Sebastian Gürtl, BSc

MOSAIC: Empowering a Modular Framework for Configurable and Tailored Web Search based on an Open Web Index

Master's Thesis
to achieve the university degree of
Master of Science
Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor
Gütl, Christian, Assoc.Prof. Dipl.-Ing. Dr.techn.

Co-Supervisor
Nussbaumer, Alexander, Dipl.-Ing. Dr.techn.

Institute of Interactive Systems and Data Science
Head: Kappe, Frank, Univ.-Prof. Dipl.-Ing. Dr.techn.

Graz, July 2024

Sebastian Gürtl, BSc

MOSAIC: Aufbau eines modularen Frameworks für eine konfigurierbare und angepasste Websuche auf der Grundlage eines offenen Webindex

Masterarbeit
zur Erlangung des akademischen Grades eines
Diplom-Ingenieur
Masterstudium: Software Engineering and Management

eingereicht an der

Technische Universität Graz

Betreuer
Gütl, Christian, Assoc.Prof. Dipl.-Ing. Dr.techn.

Mitbetreuer
Nussbaumer, Alexander, Dipl.-Ing. Dr.techn.

Institute of Interactive Systems and Data Science
Vorstand: Kappe, Frank, Univ.-Prof. Dipl.-Ing. Dr.techn.

Graz, Juli 2024

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Abstract

In an era where information is a key resource, the ability to effectively search and retrieve relevant data from the web is vital. Modern web search engines are essential tools that enable users to efficiently navigate and retrieve relevant information from extensive digital repositories. As the amount of digital content continues to grow, the need for effective and alternative search capabilities becomes increasingly crucial.

The proprietary nature of many existing web search engines limits transparency and user customization, thereby raising concerns about privacy and the concentration of control over information access. In addition, the indices used by these search engines are often not accessible to the public or only available at a cost. This highlights the need for more transparent alternatives that provide greater control over how information is accessed and managed.

Therefore, this work proposes a modular and scalable web search engine framework utilizing the *Open Web Index* from the OpenWebSearch.eu initiative to tackle these issues. The framework is designed to be open and it allows for extensive customization and integration of various components to meet specific user needs. By leveraging the *Open Web Index*, its goal is to promote transparency and user control and to support a more democratic approach to information retrieval of web documents.

In this thesis, we develop the *Modular Search Application based on Index Fractions* (MOSAIC) framework which demonstrates the capability to utilize *Open Web Index* partitions for creating tailored search engines. The modular architecture of the MOSAIC framework ensures flexibility, scalability, and adaptability, which makes it an effective solution for developing customizable search engines.

Through a comprehensive practical user study and focus group discussions with experts, we evaluate the framework's technical approach, modular architecture, and usability and applicability. The results offer valuable insights into the framework's potential to enhance the web search landscape by fostering innovation and improving access to information. These assessments highlight the framework's strengths and areas for improvement, thereby providing direction for future developments.

Hence, this thesis introduces the MOSAIC framework as a valuable addition to web search technology. It integrates features such as the incorporation of *Open Web Index* partitions, modular architecture, and customization options in an open and accessible manner to support search engine developers and operators across a wide range of technical expertise.

Kurzfassung

In einer Zeit, in der Informationen eine zentrale Ressource sind, ist die Fähigkeit, relevante Daten im Internet effektiv zu suchen und abzurufen, von entscheidender Bedeutung. Moderne Websuchmaschinen sind unverzichtbare Werkzeuge, die es den nutzenden Personen ermöglichen, effizient zu navigieren und relevante Informationen aus umfangreichen digitalen Beständen abzurufen. Da die Menge der digitalen Inhalte weiter wächst, wird der Bedarf an effektiven und alternativen Suchfunktionen immer wichtiger.

Der proprietäre Charakter vieler bestehender Websuchmaschinen schränkt die Transparenz und die individuelle Anpassung durch die benutzende Person ein, was Bedenken hinsichtlich des Datenschutzes und der Konzentration der Kontrolle über den Informationszugang aufkommen lässt. Darüber hinaus sind die von diesen Suchmaschinen verwendeten Indizes oft nicht öffentlich zugänglich oder nur gegen Bezahlung abrufbar. Dies hebt den Bedarf an transparenteren, benutzerfreundlicheren Alternativen, die eine größere Kontrolle über den Zugriff auf Informationen und deren Verwaltung ermöglichen, hervor.

Aus diesem Grund wird in dieser Arbeit ein modulares und skalierbares Websuchmaschinen-Framework vorgeschlagen, das den *Open Web Index* der OpenWebSearch.eu-Initiative nutzt, um diese Probleme zu lösen. Das Framework ist offen gestaltet und ermöglicht eine umfassende Anpassung und Integration verschiedener Komponenten, um spezifische Anforderungen von benutzenden Personen zu erfüllen. Durch die Nutzung des *Open Web Index* sollen Transparenz und Benutzerkontrolle gefördert und ein demokratischerer Ansatz für die Informationsbeschaffung von Webdokumenten unterstützt werden.

In dieser Arbeit entwickeln wir das *Modular Search Application based on Index Fractions* (MOSAIC)-Framework, das die Fähigkeit demonstriert, *Open Web Index*-Partitionen für die Erstellung maßgeschneiderter Suchmaschinen zu nutzen. Die modulare Architektur des MOSAIC-Frameworks gewährleistet Flexibilität, Skalierbarkeit und Anpassbarkeit, was es zu einer effektiven Lösung für die Entwicklung personalisierter Suchmaschinen macht.

Durch eine umfassende praktische Nutzerstudie und Fokusgruppendiskussionen mit Experten bewerten wir den technischen Ansatz, die modulare Architektur sowie die Nutzbarkeit und Anwendbarkeit des Frameworks. Die Ergebnisse bieten wertvolle Einblicke in das Potenzial des Frameworks, die Websuchlandschaft durch die Förderung von Innovationen und die Möglichkeiten des Zugangs zu Informationen zu erweitern. Diese Bewertungen heben sowohl die

Stärken als auch die Verbesserungsperspektiven des Frameworks hervor und geben so die Richtung für zukünftige Entwicklungen vor.

In dieser Arbeit wird daher das MOSAIC-Framework als wertvolle Ergänzung zur Websuchtechnologie vorgestellt. Es integriert innovative Funktionen wie die Einbeziehung von *Open Web Index*-Partitionen, modularem Design und Anpassungsoptionen in einer offenen und zugänglichen Weise für benutzende Personen mit einem breiten Spektrum an technischem Fachwissen.

Acknowledgment

Primarily, I would like to express my deepest gratitude to my co-supervisor Dr.techn. Dipl.-Ing. Alexander Nussbaumer and to my supervisor Assoc.Prof. Dr.techn. Dipl.-Ing. Christian Gütl for their professional support and invaluable advice and suggestions throughout all phases of this project. Their insightful ideas, constant availability, and encouragement were instrumental in guiding me through my thesis. I would also like to extend my heartfelt thanks for the opportunity to be part of the OpenWebSearch.eu project and to conduct this thesis within this remarkable initiative.

I am deeply thankful to Dr.rer.nat. Mag.rer.nat. Alexander Steinmaurer who guided me for years and from whom I have learned a great deal. His mentorship and support have been invaluable throughout my academic journey. Moreover, I would like to thank the entire CoDiS Lab team for their support, collaboration, and for creating an enjoyable working environment throughout this project.

Finally, I would like to express my heartfelt gratitude to my entire family for their unwavering support, especially during the challenging times. Their patience and belief in me fueled my determination and drive to succeed. I am also grateful to all my friends for their constant encouragement and for providing enjoyable distractions that helped me stay balanced and motivated.

Contents

List of Figures

List of Tables

Listings

Acronyms

AI Artificial Intelligence. 5, 25, 40, 93, 104, 109–111, 114, 115
API application programming interface. 36, 37, 39, 42, 44, 45, 47–49, 53, 55, 56, 58, 63, 64, 68, 69, 76, 79–81, 83, 87, 90, 96, 101, 110

BBB BigBlueButton. 97
BERT Bidirectional Encoder Representations from Transformers. 15

CIFF Common Index File Format. 31, 32, 43–46, 48, 50, 55, 57, 76, 87, 100–103, 108, 109, 113, 114
CIR conversational information retrieval. 13
CLI command line interface. 43, 76, 77, 87
CoDiS Cognitive and Digital Science. 86
CSS Cascading Style Sheets. 49, 56, 76

DLR German Aerospace Center. 97

FTP File Transfer Protocol. 15

GPT-4 Generative Pre-trained Transformer 4. 14

HITS Hyperlink-Induced Topic Search. 24
HPC high-performance computing. 30, 31
HTML HyperText Markup Language. 5, 16, 20, 30, 35, 49, 56, 68, 69, 76
HTTP Hypertext Transfer Protocol. 30, 44, 56, 64, 65, 68

IDE integrated development environment. 127
IR Information Retrieval. 2, 3, 5–15, 18, 26, 33, 35–37, 39–41, 48, 98–100, 103, 109, 113
ISDS Institute of Interactive Systems and Data Science. 86

JDBC Java Database Connectivity. 49, 55
JSON JavaScript Object Notation. 37, 39, 42, 55, 56, 63, 64, 66, 67, 69, 71, 83

LLM Large Language Model. 6, 13, 14, 21–23, 55, 58, 67, 70, 90, 92, 93, 95–97, 105, 108, 110, 115
LTR Learning to Rank. 25

MOSAIC Modular Search Application based on Index Fractions. 3, 41, 42, 44–51, 53, 54, 56–58, 61, 63, 66, 67, 69, 70, 75–81, 83, 85–92, 94, 95, 97–100, 102–111, 113–116

NLP Natural Language Processing. 7, 13, 15, 22, 24

OWI Open Web Index. 1–3, 5, 28–31, 33, 40–42, 44–46, 51, 53, 55–57, 69, 72, 76, 83, 101–105, 108, 113, 114, 116
OWSAI Open Web Search and Analysis Infrastructure. 28, 76
OWSE-Hub Open Web Search Engine Hub. 32, 33

PDF Portable Document Format. 20

RAG retrieval-augmented generation. 13, 21, 96, 97
REST representational state transfer. 44, 45, 47, 53, 56, 58, 63, 65, 66, 79–81, 83, 87, 90
RQ research question. 2, 85, 102, 108

SEO Search Engine Optimization. 42
SOIF Summary Object Interchange Format. 34
SQL Structured Query Language. 12, 49, 71
SSPL Server Side Public License. 36

TF-IDF term frequency-inverse document frequency. 8
TIRA TIRA Integrated Research Architecture. 31
TTS text-to-speech. 93

URL Uniform Resource Locator. 16, 18, 21, 29, 30, 43, 45, 47, 56, 68–70, 72, 74, 80, 81, 83

WARC Web ARChive. 29, 30, 67, 69, 72
WWW World Wide Web. 1, 2, 7, 14–16, 50

XML Extensible Markup Language. 42, 44, 55, 56, 63, 65, 69, 72, 77, 83

1. Introduction

The pursuit of knowledge is an intrinsic part of human nature, propelling innovation and discovery throughout history (Kvanvig, 2003). From the ancient libraries of Alexandria to the meticulous cataloging of the natural world by Renaissance scholars, humanity has consistently endeavored to collect, organize, and access information. In the modern era, web search engines have emerged as the latest evolution in this quest for knowledge, fundamentally transforming how we navigate the immense amount of digital information accessible in the World Wide Web (WWW).

Web search engines have become an indispensable tool in our daily lives, as they shape the way we access and interact with the extensive volume of information available on the internet. From their humble beginnings with simple keyword searches, search engines have evolved into sophisticated systems capable of understanding and processing complex queries. The importance of search engines in modern society cannot be overstated, as they facilitate the efficient retrieval of information, support academic research, drive business decisions, and even influence social interactions. Despite their crucial role, many existing search engines operate as black boxes, offering little transparency or control over the data they process. This lack of transparency underscores the need for more open, customizable, and transparent search engine frameworks.

The OpenWebSearch.eu[1] project aims to address these issues by promoting open and transparent search technologies to counteract the dominance of proprietary search engines, thus offering an alternative. By maintaining digital sovereignty in Europe, the initiative intends to ensure that web search infrastructure remains under public control and accessible to all. It aims at creating an open and transparent search infrastructure for the web and to democratize access to web data by developing decentralized, community-driven search technologies that prioritize privacy and user control. By providing access to the Open Web Index (OWI)[2], the project enables researchers and developers to build customized search engines and innovative web services.

[1] https://openwebsearch.eu
[2] https://openwebindex.eu

1.1. Aims and Objectives

The primary aim of this thesis is to develop a scalable and modular web search engine framework leveraging the OWI from the OpenWebSearch.eu initiative. This framework seeks to address the limitations of existing web search engines by offering greater transparency, control, and customization, although it is not intended to serve as a large-scale search engine for the entire WWW but as a demonstration framework for utilizing partitions of the OWI.

Additionally, the framework aims to support a modular architecture, allowing for easy integration and customization of various components to meet specific domain and user requirements. Ensuring compatibility with diverse data formats and external systems is another important objective, maximizing the framework's applicability and flexibility.

The system's design prioritizes sustainability and applicability, thereby ensuring it can adapt and evolve in response to emerging needs and technologies. By contributing to the OpenWebSearch.eu ecosystem, the framework aims to foster a collaborative environment where continuous improvements and innovations can be shared and integrated. This contribution not only enhances the system's long-term viability but also supports the broader goals of creating open and transparent search solutions. Through these efforts, the thesis aims to provide a versatile and effective solution for developing custom search engines based on OWI partitions.

Subsequently, the following research questions (RQs) emerge concerning the development and implementation of a modular search engine framework utilizing the OWI:

- **RQ1**: How can partitions of the OWI be effectively integrated and utilized to enhance web search engine capabilities?
- **RQ2**: What are the advantages and limitations of incorporating a modular web search engine architecture, and how does it affect the system's flexibility and scalability?
- **RQ3**: How applicable and user-friendly is a modular framework that utilizes partitions of the OWI for developing and deploying custom web search engines tailored to specific domains or needs?

1.2. Methodology and Contribution

The methodology of this thesis involves designing a framework that addresses the key features and gaps identified in existing Information Retrieval (IR) and web search solutions, while also adhering to the technical specifications of the OpenWebSearch.eu initiative. The design process was informed by a comprehensive literature review, which ensured the integration of best practices and innovations from the field into the framework. Utilizing an iterative approach,

the development process incorporated feedback continuously to refine and enhance the system. The framework's evaluation included a practical user study to assess its applicability and effectiveness and expert evaluations through focus group discussions to gain insights into its practical applicability.

This thesis advances the field of IR and web search systems by demonstrating the practical utilization of OWI partitions by the Modular Search Application based on Index Fractions (MOSAIC) framework. By leveraging the OWI from the OpenWebSearch.eu initiative, it highlights a solid approach to integrate open and transparent search technologies. The modular design of the MOSAIC framework not only enhances flexibility, scalability, and customization but also provides a valuable blueprint for developing adaptable and even declarative search engines. This work enhances the OpenWebSearch.eu ecosystem by providing a system that can be seamlessly integrated into the existing infrastructure, thereby extending its overall capabilities. Moreover, the insights and recommendations presented in this thesis aim to inspire future research and development, further advancing the field of open and transparent search technologies particularly within the OpenWebSearch.eu project.

1.3. Structure

This thesis is organized into seven chapters, each detailing a distinct phase of the project. Chapter 2 explores the structure and functionality of IR systems and web search systems. It delves into the technical aspects of the OpenWebSearch.eu project, including the OWI generation pipeline. Additionally, it reviews existing IR and web search systems, particularly highlighting their features and identifying the gaps they present.

Building on the background and related work, Chapter 3 identifies both functional and non-functional system requirements. It then outlines a conceptual architecture based on these requirements and discusses the design decisions for specific components of the architecture. The chapter also addresses the limitations reflected upon the design phase.

Chapter 4 details the development phase of the project, thereby presenting the technical architecture and key features such as index utilization, metadata enrichment and modular aspects.

The system is evaluated through a practical user study and expert assessments conducted in focus group discussions. Chapter 5 provides insights into the system's effectiveness and usability from both user and expert perspectives.

Chapter 6 reflects on the lessons learned throughout the project, discussing challenges encountered and how they were addressed, as well as insights gained during the development process.

Eventually, Chapter 7 synthesizes the overall research process by summarizing major findings and takeaways. It proposes potential directions for future

research and development, and offers a forward-looking perspective on the project's impact and opportunities for further exploration.

2. Background and Related Work

This chapter introduces relevant concepts that are pertinent to this thesis, thereby providing a thorough background for the subsequent sections. It starts by exploring IR in Section 2.1, detailing key processes such as indexing, query processing, and ranking, and discussing models such as Boolean, Vector Space, and Probabilistic models. The focus then shifts in Section 2.2 to the architecture of web search engines and algorithms that assess page relevance based on several factors and signals. Modern advancements are highlighted, including the integration of Artificial Intelligence (AI) to improve query understanding and result accuracy. Section 2.3 examines technical details of the OpenWebSearch.eu project, as well as vertical search engines, which specialize in specific domains, and modular search frameworks. This overview provides a foundation for a more detailed exploration of modern search technologies and is utilized to define requirements to create a search framework that leverages OWI partitions generated by the OpenWebSearch.eu infrastructure.

2.1. Information Retrieval

IR is the process of extracting relevant information from a collection of resources in response to a user's query. It encompasses both the techniques and methodologies used to find and retrieve documents, data, or other forms of information that meet specific criteria determined by particular information needs. The primary goal of IR is to minimize the gap between the information a user needs and the immense amount of data available, thereby ensuring that the retrieved information is both relevant and of high quality. In contrast to data retrieval, which mainly deals with structured data in databases, IR frequently tackles semi- or unstructured data such as text documents, particularly HyperText Markup Language (HTML) from web pages (Baeza-Yates & Ribeiro-Neto, 2010). Essentially, an IR system endeavors to understand the user's information needs and retrieve relevant documents that are likely to assist the user in their particular task. Typically, users articulate their information needs through queries consisting of keywords, which the IR system then processes and interprets accordingly. IR systems can be applied to various types of data, spanning from text documents to multimedia data. Although web search is presumably the most prominent application of IR nowadays, searchable systems are also used in other application areas such as digital libraries or knowledge management

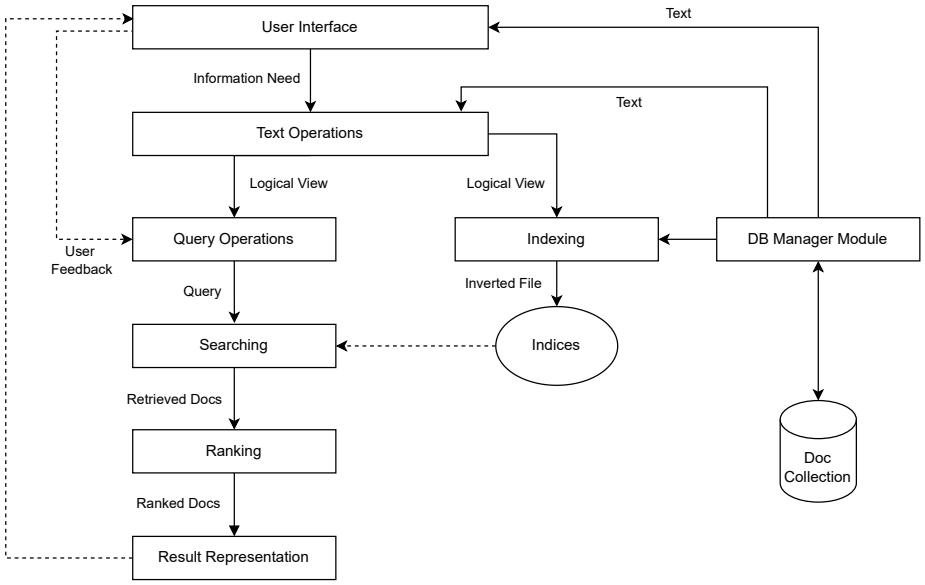


Figure 2.1.: Conceptual architecture of the information retrieval process based on Baeza-Yates and Ribeiro-Neto (2010)

systems. In contrast to browsing, which entails users exploring content more broadly without a specific target in mind, searching focuses on tracking down the relevant information (Ellis, 1989).

2.1.1. Information Retrieval Process

The retrieval process in IR is a complex and multifaceted sequence of actions designed to locate and present relevant information from large datasets based on user queries. This process is fundamental to the functioning of different application areas of IR systems where users seek specific pieces of information from vast collections. According to Baeza-Yates and Ribeiro-Neto (2010), at its core, the general retrieval process transforms a user's information need, expressed through a query in a user interface, into a set of relevant documents or data items (see Figure 2.1). Beyond this fundamental concept, modern approaches in the IR pipeline, such as the use of Large Language Models (LLMs), have since emerged.

2.1.1.1. Indexing

Indexing is a fundamental step in the IR process and plays a crucial role in enabling efficient and effective search capabilities within large datasets. It involves creating a structured representation of the document collection, which facilitates quick and accurate retrieval of relevant documents in response to

user queries. Indexing transforms raw data into a format that is optimized for searching, thereby significantly enhancing the performance of an IR system. The primary purpose of indexing is to organize the information in a way that allows the system to quickly locate and retrieve documents that match the user's information need. Without indexing, a search system would have to scan every document in the collection each time a query is made, which would be computationally expensive and time-consuming, especially for large datasets such as the WWW.

The process of indexing documents consists of different steps to organize and optimize data for efficient and quick retrieval. First, documents are collected and processed to extract text. This text is then normalized through established Natural Language Processing (NLP) techniques such as tokenization, stemming, and removal of stop words, resulting in extracted terms. In the next step, an index structure using the extracted terms is constructed. Ultimately, compression and optimization techniques such as *delta encoding* and *variable-byte encoding* are applied to reduce storage requirements and improve search performance (C. D. Manning, 2008).

The most widely used indexing technique in text retrieval systems is the creation of an *inverted index*. It involves constructing a list of all unique terms in the document collection and mapping each term to a list of documents in which it appears. This allows the system to quickly find all documents containing a given term. The inverted index structure comprises two main components: a *vocabulary* and a *posting list*. The vocabulary is a set of all distinct terms in the collection, while each posting list is a list of document identifiers where the corresponding term appears. The efficient construction and storage of the inverted index are critical for achieving fast query response times and minimizing resource usage (Zobel & Moffat, 2006).

Among many others, another indexing technique worth mentioning is the *forward index*. It is a data structure used in IR systems that maps each document to the terms it contains. This is the inverse of an inverted index, which associates terms with the documents in which they are found. In particular, forward indexes are useful for tasks that require processing documents as whole entities, such as document classification and clustering (C. D. Manning, 2008).

Indexes utilize algorithms that consider the frequency and distribution of terms to help prioritize documents that are more likely to be relevant to the user's information need. This capability also supports the handling of complex queries, including Boolean operations and phrase searches, which allows for more precise and nuanced search capabilities. Moreover, modern indexing systems can update dynamically, thereby efficiently incorporating new documents or changes to existing ones without needing a complete reindexing. This ensures that the IR system remains current and responsive to new information to maintain its usefulness and accuracy over time (Croft et al., 2010; C. D. Manning, 2008).

2.1.1.2. Query Processing

In order to search in indexes, users express their information needs in the form of queries. The process from *query formulation* to document ranking is critical for enhancing search effectiveness and efficiency. In the first step of query processing, users define their search based on perceived information needs. However, initial user queries can often be vague or imprecise and may require refinement to improve retrieval accuracy.

Once the query is received by the IR system, *query parsing* is performed. During this stage, the query is analyzed to determine its syntactic structure (C. Manning & Schutze, 1999). This helps in understanding the parts of the query that are crucial for retrieving relevant information.

An optional yet commonly used next step is the application of *query expansion* to improve the retrieval effectiveness. This involves modifying the original query to include additional terms that are related to the terms in the user's initial query. For instance, this could be synonyms, hypernyms, or other semantically related terms, aiming to bridge the gap between the user's language and the language in the document collection of the system and increasing the likelihood of retrieving relevant documents (Efthimiadis, 1996).

Additional steps can involve *term reweighting*, where the significance of each query term is adjusted based on its distribution across the document corpus, as well as *query optimization* by refining the query execution strategy against the database index reducing the computational resources required, and enhancing retrieval speeds (Buttcher et al., 2016; Croft et al., 2010).

Each of these steps contribute to the overall effectiveness of IR systems, which demonstrates an interplay of linguistic, statistical, and computational techniques designed to optimize both the relevance and precision of retrieved information.

2.1.1.3. Ranking

The core function of the retrieval process is to match the processed query against the indexed data and subsequently rank the matched documents accordingly. The process begins with the matching phase, where the system evaluates which documents in the collection or database correspond to the user's query based on the occurrence of query terms within the documents. Typically, this is achieved through different models such as the Boolean model, the Vector Space model, or Probabilistic models, each providing a different approach to determine how well documents match a query (Baeza-Yates & Ribeiro-Neto, 2010).

Once documents are identified as matches, the ranking phase sorts these documents by relevance. This relevance is often calculated using scores based on textbook approaches such as cosine similarity or term frequency-inverse document frequency (TF-IDF), which assesses the significance of a word to a document within a collection or corpus (C. D. Manning, 2008).

However, the relevance scoring might also involve more sophisticated algorithms such as the feature weighting algorithm *BM25* or language modeling approaches, which consider the probability of terms appearing in documents to estimate their relevance to a query (Robertson et al., 1995).

The system may further refine these scores using additional heuristics or metrics, such as document freshness, user engagement metrics, or the semantic relationships between query terms and document content. For instance, recent documents or those with higher click-through rates from users might be ranked higher under the assumption that they are more useful or relevant. In addition, these ranking algorithms are also capable of adjusting for query-specific features. For instance, if a query implies a need for very recent information, the ranking algorithm may prioritize recency more heavily than it would for a general informational query (Baeza-Yates & Ribeiro-Neto, 2010).

