Project Report

Modern Search Engines, Summer 2024

Group members: Niclas Lietzow, Timo Lübbing, Florian Sprick and Sebastian Volz

1 Crawler

1.1 Frontier management

The crawling frontier is managed using a data frame. At the beginning of each crawl, the frontier is initialized with a set of seed URLs and periodically saved to a CSV file. This mechanism allows the crawl to resume from the last saved state if interrupted. Each entry in the frontier records various details: the URL of the site, its crawling status (Pending, Finished, Failed), priority level (higher for URLs from English sites, lower for others), depth (the number of links or hops it took to reach the URL from the initial seed URLs), domain, whether the site contains the term "Tübingen", and whether the site's language is English.

1.2 Crawling loop

The crawling process is organized in a loop with three main steps. After each iteration, the updated frontier is saved to a CSV file.

Fetching pending frontier entries. During each iteration, the system queries the frontier for entries marked "pending" that have a depth of less than 10. These entries are sorted by priority in descending order and by depth in ascending order. Only the first entry per domain and a total of 256 entries are selected. This ensures that we do not send too many requests to a single domain. Asynchronous requests to these URLs are made using the httpx library, with various request headers (such as user-agent) included to mimic browser interactions. For failed requests, an exponential backoff strategy from the tenacity library is used, and each request has a timeout of 10 seconds.

Validating responses. Each retrieved response is validated using the response headers to ensure that it is HTML. Any response that does not conform to this requirement or exceeds 10MB in size is discarded, and the associated frontier entry is marked as "failed". Valid HTML responses are saved locally.

Extracting new URLs. New URLs are extracted from the parsed HTML content using the lxml library. Initial checks determine whether the term "Tübingen" or its variations are present. If the term is absent, no new links are extracted. The document's language is assessed from the HTML tag. For documents identified as English, we assign higher priority to all new links extracted from the page. Non-English pages are checked for English versions by setting the URL path to "/en". Extracted URLs are stripped of query parameters and fragments, leaving only the scheme, host, and path. Each URL is verified to ensure it is absolute, adheres to HTTPS or HTTP protocols, and is not in a specified list of denied domains. Valid URLs are added back to the frontier as "pending", and the original entries are updated to "finished".

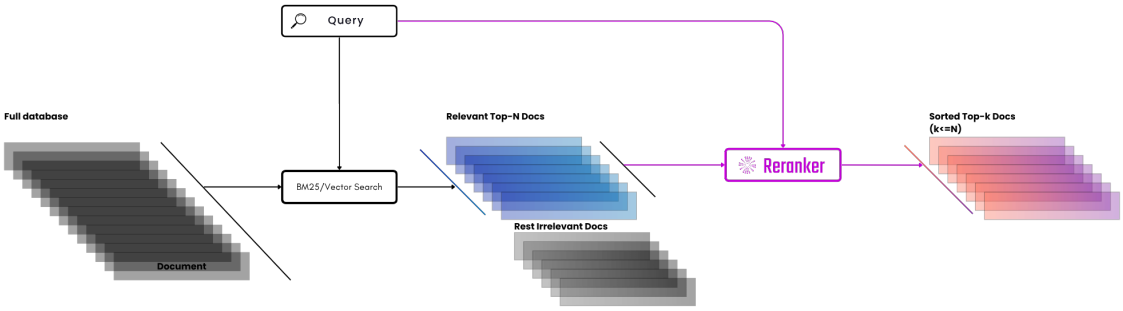

Figure 1: Our retrieval pipeline utilizes an ensemble of three models. The process starts with initial retrieval using BM25 and vector search. Subsequently, the results are reranked with an NLI model. The visualization of this process is adapted from [1].

1.3 Postprocessing

This step involves selecting finished entries from the frontier that mention "Tübingen" and removing duplicate URLs, which may arise from redirects. The corresponding HTML files are parsed using the BeautifulSoup library. Content inside tags such as script, style, header, or footer is removed in order to focus on the main content. Text is extracted from titles, headings, and paragraphs, ignoring text elements shorter than three words for relevance. The extracted texts are cleaned of newlines and concatenated. Texts shorter than 32 words in total are discarded to ensure adequate content length. The Lingua library, which is particularly useful for mixed-language or short texts, is used to accurately detect the language of the texts. Finally, texts identified as English are saved as text files.

