files.

Input

- **Search Directory:** Location of folder to be searched
- **Search Query:** Phrase/keywords to be searched

Output

- **Relevant Files:** Location of top matched files

Files Supported

- **File Content:** .docx, .txt, .pdf, .pptx, .csv
- **File Name:** .docx, .txt, .pdf, .pptx, .csv, .xlsx, jpg, .png, .mp3, .mp4

3. Design Aspects

As shown in Figure 1, the tool takes directory path and query phrase as input and outputs the relevant documents. The system design includes the following aspects.

1. Read and Check Input: The tool takes directory path and query phrase as input. Sanity checks implemented include checking for empty string in input and existence of the directory to be searched. Error is raised whenever a check fails.

2. Document Parsing: The tool recursively lists all the files present in the search directory and its sub-directories. For every file, a custom class object is created that stores the following information - filename, file extension, file path, title (processed filename), processed file content, title embedding, content embedding. The tools

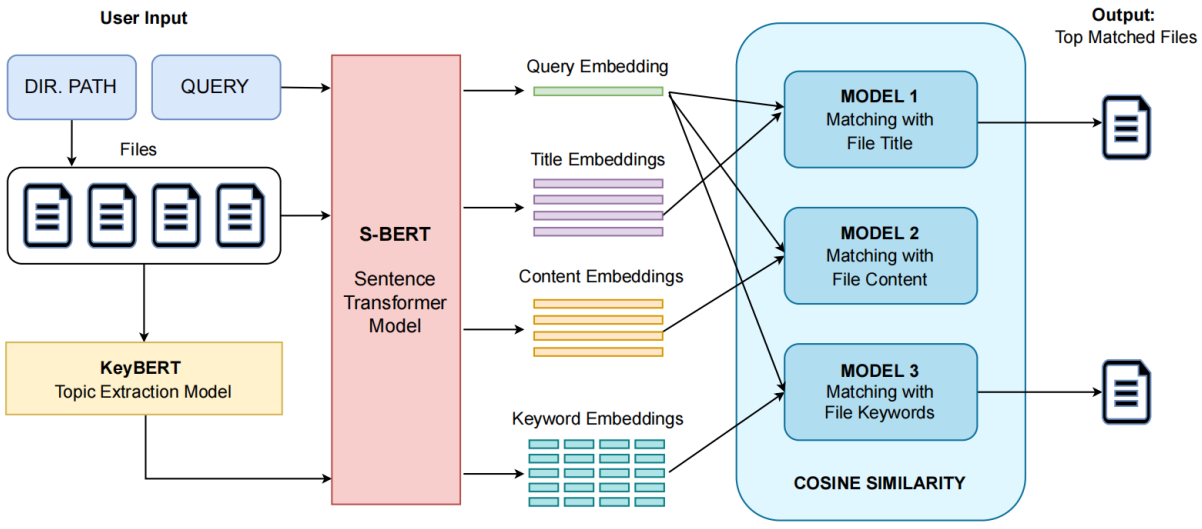

FIGURE 1: System Design

supports filename processing for .docx, .txt, .pdf, .pptx, .csv, .xlsx, jpg, .png, .mp3, .mp4 files and file content processing for .docx, .txt, .pdf, .pptx, .csv files.

3. Query Phrase Processing: The search query is tokenized using NLTK Library's Word Tokenizer[4] . Every word is then converted into its root form by WordNet Lemmatizer[5]. The clean query tokens are then passed to a pretrained SBERT[6] Model to obtain query's latent representation in the form of an embedding (feature vector) and is referred to a "query embedding".

4. File Embedding: Like the search query, filename is also tokenized and lemmatized and encoded to obtain "title embedding". File content is stored in the form of a string and sentence level latent representation is obtained and is referred as "content embedding". Both the embeddings are obtained using SBERT Model.

5. Matching Algorithm: After obtaining the 3 embeddings namely - query, title and content; we use 3 strategies to find the relevant documents. These strategies and the associated models are discussed in detail in the next section. Each strategy returns a list of top matched documents. The final model is an ensemble of the 3 core models which takes union of the results to output the relevant documents.

6. Cache Optimizations: The bottleneck of the pipeline in terms of inference time is reading file content and creating embeddings. To save the computation time, we implemented cache optimization. The title and content embeddings of each new file that is read are saved as.npy files. When the same file is searched again, the embeddings are immediately loaded into memory rather than re-reading content from the files again. This optimization increased the inference speed by up to 70% for previously queried directories.

4. Matching Algorithm

The task at hand is to find documents that are closely related/similar to the query phrase. The three embeddings namely - query, title and content carry the semantic meaning of the natural language in the form of feature vectors.

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and often used to measure document similarity in text analysis. We've used cosine similarity through three different strategies as mentioned below:-

1. Matching with Title: The query embedding is matched with each file's title embedding using cosine similarity. Files with score greater than 0.80 are returned.

2. Matching with Content: The query embedding is matched with each file's content embedding using cosine similarity. Files with score greater than 0.70 are returned.