Eventually, the goal of the matching and ranking functionality in IR systems is to deliver the most relevant, useful, and timely information to users in a ranked list. This makes it easier for them to find what they are looking for, thus increasing the user experience.

2.1.1.4. Result Representation

The result representation in IR systems is a critical aspect that directly impacts user satisfaction by determining how search results are displayed. After the matching and ranking phase, the IR system presents the results in a manner that users can easily interpret and use. This typically involves displaying a list of matched documents, where each document is usually accompanied by a context indicating why the document is relevant to the query (Baeza-Yates & Ribeiro-Neto, 2010).

In particular, result snippets are important as they often include highlighted query terms, allowing users to quickly assess the relevance of the document without needing to open it (Buttcher et al., 2016). These snippets are generated either by extracting relevant sentences directly from the document or by creating summaries that capture the essence of the content where the query terms appear.

Furthermore, the organization of results on a page is crucial. Systems often employ user-centric design principles to prioritize clarity and usability. This may include categorizing results into tabs or sections (e.g., images, videos, news) and providing filters or facets that allow users to refine their search based on specific criteria such as date, location, or document type (Hearst, 2009).

Result representation aims not only to inform but also to facilitate further interaction with the retrieved information, for instance, through refining the search, accessing detailed document views, or exploring related topics. Consequently, this functionality comprises the fields information science, graphic design, and human-computer interaction principles.

2.1.1.5. User Interaction and Relevance Feedback

An essential component of IR systems is the user interaction and relevance feedback, allowing the system to enhance search accuracy and user satisfaction through iterative refinement. User interaction involves how users engage with the system, from the initial query input to retrieving, navigating and manipulating search results. This interaction can provide implicit feedback on user preferences and behaviors, such as which links they click on, how long they view a page, or how they modify their queries in response to search results (Baeza-Yates & Ribeiro-Neto, 2010).

In contrast, relevance feedback is a more explicit method of interaction where users directly inform the system about the relevance of retrieved documents. For example, this can be done by marking results as *relevant* or *not relevant*, which the IR system then uses to adjust the search algorithm. This feedback helps the system to better understand the user's specific needs and refine the search results accordingly. The use of relevance feedback loops enables the system to learn and adapt, potentially improving the precision and recall of the search results over time. For instance, if a user consistently marks documents containing certain terms as relevant, the system may boost the weighting of these terms in future searches (C. D. Manning, 2008).

The opportunity of user interaction and relevance feedback enables IR systems to evolve with user needs and preferences, ultimately improving the general search experience.

2.1.2. Information Retrieval Models

The foundation of any effective search engine or database system is built upon its underlying retrieval model that makes the IR process possible. These models are theoretical constructs that describe how systems can determine the relevance of documents to a user's query (Baeza-Yates & Ribeiro-Neto, 2010). Each model offers a unique approach to handling and interpreting the large amounts of data encountered in search tasks, varying significantly in terms of complexity, effectiveness, and suitability for different types of data or queries. Baeza-Yates and Ribeiro-Neto (2010) formally define the characterization of IR models using the four components D, Q, F and $R(q_i, d_j)$.

1. D denotes the representations for the documents in a database or a collection.
2. Q denotes the representations for the information needs of a user, also named queries.
3. F denotes the framework that is utilized for modeling representations of documents, information needs and the relationships between document representations and queries.

4. $R(q_i, d_j)$ denotes the ranking function constructed by the framework. For a query $q_i \in Q$ and a particular document representation $d_j \in D$, the ranking function yields a real number contributing to an overall ordering of the documents depending on the query.

In general, IR models serve as planning tools for actual implementations of IR systems. Focusing on classic IR, the fundamental models comprise the *Boolean model*, *Vector Space model* and *Probabilistic model*.

2.1.2.1. Boolean Model

As an exact match model, Boolean models operate based on Boolean logic, using operators such as *AND*, *OR*, and *NOT* to combine search terms. Documents are retrieved based solely on whether they contain or do not contain the specific terms stipulated by the query, making the results very explicit and dependent on exact matches with the query terms. This model treats each document as a set of keywords, allowing for straightforward matching but without considering the context or frequency of the terms within the documents. While Boolean models provide clear and easy-to-understand results, they lack nuance in ranking since they do not differentiate documents by relevance beyond meeting the query criteria in its classic form (Salton et al., 1983). Consequently, this can lead to either overly broad or overly restrictive results, depending on the user's formulation of the query.

2.1.2.2. Vector Space Model

In order to be able to rank documents, Salton (1983) introduced an approach to measure the similarity between a query and documents. Given a multidimensional Euclidean space, both the query and the index representations are vectors that are embedded in this space, where each dimension corresponds to a unique term in the overall corpus. Similarity between a document and a query is determined by calculating the cosine of the angle between their respective vectors, with a smaller angle indicating higher relevance. The Vector Space model allows for partial matching and ranking of documents based on their cosine similarity scores, which offers a more nuanced approach to retrieval than Boolean models (Hiemstra, 2009).

2.1.2.3. Probabilistic Model

An alternative approach that utilizes probability theory to determine the likelihood that a document is relevant to a given user query is defined by the Probabilistic model. These models calculate the probability of relevance based on the presence or absence of terms in the documents, assuming that each term independently contributes to the document's relevance. It operates on the

principle that there exists an "ideal answer set" of documents which are exactly relevant to a user's query. However, the challenge lies in defining the properties of this ideal set, which are not known at the query time but are estimated using index terms to improve the retrieval process. During the search, initial guesses of these properties help generate a preliminary probabilistic description, which is refined through user interaction as they identify relevant documents. This iterative process aims to converge the system's understanding of an ideal answer set closer to the actual set, thereby enhancing the accuracy of document retrieval based on the calculated probabilities of relevance (Robertson, 1977).

2.1.3. Query Languages

A compelling component of a model in an IR system are queries specifying the information need of users. Serving as the bridge between user queries and the comprehensive document collections and databases of information these systems search in order to assist users in finding and retrieving the information they need, query languages are a fundamental tool. They enable users to articulate their information needs in a structured format that the system can understand and process efficiently. The development and refinement of query languages have been central to the evolution of IR systems, reflecting advances in technology and changes in user behavior ranging from simple keyword-based systems to complex languages capable of handling structural and natural language queries (Baeza-Yates & Ribeiro-Neto, 2010).

Keyword-based queries are the most basic form of query language used in information retrieval systems. Users input one or multiple keywords or phrases that they believe are relevant to their information need, and the system searches for these terms across its database. While this type of query language is easy to express and intuitive, it may not always deliver precise results due to its reliance on exact matches between the query terms and document content (Baeza-Yates & Ribeiro-Neto, 2010).

Boolean queries enhance the precision of search results by allowing users to combine keywords with Boolean operators such as *AND*, *OR*, and *NOT*. This approach enables users to refine their search criteria significantly, dictating how different terms relate to each other within the search process, thus exploiting the compositional scheme by using combinations to create complex queries (C. D. Manning, 2008).

Structural queries are used predominantly in database management systems but are relevant to IR when dealing with structured data. For instance, the Structured Query Language (SQL) allows for precise queries about specific data attributes stored in structured database formats. In IR systems, the usage of such languages and query protocols is beneficial for queries that require specific criteria to be met, such as dates, tags, or metadata fields, providing a high level of precision in searches. Additionally, structural queries can be

utilized to enrich relevant aspects and search in certain parts (Croft et al., 1991).

Opposed to structural queries, *natural language queries* represent a more advanced type of query language that allows users to enter search terms in the form of conversational language. This type of query language uses NLP to parse and understand the user's intent, which makes the search process more intuitive and less restrictive. While such languages offer a more user-friendly interface, they present significant challenges in accurately interpreting and processing the user's intent due to the complexity and ambiguity of natural language (Lewis & Jones, 1996).

2.1.4. Advanced Techniques in Modern Information Retrieval

Modern approaches in IR systems encompass several advanced techniques aimed at improving the accuracy and relevance of search results. One significant development is the use of neural networks and deep learning models, which enhance traditional methods by better understanding the semantic context of queries and documents. To process and generate human-like text, thus significantly improving query processing and document ranking, transformer models and LLMs can be integrated (Tang et al., 2024).

Neural approaches to conversational information retrieval (CIR) have gained prominence, enabling interactive and multi-turn conversations with users to refine and clarify their search intents, leading to more accurate results. CIR systems utilize advanced NLP and dialogue management techniques to maintain context and coherence across interactions, making the search experience more intuitive and user-friendly (Gao et al., 2023).

Particularly important for addressing the user's information needs is the incorporation of retrieval-augmented generation (RAG), where LLMs are used to generate responses based on retrieved external knowledge, thus reducing the likelihood of generating irrelevant information. Furthermore, modern IR systems often utilize topic modeling techniques to improve the representation and indexing of text documents by identifying the underlying topics within them. The adoption of multilingual retrieval models has also been critical in making IR systems more accessible and effective across different languages and regions (Tang et al., 2024). Word embeddings provide dense vector representations of words that encapsulate semantic meanings and relationships. Useful in various NLP tasks, including document classification, question answering, and named entity recognition, they offer a more sophisticated representation of text compared to traditional keyword-based methods. Consequently, word embeddings improve the indexing of documents and the retrieval accuracy by capturing semantic relationships between terms, as well as by allowing the system to understand contextual nuances and synonyms in queries and documents (Kusner et al., 2015).

Additionally, the use of graph databases is gaining traction, particularly for

handling complex, multi-hop queries by modeling data as interconnected nodes and edges, which improves retrieval accuracy and efficiency (Barceló Baeza, 2013).

Zhu et al. (2023) outline the integration of traditional term-based approaches with modern neural architectures to combine the strength of both. Traditional methods guarantee fast response times and high efficiency, while neural models are characterised by their ability to capture complex contextual signals and semantic nuances. For instance, the introduction of LLMs such as Generative Pre-trained Transformer 4 (GPT-4) has further impacted IR by improving key components including query rewriters, retrievers, rerankers, and readers. This enhancement leads to increased accuracy and relevance of retrieved information. Subsequently, the integration of traditional and modern techniques creates a robust and efficient IR pipeline, capable of effectively addressing user queries and challenges such as data scarcity and interpretability. (Zhu et al., 2023).

These advancements collectively improve the performance and user experience of modern IR systems, making them more robust and adaptable to handling complex search queries.

2.2. Web Search Engines

One particular application area of an IR system is the possibility to search having digital information stored in the WWW as underlying large-scale dataset. In the rapidly evolving landscape of digital information with a current estimated size of around 60 billion web pages[1] in the WWW, web search systems stand as key tools that empower users to navigate the immense expanse of the internet efficiently. These systems, which range from ubiquitous search engines such as Google[2] and Bing[3] to specialized academic search platforms such as Google Scholar[4], utilize complex algorithms to fetch, index, and retrieve information in response to user queries. The effectiveness of a web search system is significantly influenced by its ability to provide relevant and timely results. Relevance algorithms, which assess the significance of a web page to a given query, are continually refined to improve accuracy and user satisfaction (C. D. Manning, 2008).

Moreover, these systems must also navigate the challenges posed by the dynamic, heterogeneous and rapidly changing nature of web content, the large scale of data, and the diverse and changing information needs of users (Lewandowski, 2015).

[1] https://www.worldwidewebsize.com
[2] https://google.com
[3] https://bing.com
[4] https://scholar.google.com

2.2.1. History and Evolution

While the first hypertext systems had been developed in the 1960s and 1970s, Tim Berners-Lee and his colleagues started the development of today's famous global information medium at the international scientific organization CERN in Geneva, Switzerland in 1989 (C. D. Manning, 2008). Since then, the history of web search engines is marked by significant milestones and continuous evolution. The journey began with Archie in 1990, the first search engine created to index File Transfer Protocol (FTP) archives. This was followed by the development of web crawlers such as JumpStation and WebCrawler in the early 1990s, which automated the process of indexing web pages. For instance, WebCrawler, launched in 1994, was the first to index entire pages, allowing full-text search capabilities. During the mid-1990s, various search engines such as Lycos, AltaVista, and Ask Jeeves introduced innovative features that improved the relevance and user-friendliness of search results (Seymour et al., 2011).

Also developed in the 1990s by Bowman et al. (1995), Harvest offered a decentralized approach to IR on the WWW by distributing the tasks of collecting, organizing, and searching documents across multiple nodes. Unlike traditional methods that centralize all web documents into a single collection, Harvest uses customizable tools, so-called *Gatherers* and *Brokers*, to efficiently gather, summarize, replicate, distribute, and search documents. In consequence, this significantly reduced server load and network traffic while directing user queries to the most relevant sources (Bowman et al., 1995).

Google emerged in 1998 and revolutionized web search with its PageRank algorithm, which assessed the importance of web pages based on link structures. This approach significantly improved the relevance of search results, making Google the dominant search engine by the early 2000s. The introduction of advanced filters, crawling patterns, and later, machine learning algorithms such as RankBrain and Bidirectional Encoder Representations from Transformers (BERT), continued to enhance Google's search capabilities (Kathuria et al., 2016; Padaki et al., 2020).

The evolution of web search engines also saw the rise of mobile and personalized search, driven by the emergence of mobile devices in the 2000s. This shift made search engines context-aware and highly personalized, further improving user experience. Emerged in the late 2000s and utilizing NLP and knowledge graphs, semantic search enabled search engines to understand the intent behind queries and deliver more accurate results (Seymour et al., 2011).

Early search engines often returned irrelevant results, and the rise of automated crawlers led to issues such as spam. Recent and modern search engines, while highly sophisticated, face challenges related to privacy, data security, and the complexity of maintaining and updating algorithms to handle the large-scale and dynamic nature of the web (Cambazoglu & Baeza-Yates, 2022).

2.2.2. Anatomy

At their core, web search systems are built upon three fundamental processes: crawling, indexing, and querying (see Figure 2.2). Web crawlers, also known as spiders, systematically browse the WWW to collect data from web pages (Brin & Page, 1998). This data is then organized into indexes, massive databases that use sophisticated data structures to efficiently store and access information (Page et al., 1999).

The architecture of these systems encompasses several key components to handle these processes effectively. The web crawler, or spider, navigates through web pages using a Uniform Resource Locator (URL) Frontier to manage and queue URLs for visitation. Once web pages are fetched, an HTML parser processes the pages and extracts links for further crawling and content for indexing. The deduplication component ensures that only unique content is indexed, preventing redundancy in the search index (Brin & Page, 1998; C. D. Manning, 2008).

As the data is collected, the indexer tokenizes the text into individual terms, applies stemming and lemmatization to normalize the terms, and organizes them into an inverted index. This inverted index maps terms to their occurrences in the document collection, enabling efficient query processing. During query processing, the system may employ advanced approaches such as query expansion techniques to enhance the query with related terms, which can substantially improve the retrieval of relevant documents (C. D. Manning, 2008).

When a user submits a query, the query parser processes it by extracting key concepts and generating a structured query. The search engine then uses this structured query to retrieve relevant documents from the inverted index. The ranking component, which may involve both large-scale and small-scale ranking algorithms, evaluates the relevance of these documents based on various signals and factors, such as term frequency, document popularity, and user engagement metrics (Gudivada et al., 2015; C. D. Manning, 2008).

Eventually, the search results aggregation component compiles the ranked results and they are presented to the user through a search interface. User engagement and relevance feedback mechanisms are employed to continuously refine and improve the search algorithm to ensure that the search engine delivers the most relevant results over time (Kopliku et al., 2014; C. D. Manning, 2008).

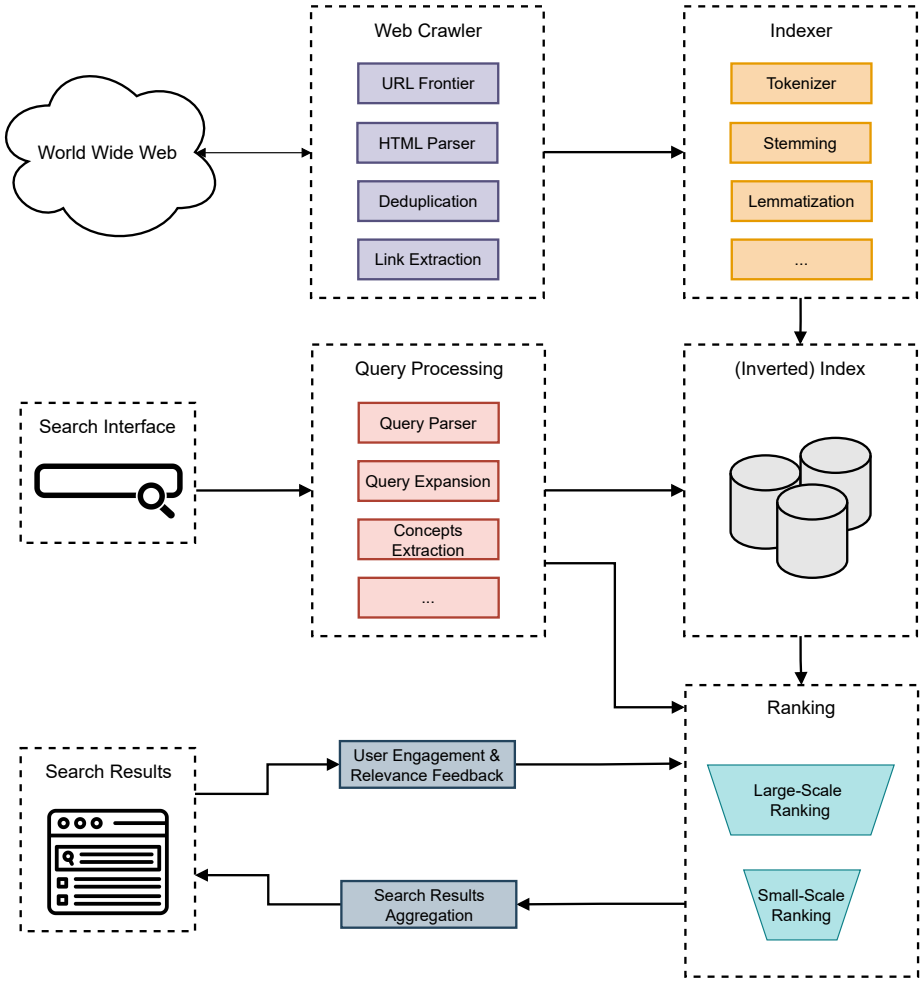


Figure 2.2.: Functionality of web search engines based on the crawler-indexer architecture based on Jones et al. (2004), Kopliku et al. (2014), and Wachsmuth et al. (2017)

2.2.3. Crawling

Textbook IR systems typically handle structured and homogeneous data sources, such as academic databases, where documents are well-organized and the user base consists of researchers and professionals who are experienced at forming precise search queries. However, the web contains unstructured information from various sources, necessitating more complex indexing and retrieval mechanisms (Garg & Sharma, 2012).

A crawler, also known as a spider or bot, is a fundamental component of web search systems, designed to systematically browse the web and collect data for indexing. The crawling process begins with a seed URL list, which the crawler visits to fetch web pages. Upon fetching these pages, the crawler extracts hyperlinks and adds them to its list of URLs to visit next, ensuring a comprehensive exploration of the web. This process involves several stages: downloading web pages, parsing the content, and storing relevant information in a structured format (Brin & Page, 1998).

Crawlers must handle vast amounts of data and operate efficiently to keep up with the dynamic nature of the web. They employ techniques such as parallel processing and distributed crawling, where multiple bots work simultaneously across different segments of the web to speed up data collection. To avoid overloading web servers, crawlers adhere to the *robots.txt* protocol, which specifies rules and restrictions set by websites on how they should be crawled (Garg & Sharma, 2012).

One of the key challenges in web crawling is dealing with duplicate content and ensuring that the index remains free of redundant information. Advanced algorithms help in detecting and filtering out such duplicates. Additionally, crawlers prioritize URLs based on various factors, including the freshness of content and the importance of the page, often determined by relevance algorithms and metrics such as PageRank (Brin & Page, 1998).

Efficient data storage and management are crucial, as crawlers gather large volumes of web pages. The collected data is then parsed and indexed, creating a searchable database that enables quick retrieval of relevant information in response to user queries. This process is essential for maintaining an up-to-date and comprehensive index, which forms the backbone of any effective web search system (Garg & Sharma, 2012).

Figure 2.3 illustrates a typical architecture of a distributed web crawler, featuring multiple processes across different machines. Each process contains multiple worker threads that continuously cycle through tasks. During each cycle, a worker thread fetches a URL from the Frontier data structure, downloads the web page, extracts hyperlinks, filters out unwanted or duplicate URLs, and prioritizes them for future crawling (Olston, Najork, et al., 2010).

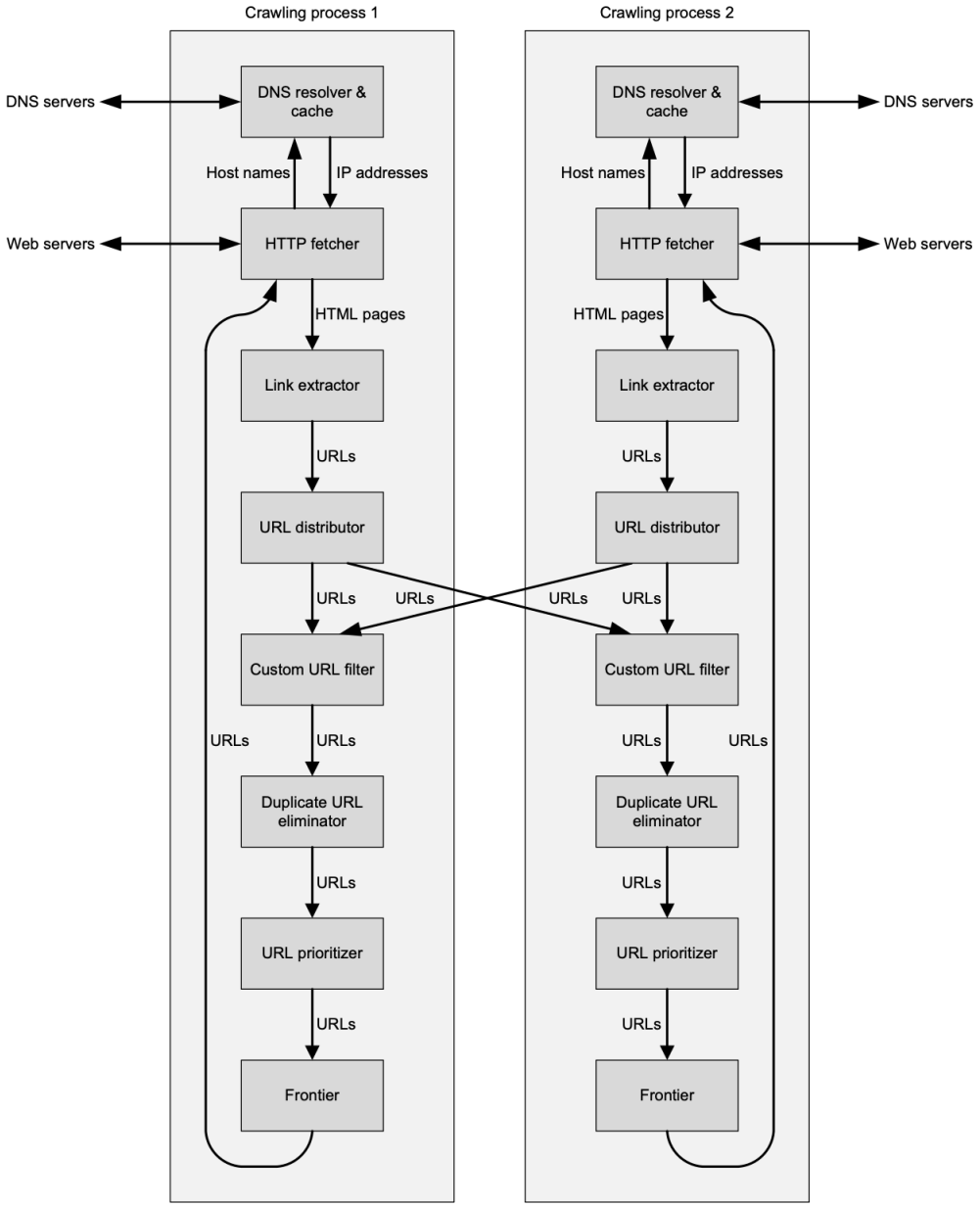


Figure 2.3.: Prototypical distributed web crawler architecture (Olston, Najork, et al., 2010)

2.2.4. Web Documents Indexing

Once a crawler fetches a web page, the indexer processes its content, extracting relevant information such as text, metadata, and hyperlinks. This extracted data is then converted into a structured format, most commonly into an inverted index. The indexing process itself begins with parsing the HTML content of web pages to remove unnecessary elements such as scripts and styles, focusing on the extraction of the main text and important metadata. To ensure uniformity, the text is then tokenized into individual terms, which are standardized through processes such as stemming and lemmatization. For instance, different inflected forms of a word (e.g., ”search”, ”searching”, ”searched”) are grouped to a single root form (e.g., ”search”) (Hyusein & Patel, 2003).

To enhance search relevance, indexers also incorporate additional contextual information, such as the importance of a term based on its location within the web page (e.g., title of the page, headers) and the anchor text of hyperlinks that point to the page. This helps in ranking the pages more effectively during search queries. The index is stored in a highly optimized and compressed format to ensure quick access and efficient use of storage space. Advanced indexing algorithms and data structures, such as B-trees, Skip-Lists and hash tables, are employed to manage the index and facilitate rapid lookup operations (Malki, 2016).

However, indexing is not limited to HTML content of web pages. For instance, Google indexes images by analyzing their alt text and surrounding context, and it also processes structured data using schema.org markup to provide rich search results for specific types of content such as events and reviews. Additionally, Google considers various document types in its indexing process, including Portable Document Format (PDF), Word documents, and PowerPoint presentations, extracting their text content for indexing similarly to HTML pages. This extensive indexing capability allows Google to deliver accurate and diverse search results, responding to the wide-ranging needs of its users (Adnan & Akbar, 2019).