2 Retriever

The retrieval process consists of an initial retrieval stage followed by subsequent reranking, as illustrated in Figure 1. During the initial retrieval, a search system using BM25 and embedding similarities was employed to identify a preliminary set of potentially relevant documents. The reranker then assessed these results in greater detail, considering more intricate interactions between the query terms and the document content.

2.1 Initial Retrieval

Lexical search with BM25. To efficiently retrieve the most relevant documents based on term frequency, the Best Matching 25 (BM25) algorithm was used. BM25 is highly effective at quickly identifying documents with exact term matches, managing the influence of common terms, and emphasizing rare terms to enhance their visibility in the dataset. To optimize BM25's performance, essential precomputations were conducted on term frequencies, document lengths, and inverse document frequencies across all documents in the crawl. This implementation follows the original formulation by Robertson et al. [2], using parameter values of $k1 = 2$ and $b = 1$ to adjust term frequency and document length normalization, respectively.

Semantic search with text embeddings. While BM25 effectively retrieves documents based on exact term matches, it may miss documents that are semantically related to the query but lack the specific query terms. To address this limitation, the document selection process is expanded to include documents that are semantically related to the query. This semantic matching is achieved by computing embeddings for each text paragraph using the "all-MiniLM-L12-v2" sentence transformer model [3, 4]. Given the average structure of the crawled documents, a cutoff of 64 paragraphs has been set; documents exceeding this limit are randomly sampled to retain only 64 paragraphs for processing. During the query phase, embeddings for each query token and for the entire query itself are generated by the same sentence transformer model. Inspired by the ColBERT methodology [5], these query embeddings are compared with all paragraph embeddings. The relevance score for each document is then calculated by summing the highest similarity scores obtained for each query embedding against the paragraph embeddings.

We combine the results of both retrieval approaches and select the top 512 documents.

2.2 Reranking

For the final reranking of documents, we use a language model fine-tuned for binary natural language inference (NLI). This task involves determining whether a given hypothesis logically follows from a specified premise. The versatility of this framework allows many classification tasks to be adapted in a zero-shot manner. Specifically, we use the entailment score to assess a document's relevance to a particular query. We treat the document as the premise and craft a hypothesis in the format: "This text is about {query}". This approach enables us to accurately measure the alignment of a given document with a query. For long documents, we divide them into multiple segments and calculate the relevance score based on the highest score across all segments. We employ the Transformers library and the language model "deberta-v3-large-zeroshot-v2.0" for binary NLI [6]. DeBERTa [7] is an advanced version of the BERT model, specifically designed for complex natural language understanding tasks. Its ability to recognize intricate linguistic patterns and semantic relationships makes it highly effective for zero-shot text classification and applicable to a variety of tasks without the need for additional training.

Finally, we select the top-k documents with the highest unnormalized entailment scores.

3 User Interface

The user interface (UI) for our search engine was developed using the React library and Next.js framework for the frontend, and FastAPI library for backend communication.

React's component-based architecture facilitates efficient updates and rendering of UI components. Next.js builds on React by offering server-side rendering for faster page loads and supporting client-side rendering for dynamic content, ensuring quick responses and real-time updates, which are essential for a search application. FastAPI was selected to manage API requests due to its high performance and lightweight nature. Its compatibility with our Python-based retrieval models and crawler code ensures seamless integration. Additionally, its support for asynchronous code allows the system to scale efficiently, which is crucial for a search application expected to handle a large volume of queries.