3. Matching with Keywords: Query embedding carries information of 2-3 tokens, while the content embeddings compress the information of an entire file thus leading to a lot of loss of information. Also, comparing token-level embeddings with sentence level embeddings is not efficient; thus we first find top keywords that best represent the contents' of the file. For this we used KeyBERT Topic Extraction Model [7], that gives top 5 topics for every file. We use SBERT Model to encode every topic and then match it with query embedding using cosine similarity. Files with similarity score greater than 0.70 are returned

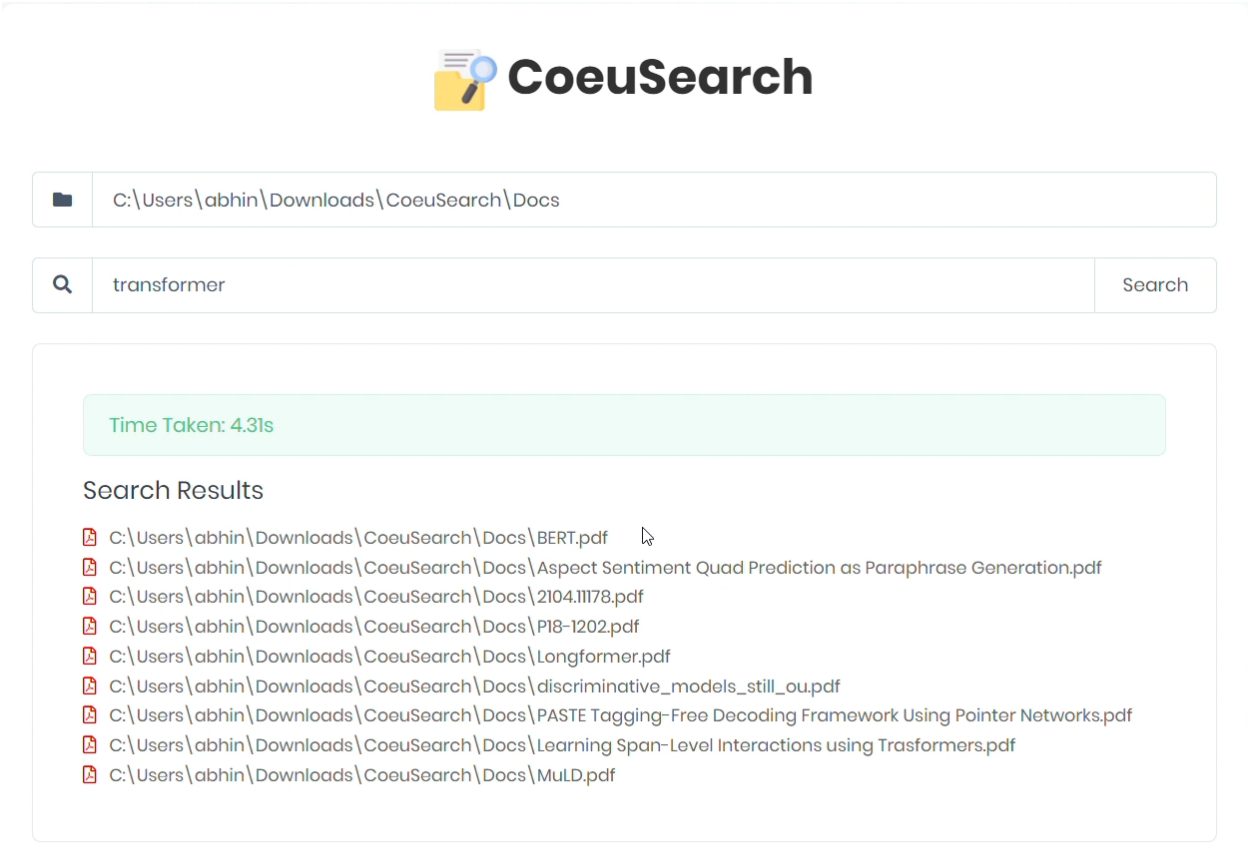

FIGURE 2: Neural File Search Engine - User Interface

6. Future Work

In future, we can extend this tool to a multi-modal search engine that supports image, audio and video data by utilising Vision Transformers[8] and Automatic Speech Recognition. By using pre-trained language models like mBERT and XLM[9], we can enable multilingual search queries as well. We can also increase inference speed by using Approximate Nearest Neighbors for semantic similarity.

Code Link:https://github.com/abhinav-bohra/CoeuSearch

Bibliography

[1] T. Roberts, "Bloomreach," 2019, https://www.bloomreach.com/en/blog/2019/semantic-search-explained-in-5-minutes.

[2] Microsoft, 2022, https://docs.microsoft.com/en-us/windows/win32/search/-search-3x-wds-overview.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] S. NLP, "Toknizer," 2022, https://www.nltk.org/modules/nltk/tokenize/stanford.html.

[5] S. NLP, "Wordnet lemmatizer," 2022, https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.

[6] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[7] M. Grootendorst, "Keybert: Minimal keyword extraction with bert.," 2020.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[9] G. Lample and A. Conneau, "Cross-lingual language model pretraining," *arXiv preprint arXiv:1901.07291*, 2019.