2.2.5. Web Search Interface

The search interface is an essential component of web search engines, acting as the main point of interaction between users and the search system. A well-crafted search interface not only facilitates the retrieval of relevant information but also enhances the overall user experience by making the process intuitive and efficient.

2.2.5.1. Key Components

One of the primary elements is the search box, which allows users to input their queries. Auto-suggestions and spell-checking enhance user experience by

predicting queries and correcting errors. Search buttons and controls enable query submission and refinement. Advanced search options provide filters to narrow down results by various criteria. Navigation tools such as pagination and breadcrumbs streamline the process of moving through search results, while personalization options including customized layouts and personalized suggestions adapt the search environment to meet individual preferences. (Wilson et al., 2010).

The representation and visualization of search results in web search engines are particularly relevant to user interaction and information comprehension. Fox et al. (2005) outline that effectively displayed results significantly enhance user satisfaction and engagement. Typically, results are organized in a list format that includes the title along a snippet of content, and the source URL, which provides a quick overview of the information relevance. Modern search engines employ rich snippets and visual previews, such as images and summarizations, to provide more context and assist in quicker decision-making. In addition, graph-based visualizations are emerging as they illustrate relationships between different search outcomes or their relevance over time, thus making the data more accessible and easily understandable (Hearst, 2009).

Timeline visualizations for search results have been particularly beneficial for queries related to historical data or progression, enhancing the user experience by facilitating a more natural exploration of information (Alonso et al., 2009).

2.2.5.2. Enhancements in Modern Search Interfaces

Modern search interfaces incorporate voice search and visual search capabilities. Voice search enables users to vocalize their queries instead of typing them by leveraging speech recognition technology, while visual search leverages image processing technologies. These features increase the number of possibilities users can interact with search engines (Schalkwyk et al., 2010; Smith & Chang, 1997).

Interactive features such as maps, instant previews and carousels, combined with a focus on accessibility and intuitive design, are essential for enhancing the overall user experience (Qvarfordt et al., 2013).

Recent advancements in search interfaces include the integration of LLM and RAG systems. These technologies enhance the search experience by providing more accurate and context-aware responses directly within the interface. LLMs can understand and process complex natural language queries, so the search interface can return more precise and relevant results, often displayed in a more conversational and human-like manner. RAG systems combine the generative capabilities of LLMs with traditional retrieval methods to ensure that the information provided is both relevant and grounded in factual data. This integration allows search interfaces to present results that are not only accurate but also contextually enriched, thereby improving user satisfaction. Additionally,

these systems can dynamically generate answers and summaries, which enhance the interactivity and responsiveness of the search interface (Zhu et al., 2023).

Query expansion techniques powered by LLMs improve the search interface by suggesting additional relevant terms or concepts that can assist users refine their queries and explore related topics effortlessly. This leads to a broader and more accurate set of search results, yet displayed in an intuitive manner. Word embeddings further enhance this by understanding and representing semantic relationships between words. Consequently, this allows for more sophisticated search result matching and ranking (Bacciu et al., 2024; Kusner et al., 2015).

Moreover, advanced re-ranking algorithms, often enhanced by LLMs, refine the order of search results based on deeper contextual relevance. This enhances the search interface by making the results more immediately useful and accessible to users (Xu et al., 2024).

2.2.5.3. Challenges

Web search interfaces face several challenges that can affect their efficacy and user satisfaction. One major challenge is the diverse structure and functionality of online resources, which makes consistent and effective searching difficult. Each resource may require different search techniques, complicating the design of universal search strategies and interfaces (Stansfield et al., 2016).

Advanced algorithms are needed to ensure that search results align with user expectations. In addition, privacy concerns require secure data handling to protect personal information, yet maintaining transparency with users about how their data is used. Enhancements in technology must balance quick response times with the computational demands of processing large data volumes (Hearst, 2009; Shou et al., 2012).

Moreover, the rapid evolution of web technologies and the continuous expansion of information on the internet demand ongoing updates and improvements to search interfaces. This requires the development of sophisticated algorithms and designs that are capable of adapting to evolving user behaviors and expectations, incorporating increasingly personalized and context-sensitive search features, also addressing the accessibility needs of users with disabilities, particularly blind users (Naseer et al., 2023).

2.2.6. Query Processing

The query processing component is responsible for interpreting and refining user queries to enhance search accuracy and relevance. This process involves parsing and semantic analysis of the query, often employing NLP techniques to grasp the contextual meanings of words. Techniques such as query expansion are used to broaden the search by including synonyms or related terms, thereby enhancing the retrieval effectiveness. Additionally, modern search engines

may apply machine learning models to predict and refine query intent based on previous user interactions and other contextual data. This sophisticated approach helps in accurately aligning search results with what the user is truly seeking, improving both user satisfaction and the efficiency of the search engine (Chen et al., 2021; Rose & Levinson, 2004).

2.2.6.1. Query Expansion

By appending additional, contextually relevant terms to the original search query, query expansion improves the breadth and precision of search results. It is especially effective in resolving ambiguities within short queries by drawing on user-specific data, such as personal information repositories, to personalize and refine search outputs (Chirita et al., 2007).

Furthermore, utilizing LLMs for query expansion represents a significant advancement in enhancing search engine performance as they are able to interpret the nuanced intent behind user queries. This allows web search systems to automatically expand queries with additional relevant terms or phrases. This process not only enriches the query but also increases the likelihood of retrieving highly relevant and comprehensive search results. Moreover, the use of LLMs for query expansion helps in addressing the ambiguity and context-specific nature of queries, thereby providing a more tailored and insightful search experience (Wang et al., 2023).

2.2.6.2. Query Rephrasing

Query rephrasing is an approach in web search engines that involves transforming user queries into alternative versions to improve the retrieval of relevant documents. For instance, this process can be achieved using LLMs, which can generate multiple paraphrases of the original query. This enables to capture different expressions of the same intent. By rephrasing queries, search engines can match a broader range of documents, thereby increasing the likelihood of retrieving relevant information. This technique enhances the search experience by addressing issues such as vocabulary mismatch and varying ways users may phrase their search requests (J. Liu & Mozafari, 2024).

2.2.6.3. Query Contextualization

Query contextualization, for instance, by employing user-embeddings, leverages a user's historical search data to enhance the understanding of their current queries. By creating embeddings that represent a user's search behavior and preferences, search engines can more effectively interpret the context of a query within the user's broader search session. This approach ensures that search results are more personalized and relevant, effectively addressing the user's specific information needs (Ning et al., 2024).

2.2.7. Matching and Ranking

The ranking of search results in web search systems is determined by evaluating various on-page and off-page factors. On-page factors include the presence and frequency of keywords, the structure of content, and the use of metadata such as title tags and descriptions. Off-page factors, such as the number and quality of backlinks, indicate the authority and relevance of a web page. User engagement metrics, such as click-through rates and bounce rates, also play a crucial role in assessing the relevance of search results. Advanced algorithms incorporate NLP and machine learning to understand query context and improve accuracy (Adnan & Akbar, 2019). By combining these elements and employing various ranking approaches, web search systems aim to deliver the most relevant and proper results to users.

2.2.7.1. HITS

The Hyperlink-Induced Topic Search (HITS) algorithm, developed by Kleinberg (1999), is a foundational link analysis algorithm used in web search systems to identify authoritative web pages. It assigns two scores to each page: an authority score, which reflects the value of the content, and a hub score, which measures the quality of its outbound links to other pages. The algorithm operates on a recursive principle where good hubs link to good authorities, and vice versa, enhancing the effectiveness of search results by emphasizing both the quality of content and the structure of the web. (C. D. Manning, 2008; Najork et al., 2007).

2.2.7.2. PageRank

Another popular and widely used algorithm is Google's PageRank. The PageRank algorithm is a method utilized to rank web pages in search engine results based on their importance, which is determined by analyzing the quantity and quality of links pointing to them. Developed by Larry Page and Sergey Brin, PageRank treats each link to a page as a vote, where votes from more significant or highly-ranked pages carry more weight. Initially, all pages are assigned an equal rank, which is then adjusted iteratively based on the link structure of the web. This process involves the redistribution of rank through outbound links, with each iteration refining the ranks until they converge (Brin & Page, 1998).

A key component of PageRank is the damping factor, typically set around 0.85, which simulates the likelihood that a user will continue clicking on links rather than starting a new search. This factor helps ensure a more even distribution of rank and prevents certain pages from accumulating excessive influence. The algorithm effectively captures the idea that important pages are those linked to by many other important pages (Page et al., 1999).

The computational efficiency of PageRank is achieved through the power method, an iterative technique that finds the dominant eigenvector of the link

matrix, representing the PageRank scores. This method allows for the practical application of PageRank on the web's vast and complex structure, enabling it to scale effectively with the size of the web (Franceschet, 2011).

2.2.7.3. Weighted PageRank

The Weighted PageRank algorithm introduces an enhancement to the traditional PageRank by incorporating the importance of both the inbound and outbound links of pages into the ranking process. Unlike the standard PageRank, which equally distributes the rank value among all outbound links, Weighted PageRank assigns weights to these links based on the popularity of the linked web pages. This popularity is measured by the number of inlinks and outlinks a page has, allowing Weighted PageRank to differentiate between links based on their relevance and importance. Weighted PageRank thus distributes rank scores not uniformly but proportionately to the significance of each link, aiming to return more relevant and higher quality pages in response to a user's query (Xing & Ghorbani, 2004).

2.2.7.4. Fairness-Aware Ranking

Fairness-Aware ranking frameworks are designed to address and mitigate algorithmic bias in systems used for ranking individuals, typically in web-scale search and recommender systems. This is accomplished using specialized re-ranking algorithms that modify the order of results to meet fairness criteria, with the goal of achieving a balanced representation among diverse groups. This not only enhances fairness but also adheres to principles such as equality of opportunity and demographic parity, tailored to the specific distribution goals set for the protected attributes. Subsequently, this methodology represents a significant step towards more ethical web search and AI practices by actively counteracting bias and promoting diversity (Geyik et al., 2019).

2.2.7.5. Advanced Matching and Ranking Techniques

Improving the relevance and accuracy of search results in web search engines relies on modern sophisticated ranking algorithms. Learning to Rank (LTR) models utilize machine learning techniques to rank search results based on their predicted relevance. These models are trained on extensive datasets to optimize performance. Neural ranking models enhance this process by applying deep learning to understand and evaluate the semantic relevance of documents, which offers a more nuanced and precise ranking. Furthermore, these algorithms adapt rankings in real-time by incorporating user feedback and engagement metrics. Additionally, leveraging embeddings significantly enhance the semantic understanding of queries and documents. By capturing and representing semantic relationships between terms, embeddings improve

the matching process, resulting in more accurate and contextually relevant search rankings (Guo et al., 2020; T.-Y. Liu et al., 2009).

2.2.8. Types of Web Search Engines

Web search engines come in various types, each designed to meet specific requirements or to enhance particular types of IR.

2.2.8.1. General Search Engine

General search engines are the most widespread type of search platforms, designed to address a broad spectrum of queries by indexing extensive amounts of diverse and unstructured content from the internet. The strength of general search engines is their ability to quickly deliver extensive results across various types of content, including news articles, academic papers, multimedia, and social media posts. However, they can sometimes prioritize popular or trending content, which may not always align with more specific or academic research needs (Chau et al., 2005; Shu et al., 2017).

2.2.8.2. Vertical Search Engine

In contrast to general search engines, specialized search systems, called vertical search engines, are platforms that focus on a specific segment of online content, such as academic literature, images, or news. These search engines are designed to provide more precise and contextually relevant results within a specific domain. They utilize appropriate algorithms to deeply index and understand the specific types of data they handle. Vertical search engines enhance user search experiences by filtering out unrelated information and providing quicker and more accurate access to the needed resources within a particular field (Bostoen, 2018).

2.2.8.3. Meta Search Engine

Meta search engines are a unique type of search tool that aggregates results from multiple other search engines without conducting their own crawling and indexing. By sending a user's query to several other engines simultaneously and compiling the results, meta search engines provide a comprehensive overview of available information across various platforms. This approach can offer a broader search coverage and help users discover a diverse range of sources and content types more quickly than using a single search engine. However, the quality of meta search results is largely influenced by the choice of search engines used for querying. In addition, there may be challenges related to duplicate results and the integration of results from different sources (Meng et al., 2002).

2.2.8.4. Private Search Engine

With a focus on preserving privacy and ensuring security, private search engines do not track personal data, such as search history or location, which contrasts with many search engines that often collect extensive user data for advertising and personalization purposes. These engines, such as DuckDuckGo[5], StartPage[6] and Brave Search[7], offer anonymous browsing experiences and do not use tracking cookies, ensuring that users' search activities remain confidential. Moreover, private search engines typically utilize encryption techniques to secure search queries and prevent unauthorized data access, aiming to provide a secure and private search environment for users concerned about digital privacy and data security. Such features are crucial in maintaining user trust and providing safe browsing experiences in an era where data privacy is a significant concern (Castellà-Roca et al., 2009).

2.2.8.5. Enterprise Search Engine

Designed for use within corporations, enterprise search engines index and retrieve content from private repositories within a corporate environment. These search engines are tailored to handle the specific needs of businesses, allowing employees or members to quickly find internal documents, data, and other resources essential for their work. Unlike public web search engines, enterprise search engines focus on indexing private databases, intranets, and other internal data sources, ensuring that sensitive information remains secure and accessible only to authorized users (Hawking, 2004).

2.2.8.6. Hybrid Search Engine

Hybrid or federated search engines integrate features from both traditional and innovative search technologies to enhance search capabilities and performance. By blending automated indexing with curated content or user feedback, hybrid search engines can effectively address the diverse needs of users. For example, a hybrid engine might incorporate both general web search and deeper, topic-specific searching capabilities (Ding et al., 2012).

2.3. Related Work

In this section, the infrastructure and its single components of the OpenWebSearch.eu project are explored. In addition, existing modular web search engine frameworks are discussed and compared.

[5]https://duckduckgo.com
[6]https://www.startpage.com
[7]https://search.brave.com

Search engine	Years active	Alexa rank	Country	Ind.	Scale	User data	Funding	Transparency
MetaGer	1996–today	64,210	DE	No	Uses Bing + Scopia	None	Ads, donations	Open source
Google	1997–today	1	USA	Yes	Own datacenters	Own traffic	Ads	Closed
Yandex	1997–today	62	RU	Yes	Own datacenters	Own traffic	Ads	Closed
Startpage.com	1998–today	1895	NL	No	Uses Google	None	Ads	Closed
Naver	1999–today		KR	Yes	Own datacenters	Own traffic	Ads	Closed
Baidu	2000–today	5	CN	Yes	Own datacenters	Own traffic	Ads	Closed
Gigablast	2002–today	19,819	USA	Yes	Own datacenters	None	B2B, donations	Open source
YaCy	2003–today			Yes	Decentralized	None	Donations	Open source
Exalead	2004–today	47,873	FR	Yes	Own datacenters	Own traffic	B2B	Closed
Mojeek	2004–today	414,308	UK	Yes	Own datacenters	None	B2B	Closed
Wikia Search	2007–2009		USA	Yes	Community-moderated	User contribution	Ads	Open source
DuckDuckGo	2008–today	182	USA	Hybrid	Uses Yahoo, Bing	None	Ads	Open source
Bing	2009–today	38	USA	Yes	Own datacenters	Own traffic	Ads	Closed
Ecosia	2009–today	471	DE	No	Uses Bing	None	Ads	Closed
Qwant	2013–today	7408	FR	Hybrid	Uses Bing + own index	None	Ads	Closed
Cliqz	2015–2020	52,948	DE	Yes	Own index	Human web	Ads	Mostly closed
Brave Search	2021–today							
Neeva	2021–today							
You.com	2021–today		USA	No	Uses Bing	Hybrid	Venture capital	Closed

Figure 2.4.: Comparison of established web search engines (Granitzer et al., 2024)

2.3.1. OpenWebSearch.eu

As the internet continues to grow, both in size and in importance to daily life, the role of web search engines becomes increasingly critical. Therefore, a large number of established web search engines exist (see Figure 2.4). Most of them operate with indexes that are not public or transparent, concealing the mechanisms and data that drive their search results. Furthermore, many web search engines do not maintain their own proprietary indexes but instead access indexes provided by third-party providers. This approach can impact the autonomy and customization of the search services they offer, as they rely on external sources for key components of their search process (Granitzer et al., 2024).

This dependency raises questions about the control and integrity of search results, underscoring the need for more open and transparent indexing practices in the field of web search. The OpenWebSearch.eu project aims to address these issues by providing a more accountable and understandable search ecosystem with the OWI as central component.

2.3.1.1. Open Web Index

In Figure 2.5, Granitzer et al. (2024) describe the creation pipeline of the OWI within the Open Web Search and Analysis Infrastructure (OWSAI) by the OpenWebSearch.eu project. It involves a structured pipeline designed to foster

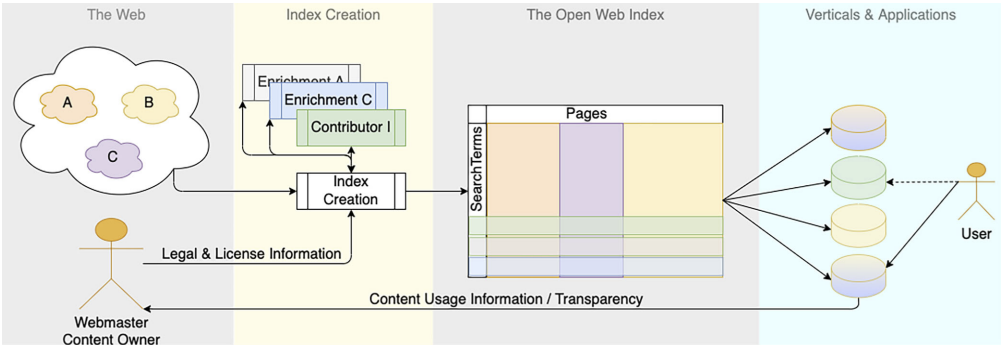


Figure 2.5.: Fundamental concept of the Open Web Index, serving as the foundation for an open, scalable, transparent, and legally compliant web search ecosystem. Different colors represent various segments of the web and the contributions from multiple third parties in constructing a web index. (Granitzer et al., 2024)

an open and collaborative web search ecosystem. This pipeline emphasizes principles such as open data access, adherence to legal standards, and joint technology development, ensuring the index is both transparent and scalable. Central to this is the construction of the OWI through coordinated web crawling, advanced data processing, and indexing across decentralized computing centers. This approach not only enhances the accessibility and utility of the index but also supports the creation of various specialized search engines as well as innovative web data products, such as language models and knowledge graphs (Granitzer et al., 2024).

2.3.1.2. Web Documents Crawling

The OpenWebSearch.eu project employs an innovative web crawling approach to build its OWI. Dinzinger et al. (2024) highlight that this crawling process is designed to be collaborative and distributed across multiple computing sites in Europe, enhancing both the scale and the efficiency of data collection. A specialized web crawler called OWLer, developed specifically for this project, is central to this operation. It is based on the StormCrawler (Nioche, 2019) framework but has been extensively modified to meet the specific needs of the OpenWebSearch.eu project. For instance, the modifications include the integration of advanced topic classification functionalities to enable more targeted crawling. The OWLer operates by systematically downloading web pages, which are then processed and stored in Web ARChive (WARC) file format. This approach ensures that they are preserved with comprehensive metadata. This crawler is distinctive because it employs a decentralized architecture, which allows multiple crawler nodes to work simultaneously in different locations, coordinated through a central component known as the URL Frontier. With this setup, the OWLer can efficiently manage and distribute URLs based on thematic or geographic criteria, thus optimizing the crawling process for relevance and

efficiency (Dinzinger et al., 2024).

The URL Frontier service acts as a central repository that maintains known URLs and assigns new ones to the appropriate crawler nodes based on their specialization. This system ensures that each part of the crawler network can focus on its specific area of interest, whether that be a particular topic or region, thus maximizing the relevance of the collected data. The flexibility and scalability of this system are crucial to its ability to handle the large-scale and unstructured nature of web content (Dinzinger et al., 2024).

Furthermore, the OWLer's ability to classify content at the point of crawling allows for the immediate categorization of data, which significantly enhances the efficiency of the indexing process. By integrating a machine learning model directly into the crawler, the OWLer can make intelligent decisions about the relevance of content in real-time. In doing so, the overhead associated with processing and storing irrelevant data can be reduced (Dinzinger et al., 2024).

The crawling and archiving as WARC files happens on a daily basis, which enables the provision of fresh and incremental index updates (Hendriksen et al., 2024).

The OpenWebSearch.eu infrastructure introduces *crawling on demand*, a service created to reduce the substantial costs and resources usually associated with web crawling. This feature allows authorized users to initiate a customized crawling process by submitting a selected list of initial URLs. Subsequently, specifically configured for this task, a containerized crawl cluster is triggered. The system checks if a URL has already been crawled to avoid redundancy, thus optimizing resource use. Users are ensured access to the most current and relevant data through daily updates of WARC files. Moreover, an *index on demand* feature can clean, enrich, and convert the crawled data into index files ready for immediate use in web search engines (Hendriksen et al., 2024).

Overall, Dinzinger et al. (2024) and Hendriksen et al. (2024) outline that the crawling strategy employed by the OpenWebSearch.eu project represents a significant advancement in web crawling technology, with its distributed and customizable approach building an open and accessible web index.

2.3.1.3. Data Preprocessing

The preprocessing step in the OWI generation pipeline is essential for preparing data for indexing and further analysis and use. After daily web crawling activities, WARC files generated are processed to extract web page-level metadata and cleaned text. This preprocessing occurs in daily batch jobs across various data centers, utilizing Apache Spark[8] on high-performance computing (HPC) clusters set up via the Magpie[9] script collection. The extracted data, including plain text and metadata from HTML, WARC, and Hypertext Transfer Protocol

[8]https://spark.apache.org
[9]https://github.com/LLNL/magpie

(HTTP) headers, is then stored in Apache Parquet[10] file format (Hendriksen et al., 2024).

Metadata such as the language of the document and domain labels, which are based on Curlie[11] community data, are extracted to facilitate the partitioning of index files. This allows for the efficient organization and retrieval of data tailored to specific linguistic or domain-based queries (Granitzer et al., 2024; Hendriksen et al., 2024).

The preprocessing pipeline's modular architecture supports the integration of third-party content analysis modules. This enables the expansion of metadata extraction capabilities. Using the TIRA Integrated Research Architecture (TIRA) developed by Gollub et al. (2012) and Potthast et al. (2019), these modules are evaluated, which ensures quality and reproducibility by running software in a controlled environment. This evaluation process leverages task-specific benchmark data which encourages the development of strong content analysis tools that comply with stringent research standards. As part of continuous improvement, efforts are made to increase the diversity and number of benchmark datasets and modules, thereby advancing the quality and comprehensiveness of the OWI (Hendriksen et al., 2024).

2.3.1.4. Indexing and Representation

The indexing task of the OWI process begins with the cleaned text obtained from the preprocessing pipeline, which is then transformed into a comprehensive full-text index. Using this full-text index, it is partitioned into distinct index shards based on various metadata values such as top-level range, language, and specific topics from Curlie. Each shard, stored as a separate Common Index File Format (CIFF) index, can be downloaded alongside Apache Parquet files that include the pertinent metadata and cleaned text. With this, custom search applications for public, commercial, or personal needs can be created. Following the completion of preprocessing for the day's content, the indexing operation is conducted daily as an Apache Spark batch job, with Magpie facilitating the setup of the Apache Spark cluster within a HPC environment. This systematic approach ensures the index is continuously updated and particularly accessible for creating vertical search applications (Hendriksen et al., 2024).

Within the OWI, CIFF serves as a standardized method for search engines to share and access index structures. Lin et al. (2020) introduced CIFF to enhance interoperability between different search engines by allowing them to export their inverted indexes into a unified format, which can then be imported and utilized by other systems. This standardization is achieved through the use of *Protocol Buffers*, which offer a platform-agnostic, language-neutral approach to serializing structured data. Therefore, the index data remains consistent

[10]https://parquet.apache.org
[11]https://curlie.org

and accessible across different platforms and search engine implementations. The use of CIFF allows for the efficient exchange of index data, subsequently facilitating more accurate and fair comparisons between different search systems (Lin et al., 2020).

One possibility to consume CIFF files is provided by the application *lucene-ciff*[12], which can import a CIFF index in Apache Lucene[13]. Lucene is an open-source search library widely used for full-text indexing and search capabilities. It provides comprehensive features for text analysis, various query mechanisms, and sophisticated indexing technologies. It supports efficient high-volume text retrieval and enables complex search functionalities crucial for modern search engines (Białecki et al., 2012).

Since the CIFF files only store the inverted index but no metadata, Apache Parquet files are used that are associated to a specific index partition. The columnar storage of the open-source format Parquet is ideal for big data processing frameworks. It is particularly well-suited for complex data processing tasks involving large-scale querying and analysis because it enables efficient data compression and encoding schemes. Parquet files provide excellent support for read-heavy operations, allowing for highly efficient data retrieval by storing data in a way that allows for selective reading of specific columns without needing to process entire rows. This file format is widely used in data analysis ecosystems due to its compatibility with different data processing tools, for instance, Apache Hadoop[14], Apache Spark, and data analysis libraries in Python (Vohra & Vohra, 2016).

One effective data processing tool that seamlessly integrates with Apache Parquet for enhanced data analysis is DuckDB. It is an embeddable analytical database that is specifically designed for efficient analytical SQL query execution within host processes. DuckDB is particularly beneficial for interactive data analysis and edge computing scenarios where local data processing is essential. Unlike traditional client-server database systems, DuckDB operates directly within the application's process, avoiding overhead and increasing data processing speed (Raasveldt & Mühleisen, 2019).