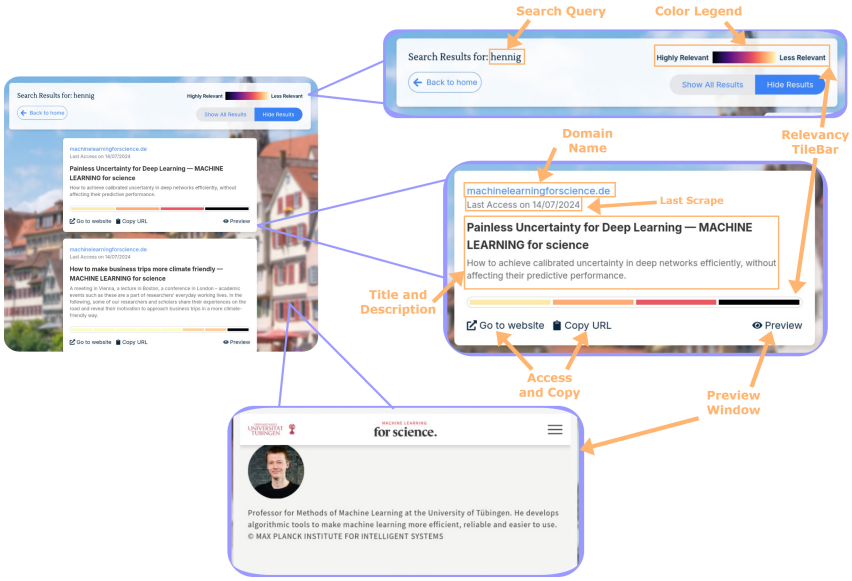

Figure 2: Anatomy of our Search Engine Results Page (SERP): Key Elements and User Interface Components.

The search interface consists of two parts: the search page and the search results page (SERP), as shown in Figure 2. The search page includes a Searchbar component with a category selection feature. Users can choose from various categories (such as tourism or travel) to refine their search queries. Depending on the selected category, the user query is enhanced through a query expansion method, adding related terms or synonyms from WordNet [8] to retrieve a broader set of relevant results and improve the overall search experience.

On the SERP, textual and visual cues assist users in quickly assessing relevance and content. Each result features a TileBar — a horizontal bar with a color gradient ranging from dark purple (highly relevant) to light yellow (less relevant). This gradient enables users to visually gauge both the relevance of each section of the result page and the document's total length at a glance. TileBars vary in length, with longer documents containing up to 8 tiles, each with its own relevance score and color. The number of tiles indicates the document's length, while the tile with the highest relevance determines the overall ranking of the results, presented from top to bottom. This presentation mirrors the decision process of our underlying NLI-based retrieval model, which segments webpage content due to its input size limitations. Consequently, TileBars directly reflect the retrieval process.

Other important elements of each search result include the domain name, title, description texts, and interactive options such as a preview button that toggles an iframe embedding of the respective webpage. This preview feature allows users to view page content without leaving the SERP.

In summary, the design choices for our search application's user interface are driven by the need for speed, modularity, and user-centric features. The main innovations include relevancy-based TileBars, interactive preview features, and other metadata. The design aims to balance comprehensive information with a clean and uncluttered layout.

References

[1] Jina AI. *Reranker Image*. Accessed: 2024-07-19. 2024. URL: https://jina.ai/reranker.

[2] Stephen E Robertson et al. "Okapi at TREC-3". In: *Nist Special Publication Sp* 109 (1995), p. 109.

[3] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: https://arxiv.org/abs/1908.10084.

[4] Wenhui Wang et al. *MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*. 2020. arXiv: 2002.10957 [cs.CL]. URL: https://arxiv.org/abs/2002.10957.

[5] Omar Khattab and Matei Zaharia. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. 2020. arXiv: 2004.12832 [cs.IR]. URL: https://arxiv.org/abs/2004.12832.

[6] Moritz Laurer et al. *Building Efficient Universal Classifiers with Natural Language Inference*. 2024. arXiv: 2312.17543 [cs.CL]. URL: https://arxiv.org/abs/2312.17543.

[7] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2021. arXiv: 2006.03654 [cs.CL]. URL: https://arxiv.org/abs/2006.03654.

[8] George A. Miller. "WordNet: A Lexical Database for English". In: *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*. 1992. URL: https://aclanthology.org/H92-1116.