2.3.1.5. OWSE-Hub

The Open Web Search Engine Hub (OWSE-Hub), conceptualized as part of the OpenWebSearch.eu project, is intended to function similarly to the Docker Hub as a distributed information system, but tailored specifically for search engines. It aims to facilitate the easy creation and customization of search engines by providing a platform where users can access prebuilt indexes and search engine stacks. Similar to pulling container images from Docker Hub, users can *pull*

[12]https://github.com/informagi/lucene-ciff
[13]https://lucene.apache.org
[14]https://hadoop.apache.org

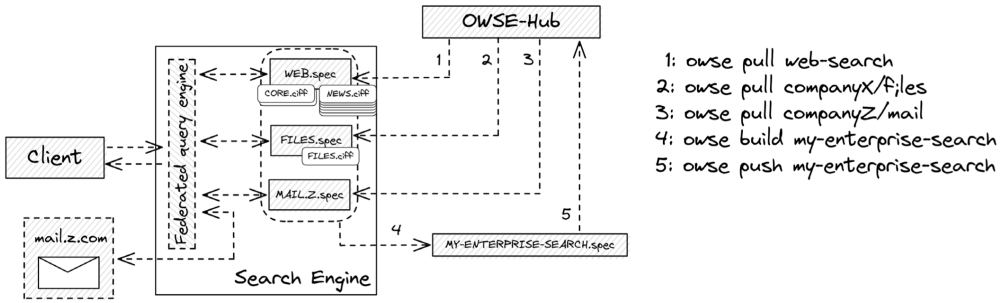


Figure 2.6.: Overview of the OWSE-Hub architecture allowing specifications for declarative search engines (Granitzer et al., 2024)

search engine configurations from the OWSE-Hub. Additionally, they can *build* their own customized search engines using these configurations, as well as then *push* their custom solutions back to the hub so that the configurations are accessible by others, as illustrated in Figure 2.6 (Hendriksen et al., 2024).

This system supports the creation of diverse search applications, ranging from personal to commercial use, and accommodates both centralized and federated search setups. By emulating the functionality of Docker Hub within the context of search engine development, the OWSE-Hub simplifies the process of search engine customization and deployment. Therefore, this approach makes it easier for developers and organizations to utilize specific segments of the OWI for various applications (Granitzer et al., 2024; Hendriksen et al., 2024).

2.3.2. IR and Web Search Engine Frameworks

Existing frameworks for IR and web search engines are available that offer interactive, configurable environments for specific needs. These frameworks allow users to engage directly with the system, customizing and extending functionalities through modular components to optimize search processes and outcomes. Therefore, this subsection explores and compares selected systems and frameworks that are central to the field of modular or customizable IR and web search technologies.

2.3.2.1. Harvest

Harvest is an distributed information discovery and retrieval system developed in the 1990s designed to manage the increasing volume and diversity of internet data. It features customizable tools for gathering, replicating, and searching documents, which significantly reduce server load and network traffic. Additionally, Harvest's architecture supports the use of partial indexes through its flexible and modular components, called *Gatherers* and *Brokers*. These partial indexes can then be managed and queried by *Brokers*, which can aggregate information from multiple *Gatherers* and other *Brokers* (Bowman et al., 1995).

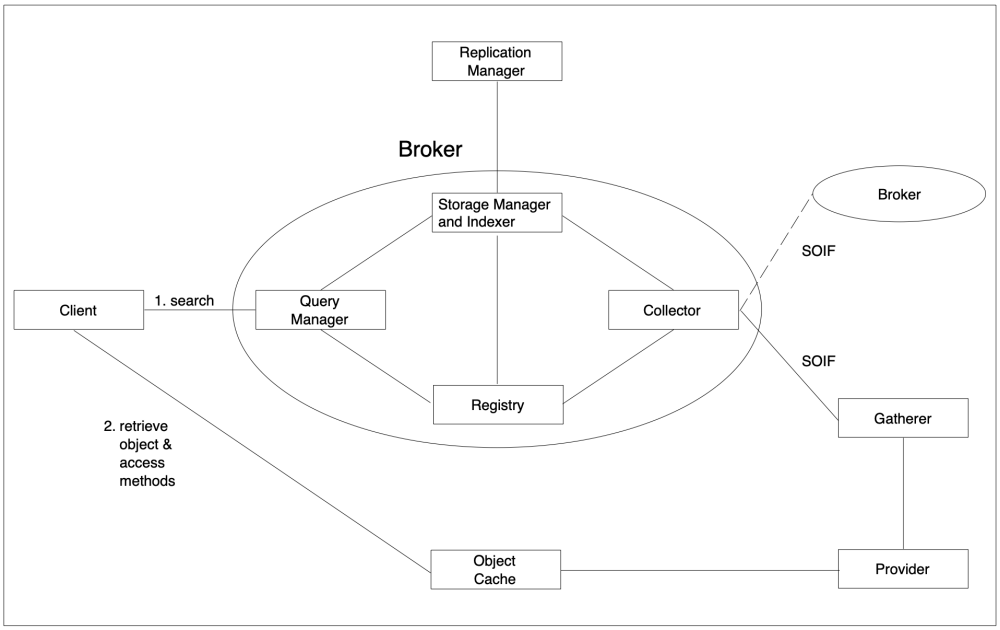


Figure 2.7.: Decentralized architecture of Harvest for efficient indexing and data access (Bowman et al., 1995)

As depicted in Figure 2.7, the architecture of the Harvest system is designed to decentralize access to crawled data and facilitate common indexing through a modular and distributed approach. The system includes *Gatherers*, which collect and update indexing information periodically from various sources. The information is then processed and indexed by *Brokers*, which provide an indexed query interface and support incremental updates. The use of the Summary Object Interchange Format (SOIF) allows efficient sharing and integration of indexing information across different nodes. This architecture enables the distributed management of data and ensures that the indexed data is accessible and efficiently searchable across the network.

2.3.2.2. Apache Solr

Apache Solr[15], built on Apache Lucene, is a powerful open-source search platform that provides flexible and reliable search functionality. Offering a range of features including full-text search, faceted search, hit highlighting, dynamic clustering, and advanced document handling, it is designed to handle diverse search applications across websites. Solr supports complex configuration and customization, enabling it to meet a wide range of enterprise search requirements. Its architecture ensures high scalability and supports both distributed search and replicating indexes (Shahi, 2016).

In particular, Solr is capable of indexing a variety of data types, including

[15]https://solr.apache.org

HTML content from web pages. The file `schema.xml` defines the schema of the documents that Solr will handle. It specifies the fields and field types for the documents, so Solr knows how to index and process incoming data. This schema configuration allows for customization of how fields are indexed, such as setting fields to be searchable, stored, tokenized, or used for sorting. Essentially, `schema.xml` serves as a blueprint for how Solr interacts with the data (Shahi, 2016).

2.3.2.3. Terrier

The Terrier IR platform, developed at the University of Glasgow, is an IR software specifically designed for research and educational purposes. It is based on the programming language Java and offers a comprehensive framework for quickly developing large-scale retrieval applications. Terrier is highly modular, which allows researchers to experiment with different components and algorithms. This makes it particularly valuable for academic research in IR. Moreover, Terrier's architecture supports a variety of weighting models and provides tools for performance evaluation. Additionally, PyTerrier[16] is an open-source Python IR platform that is based on Terrier and provides the declarative formulation of search engines (Ounis et al., 2005).

Terrier is capable of indexing web pages and retrieving information from them. It includes functionalities for web crawling and processing HTML content. With this, the platform can gather and index data from web pages effectively. This makes Terrier suitable for web search applications, where it can be used to build and manage searchable indexes of web content.

2.3.2.4. Infret

Focusing on the educational perspective, Infret is an interactive e-learning platform specifically designed for teaching IR concepts through hands-on coding exercises. Bobić et al. (2020) highlight that this tool allows students to engage directly with IR theories by programming solutions, which are then evaluated within the system. Infret's architecture integrates a user-friendly interface that manages the complexities of IR tasks while providing an instrumental learning environment. The platform's design effectively bridges the gap between theoretical knowledge and practical application and supports students in computer science and information technology courses (Bobić et al., 2021).

2.3.2.5. Elasticsearch

Elasticsearch, built on Apache Lucene, is a prominent search and analytics engine noted for its rapid indexing and retrieval capabilities which are essential

[16]https://github.com/terrier-org/pyterrier

for handling large-scale web data. It supports the creation of custom indices, making it highly suitable for indexing web pages and facilitating complex search functionalities across diverse digital content. A notable advantage of Elasticsearch is its ability to perform real-time analysis and scalable searching. However, challenges such as managing large data volumes and ensuring consistency during data indexing require robust system configuration and maintenance to leverage its full potential effectively (Kononenko et al., 2014).

2.3.2.6. OpenSearch

Established after Elastic moved Elasticsearch from an Apache 2.0 license to a Server Side Public License (SSPL), OpenSearch is forked from Elasticsearch and is an open-source search and analytics suite. OpenSearch includes capabilities for full-text search, distributed search, and analytics, similar to Elasticsearch. It allows users to maintain compatibility with existing Elasticsearch application programming interfaces (APIs) and manage large volumes of data effectively. OpenSearch aims to differentiate by fostering a broader open-source community and ensuring long-term flexibility and openness in its licensing (Papadopoulos et al., 2024).

2.3.2.7. Sphinx

Sphinx, an open-source full-text search server, offers fast, scalable, and relevant search functionalities across various platforms. It supports indexing of custom data sources, which includes web pages, and allows fine-grained control over indexing and querying processes. Sphinx is particularly noted for its high-performance capabilities in handling large datasets and complex queries efficiently. However, it demands considerable technical expertise for configuration and optimization, and it might offer fewer ready-to-use web-specific features compared to other search engines (Ali, 2011).

2.3.2.8. Indri

Designed for efficient large-scale IR, Indri is a search engine that integrates language modeling and inference networks. This system facilitates complex query constructions that integrate various types of evidence from document attributes such as text, structure, and metadata. Additionally, Indri's modular architecture accommodates various data types and retrieval strategies, which enhances its adaptability and effectiveness across diverse information retrieval applications (Strohman et al., 2005).

2.3.2.9. Google Programmable Search Engine

Particularly relevant to vertical search, Google's Programmable Search Engine[17] allows users to create a custom search engine tailored specifically to their websites or specified pages. This tool enables the search through web pages from own selected domains and a direct integration into personal or organizational websites. A key advantage is the ability to harness Google's powerful search algorithms, providing accurate and fast search results with customization options such as prioritizing or excluding specific content. Furthermore, the tool offers an API known as the Custom Search JavaScript Object Notation (JSON) API. With this API, developers can programmatically request search results in JSON format by utilizing the same capabilities as the Programmable Search Engine interface. However, the scope of indexing is limited to the domains specified by the user, and it relies on Google's infrastructure, which may limit some aspects of search customization and data privacy (Shamaeva & Galley, 2021).

2.3.2.10. Spinque

Spinque[18] is a versatile proprietary search engine design tool that allows users to tailor search solutions specifically to their needs. By integrating data from various sources into a unified knowledge graph, this tool enables users to create customized search functionalities using its visual editor. This platform supports the deployment of these search solutions via a web API, so that the solution is easily and broadly accessible. Spinque is particularly useful for professionals and organizations aiming to optimize their search processes with precision and efficiency.

2.3.2.11. Analysis and Discussion

In comparing diverse search engine frameworks, builder and tools, each has unique features tailored to specific needs. Apache Solr offers scalability and robust data handling, ideal for enterprise applications, while Terrier provides a flexible platform for academic research with its modular design. Elasticsearch and its fork OpenSearch excel in rapid indexing and real-time analytics, with OpenSearch further focusing on community-driven development. Google's Programmable Search Engine allows for customized search within specific domains but may restrict customization due to reliance on Google's infrastructure. Sphinx, although powerful for complex queries, requires substantial technical expertise. Focusing on IR methods, Indri combines language modeling with inference networks for complex queries. Similar to Google's Programmable

[17]https://programmablesearchengine.google.com
[18]https://spinque.com

Search Engine, Spinque offers visual configuration to tailor search solutions, whereas Spinque is also able to integrate varied data sources into a knowledge graph. Table 2.1 compares the findings of the exploration of these tools by highlighting their features related to configuration, customization and modularization, as well as their key advantages, key disadvantages and the required technical expertise.

Tool	**Features**	**Advantages, Concepts & Ideas**	**Disadvantages & Problems**	**Technical Expertise Required**
Harvest	Distributed retrieval, customizable indexing	Flexible configuration, modular components	Complex configuration, no visual interface	Moderate
Apache Solr	High customization with schema configuration, modular plugins	Visual editor for index management	Requires rather complex configuration	Moderate
Terrier	Highly modular, supports various IR models and components	Modular architecture, research-oriented	Complex configuration, limited visual tools	High
Infret	Moderately customizable for educational settings	Visually facilitates practical and learning	Limited to educational use	Low
Elasticsearch	API-driven customization, plugins	Easy setup of fields mapping	Requires solid system maintenance	Moderate
OpenSearch	Distributed search, toolkit for components	Open-source community, flexible licensing	Complexity of managing architecture	Moderate
Sphinx	Fine-grained control over indexing and querying	Faceted search, real-time indexing	Complex configuration, limited scalability	High
Indri	Modular architecture, supports complex query constructs	Supports complex queries, adaptable	Complex configuration and customization	Moderate
Google Programmable Search Engine	Customization to specified domains, offers API for JSON results	Customizable search experience, enhanced precision	Relies on Google's infrastructure	Low
Spinque	Highly customizable with visual editor, integrates various data	Highly customizable search processes	Proprietary tool	Moderate

Table 2.1.: Comparison of selected IR and web search frameworks

2.4. Summary

This chapter introduced important concepts related to this thesis and offered a detailed background. It began with an exploration of IR and an explanation of associated essential processes such as indexing, query processing, and ranking. Additionally, various IR models, including the Boolean model, Vector Space model, and Probabilistic model, were discussed to illustrate their roles in enhancing the efficiency and effectiveness of IR systems. Understanding these foundational concepts is crucial as they underpin the development of more advanced search technologies.

Transitioning to web search engines, the chapter traced the history and evolution of these technologies, from early web crawlers to sophisticated algorithms such as Google's PageRank. Fundamental components such as crawling, indexing, and querying were examined, along with modern advancements such as AI integration to improve query understanding and result accuracy. The ongoing development of search engine technologies was emphasized, particularly highlighting the increasing ability to process complex queries and large datasets by demonstrating the increasing sophistication of these systems.

The chapter then delved into the technical details of the OpenWebSearch.eu project, highlighting its aim to create a more open, transparent, and collaborative web search ecosystem. This initiative involves the creation of the OWI through coordinated web crawling, data processing, and decentralized indexing. By providing prebuilt indexes and search engine stacks, the project seeks to address transparency issues in current web search engines, with the goal to provide greater accessibility and fairness in the digital information landscape. The project's architecture and potential impact on web data indexing and search were discussed, emphasizing its innovative approach.

Finally, the chapter explored and compared existing modular web search engine frameworks, including Apache Solr, Terrier, Elasticsearch, and others. These tools offer varying levels of customization and modularization, which enable users to tailor search functionalities towards vertical search engines to specific needs. In particular, Terrier, Google Programmable Search Engine and Spinque were highlighted for their flexibility and possibility for configuration and modularization. This comparison (see Table 2.1) summarizes the understanding of the strengths, limitations and room for improvements of these tools.

In conclusion, this chapter provided a thorough background on the fundamental concepts and advancements in IR and web search technologies. It explored the evolution and increasing sophistication of search engines, discussed the technical details of the OpenWebSearch.eu project, and compared various modular search engine platforms. This comprehensive overview establishes a solid foundation for understanding the development and selection of search technologies for diverse applications.

3. Requirements and Design

This chapter transitions from the theoretical foundations of IR systems and web search engines, and the technical background of the OpenWebSearch.eu project as well as a comparative analysis of existing search technologies to the practical application of these insights in developing a new system. It starts by outlining the motivation for the research and the approach taken to integrate the identified requirements into the design process in Section 3.1. The chapter then outlines the specific requirements for the system in Section 3.2, followed by a detailed conceptual architecture that highlights the abstract design decisions for each component as described in Section 3.3. Finally, it addresses the rationale behind these decisions in Section 3.4, as well as the limitations relevant to developing the system detailed in Section 3.5.

3.1. Motivation

As outlined by Granitzer et al. (2024), given the immense resources required for crawling and indexing the entire web, only few entities besides Google, Microsoft's Bing[1], and long-established regional search engines such as Baidu[2] and Yandex[3] maintain their own index infrastructure and ranking algorithms. The indexes of these big players can be accessed, but the process behind their creation often lacks transparency and offers no control over their formation. Additionally, reliance on these indexes means bearing associated access costs and having limited insights into the underlying data and ranking methodologies. The goal of this thesis is to support the development of web search applications based on OWI partitions by introducing the MOSAIC.

Nussbaumer et al. (2023) showed through a prototype application that indexes generated by the OpenWebSearch.eu index generation pipeline can be used for searching and retrieving information. However, this application stored all index data in memory, thus problems occurred as soon as the index partitions became too large. Therefore, the primary motivation is to demonstrate the effective utilization of OWI partitions in search engines as they provide a technological foundation for custom applications while maintaining a modular and configurable framework.

[1]https://www.bing.com
[2]https://www.baidu.com
[3]https://yandex.com

Bevendorff et al. (2024) found that search engines such as Google, Bing, and DuckDuckGo are significantly impacted by Search Engine Optimization (SEO) spam, especially in product review queries, with persistent low-quality, affiliate-marketed content despite periodic algorithm updates to combat it. Therefore, developing independent web search engines becomes essential to ensure greater control over indexing and improve the relevancy and effectiveness of search results. In addition, the integration of index partitions and the creation of a search engine like this should be designed to require minimal technical expertise while still offering extensive customization and configuration options. Essentially, the framework presented in this thesis should provide the possibility to support developers creating a configured and personalized web search engine utilizing the OWI, while preserving independence and control over the entire process. Moreover, the system should incorporate important key features of existing solutions while addressing and bridging the gaps and problems present in these tools.

3.2. Analysis of Requirements

The system's requirements of MOSAIC were classified as either functional or non-functional requirements based on the distinctions made by Sommerville (2011). While functional requirements specify the core tasks and services the system must provide, non-functional requirements describe the overall properties and constraints of these tasks. The requirements were identified, analyzed, and derived from a review of the background literature and the related work to ensure they align with established theories and practices.

3.2.1. Functional Requirements

1. **Search Functionality:** The system is required to offer comprehensive search functionality that enables users to perform search queries through API endpoints. Users can perform searches by entering keywords or phrases, which the system then processes. The system should deliver search results in both JSON and Extensible Markup Language (XML) formats which is essential for meeting diverse user needs and providing relevant data in the preferred format. In addition, the system must offer the option to paginate search results.
2. **Faceted Search:** The system must include the functionality of a faceted search so that users can refine and filter their search results based on certain attributes. Additionally, the filtering capabilities should be managed via customizable modules.
3. **Index Management:** The system must support OWI data created by the index generation pipeline. In addition, the support of index partitions

created by the local indexing pipeline[4] is required. In particular, the system should support the import of an arbitrary number of index partitions. This includes the import of CIFF files into Apache Lucene indexes together with the associated metadata as Apache Parquet files. Additionally, it is required that the search can be performed in either one or multiple index partitions simultaneously. Updating indexes should be achievable by replacing outdated CIFF files and their corresponding Parquet files with updated versions.

4. **Metadata Enrichment:** Metadata enrichment capabilities are essential for enhancing the system's search results. While CIFF files store the terms and their occurrences in web documents for matching and initial ranking, metadata enriches this information by adding contextual details such as the title of the web page, URL, full text, and more. This enrichment allows users to see more comprehensive information when viewing search results to improve the relevance and usability of the retrieved data.
5. **Module Management:** The system should support seamless module management, which allows for the easy incorporation of various metadata fields into search results. Specifying fields for both display and filtering should be straightforward to enable customization of search results based on different metadata. This also includes user-defined fields added at a later date that are not part of the fixed metadata schema[5]. Furthermore, integrating additional components, such as custom search query analyzers, should be uncomplicated to ensure the system remains flexible and adaptable to diverse user needs.
6. **Snippet Creation:** The system must support the creation of text snippets to provide concise and relevant portions of documents in search results. These snippets should be generated dynamically based on the search query to highlight the relevant information. Considering performance, the user should be able to specify whether the full plain text of a web document or only a part of it is used for the creation of the snippet.
7. **Full-Text Retrieval:** Functionality for accessing the full plain text of a document within the index on demand is essential to allow that the complete text information can be retrieved when needed.
8. **Setup and Execution:** The setup and execution of the system must be streamlined. It has to incorporate steps for compilation, index management, and other necessary configurations. To facilitate system execution, a command line interface (CLI) should be provided with various options, enabling users to specify parameters such as index paths, configuration files, and server ports. This ensures that a customized search engine can be

[4]https://openwebsearcheu-public.pages.it4i.eu/ows-the-book/content/howto/local_index.html

[5]https://opencode.it4i.eu/openwebsearcheu-public/preprocessing-pipeline#fixed-columns

deployed and operated efficiently while minimizing the technical expertise required for initial setup and ongoing management and maintenance.

3.2.2. Non-Functional Requirements

1. **Performance:** The system is required to maintain high performance, capable of efficiently handling large volumes of data. Optimizing query processing and result retrieval is essential to deliver rapid search results and minimize latency as it must be able to operate the system on a computer or server without requiring high-end hardware.
2. **Usability:** Clear and comprehensive documentation should be available to guide users through setup, configuration, and operation processes. Additionally, the system should offer straightforward configuration options and customizable settings to accommodate diverse user needs and preferences.
3. **Maintainability:** The system must be designed to allow for easy updates and modifications. Its modular architecture along comprehensive documentation should facilitate the addition or removal of components without disrupting the overall system.
4. **Reliability:** It is essential that the system implements solid error handling and recovery to minimize downtime and potential data loss.
5. **Compatibility:** The system must ensure broad compatibility and interoperability with various data formats and external systems, including the OpenWebSearch.eu index generation pipeline, the OpenSearch[6] protocol and other external services. Additionally, the system must support the specification that each index partition of the OWI consists of a CIFF file and Parquet files, with the Parquet files adhering to a specific column format.
6. **API Integration:** The system must include a representational state transfer (REST) API that is accessible by any external service through HTTP requests, thereby ensuring broad interoperability and integration possibilities. Additionally, the API must support the OpenSearch protocol in XML format to facilitate standardized search and aggregation across various platforms. The API must also provide essential information about the indexes available within the system, as well as information about the system configuration and errors that have occurred.

3.3. Conceptual Architecture Design

Building upon the identified both functional and non-functional requirements, a conceptual architecture of MOSAIC was designed. Figure 3.1 illustrates

[6]https://github.com/dewitt/opensearch

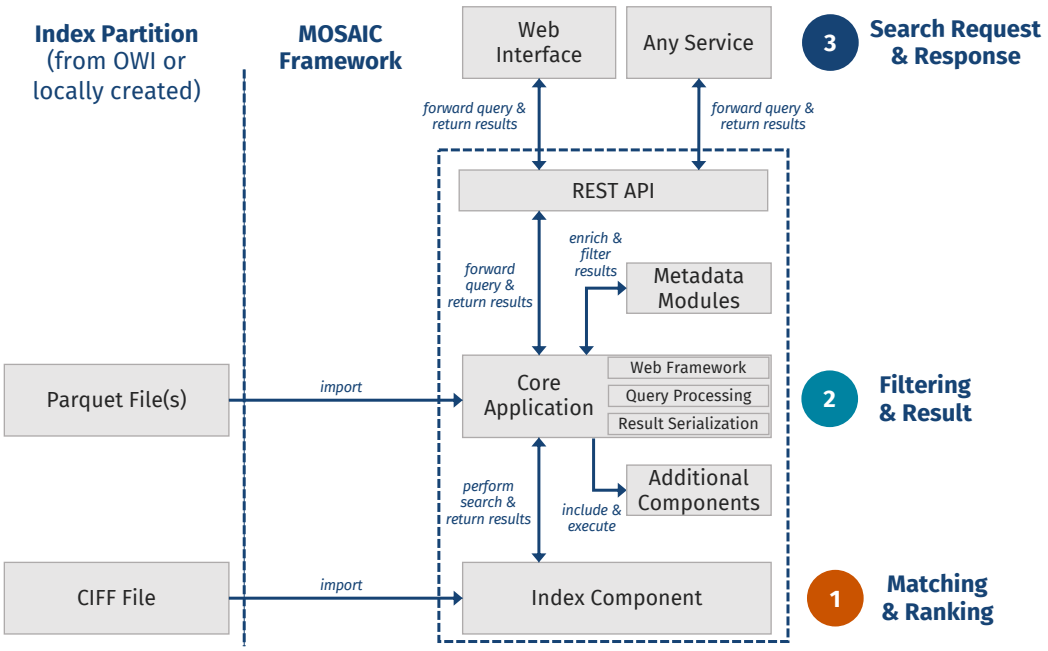


Figure 3.1.: Conceptual architecture of the MOSAIC framework, proposing the integration of index partitions, core application components, metadata modules, REST API, and web interface for efficient search and retrieval operations.

the main components and how they are related to each other. This design allows to use MOSAIC depending on individual needs: it can be used out-of-the-box including a web interface, configured as a service with a REST API for integration into a search application, extended with custom metadata modules, and its source code can be modified before utilizing the search engine. Furthermore, this conceptual architecture lays the basis for the system's development.

3.3.1. Index Partitions

OWI partitions encompass CIFF files and Parquet files which are crucial components of the MOSAIC framework, and each serve different but complementary purposes. CIFF files store the occurrence of terms and document identifiers and provide a standardized format for efficient indexing and retrieval. Parquet files, utilizing columnar storage, contain additional metadata such as titles, URLs, and language to improve retrieval performance and storage efficiency. The high-level schema of an OWI partition, consisting of a CIFF file and one or more Parquet files, is depicted in Figure 3.2. Collectively, these files ensure that search results are both comprehensive and contextually enriched, thereby optimizing data management and improving the usability of the MOSAIC system.

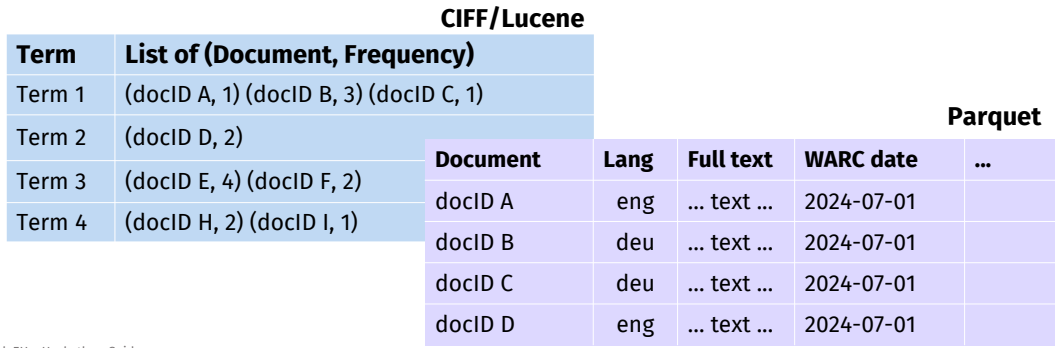


Figure 3.2.: Schema of an OWI index partition, illustrating the integration of CIFF and Lucene for inverted indexing and Parquet for tabular metadata storage, as created by the OpenWebSearch.eu index generation pipeline

3.3.2. Indexing Component

To offer robust and scalable search functions, an index component acts as the central search engine within the MOSAIC framework. It efficiently handles large volumes of data, enables fast query processing, and ensures powerful and sophisticated search processes. By utilizing advanced indexing and search features, the MOSAIC framework delivers precise and accurate search results. Essentially, the integration of this index component provides a reliable and effective foundation for the entire system.

3.3.3. Core Application

The core application in the MOSAIC framework centralizes and orchestrates the interaction between various components and manages the overall search process. Integrating CIFF and Parquet files, it constructs and maintains search indexes to ensure that data remains up-to-date and accessible. The core application is responsible for executing search queries, performing tasks such as matching, ranking, and filtering to deliver relevant results. Additionally, it supports the modular addition of metadata and other functionalities, thereby enhancing the system's flexibility and adaptability.

A sub-component of the core application is the query processing, which involves a series of essential steps to ensure accurate and efficient search results. Initially, the user's query is parsed to identify its structure and components by breaking it down into manageable elements. Subsequently, the query is used to perform a search utilizing the index component.

The core application also coordinates the interaction between various modules and components within the MOSAIC framework. It ensures seamless communication and coordination among different parts of the system, such as the index component, metadata modules, and query processors. By managing these interactions, the core application maintains a cohesive and efficient

operation, thereby allowing the system to function as a unified whole.

In addition, an integral sub-component of the core application is a web server framework to facilitate the efficient handling of web requests and responses. This framework supplies the essential infrastructure for managing client-server interactions and enables the system to process search queries and deliver results promptly and reliably.

Eventually, the core application is also responsible for preparing the search results to be exposed via the REST API. This involves serialization by formatting the results appropriately, incorporating necessary metadata, and ensuring that the data is structured according to the API specifications.

3.3.4. Modules and Components

Metadata modules in the MOSAIC framework enrich search results with associated metadata values and specify filters. These loosely coupled modules process metadata from Parquet files, integrating it into the filtering process and search results to provide users with comprehensive details about each document, including titles, URLs, and language. Additional components, comprising custom query analyzers and specialized query operations such as query expansion and query rephrasing, further enhance the system's functionality. For instance, this design allows users to include custom query analyzers to optimize query processing and improve result accuracy.

3.3.5. REST API

The REST API in the MOSAIC framework offers a flexible and versatile interface for interacting with the search engine. Most importantly, this allows external services to execute search queries and retrieve results. Supporting standard protocols and query formats, it ensures broad compatibility, interoperability, seamless communication, and easy integration with other applications and services. The API also provides endpoints for accessing essential information about available indexes and the system's configuration information.

3.3.6. Web Interface

The web interface of the MOSAIC framework provides a basic yet functional platform for demonstrating search capabilities and presenting search results. It is designed to be easy and straightforward, thereby allowing users to perform search queries and view the results without complexity. This interface showcases the core functionalities of the search engine and ensures that even users with minimal technical expertise can navigate and utilize the system effectively. Additionally, the system can be used with another service and even without a

web interface at all, by directly accessing the search service via the provided API.

3.4. Design Decisions

Following the design of the conceptual architecture, several decisions were made regarding the selection of specific approaches as well as technologies.

Although there exist importing tools of CIFF files to JASSv2[7], Pisa[8], OldDog[9] and Terrier, Lucene was chosen as the core search engine for the MOSAIC framework due to its reliable and scalable indexing and search capabilities, as well as its independence of the existing IR solutions mentioned above. CIFF files can be imported to Lucene indexes using the standalone application *lucence-ciff* via the Anserini toolkit[10]. Lucene's comprehensive documentation, active community support, and proven success in high-performance search applications contributed to its selection as a reliable choice. Even though a Python wrapper around Java Lucene named PyLucene[11] exists, PyLucene was not selected due to its relatively cumbersome installation process compared to Lucene, as well as the less active community support for PyLucene. These factors could lead to potential performance overhead and compatibility issues in future.

Influenced by the choice to use Lucene, which is inherently Java-based, Java[12] was selected as the primary programming language for the MOSAIC framework. Furthermore, Java features important aspects such as robustness, platform independence, and an extensive ecosystem of libraries and tools, which support the development of scalable and high-performance applications. Java's strong community support and well-documented best practices further enhance reliability and ease of maintenance.

Building on the decision to use Java, Apache Maven[13] was selected for dependency management and to support the modular structure of the MOSAIC framework. Maven's robust dependency management capabilities streamline the integration of necessary libraries and tools and ensure consistent and reliable builds. Additionally, Maven facilitates a modular architecture by allowing the project to be divided into discrete, manageable modules, each with its own set of dependencies and build configurations. This modular approach not only simplifies development and maintenance but also enhances the scalability and

[7] https://github.com/andrewtrotman/JASSv2
[8] https://github.com/pisa-engine/pisa
[9] https://github.com/chriskamphuis/olddog
[10] https://github.com/castorini/anserini
[11] https://lucene.apache.org/pylucene/
[12] https://www.oracle.com/java
[13] https://maven.apache.org

extensibility of the MOSAIC framework, thereby making it easier to incorporate new features and improvements over time.

The prototype application developed by Nussbaumer et al. (2023), which stores the metadata in memory, has shown that problems appear when index partitions become larger and the machine runs out of memory. Therefore, the incorporation and handling of metadata in the MOSAIC framework is facilitated using DuckDB[14], a high-performance in-process SQL database management system. DuckDB can efficiently read Parquet files and generate tables from them, which enables seamless integration and querying of structured data within the MOSAIC framework. Essentially, DuckDB enables efficient storage, retrieval, and querying of metadata, which enhances the overall performance of the search engine. Its ability to handle complex queries and large datasets makes it a practical choice for managing the extensive metadata associated with search indexes. In addition, DuckDB integrates seamlessly with Java via the Java Database Connectivity (JDBC) API, thereby allowing for straightforward incorporation into the MOSAIC framework.

Another key design decision was to include a web server framework in the system, with Quarkus[15] being selected for this role. As compared to Spring[16] and similar frameworks, Quarkus offers several advantages, including its ability to provide fast startup times and low memory usage, which are essential for developing efficient and responsive applications. Furthermore, Quarkus supports a wide range of standards and libraries, which facilitates integration with other components of the MOSAIC framework. Its developer-friendly features, such as live coding and streamlined configuration, further enhance productivity and ease of development. Moreover, Quarkus offers a strong foundation for developing a scalable and efficient API. By leveraging Quarkus, the system can achieve high performance and scalability while maintaining a flexible and efficient development process.

The design decision for the web user interface within the MOSAIC framework emphasizes simplicity and accessibility, utilizing HTML, JavaScript, and Cascading Style Sheets (CSS) and deliberately leaving out JavaScript-based front-end libraries. These technologies were chosen for their widespread support and ease of use, which ensures that the interface is both functional and user-friendly.

Considering the requirement that the system should be easy to install and deploy, the design decision to use Docker[17] for containerization was made. This approach reduces setup complexity, enhances portability, and ensures that all parts of the MOSAIC framework can be easily managed and operated coherently.

[14]https://duckdb.org
[15]https://quarkus.io
[16]https://spring.io
[17]https://www.docker.com

3.5. Limitations

The OpenWebSearch.eu project is an ongoing initiative aimed at creating a comprehensive and transparent search infrastructure for the web. As an evolving project, it continuously integrates new features, tools, and improvements. Particularly relevant to this system is the incorporation of additional metadata. Being in an active development phase, the project offers opportunities for early adopters to contribute to and benefit from the latest advancements. As new components and updates are regularly added, users and developers can expect a dynamic and evolving platform that adapts to emerging needs and technologies. This ongoing development phase presents both opportunities for innovation and challenges in terms of ensuring reliability and performance.

Scalability presents a significant challenge for the MOSAIC framework, as it is not designed to operate as a search engine for the entire WWW, but rather for specific, targeted areas by providing a vertical search engine. The system is optimized for handling special data sets and domains and ensures high performance and relevance in these special contexts. Moreover, MOSAIC is designed to be able to run on a computer or server without high-end hardware configurations while maintaining adequate performance. However, the integration of extensive metadata and custom modules can put a further strain on system resources, so that ongoing optimization efforts are required.

3.6. Summary

This chapter acts as a link connecting the review of background literature and related work, and the practical section of the thesis. It starts by emphasizing the motivation driving this research, particularly focusing on the need in establishing a web search framework that leverages indexes created by the index generation pipeline within the OpenWebSearch.eu project. The development of such a framework is crucial to address the limitations of existing search engines, which often lack transparency and control, thereby enhancing the ability to create customized and efficient search solutions.

The chapter outlines both functional and non-functional requirements for the MOSAIC system. Functionally, the system must include key features of existing solutions, such as reliable search functionality, efficient index management, and customizable metadata enrichment. Non-functional requirements emphasize performance, usability, maintainability, and reliability, ensuring the system is not only effective but also user- and developer-friendly and adaptable to various needs.

The conceptual architecture of the MOSAIC framework is then detailed, focusing on design decisions such as the use of Lucene for indexing and search capabilities, and the integration of CIFF and Parquet files for data

management. Furthermore, limitations of the system are also discussed, which include scalability challenges and dependency on ongoing developments within the OpenWebSearch.eu project. These design choices ensure a balance between functionality, efficiency, and ease of use.

In conclusion, this chapter establishes a solid foundation for the development of the MOSAIC system by providing an overview of its design and requirements. This groundwork supports the development of a modular, scalable, and user-friendly framework for vertical search engines using OWI partitions tailored to specific domains.

4. Development

This chapter details the development of the MOSAIC framework. The primary objective of this system is to create a modular and scalable vertical search engine that leverages indexes from the OpenWebSearch.eu initiative. The development of MOSAIC represents a critical phase of this thesis, laying the groundwork for its practical application and subsequent evaluation.

This chapter delves into the development process, critical design considerations, as well as technical components and the startup process of the system, which is discussed in Section 4.1. The primary phase during development, outlined in Section 4.2, involves integrating OWI partitions into the MOSAIC framework, as well as the processing of queries, matching and initial ranking and the representation of search results. Section 4.3 delves into the accessibility of the system via a REST API, focusing on the endpoints provided by the MOSAIC framework. The web user interface is detailed in Section 4.4. Furthermore, the modular aspect of the system is explored in Section 4.5, highlighting the ability to add customizable components and modules. Section 4.6 delves into the setup, installation and configuration capabilities of the MOSAIC framework. Eventually, Section 4.7 explores the MOSAIC2go tool to visually configure a tailored search engine.

4.1. Architecture

Based on the conceptual architecture outlined in Section 3.3 and the design decisions made in Section 3.4, Figure 4.1 presents a simplified technical version of the overall architecture of the MOSAIC system. Additionally, the subsequent sections delve into the development and integration of the subsystems and components within the system.

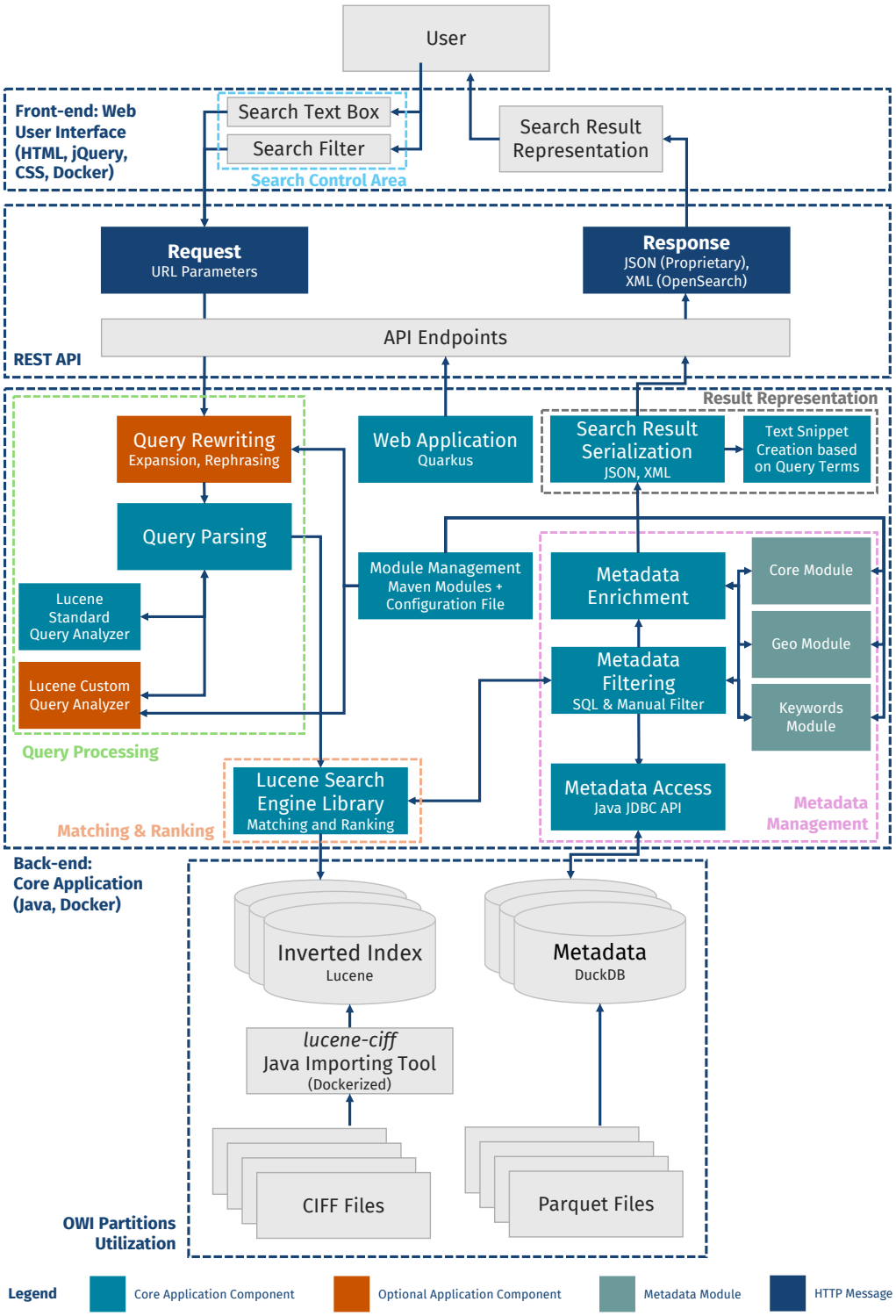


Figure 4.1.: Simplified technical system architecture of the MOSAIC framework, including important subsystems and components and how they are related to each other

At its core, the architecture integrates multiple subsystems and components to facilitate efficient search and retrieval processes. The utilized OWI partitions, comprising CIFF and Parquet files, are managed by the inverted index of the Lucene core search engine library (9.5.0)[1] and enriched with metadata stored in DuckDB through the Java JDBC API[2]. This setup ensures a separation of concerns, where Lucene handles the indices and DuckDB manages metadata. Thus, multiple index partitions can be efficiently managed and easily integrated.

The query processing pipeline includes an optional query rewriting component for expansion and rephrasing with the capability to easily integrate LLMs. This is realized by providing a simple interface where the query string can be modified. In the following, queries are parsed and analyzed using Lucene's standard or optionally custom query analyzers. The system achieves this by offering an interface that allows either the direct use of Lucene's custom query analyzers or the development of a custom analyzer from scratch. The processing ensures that queries are interpreted accurately and comprehensively.

The core application includes a matching and ranking subsystem based on Lucene's advanced algorithms to deliver relevant search results. This phase evaluates the relevance of documents based on various factors, which ensures that the most pertinent information for the given query is presented to the user. The utilization of the sophisticated ranking algorithm BM25[3] as default ranking method helps prioritize search results.

The architecture also includes a metadata module management subsystem, which enhances the search engine's capabilities. This subsystem handles the integration and operation of various metadata modules that enables enriched search results and more precise filtering options. Each module can be incorporated into the core application to ensure that metadata is effectively processed after matching and ranking.

After filtering and enriching the search results with metadata, the results are passed to the result representation subsystem. The search result serialization supports both JSON and XML formats to ensure compatibility with diverse client applications. With this, the core application ensures that search results are accurately serialized and presented to maintain consistency and reliability in the user experience.

The architecture also includes a central plugin management component, which is crucial for incorporating various modules and optional components into the core application. This component is responsible for managing the integration of these modules. The plugin management is realized using Apache Maven modules, which provide a structured and efficient way to handle dependencies and module integration. Modules and components can be easily

[1] https://lucene.apache.org/core/9_5_0/index.html

[2] https://duckdb.org/docs/api/java.html

[3] https://lucene.apache.org/core/9_5_0/core/org/apache/lucene/search/similarities/BM25Similarity.html

enabled or disabled through a configuration file, which allows for flexible customization to add or remove modules as needed as well as maintenance.

HTTP requests from the front-end or other services, including URL query strings and filter parameters, are processed through a REST API built on the Quarkus web framework. The system supports both JSON and XML responses, thereby complying with a proprietary format as well as the OpenSearch protocol. API endpoints handle different functionalities, most importantly searching in OWI partitions. The REST API architecture ensures that the system can seamlessly interact with modern web services and applications, which makes it versatile and adaptable to a broad range of use cases.

The front-end of the framework is a web user interface, developed using HTML, jQuery (3.6.0)[4], and CSS, which provides user interaction points, including a search box, filters for specific metadata, and search result representation. This interface is designed to demonstrate the search functionality and to be user-friendly in order to provide the system across a wide range of technical expertise.

The system offers flexible usage options to accommodate various user needs and preferences. It can be used out-of-the-box by deploying a pre-built Docker image[5]. The system offers a selection of Docker images for users to choose from: one image that contains the whole framework including the index partitions available in the repository, one image that only contains the back-end without direct integration of OWI partitions, one image to easily utilize the index importing tool, and one image that contains the front-end. Docker containers can be created using either the images provided in the container registry or using images created locally. This allows users to quickly set up and run the application with minimal configuration. For those who wish to customize or extend the framework, the source code can be modified, and the entire application can be built and run manually.

In addition to these options, the MOSAIC framework supports live coding, which enables developers to see their changes in real-time without the need for manual compilation after every code modification. This feature can significantly enhance the development experience by streamlining the code update process.

Overall, this architecture aligns with the conceptual design by incorporating modular components such as metadata enrichment and query analyzers, so the system is adaptable and can evolve with future requirements. The combination of Lucene for indexing and DuckDB for metadata management ensures a separation of concerns, while the use of a REST API and Quarkus framework supports straightforward integration and scalability.

[4]https://jquery.com

[5]https://opencode.it4i.eu/openwebsearcheu-public/mosaic/container_registry

```
String sql = "CREATE TABLE " + indexName.replace('-', '_') + " AS " +
             "SELECT " + columns + " " +
             "FROM read_parquet('" + CoreUtils.getParquetDirPath() +
   indexName + File.separator + "*.parquet*') " +
             "ORDER BY " + CoreUtils.getIdColumn();
try {
    conn.createStatement().execute(sql);
} catch (SQLException e) {
    LOGGER.error("Failed to create table using DuckDB for index " +
   indexName, e);
}
```

Listing 4.1: Import of Parquet files from an index partition to a database table using DuckDB

4.2. Core Application

This section highlights technical aspects of the core application development. It comprises the integration of several key components and how they interact with each other.

4.2.1. OWI Partitions Utilization

The MOSAIC framework utilizes OWI partitions by importing CIFF files and Parquet files to facilitate efficient search operations. CIFF files are imported into the Lucene index using the `lucene-ciff` tool, which is executed through a dedicated `import_index` script. The tool generates a Lucene index from CIFF files by utilizing Lucene's codecs module, thereby creating wrappers for the *reading* part to convert CIFF data into Lucene's internal formats and then using the *writing* part to produce an index that is compatible with various codecs. This process ensures that the term-document mappings are accurately integrated into the search engine.

In addition, one or multiple Parquet files are imported into a database table using DuckDB which creates the table based on the schema of the provided Parquet files. For this, a DuckDB database file is created on the file system by default. The import of Parquet files of a particular index partition is illustrated in Listing 4.1.

This dual-import strategy ensures that both the raw index data and the associated metadata are seamlessly integrated into the MOSAIC framework. Moreover, the MOSAIC framework supports both compressed and uncompressed index partition files. Figure 4.2 illustrates an example of the structure for two imported index partitions.

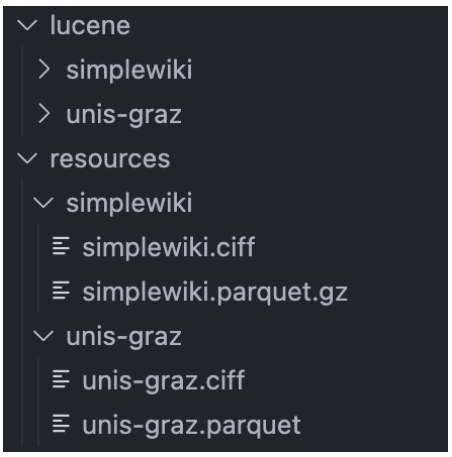


Figure 4.2.: Example structure of index partitions imported into the MOSAIC framework

4.2.2. Query Processing

Once the index partitions are imported, users can perform search queries through the MOSAIC framework. A search query is submitted via the web user interface, any other service or directly via the REST API, which then forwards the query. Before the query is actually analyzed, query manipulation operations can be performed.

4.2.2.1. Query Rewriting

The query rewriting component in the MOSAIC framework is an optional feature that provides an interface for enhancing search queries. It can be used to expand, rephrase, or modify queries to improve search accuracy and relevance. This component ensures that the search engine can interpret and respond to user queries more effectively, even when the initial input may be ambiguous or incomplete. Additionally, a LLM can be integrated into this component to further enhance the sophistication and precision of the query rewriting process. Listing 4.2 shows the method stub where users can implement custom query rewriting operations before the query is actually parsed analyzed.

4.2.2.2. Query Parsing

The search query is parsed and analyzed prior to executing the actual search in the Lucene index. The optional query analysis component in the MOSAIC framework is responsible for parsing and understanding the search queries. By default, Lucene's StandardAnalyzer[6] is used, which tokenizes the input text, removes common stop words, and applies lowercase normalization to

[6]https://lucene.apache.org/core/9_5_0/core/org/apache/lucene/analysis/standard/StandardAnalyzer.html

```java
/**
 * Modifies the query string before it is analyzed and parsed.
 * Change the implementation of this method stub to modify the query
    string.
 * @param q Original query string
 * @return Modified query string
 */
@Override
public String modifyQuery(String q) {

    String modifiedQuery = q;

    // Implement the desired query modification

    return modifiedQuery;
}
```

Listing 4.2: Method stub providing the possibility to rewrite the original query

standardize the query terms. Users have the flexibility to choose an alternative existing analyzer or implement their own custom analyzer to suit their specific requirements, as shown in Listing 4.3.

4.2.3. Matching and Ranking

After the query has been analyzed within the MOSAIC framework, the matching and ranking process begins. The system accesses the appropriate Lucene index stored in the file system, thereby verifying the existence of the specified index. A Lucene IndexReader[7] is then created to read the index from the file system, and a Lucene IndexSearcher[8] is initialized to perform the search operations. The IndexSearcher uses the BM25Similarity[9] algorithm to rank the search results based on their relevance to the query, as shown in Figure 4.3. This algorithm, a state-of-the-art ranking function, calculates the similarity between queries and documents based on term frequency and document length. The search results are fetched from the Lucene index, processed, and returned as a list of relevant documents. This method guarantees the identification and effective ranking of the most relevant documents by delivering accurate and valuable search results to users.

[7]https://lucene.apache.org/core/9_5_0/core/org/apache/lucene/index/IndexReader.html

[8]https://lucene.apache.org/core/9_5_0/core/org/apache/lucene/search/IndexSearcher.html

[9]https://lucene.apache.org/core/9_5_0/core/org/apache/lucene/search/similarities/BM25Similarity.html

```java
/**
 * Returns the appropriate Lucene Analyzer. Use this stub to define
   the Analyzer for the Lucene index.
 * As an alternative to the default analyzer, use either an existing
   Lucene Analyzer or create a custom one.
 * Change the implementation of this method stub to return the
   desired Analyzer.
 * @param defaultAnalyzer Default Lucene Analyzer defined in the core
    module
 * @return Custom Lucene Analyzer
 */
public Analyzer getAnalyzer(Analyzer defaultAnalyzer) {

    Analyzer analyzer = defaultAnalyzer;

    // Implement the desired analyzer

    return analyzer;
}
```

Listing 4.3: Method stub providing the possibility to use another existing query analyzer or an own custom analyzer

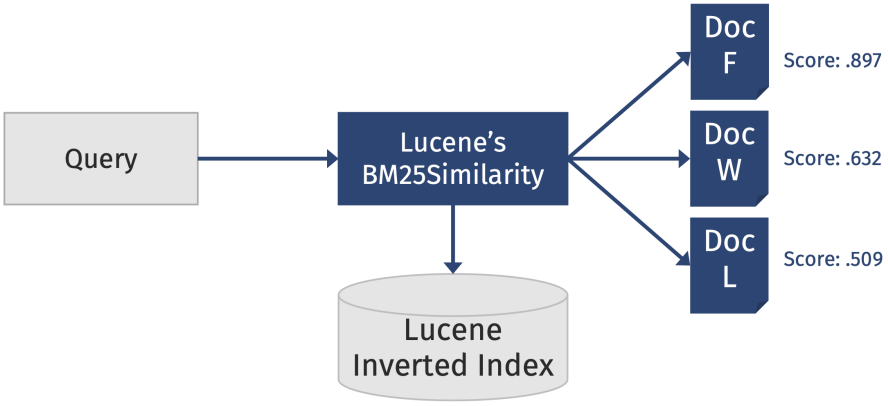


Figure 4.3.: Initial ranking of documents based on Lucene's BM25Similarity

4.2.3.1. Optional Re-Ranking

The MOSAIC framework also includes the capability to re-rank search results after they have been retrieved from the Lucene index. Currently, a demonstration implementation provides the option to re-rank documents based on their word count, which illustrates the potential for adjusting ranking criteria post-retrieval. This example underscores the flexibility of the framework to incorporate additional re-ranking algorithms tailored to specific needs. Essentially, the re-ranking process allows for further refinement of search results.

4.2.4. Metadata Filtering

The system incorporates a dual approach comprising metadata filtering and enrichment. User-provided filtering parameters are processed by the MOSAIC core application to refine search results based on specific criteria. This filtering can be executed directly using metadata columns or through advanced algorithms, so-called manual filtering, that leverage these metadata columns for more sophisticated filtering. The manual filtering procedure for a single document is demonstrated in Listing 4.4.

4.2.5. Metadata Enrichment

Once the filtering process is complete, the system enriches the search results by adding relevant metadata from the Lucene index. This metadata enhancement ensures that the final search results are comprehensive and contextually informative, ready to be returned by the MOSAIC back-end to the user. Further details on the incorporation of metadata is explored in Section 4.5 which describes the modular architecture of the MOSAIC framework.

4.2.5.1. Text Snippet Creation

Metadata enrichment in the MOSAIC framework includes creating text snippets, which are crucial for providing users with quick, relevant previews of search results. Within MOSAIC, text snippets are generated from the full text available in the document's metadata table record. The snippet creation process identifies the most pertinent section of the text, particularly the sentences where the query terms appear most frequently.

In scenarios where only a portion of the full plain text of documents is stored in the table to optimize storage and performance, MOSAIC offers an on-demand loading mechanism, as depicted in Figure 4.4. This mechanism allows the complete plain text to be retrieved from the Parquet file when it is necessary to generate more accurate and satisfying text snippets for documents. Dynamically loading the full text as needed ensures that, despite storage optimizations, the quality and relevance of the text snippets are maintained.

```java
// Create a map of metadata columns and their values for the search result
Map<String, String> result = new TreeMap<>();
for (int i = 1; i <= rsMetadata.getColumnCount(); ++i) {
    result.put(rsMetadata.getColumnName(i), rs.getString(i));
}
result.put("index", indexName);

// Check if the search result passes the manual filter of the metadata modules
boolean passedManualFilter = true;
for (MetadataModule module : PluginManager.getInstance().getModules().values()) {
    if (!module.inManualFilter(result, queryParams)) {
        LOGGER.info("Search result did not pass manual filter of module: {}", module.getClass().getSimpleName());
        passedManualFilter = false;
        break;
    }
}

// Add the enriched search result to the list of results
if (passedManualFilter) {
    rs.close();
    dbConn.closeConnection();
    return result;
}
```

Listing 4.4: Snippet of the metadata filtering and enrichment procedure demonstrating the manual filtering process

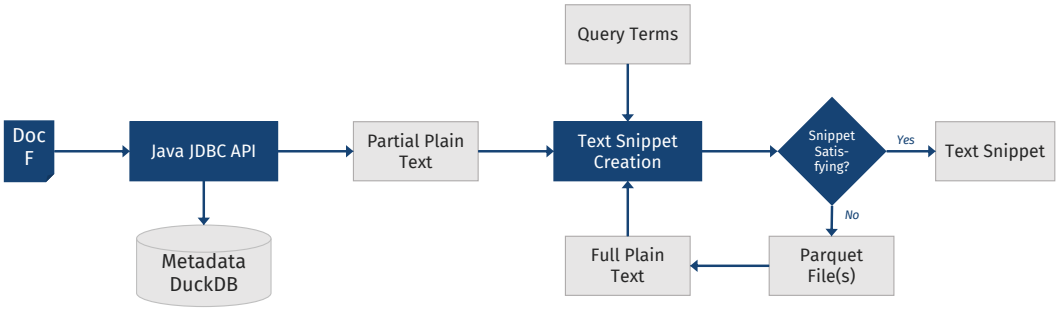


Figure 4.4.: Text snippet creation based on the plain text with optional on-demand loading of the full plain text of a document

4.2.6. Result Representation

After filtering and enrichment of metadata, the search results are formatted and prepared for presentation. The framework supports multiple formats, including JSON and XML serialization, to accommodate various user needs and integration requirements. Section 4.3 delves into the exact format of the result representation of search results within the MOSAIC framework.

In JSON format, search results are grouped by index partition, presenting a structured list where each group contains the relevant documents from a specific index partition. This grouping can help users easily identify and navigate through results from different data sources and facilitates parsing in the web user interface since the results are already categorized.

For the XML format, which aligns with the OpenSearch protocol (1.1, Draft 6), the results are not grouped by index partition. Instead, each search result includes an additional field that indicates the index partition from which the document originates. This approach maintains compliance with the OpenSearch standard while providing users with essential context about the source of each search result. The OpenSearch protocol not only allows the application to accept search requests via defined query parameters and process these requests, but also generate Atom-based[10] XML responses.

4.3. REST API

The MOSAIC framework offers a comprehensive REST API for receiving search queries and returning responses containing the search results. In its current version, four distinct endpoints are available, which can be accessed using a web browser or any API testing platform, for instance, Postman[11]. Table 4.1 shows an overview of the endpoints provided by the MOSAIC framework. By default, the MOSAIC back-end is accessible via the API on port `8008`. The port definition can be adjusted according to user requirements and the specific environment in which the framework is deployed. Further details on the endpoints and their functionalities are provided in the following subsections.

4.3.1. Search with JSON Response

The endpoint `/search` performs searches within one or multiple index partitions using the provided REST query parameters and returns the results in JSON format. The available query parameters may vary depending on the enabled modules, which are explored in Section 4.5. Generally, while no query parameter is strictly required, the parameter `q` is commonly used to specify

[10] https://datatracker.ietf.org/doc/html/rfc4287
[11] https://www.postman.com

Endpoint	Description
`/search`	Handles search queries and returns search results in a proprietary JSON format.
`/searchxml`	Handles search queries and returns results in XML format compliant with the OpenSearch protocol.
`/index-info`	Provides a list of available index partitions including essential properties of each index partition in JSON format.
`/full-text`	Retrieves the full text of a document based on its identifier.

Table 4.1.: REST API endpoints supported by MOSAIC

query terms. A trivial HTTP GET request for this endpoint using the default port `8008` is `http://localhost:8008/search?q=europe`.

Through the API, the core application returns a response containing a list of search results, with each result comprising the fields specified by the enabled modules. If no index name is provided as a parameter, the back-end searches across all available index partitions and returns a separate list of results for each index partition.

For instance, a response having only the `core` module enabled could yield a JSON object as illustrated in Listing 4.5.

```
{
  "results": [
    {
      "simplewiki": [
        {
          "id": "cfe49c84-2244-430e-83e3-fe3f30aae21e",
          "url": "https://simple.wikipedia.org/wiki/Anthem_of_Europe
   ",
          "title": "Wikipedia: Anthem of Europe",
          "textSnippet": "https://simple.wikipedia.org/wiki/
   Anthem_of_Europe",
          "language": "eng",
          "warcDate": "2024-01-15T22:19:46Z",
          "wordCount": 11
        },
        ...
      ]
    },
    {
      "unis-graz": [
        { ... },
```

```
      ],
      ...
    },
    ...
  ]
}
```

Listing 4.5: Example response of the `/search` endpoint

4.3.2. Search with XML Response

Similar to the `/search` endpoint, the endpoint `/searchxml` facilitates searching within one or multiple index partitions based on REST query parameters provided in the HTTP GET request. However, the results are returned in XML format in accordance with the OpenSearch protocol. Listing 4.6 exemplarily shows the structure of such an XML response. In the response, the node `item` represents one search result and contains the fields specified in the enabled modules.

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
  <title>MOSAIC Search: {searchTerms}</title>
  <description>Search results for "{searchTerms}" at MOSAIC
               Search Service</description>
  <author>
    <name>OpenWebSearch.eu</name>
  </author>
  <opensearch:totalResults>1121</opensearch:totalResults>
  <opensearch:startIndex>1</opensearch:startIndex>
  <opensearch:itemsPerPage>20</opensearch:itemsPerPage>
  <opensearch:Query role="request" searchTerms="{searchTerms}"
                    startPage="1"/>
  <link rel="alternate"
        href="{baseUrl}/search?q={searchTerms}&pw=1&limit=20"
        type="application/json"/>
  <link rel="self"
        href="{baseUrl}/searchxml?q={searchTerms}&pw=1&limit=20"
        type="application/atom+xml"/>
  <link rel="next"
        href="{baseUrl}/searchxml?q={searchTerms}&pw=2&limit=20"
        type="application/atom+xml"/>
  <link rel="last"
        href="{baseUrl}/searchxml?q={searchTerms}&pw=56&limit=20"
        type="application/atom+xml"/>
  <link rel="search"
        type="application/opensearchdescription+xml"
        href="{baseUrl}/opensearch.xml"/>
```

```
  <item> ... </item>
  ...
</feed>
```

Listing 4.6: Example response of the `/searchxml` endpoint

4.3.3. Index Information

The `/index-info` endpoint is not related to search in the narrower sense but provides more contextual and detailed information about the index partitions used in the current MOSAIC core application execution. It does not require any REST query parameters and returns the response in JSON format. For each index partition, the information includes the index partition name, the number of indexed documents, and a list of languages present in the index partition. The number of indexed documents is determined using Lucene's indexing capabilities, while the list of languages present in the index partition is determined by referencing the associated metadata stored in the database table. An example response of the `/index-info` endpoint is shown in Listing 4.7.

```
{
  "results": [
    {
      "simplewiki": {
        "documentCount": 285392,
        "languages": [
          "deu",
          "eng",
          "est",
          "fra",
          "ltz",
          "pol",
          "unknown",
          "zho
        ],
      }
    }
  ]
}
```

Listing 4.7: Example response of the `/index-info` endpoint

This endpoint can be utilized to display the available indexes in the front-end interface, thereby providing users with a clear overview of the data sources being queried. By accessing detailed information about each index partition,

such as the number of indexed documents and the languages present, users can better understand the scope and content of the available data present in the current MOSAIC core application execution. This transparency enhances user interaction and allows for more informed query formulation.

4.3.4. Full Plain Text Retrieval

As detailed in Table 4.5, the option `-n <num>` allows importing only a portion of the full plain text of each document into the database. This approach is particularly beneficial for larger index partitions, as it helps to reduce both the creation time and the file size of the database. To retrieve the full plain text of a web document, the MOSAIC framework provides the `/full-text` endpoint. This endpoint requires the parameter `id` to identify the specific document. Additionally, the parameter column can be used to specify which metadata column the provided document ID should match, with the default being `record_id`. For scenarios involving multiple index partitions, the optional parameter `index` can be included if the index partition name containing the document is already known. Listing 4.8 illustrates the structure of the JSON response.

```
{
  "id": <id>,
  "fullText": <fullText>
}
```

Listing 4.8: JSON response structure of the `/full-text` endpoint

This endpoint can be used for tasks such as retrieving the full content of documents for detailed analysis or for automatic summarization by LLMs. It enables comprehensive access to document text and facilitates advanced processing and content generation applications.

4.4. Web User Interface

Mainly conceptually designed and further developed by Alexander Nussbaumer, the web user interface[12] within the MOSAIC framework is intentionally kept very simple to demonstrate the functionality of MOSAIC and not to distract end users from the actual purpose. The structure and functionality of the web user interface are illustrated in Figure 4.5. Below this, the search results are displayed in a list format, similar to established search engine user interfaces. Depending on the module configuration, each result returned by the MOSAIC back-end includes the title of the web page, a text snippet, the WARC

[12]https://qnode.eu/ows/mosaic/webinterface/

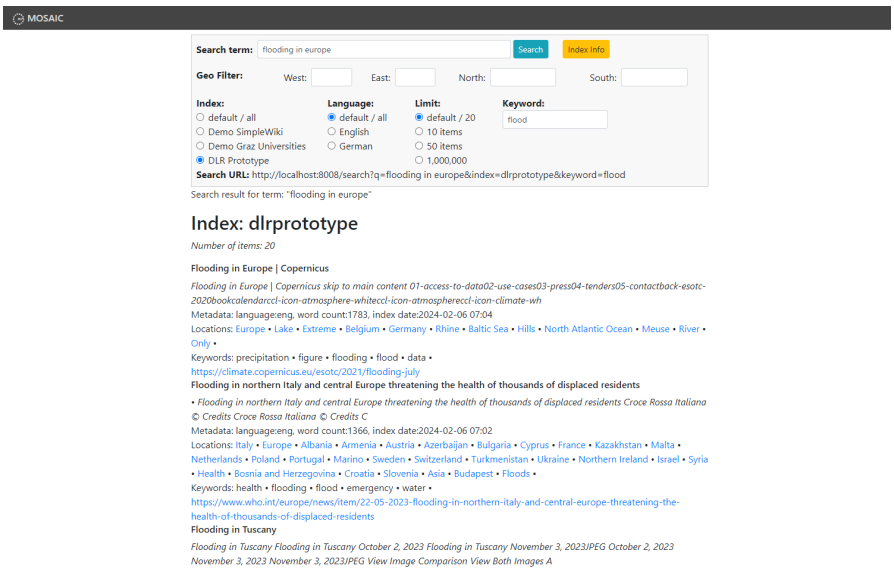


Figure 4.5.: Basic web user interface implementation within the MOSAIC framework

date, the extracted locations from the full plain text, the extracted keywords, and the URL of the web page. Furthermore, when a search is performed across multiple index partitions, the results are separated by index partition, which allows users to easily identify which index partition each result originates from. Additionally, the retrieval time for each search query is measured in the front-end and displayed to the end user. In general, this structured approach ensures that end users can efficiently navigate and interpret the search results.

4.4.1. Search Control Area

At the top of the web user interface, there is a search control area where end users can enter their search queries in a search box and utilize various filtering options. Search criteria specified by end users are used to assemble the HTTP query string.

4.4.1.1. Search Box

The search box in the front-end is a fundamental component designed for end user interaction. Implemented as a simple HTML input element, it allows end users to formulate their search queries. Upon submission per clicking the button or pressing the enter key, these queries are transmitted to the back-end via the API endpoint `/search` for further processing. The content of the search box is used as the content of the URL query parameter `q`.

4.4.1.2. Search Filter

Depending on the integrated OWI partitions and the enabled metadata modules, end users can select and specify filter options to narrow down search results. End users can choose available indices for searching and apply filtering options such as the language of indexed documents, coordinates for bounding boxes for geospatial data, and keywords extracted from the full plain text. The available indices and languages present in these indices are fetched from the back-end via the API endpoint `/index-info`.

4.4.2. Search Result Representation

The search result representation in the front-end is organized by the indices in which the search was performed, with results displayed in a structured list format. Each search result, received either as a JSON object or XML node, is presented with several key attributes. These include the title, a text snippet, the URL as link to the web document, and the WARC date, which is converted to an ISO 8601 date string[13]. Additionally, if available in the metadata, the locations and keywords associated with each result are also displayed.

End users have the option to interact with the location data by clicking on the displayed locations, which will then open the coordinates in OpenStreetMap[14]. It is important to note that the extracted locations may not always be completely precise due to limitations in the OpenWebSearch.eu preprocessing pipeline. This feature allows end users to visually explore the geographical context of the search results.

4.4.3. Index Information

The web user interface also provides detailed information about the available index partitions in the current MOSAIC core application execution by utilizing the `/index-info` API endpoint. This feature is realized by providing a simple HTML button and displays upon request the available index partitions along with the corresponding number of documents and the languages present in each partition. By presenting this information transparently, the interface helps users understand the scope and content of the index partitions being searched.

4.5. Module Management

Following a plugin architecture, the MOSAIC framework adopts a modular approach that allows for flexible integration of metadata modules and optional

[13]https://www.iso.org/iso-8601-date-and-time-format.html
[14]https://www.openstreetmap.org

system components. This design enables users to customize their search engine by selectively enabling or disabling specific modules according to their needs. In addition, by supporting a wide range of modules and components, the MOSAIC framework offers a scalable infrastructure that can evolve with emerging technologies, such as LLMs, and user demands.

4.5.1. Technical Concept

In the technical implementation of the MOSAIC framework, each module and component is managed using Apache Maven, which handles the dependencies, build processes, and lifecycle management. Modules are defined with clear interfaces and can interact with the core system and other modules through interfaces. This approach allows for straightforward integration and communication between different subsystems, components and modules of the framework. Additionally, the integration of modules is managed through a configuration file, which provides the option for users to easily enable or disable specific functionalities without altering the core system. Listing 4.9 shows an excerpt of a configuration file entry for modules and components that are enabled and subsequently included in the back-end when executing the application.

```
"modules": {
    "core": "eu.ows.mosaic.CoreMetadata",
    "geo": "eu.ows.mosaic.GeoMetadata",
    "keywords": "eu.ows.mosaic.KeywordsMetadata",
    "query": "eu.ows.mosaic.CustomQuery",
    "analyzer": "eu.ows.mosaic.CustomAnalysis"
}
```

Listing 4.9: Excerpt from the configuration file showcasing the enabled modules and components

4.5.2. Metadata Modules

MOSAIC incorporates metadata modules that can be used to enrich search results with additional contextual information, such as titles, URLs, language and other metadata. These modules facilitate additional filtering of search results and can append extra fields to the response, which enhances the detail and relevance of the returned data. Table 4.2 provides an overview of currently existing metadata modules within the MOSAIC framework. Moreover, MOSAIC is based on Maven modules and allows users to simply add new modules by themselves. Thus, a few actions, which are shown in Appendix A, are required to incorporate a new metadata module for additional filtering capabilities and metadata enrichment.

Module Name	Necessity	Description
Core	Required	Responsible for the basic search and the response and contains the web framework.
Geo	Optional	Extends text-based search by utilizing geographical information.
Keywords	Optional	Incorporates extracted keywords from the full plain text.

Table 4.2.: Existing modules in the current version of MOSAIC

In order to function properly, every metadata module must include a Java class that extends `MetadataModule`[15]. Subsequently, by overriding the following methods from the base class, users can specify filtering options, as well as the inclusion of additional metadata columns:

- `getMetadataColumns()`: Returns the set of metadata columns expected to be used by the module. Subclasses should override this method to specify the necessary metadata columns available in the Parquet file(s).
- `getFilterColumns()`: Provides the set of filter columns used for filtering results via DuckDB. Subclasses in modules should override this method to define the required filter columns present in the Parquet file(s).
- `validateParams()`: Validates the query parameters passed to the module. Subclasses in modules should override this method to implement additional validation logic as needed.
- `parseQueryParams()`: Parses the query parameters, returning a map of parameters that match the defined filter columns. It ensures that only relevant parameters are processed for filtering.
- `getSqlFilterClauses()`: Generates SQL filter clauses based on the query parameters and available metadata columns. By default, it constructs additional `WHERE` clauses for each filter column using the equal sign (=) for comparison.
- `getSqlFilterValues()`: Returns a list of SQL filter values used as parameters for a *PreparedStatement*. It gathers values from the query parameters that correspond to the defined filter columns.
- `inManualFilter()`: Checks if a search result meets the criteria of a manual filter. Subclasses in modules should override this method to implement custom filtering logic that extends beyond SQL-based filters.
- `serializeJson()`: Serializes a search result into a JSON object. By default, it includes metadata columns defined in `getMetadataColumns()`

[15]https://opencode.it4i.eu/openwebsearcheu-public/mosaic/-/blob/main/search-service/shared/src/main/java/eu/ows/mosaic/MetadataModule.java?ref_type=heads

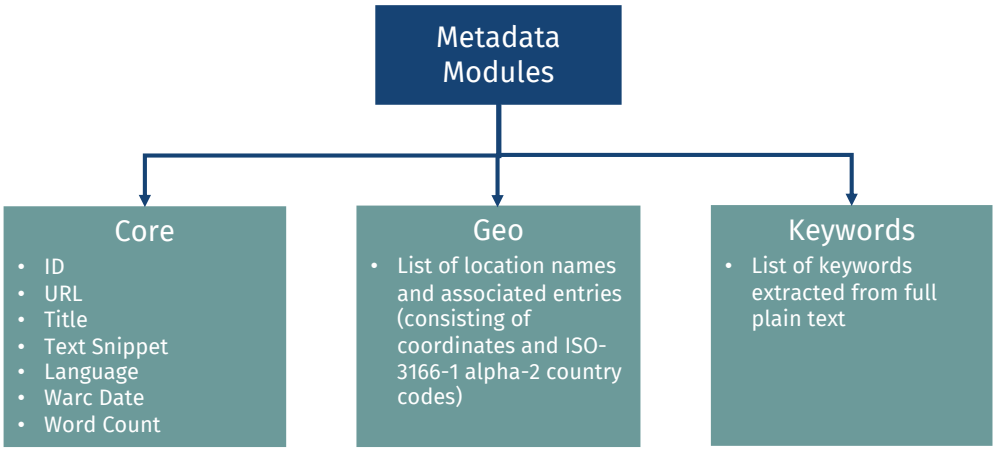


Figure 4.6.: Metadata fields added by the modules to the search results

that are present in the result.

- `serializeXml()`: Serializes a search result into an XML string. It includes metadata columns specified in `getMetadataColumns()` and processes each value to ensure it is properly formatted for XML.

Figure 4.6 depicts the metadata fields used for enrichment by each module. The core metadata fields include essential information such as ID, URL, title, text snippet, language, WARC date, and word count. The geo module contains a list of location names and the corresponding entries consisting of coordinates and ISO-3166-1 alpha-2 country codes[16]. Additionally, the keywords module includes extracted keywords from the full plain text of documents. Based on the metadata modules configuration, these fields are added to the search result after matching, initial ranking and filtering.

4.5.2.1. Core Module

As the only required module along the shared module, the core module provides the possibility to search in one or multiple index partitions from the OWI. It is the main architectural component, and all other components and modules depend on this module. The core module is responsible for the basic search, filtering and metadata enrichment. Table 4.3 highlights the URL query parameters which are processed by this module.

4.5.2.2. Geo Module

The geo module extends the text-based search by using geographical information that is stored in the metadata. This module leverages detailed metadata to accurately parse and incorporate geographic information into the search

[16]https://www.iso.org/iso-3166-country-codes.html

Parameter	Value	Necessity	Description
`q`	string	Optional	Search term(s) to be searched for in the Lucene index.
`index`	string	Optional	Specifies the Lucene index to be searched in. The passed value must match the folder name of the Lucene index. If no index is specified, a separate search in all indexes that are present is performed.
`lang`	string	Optional	Restricts the search result to only consider pages in the specified language (e.g., *eng*). If no language is specified, the search results are language independent.
`ranking`	string	Optional	Specifies the order of the search result based on the number of words a page has. Can be either *asc* or *desc*. If no ranking is specified, the order of the search result yielded by Lucene's similarity search is used.
`pw`	int	Optional	Defines the page number of the set of search results desired by the search client. If no page number is specified, 1 is used.
`limit`	int	Optional	Sets the maximum number of results to be returned. If no limit is specified, a maximum of 20 results are returned by default per page.
`fulltext`	boolean	Optional	Loads the full plain text on-demand from the Parquet file(s) to generate the text snippet if the query term(s) are not present in the plain text which is stored in the database. If not specified, the full text is not loaded dynamically.

Table 4.3.: URL query parameters processed by the core module

Parameter	Value	Necessity	Description
`east`	float	Optional	Specifies the max. longitude.
`west`	float	Optional	Specifies the min. longitude.
`north`	float	Optional	Specifies the min. latitude.
`south`	float	Optional	Specifies the max. latitude.
`operator`	string	Optional	Specifies whether all locations (i.e., *and*) or at least one (i.e., *or*) location of the search result must be inside the bounding box. The default operation is *or*.

Table 4.4.: URL query parameters processed by the geo module

results. For instance, a bounding box comprising four coordinates can be used as a filter to ensure that the extracted locations of a web page fall within specific geographic areas. Table 4.4 outlines the URL query parameters which are processed by this module.

In the response, each search result includes the location name and coordinates, thereby providing precise geographic context. This inclusion of location names and coordinates is particularly beneficial for visualizing search results on a map in the front-end interface. Essentially, this module exemplifies a sophisticated implementation, featuring extensive manual filtering, metadata parsing, and enrichment processes.

4.5.2.3. Keywords Module

In contrast to the geo module, the keywords module demonstrates the simplest approach for a metadata module. It allows for simple filtering by specific keywords and enriches the search results with previously extracted keywords. Outlined in Listing 4.10, the keywords module achieves this by specifying a single metadata column for both filtering and inclusion in the search results response.

```
@Override
public Set<String> getMetadataColumns() {
    return Set.of("keywords");
}

@Override
public Set<String> getFilterColumns() {
    return Set.of("keywords");
}
```

Listing 4.10: Simplest approach to define an additional metadata filter and inclusion in the search results illustrated by the implementation of the keywords module

4.5.3. Optional Application Components

In addition to metadata modules, optional application components are further Apache Maven modules that can be integrated into the MOSAIC framework to extend its capabilities beyond the core functionality. Two of these components are detailed in Section 4.2 as they illustrate the flexibility and extensibility of the framework. These components demonstrate how additional functionalities can be integrated into the existing system to enhance its capabilities, while the functionality of the core application and other components is still ensured.

To add these optional components, the MOSAIC framework utilizes Maven modules to manage dependencies and build processes. Depending on the purpose of the component, interfaces are defined to allow these components to interact with the core application at specific integration points. This ensures that the components can communicate effectively with other subsystems and other modules. As for the metadata module, the configuration file is used to eventually enable or disable these components.

4.6. Installation and Configuration

The MOSAIC framework is designed with usability and accessibility in mind, and is aimed to a broad spectrum of users, including developers and search engine operators with varying levels of technical experience. For those with minimal technical expertise, MOSAIC provides an easy-to-use platform where users can simply download the framework, make necessary adaptations, select the desired index partitions, and then build and run the application with minimal effort. This ensures that the inhibition threshold for working with MOSAIC remains low, thereby making it accessible to users with little to no technical background while maintaining a configurable setup and execution. Such accessibility also makes MOSAIC suitable for academic settings, where students and researchers can quickly set up and experiment or prototype with search engine technologies.

4.6.1. Framework Setup

MOSAIC is publicly available in a GitLab[17] repository[18] hosted by OpenWebSearch.eu project partners. The technical project setup for the MOSAIC framework involves organizing the core application and additional modules and components within the `search-service` directory. This directory is the central part of the project, encompassing the web framework, providing the API, and managing the integration of index partitions, as well as the incorporation of various metadata modules and optional components.

OWI partitions, consisting of CIFF and Parquet files, are stored in the `resources` directory by default, with each partition named specifically. The imported Lucene indices are stored in the `lucene` directory by default, also organized by partition name. However, it is also possible to modify the directory paths for both `resources` and `lucene` by specifying the associated CLI startup option. The `lucene-ciff` directory contains the tool for importing CIFF files into Lucene indices. Noteworthy, in the repository of the MOSAIC framework there are already two index partitions available, particularly useful for testing the framework independently of required own partitions:

- `simplewiki`: An index partition used for demonstration purposes containing abstracts of Simple Wikipedia[19] pages (241,839 documents).
- `unis-graz`: An index partition used for demonstration purposes containing documents related to the University of Graz and Graz University of Technology (1,935 documents).

Moreover, additional index partitions can be downloaded and used via the NextCloud[20] instance of Graz University of Technology. Some of these index partitions were created by the index generation pipeline of the OWSAI, while others were created using the pipeline locally.

Additionally, the scripts folder includes several shell scripts and batch files for both Unix-like operating systems and Windows respectively: one for importing CIFF files into Lucene indices, one for compiling the MOSAIC back-end and importing any unimported CIFF indices, and one for starting the core application with various configuration options via a CLI.

The `front-end` directory contains a basic web user interface using HTML, JavaScript (jQuery) and CSS, that demonstrates the core application's capabilities of the MOSAIC framework.

[17] https://gitlab.com
[18] https://opencode.it4i.eu/openwebsearcheu-public/mosaic
[19] https://simple.wikipedia.org/wiki
[20] https://cloud.tugraz.at/index.php/s/xHnJNCozT9joedt

4.6.2. Framework Startup Process

Once the framework is compiled using the script `build.sh`, the `start.sh` script executes the compiled application. During the startup process of the MOSAIC back-end, numerous actions are executed in the following order:

1. The path to the index directory containing the Lucene indices of one or more index partitions is set.
2. The path to the metadata directory containing the Parquet file(s) of one or more index partitions is set.
3. The `ID` column used to match Lucene documents with corresponding metadata in the Parquet files is specified.
4. The path to the framework configuration file is defined.
5. Components and modules are loaded according to the framework configuration file.
6. The path to the DuckDB database file, that is used for storing the metadata from Parquet files of an index partition in a table, is specified.
7. The database table containing the metadata of an index partition is created if it does not already exist. Additionally, if the option `-n <num>` is passed during startup, the full plain text in the `plain_text` column is truncated to `<num>` characters for each row.
8. The OpenSearch XML document is updated in accordance with the framework configuration file.

4.6.2.1. CLI Startup Options

To facilitate flexible and straightforward use and development based on the MOSAIC framework, the MOSAIC core application supports several CLI options which can be passed optionally at the application startup, as outlined in Table 4.5. In particular, the options `-l` and `-p` can be beneficial to map existing directory structures directly into the framework without the need to explicitly move or copy all index partition files into the directories provided for this purpose.

4.7. MOSAIC2go

Built on top of MOSAIC, the MOSAIC2go platform provides a web user interface designed to facilitate the creation of customized search engines. The motivation behind MOSAIC2go is to focus on empowering end users to create their own search engines based on the MOSAIC framework, rather than primarily supporting developers as the MOSAIC framework does.

Option	Description
`-l, --lucene-dir-path <dir>`	path of directory containing the Lucene index(es) (default = `lucene` directory as in the repository)
`-p, --parquet-dir-path <dir>`	path of directory containing the Parquet file(s) (default = `resources` directory of this repository)
`-i, --id-column <col>`	column that contains the document identifiers (default = `record_id`)
`-n, --num-characters <num>`	number of characters selected from the full plain text column to be stored in the associated database table column
`-d, --db-file-path <dir>`	path of directory containing the database file (file is created when starting the MOSAIC backend for the first time) (default = `/tmp/mosaic_db`)

Table 4.5.: Supported CLI options for MOSAIC core application startup

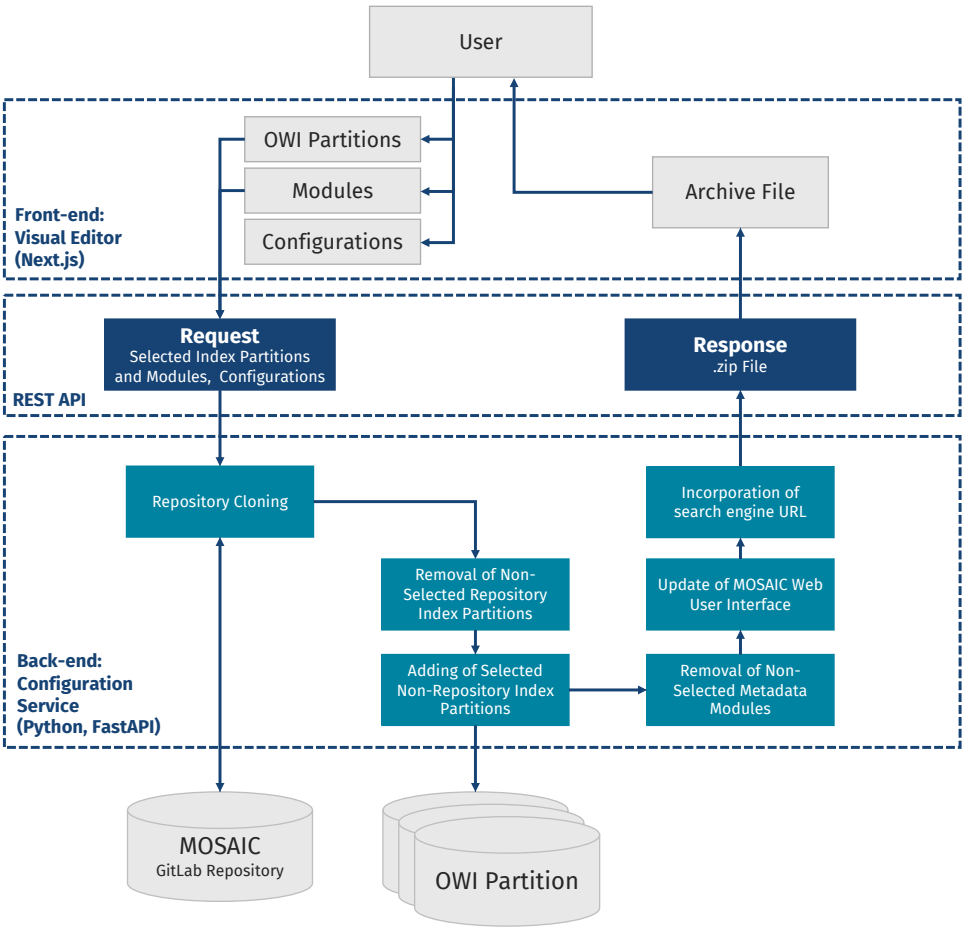


Figure 4.7.: Simplified technical system architecture of MOSAIC2go, including important subsystems and components and how they are related to each other

4.7.1. Architecture

MOSAIC2go's technical architecture, as depicted in Figure 4.7, is composed of several interconnected subsystems that work together to facilitate the creation and configuration of customized MOSAIC search engine instances.

The back-end of the MOSAIC2go architecture is developed using Python and FastAPI[21], which together provides the configuration service. This subsystem is responsible for processing user requests related to the selection of index partitions and modules, as well as managing various configurations related to the web user interface and search engine settings. Upon receiving a request through the provided API, the configuration service packages the selected components and configurations into a .zip file, which is then sent back to the user via the API.

The REST API acts as the intermediary layer between the front-end visual

[21]https://fastapi.tiangolo.com

editor and the back-end configuration service. It facilitates communication by handling requests and responses between these subsystems. Users interact with the API to submit their configurations and selections, which are then processed by the back-end service. Furthermore, the API ensures that also other services can access the configuration service of MOSAIC2go.

Developed using the framework Next.js[22], the front-end of MOSAIC2go delivers a user-friendly visual editor for configuring the MOSAIC search engine instance. This visual editor allows users to select index partitions, modules, and other configurations through an intuitive interface. The editor communicates with the back-end service via the REST API to submit user selections and receive the configured .zip file.

4.7.2. Configuration Service

The MOSAIC2go configuration service is integral to the system as it manages the processing and packaging of user-defined configurations. Each configuration request to the service is assigned an ID which is returned in the response along the customized MOSAIC instance. The customization and assembly process of a MOSAIC instance generated by the current version of the MOSAIC2go configuration service is detailed as follows:

1. **Clone MOSAIC Repository:** The MOSAIC repository is cloned into a temporary directory named after a generated unique ID. The cloning process is realized using the Python package GitPython[23].
2. **Remove Non-Selected Index Partitions:** Non-selected indexes are removed from the `resources` and `lucene` directories within the cloned repository to ensure only the selected indexes are retained.
3. **Add Selected Index Partitions:** Additional indexes specified by the user are downloaded and extracted into the `resources` directory of the cloned repository.
4. **Remove Non-selected Metadata Modules:** Non-selected modules are removed from the `search-service` directory, and references to these modules are updated in the `pom.xml` files and `config.json` file.
5. **Update Front-end:** Set the title and the surrounding color in the basic front-end within the MOSAIC framework.
6. **Modify Base URL in Config:** The `config.json` file is updated with the specified base URL to configure the OpenSearch template URL.
7. **Create a Zip Archive:** A zip file of the configured instance is created and stored in the temporary directory named after the session ID. The unique instance ID and the URL to the zip file are returned, which allows users

[22]https://nextjs.org
[23]https://github.com/gitpython-developers/GitPython

to download and subsequently share their configured search engine. The cloned repository is removed after the zip file is created to free up space.

By providing a REST API, the use of MOSAIC2go is not limited to the existing web user interface, but it can also be used with other services. For instance, it is also possible to configure and download a tailored MOSAIC search engine instance using command-line tools such as *cURL*[24].

Configured MOSAIC search engine instances are stored on the file system for 24 hours until they are deleted. Furthermore, the download link of these customized instances is publicly accessible, thus they can be shared with others.

4.7.3. Visual Editor

MOSAIC2go provides visual configuration options that facilitate the customization and management of search engines. Figure 4.8 depicts how users can select from a variety of provided index partitions and metadata modules, which enables tailored configurations to meet specific requirements and preferences. Additionally, the platform allows users to personalize their search engine by specifying a title, choosing a color theme, and defining the URL for their own search service. The available index partitions and the modules are fetched from the configuration service via the API.

Upon completion of the customization process, users can download the configured search engine as an archived file. This archive is available for 24 hours, during which users can download it again or share the download link with others. The archived file contains the source code of the customized MOSAIC instance. Once extracted, the downloaded source code of the MOSAIC instance can be executed by either running the provided scripts `build.sh` and `start.sh` as outlined in Section 4.6 or by creating a Docker container utilizing the Dockerfile shipped in the archived file.

This functionality simplifies the distribution and deployment of custom search engines. Moreover, it encourages collaboration and sharing within the community. By streamlining the customization process and providing easy access to downloadable search engines, MOSAIC2go enhances the usability and accessibility of the MOSAIC framework.

[24]https://curl.se

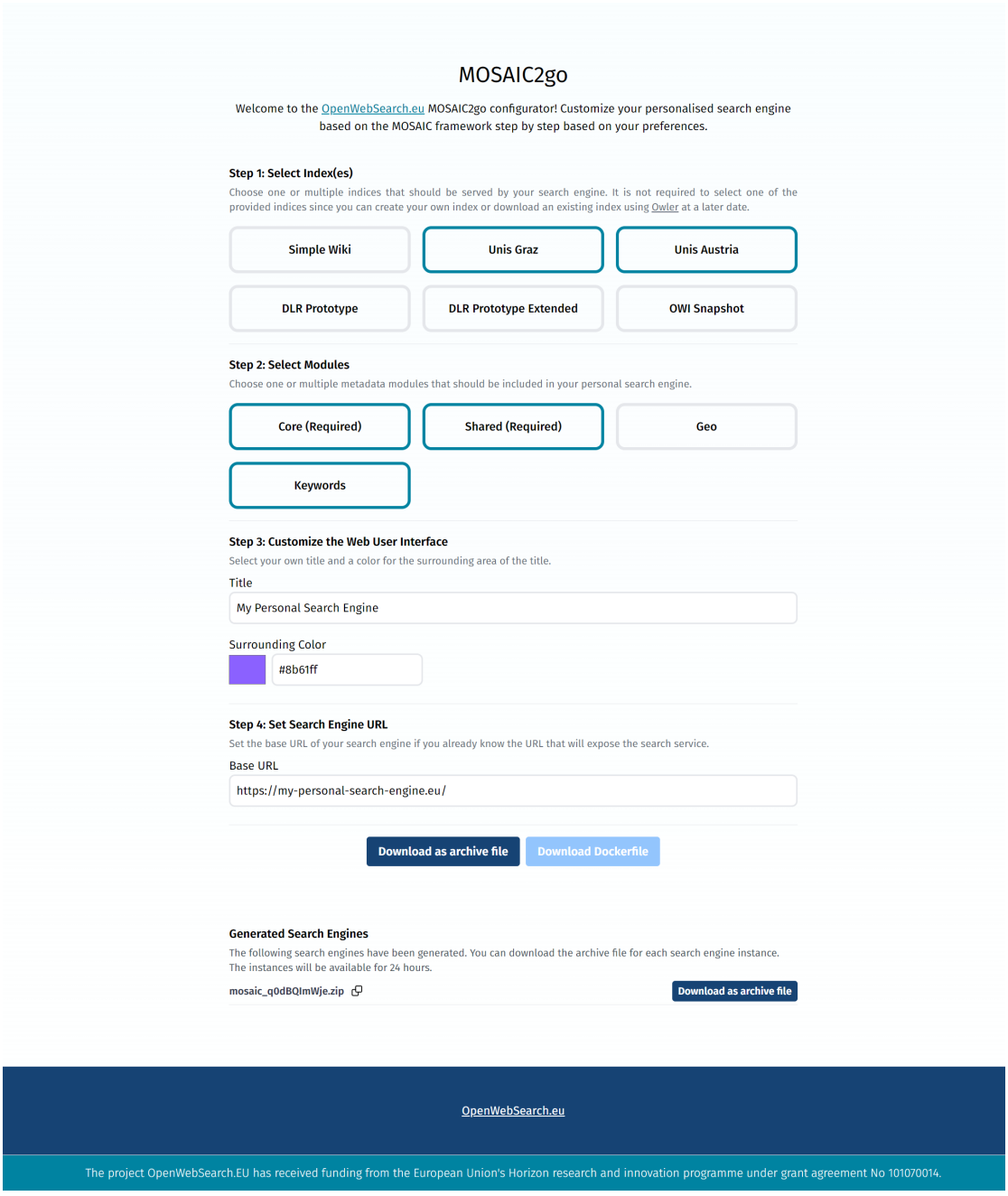


Figure 4.8.: Web user interface of MOSAIC2go, allowing users to create their own search engine by selecting or deselecting index partitions and modules

4.8. Summary

This chapter describes the development of the MOSAIC framework, particularly highlighting its modular approach and the various components involved. The MOSAIC framework is designed to leverage the OWI from the OpenWebSearch.eu initiative, in order to provide a scalable and customizable web search engine. The development phase is crucial for integrating various features and ensuring the system's overall functionality.

The core application of MOSAIC is central to its operation and encompasses components responsible for query processing, matching and ranking, metadata filtering, and enrichment. Query processing involves parsing and analyzing user queries to ensure precise search results. The matching and ranking component utilizes algorithms to retrieve and rank documents based on their relevance to the query. Metadata filtering allows for refined search results based on specific criteria, while metadata enrichment enhances the information provided in the search results.

Furthermore, the REST API serves as the primary interface for interacting with the MOSAIC framework. It offers endpoints for performing search queries, retrieving search results in both JSON and XML formats, and obtains detailed index information. This API ensures that users can effectively communicate with the search engine and retrieve relevant data as needed.

The web user interface within the MOSAIC framework provides a basic yet user-friendly platform for conducting searches. It features a search control area for query input and filtering options and displays search results in a list format, including titles, snippets, URLs, and other relevant metadata. This interface ensures that users can easily interact with the MOSAIC back-end and access the information they need.

MOSAIC's modular design allows for flexibility and customization, with metadata modules handling URL query parameters and enriching search results. Based on the modular design of the framework, MOSAIC2go was developed as a tool that enables users to create and configure their own search engine instances easily. This modular approach ensures that the system can be tailored to meet specific user requirements and integrate seamlessly into the OpenWebSearch.eu ecosystem.

Usability and accessibility are key considerations during the development of the MOSAIC framework, focusing on both novice and experienced developers. The framework can be used out of the box with Docker images or customized through direct modifications to the source code. This versatility ensures that MOSAIC is accessible to a wide range of users and can be adapted to various use cases and technical environments.

The development of the MOSAIC framework has established a comprehensive and adaptable foundation for web search engines that are based on the OWI. By integrating modular components, user interfaces, and an API, the framework

meets diverse user needs and technical requirements. This phase provides a solid groundwork for future improvements and further integration within the OpenWebSearch.eu initiative.

5. Evaluation

This chapter presents the two studies conducted to evaluate the MOSAIC framework. The initial Section 5.1 outlines the scope of these studies and depicts the evaluation goal. The subsequent Section 5.2 details the practical user study conducted at a hackathon event, where MOSAIC was a central focus, as well as the focus group discussions held with experts as described in Section 5.3. Furthermore, Section 5.4 discusses the findings from the two studies, and Section 5.5 delves into identified limitations in the evaluation of the MOSAIC framework.

5.1. Scope and Evaluation Goal

The scope of MOSAIC's evaluation encompasses two primary studies designed to assess the technical approach, the modular design and the applicability and acceptance of the system. Therefore, this chapter will significantly contribute to addressing the RQs outlined in Section 1.1 by providing valuable insights and empirical data. Focusing specifically on the strengths and weaknesses of the MOSAIC framework, a partial SWOT analysis was conducted, while the opportunities and threats aspects were not included (Mintzberg, 1998).

The first study is a user-focused evaluation conducted during a hackathon event, with the aim to gather feedback from participants with varying levels of technical expertise. This study seeks to understand how easily users can adopt, customize, and utilize the framework in real-world scenarios. The second study involves focus group discussions with experts in the field of information retrieval and web search technologies. These discussions aim to provide a deeper technical evaluation of the framework's architecture, modularity, and applicability. By combining insights from both user perspectives and expert analyses, the evaluation aims to provide a holistic assessment of MOSAIC's strengths, opportunities and areas for improvement.

5.2. Practical User Study

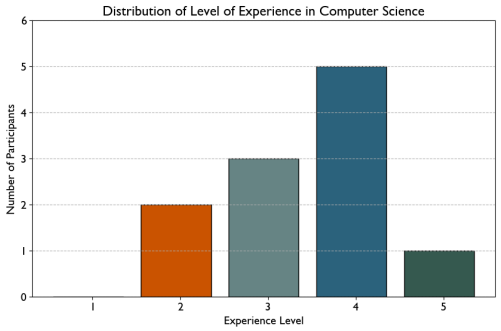
This section details the practical user study conducted to evaluate the MOSAIC framework's usability, functionality and applicability. The hackathon provided a practical setting to observe how participants interact with and implement the framework in applied scenarios. The study was carried out within a one-day

hackathon event in the context of the OpenWebSearch.eu, organized by the Cognitive and Digital Science (CoDiS) Lab, a research group of the Institute of Interactive Systems and Data Science (ISDS) at the Graz University of Technology. It took place on Friday, 24 May 2024, in the institute building in Sandgasse 36 in Graz, Austria, and online. The hackathon started at 09:00 CEST in the morning and ended at 18:00 CEST in the evening.

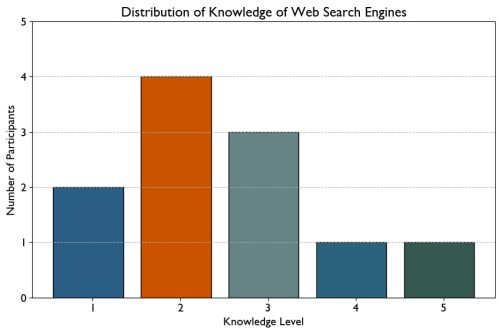
5.2.1. Participants

The practical user study for evaluating the MOSAIC framework was conducted during a hackathon event involving a total of 13 participants. Eleven participants were computer science students from Graz University of Technology (84.62%), with six enrolled in bachelor's programmes (46.15%) and five in master's programmes (38.46%). Additionally, two external participants (15.38%), both holding master's degrees in computer science, participated in the practical user study. The gender distribution among the participants included 2 women (15.38%) and 11 men (84.62%). Participants were divided into four teams, with to two to four people per team, including one team that participated online.

Of the 13 participants in total, eleven (84.62%) also completed a questionnaire at the end of the hackathon. The distributions of these participants' self-rated experience in computer science and their knowledge of web search systems are visualized in Figure 5.1a and Figure 5.1b respectively, both using a score range from 1 (less experienced) to 5 (highly experienced).



a) Distribution of level of experience in computer science



b) Distribution of knowledge of web search engines

Figure 5.1.: Practical user study participants' self-assessed levels of experience and knowledge in computer science and their knowledge of web search engines

5.2.2. Materials and Methods

The hackathon began with a presentation introducing participants to the concepts of web search, the OpenWebSearch.eu project, and the MOSAIC framework. Participants were provided with technical details and various possibilities

for utilizing MOSAIC in their applications. Throughout the event, participants had access to general explanations, documentation, and developer guides to support their development activities. Collaboration and communication were facilitated through a cloud folder for each team, a dedicated Discord[1] channel, and a Jitsi[2] meeting room for online participants.

5.2.2.1. Team Projects

The organizers suggested various development possibilities to guide participants and maximize the potential applications of the MOSAIC framework. These suggestions included:

- **Create Your Own Index Partition**: Participants could create their own index partition by moving or copying CIFF and Parquet files to the designated directory. This involved serving the files using CLI options to integrate them into the MOSAIC framework.
- **Create or Improve a Front-End**: Participants were encouraged to modify the existing front-end located in the `front-end` directory or to create a new web user interface utilizing the REST API. This allowed for enhanced customization and improved user interaction.
- **Create an Application Using MOSAIC as a Service**: This option involved integrating a new index partition into MOSAIC, modifying or creating a front-end, and accessing the REST API to use MOSAIC as a service. This approach demonstrated how MOSAIC could be utilized as a back-end service for various applications.
- **Undertake Web Data Analysis**: Participants could use existing or locally created index partitions to perform web data analysis, such as topic modeling or implementing PageRank. This showcased the analytical capabilities of the MOSAIC framework.
- **Feed a Large Language Model**: This involved using existing or locally created index partition to develop a use case and analysis goal, thereby demonstrating the integration of MOSAIC with advanced language models for enhanced information retrieval.
- **Create a New Module**: Participants were encouraged to follow the steps in the developer guide to create new modules. An example provided was using the existing, at that time unused metadata column `domain_label` in the Parquet file, allowing for further customization and extension of the framework.

[1] https://discord.com
[2] https://meet.jit.si

5.2.2.2. Questionnaire

The questionnaire used in the study was created with Tally[3] and included a range of questions designed to gather comprehensive feedback from participants. It began with demographic questions to capture basic information about the participants. Following this, the questionnaire addressed various aspects of the MOSAIC framework, including the approach, technical concept, and design. Questions were also included to assess the usability and applicability of MOSAIC, focusing on what participants were able to accomplish with the framework. Additionally, participants were asked to evaluate the strengths and weaknesses of MOSAIC and provide suggestions for improvements.

Participants were asked to rate the following five statements with 1 (Strongly Disagree) to 5 (Strongly Agree):

- I appreciate the concept of OpenWebSearch.eu to create vertical search engines.
- The technical concept and design of MOSAIC is useful to create own applications.
- The installation of MOSAIC is easy.
- Using MOSAIC in the development process is easy.
- When using and integrating MOSAIC, no problems occurred.

To assess the internal consistency of these five items, Cronbach's Alpha was used (Cronbach, 1951). The calculated value of $\alpha = 0.79$ indicates acceptable reliability. The 95% confidence interval of $[0.498, 0.935]$ further supports the reliability, which indicates a reasonable range within which the true value of α lies.

In addition, participants provided responses using predefined options to the following questions:

- What did you do with MOSAIC?
- From a developer's point of view, to what extent would you be willing to work and share the experience with MOSAIC?
- In your opinion, what are MOSAIC's strengths and/or weaknesses?
- What features do you think are missing in MOSAIC?
- What suggestions for improvement do you have?

Eventually, participants were able to share their thoughts about strengths, weaknesses, and room for improvement through open-ended responses to the following questions:

- In your opinion, what are MOSAIC's strengths and/or weaknesses?
- What features do you think are missing in MOSAIC?
- What suggestions for improvement do you have?

[3] https://tally.so

The full questionnaire used for the practical user study, including selectable answer options, can be seen in Appendix B.

5.2.3. Procedure

The hackathon began with an introductory presentation addressing general web search principles, the OpenWebSearch.eu project's approach, and the concept of the MOSAIC framework, followed by team formation and idea generation sessions. Participants were tasked with creating applications using MOSAIC or developing extensions or modules for the framework. After initial brainstorming, each team presented their ideas in a plenary session, receiving feedback and suggestions. The development phase of their projects then began, with organizers providing technical support and further discussions to refine the ideas. Throughout the day, participants engaged in collaborative problem-solving and utilized the provided materials and tools to develop their projects. The event concluded with teams presenting their final work, followed by a voting session[4] using Menti[5] to evaluate the results within this setting. Each participant was also asked to complete a questionnaire before the voting session in order to provide feedback on their experience with MOSAIC, covering aspects such as ease of installation, usability, and potential improvements. Furthermore, all participants were awarded a prize for their involvement in the hackathon. The tasks of the hackathon and the time allocated for each are detailed in Table 5.1. In general, this approach ensured a thorough evaluation of the framework from both technical and user-centric perspectives.

Task	**Allocated Time**
Group Formation	15 minutes
Idea Generation	60 minutes
Presentation of Ideas*	15 minutes
Working Session	285 minutes
Online Questionnaire	15 minutes
Presentation of Results*	45 minutes
Voting Session and Prices*	15 minutes

*Plenary session, broadcasted to online participants

Table 5.1.: Overview of hackathon tasks and allocated time for each task

[4]Each participant was allowed to award 1 (moderate) to 10 (excellent) points for the evaluation of each team

[5]https://www.mentimeter.com

5.2.4. Results

This section presents the outcomes of the hackathon event, focusing on the projects developed by the participants during the hackathon and the feedback gathered through the questionnaire.

5.2.4.1. Project Outcomes

The hackathon's project outcomes are described in the following, outlining each team's objectives, addressed problems, as well as the technical development details. In addition, Table 5.2 synthesizes the outcomes and numerous technical aspects regarding the MOSAIC framework.

Team 1: Improve Search Queries and Results Summarization with Zero-Shot LLMs[6]

- **Use Case**: Enhance user search queries and provide useful summaries to deliver interesting insights and valuable information, even if the original query was vague.
- **Addressed Problem**: Users often enter vague or poorly defined search queries. By using zero-shot LLMs, the team aimed to enhance and refine these queries and provide valuable summaries.
- **Development**:
  - A *QueryOptimizer* refines and expands user queries using a zero-shot prompt for a LLM.
  - An online demonstration version of MOSAIC searches five separate instances using the original query, the optimized query, and three sub-queries via REST API.
  - Resulting text snippets from these searches are processed by a *Summarizer* in three ways: original query results, improved query results, and combined queries.
  - Both the *QueryOptimizer* and *Summarizer* are zero-shot prompts for a LLM.
  - Technical details include using the `open-mixtral-8x7b` model via Mixtral[7] API, LangChain framework[8], and Python versions 3.9 and 3.11 with a LLM temperature of 0.7.

[6]https://github.com/vedelsbrunner/MOSAIC-LLM
[7]https://mistral.ai
[8]https://www.langchain.com

Team 2: MOSAIC-Lens[9]

- **Use Case**: Develop a mobile application that uses a camera to take a picture of an object and provides a short explanation of the object as a text summary, which is then read aloud by the phone.
- **Addressed Problem**: Providing search functionality using a picture taken with the camera of a mobile device (similar to the concept of Google Lens[10]).
- **Development**:
  - An Android[11] app developed with Flutter[12] allows users to take a photo.
  - The photo is sent to a back-end service where the image is analyzed and translated into a description using the image-to-text library xtuner[13], based on the LLaVA-LLama3-8b model[14].
  - The description is sent to MOSAIC as a search query, and the text snippets of the top three search results are used to retrieve a short summary from a summarizer.
  - A BART model[15] fine-tuned on the CNN/Daily Mail dataset[16] was used for summarization, and the text summary is translated to speech using the built-in Flutter library flutter_tts[17].
  - The back-end server uses the Express.js[18] framework.
  - Topics were extracted from the plain text column of the Parquet file using BERTopic[19] and stored in a new column in the Parquet file. A new MOSAIC module was created to filter search results based on the extracted topics.

Team 3: Next.js Front-End with MOSAIC API and OpenAI GPT-3.5[20]

- **Use Case**: Improve and enrich the front-end and user experience of MOSAIC by providing a new design and adding functionality including text summarization, translation, and re-ranking of the results.
- **Addressed Problem**: The current version of the MOSAIC front-end is in a very basic development stage. Users expect a better web user interface.

[9] https://github.com/NeXTormer/ows-hackathon-2024
[10] https://lens.google/
[11] https://android.com
[12] https://flutter.dev
[13] https://github.com/InternLM/xtuner
[14] https://llama.meta.com
[15] https://huggingface.co/facebook/bart-large-cnn
[16] https://paperswithcode.com/dataset/cnn-daily-mail-1
[17] https://pub.dev/packages/flutter_tts
[18] https://expressjs.com
[19] https://maartengr.github.io/BERTopic/index.html
[20] https://github.com/thenextmz/Hackathon-OpenWebSearch.eu

- **Development**:
  - A back-end developed to coordinate the improvement of search results, adding summaries and translations generated by GPT-3.5[21] LLM.
  - The original MOSAIC back-end was extended to add the possibility of re-ranking the search results by date.
  - The results are displayed by a front-end created with Next.js.
  - The implementation was demonstrated at the end of the hackathon.

Team 4: OWS X Enhanced[22]

- **Use Case**: Enhance the search user interface by providing summaries of retrieved search results and summarizing the most prominent narratives present in the search results using LLM capabilities.
- **Addressed Problem**: Currently, the MOSAIC front-end only offers "common" web search. The proposed solution offers a new way for users to interact and explore information from retrieved search results.
- **Development**:
  - Retrieved a set of web documents for a specific user query through the MOSAIC framework.
  - Pre-processed the retrieved documents by clustering them into similar narratives and topics (topic modeling) and then passing them into a LLM chain.
  - The goal of the LLM chain is to create summaries of retrieved documents or topic clusters using LangChain, which includes different steps of text processing and prompt engineering.
  - Due to time constraints and integration problems, the implementation could not be demonstrated.
  - Initial proposal for LLMs included Llama 2 and Mistral 7B.

5.2.4.2. Questionnaire Analysis

The analysis of the questionnaire responses involved both statistical and qualitative methods. A statistical analysis was performed to quantify the responses to closed-ended questions, thereby providing insights into overall trends and patterns. Additionally, a qualitative analysis of the free-text answers was conducted to capture more detailed and nuanced feedback, thus common themes and specific suggestions from the participants could be identified.

Figure 5.2 illustrates the distribution of participant ratings for the five statements outlined in Subsection 5.2.2 of the MOSAIC framework. In addition, Table 5.3 summarizes various aspects of MOSAIC asked in the questionnaire.

[21] https://platform.openai.com/docs/models
[22] https://github.com/johndolier/OWS-Hackathlon

	Team 1 (online)	**Team 2**	**Team 3**	**Team 4**
Index Partition	used existing ones	updated with columns on topics (AI)	used existing ones	used existing ones
Query	improvement, expansion (LLM)	image recognition (LLM)	unmodified	unmodified
MOSAIC Search Service	unmodified	created new module for topic filtering and enrichmemnt	updated re-ranking	unmodified
Search Result	summarization (LLM)	summarization (llm) and text-to-speech (TTS)	summarization (LLM), translation (LLM)	summarization (LLM), classification
Web User Interface	unmodified	Android application (Flutter) with camera (image recognition) and TTS functionality	new front-end (Next.js)	concept of visualization
Outputs	concept, implementation, demonstrator	concept, implementation, demonstrator	concept, implementation, demonstrator	concept, partial implementation
Voting	8.4 points (2^{nd} place)	9.3 points (1^{st} place)	7.2 points (3^{rd} place)	6.3 points (4^{th} place)

Table 5.2.: Summary of the projects developed by participants during the hackathon, highlighting the diverse applications and enhancements made to the MOSAIC framework

Aspect	n	$\bar{x}$	s
I appreciate the concept of OpenWebSearch.eu to create vertical search engines.	11	4.73	0.47
The technical concept and design of MOSAIC is useful to create own applications.	11	4.64	0.50
The installation of MOSAIC is easy.	11	4.00	1.10
Using MOSAIC in the development process is easy.	9	4.56	0.53
When using and integrating MOSAIC, no problems occurred.	10	3.90	0.99

Table 5.3.: Statistical analysis of various questionnaire aspects of MOSAIC, sample size (n), mean ($\bar{x}$), and standard deviation (s)

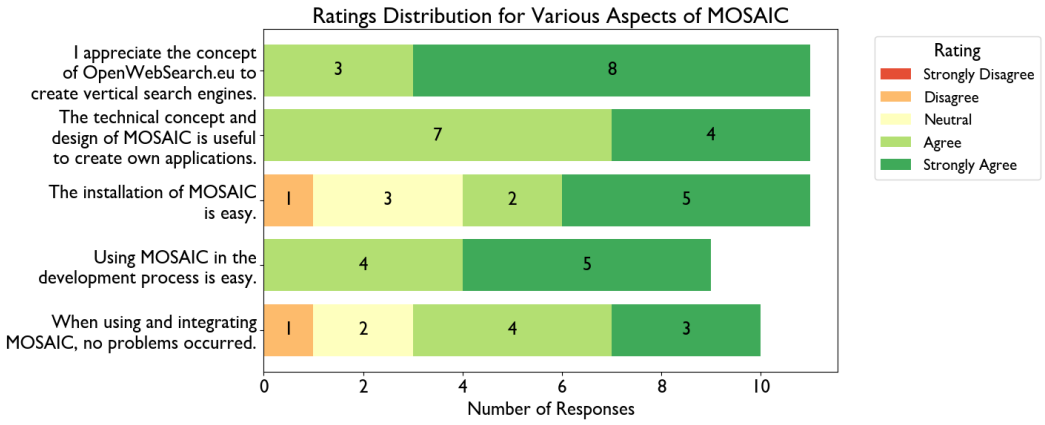


Figure 5.2.: Practical user study participants' response on different aspects of MOSAIC

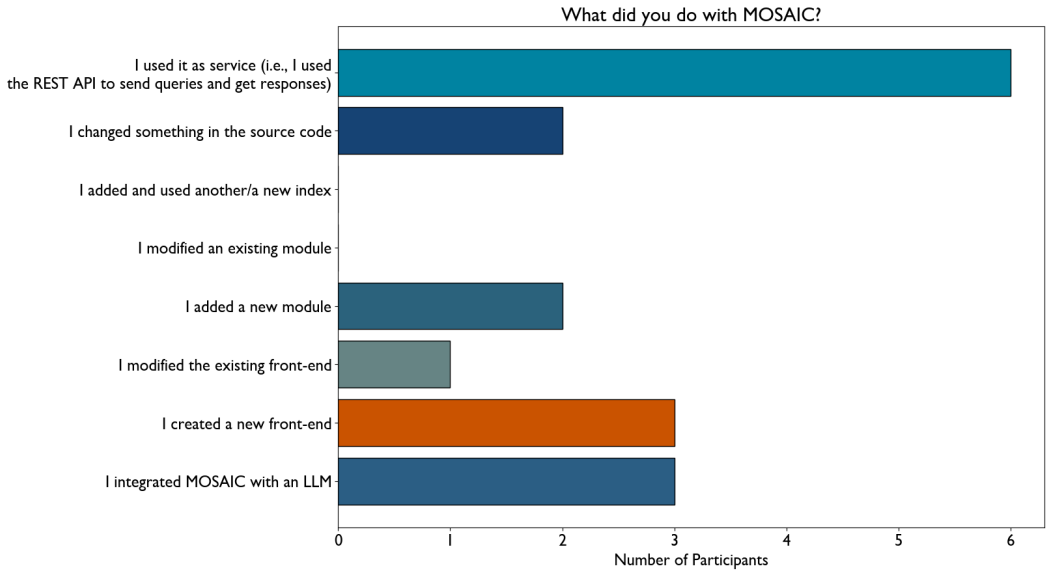


Figure 5.3.: Actions taken by practical user study participants with MOSAIC

Each bar represents the number of responses in different rating categories: *Strongly Disagree*, *Disagree*, *Neutral*, *Agree*, and *Strongly Agree*. The majority of participants rated the concepts of OpenWebSearch.eu and MOSAIC highly, with a substantial number indicating *Agree* and *Strongly Agree*. However, there was a more varied response for the ease of installation and problem-free usage, with some participants indicating *Disagree* and *Neutral*.

Furthermore, Figure 5.3 depicts the distribution of various actions taken by participants while working with the MOSAIC system. The participants had the option to select multiple actions they performed. The histogram illustrates that the most common action was using MOSAIC as a service, followed by creating a new front-end and integrating MOSAIC with a LLM.

The distribution of developers' willingness to work with and share their experiences with MOSAIC is highlighted in Figure 5.4. The participants were asked to indicate their level of willingness across several contexts and could select multiple options.

The options included using MOSAIC in a private or non-professional environment, using it in a professional environment, finding MOSAIC helpful for current or future tasks, willingness to modify or improve MOSAIC in the future, and recommending MOSAIC to other developers. The chart shows that the highest willingness is for modifying or improving MOSAIC in the future, followed by using it in a professional environment and recommending it to other developers.

Table 5.4 summarizes the qualitative analysis of MOSAIC, focusing on strengths, weaknesses, and areas for improvement as reported by participants. The strengths of MOSAIC include its quick and straightforward installation and

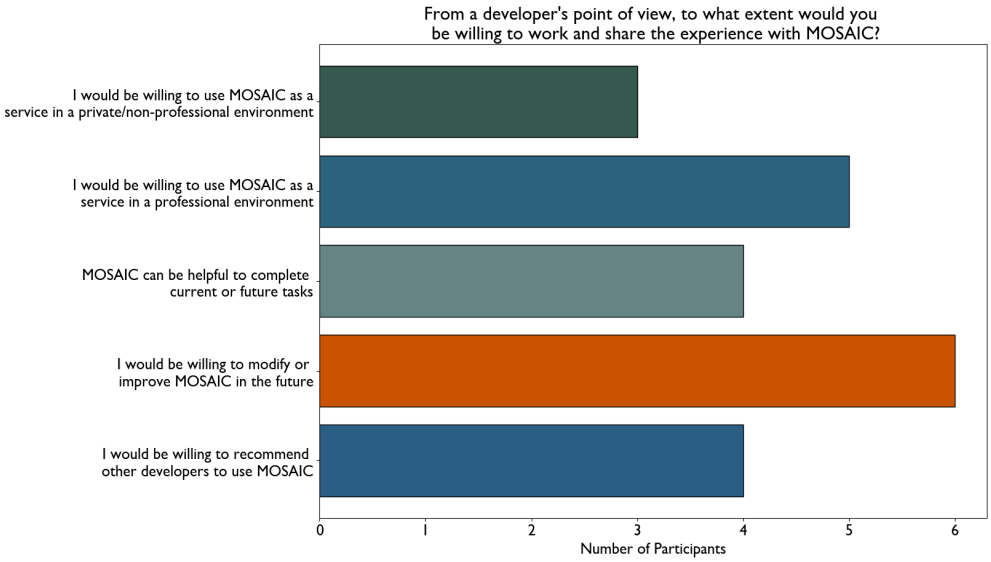


Figure 5.4.: Willingness of practical user study participants to work and share experience with MOSAIC

Category	**Observations**
Strengths	• Easy and fast installation and development • Flexibility in integration with other systems and application scenarios • Efficient and transparent search capabilities (i.e., robust API for querying and receiving responses) • Modular architecture
Weaknesses	• Limited availability of data (i.e., index partitions) • Limited performance of online demonstration version • Some components and features are not fully documented
Room for Improvement	• Proper ranking and re-ranking capabilities • Enhanced web user interface • Direct integration of LLMs and RAG features • Improved documentation for advanced features • Pagination feature

Table 5.4.: Qualitative analysis of MOSAIC's strengths, weaknesses and room for improvement, based on practical user study participant feedback

development process, flexibility in system integration, efficient and transparent search capabilities, and modular architecture. Identified weaknesses involve limited data availability, performance issues with the online demo version, and incomplete documentation for some features. Participants suggested improvements such as better ranking and re-ranking capabilities, an enhanced web user interface, integration of LLMs and RAG features, and the addition of a pagination feature.

5.3. Focus Groups

Following the practical user study conducted at the hackathon event, focus group discussions were held to gain deeper insights into the perception and assessment of the MOSAIC framework. These discussions aimed to capture the perspectives of experts and gather detailed feedback on various aspects of the system. The following section details the methodology, participants, and key findings from these focus group sessions.

5.3.1. Participants

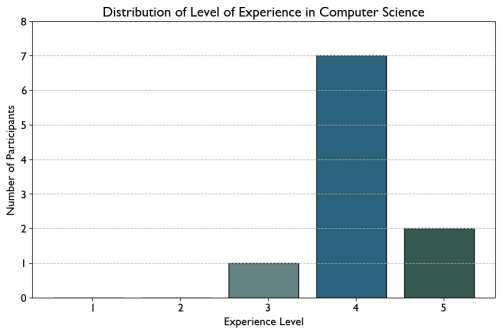
All experts who participated in the focus group discussions were members of the OpenWebSearch.eu consortium. The group comprised a diverse set of experts, including three representatives from Radboud University (30.00%), one from the University of Passau (10.00%), three from the German Aerospace Center (DLR) (30.00%), one from CERN (10.00%), and two from the Webis research group (20.00%). Among the ten participants, seven held a master's degree (70.00%) and three had a doctorate (30.00%). Four participants were PhD students (40.00%), four identified their profession as researchers (30.00%), and three were professors (30.00%). The gender distribution among the experts that participated in the focus group discussions included 3 women (30.00%) and 7 men (70.00%). This diverse representation fostered a multifaceted discussion by incorporating the varied expertise and perspectives of the consortium's members.

The distributions of these participants' self-rated experience in computer science and their knowledge of web search systems are visualized in Figure 5.5a and Figure 5.5b, respectively, both using a score range from 1 (less experienced) to 5 (highly experienced).

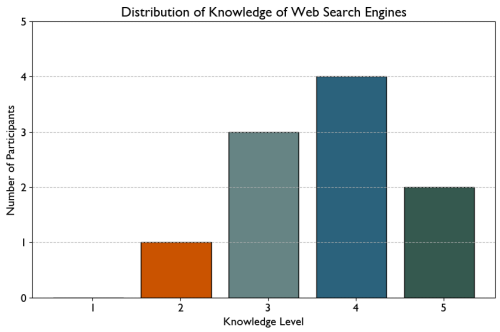
5.3.2. Materials and Methods

The focus group discussions were conducted online using BigBlueButton (BBB) meeting rooms[23]. Each partner from the OpenWebSearch.eu consortium was

[23]https://bigbluebutton.org



a) Distribution of level of experience in computer science



b) Distribution of knowledge of web search engines

Figure 5.5.: Focus group participants' self-assessed levels of experience and knowledge in computer science and their knowledge of web search engines

able to select a 30-minute time slot during which their members could participate. Table 5.5 outlines the specific questions used to initiate dialogue during the focus group discussions, categorized by technical approach, modular architecture, and applicability of MOSAIC.

Category	**Questions**
Technical Approach	What is your opinion on the overall technical approach of MOSAIC used as a web search engine? What features do you expect from a web search engine and what can be improved?
Modular Architecture	What is your opinion of the modular approach of MOSAIC? How good is this approach compared to existing IR and web search frameworks?
Applicability	How suitable is MOSAIC for creating your own search application? Would you recommend MOSAIC to others for creating an own search application?

Table 5.5.: Focus group discussion questions

Audio recordings of the sessions were made with the participants' consent and were promptly deleted after the analysis was completed. Additionally, participants were asked to complete a Tally questionnaire after the respective discussion to provide quantifiable responses. In the questionnaire, participants provided ratings for the following six statements using scores from 1 (Strongly Disagree) to 5 (Strongly Agree):

1. The overall technical concept of MOSAIC makes sense regarding the

development of search engines.
2. MOSAIC includes features that I would expect from a web search engine.
3. The modular approach of MOSAIC is sensible.
4. Compared to existing IR and web search systems, the modular approach of MOSAIC is adequate.
5. MOSAIC is suitable for creating own search applications.
6. I would recommend others to use MOSAIC as a basis for developing their own search engine.

Cronbach's Alpha for the questionnaire statements 1 & 2, 3 & 4, and 5 & 6 were calculated. The Cronbach's Alpha values for the focus group discussion's measures varied significantly. The first value, for statements 1 & 2 , with $\alpha = 0.8621$ and a 95% confidence interval of $[0.445, 0.966]$, indicates a high level of internal consistency, although some variability is present across different samples. The second value, for statements 3 & 4, with $\alpha = 0.4483$ and a wide 95% confidence interval of $[-1.221, 0.863]$, suggests questionable internal consistency and high uncertainty, and indicates potential issues with the reliability of this measure. The third value, for measures 5 & 6, with $\alpha = 0.9437$ and a 95% confidence interval of $[0.773, 0.986]$, demonstrates excellent internal consistency and strong reliability, with minimal variability across samples.

5.3.3. Procedure

An email was sent to OpenWebSearch.eu project partners inviting them to select a time slot for the focus group discussion. At the beginning of the online focus group discussion, experts were informed that the discussion was part of the evaluation of MOSAIC, alongside an already conducted practical user study. They were briefed that the discussion would cover three categories within a 30-minute session, with approximately 10 minutes dedicated to each category. Participants were also asked for their consent to audio record the session. The moderators then posed the questions, and notes were taken throughout the discussion. At the conclusion of the 30-minute session, participants were requested to complete an online questionnaire. Table 5.6 outlines an overview of the allocated time for each focus group discussion topic and the online questionnaire. The audio recordings were deleted after the qualitative analysis was completed.

5.3.4. Results

The results of the focus group discussion encompass a qualitative analysis of the experiences, insights, and feedback provided by experts during the sessions, as well as an analysis of the responses collected from the subsequent questionnaire.

Activity	Allocated Time
Discussion about the technical approach of MOSAIC	10 minutes
Discussion about the the modular architecture of MOSAIC	10 minutes
Discussion about the applicability of MOSAIC	10 minutes
Online Questionnaire	5 minutes

Table 5.6.: Overview of focus group discussion activities and allocated time for each task

5.3.4.1. Qualitative Analysis

The focus group discussions highlighted several key strengths, weaknesses, and areas for improvement of the MOSAIC framework, as summarized in Table 5.7. Participants appreciated the core functionality and modular approach of MOSAIC, noting its effective integration of index partitions and compatibility with the OpenWebSearch.eu project. The framework's ease of use and suitability for prototyping were also emphasized. However, several weaknesses were identified, including limitations in advanced index management, potential complexity, and dependency on the CIFF format. The web user interface and ranking system also required enhancements to meet user expectations. To address these issues, participants suggested improvements such as better index partition management, simplified examples, enhanced modular functionality, and more advanced search features. Additionally, they recommended extending metadata handling capabilities, providing detailed documentation and blueprints, implementing automatic updates, and developing methods for cross-index ranking.

5.3.4.2. Questionnaire Analysis

Following the focus group discussion, a questionnaire was sent to the participants and subsequently analyzed statistically. Figure 5.6 illustrates the distribution of scores for various evaluation items related to the MOSAIC framework, as assessed by experts after the focus group discussion. The items evaluated include the overall technical concept, the inclusion of expected features, the sensibility and appropriateness of the modular approach compared to existing IR and web search systems, the suitability for creating custom search applications, and the likelihood of recommending MOSAIC to others. The responses are categorized into five levels: *Strongly Disagree*, *Disagree*, *Neutral*, *Agree*, and *Strongly Agree*.

Category	Observations
Strengths	• MOSAIC effectively serves as a basic search engine, functioning as expected • The plugin functionality and the ability to add or not use modules/components are highly valued • Easy integration of index partitions and compatibility with the OpenWebSearch.eu project • Particularly useful for prototyping and low-code development • Simplifies the creation of custom search engines, including providing a nice installation process • The framework offers satisfactory performance and speed • Potential for a community-driven approach where others can add and share modules
Weaknesses	• Lacks support for index partition updates, additions, and removals • Risk of becoming too complex; currently suitable more as a demonstrator than a production system • Currently, rank results are separately for each index partition rather than a combined ranking • Dependency on CIFF format, since CIFF is not widely known as an index exchange format • The web user interface could be improved to show more detailed search results and match the user's expectations • More suitable for specific use cases using OWI schema (CIFF + Parquet), not for broader applications.
Room for Improvement	• Need to effectively combine CIFF files and support various metadata versions. • Ensure modules and components can be accessed and added easily via APIs • Implement advanced search features such as bounding boxes in the web user interface and more filters • Implement features for automatic downloading and updating of modules and index partitions • Offer blueprints and more detailed documentation for creating custom modules and components • Develop methods for combined ranking across multiple index partitions and address duplication issues

Table 5.7.: Qualitative analysis of MOSAIC's strengths, weaknesses and room for improvement, based on focus group discussions

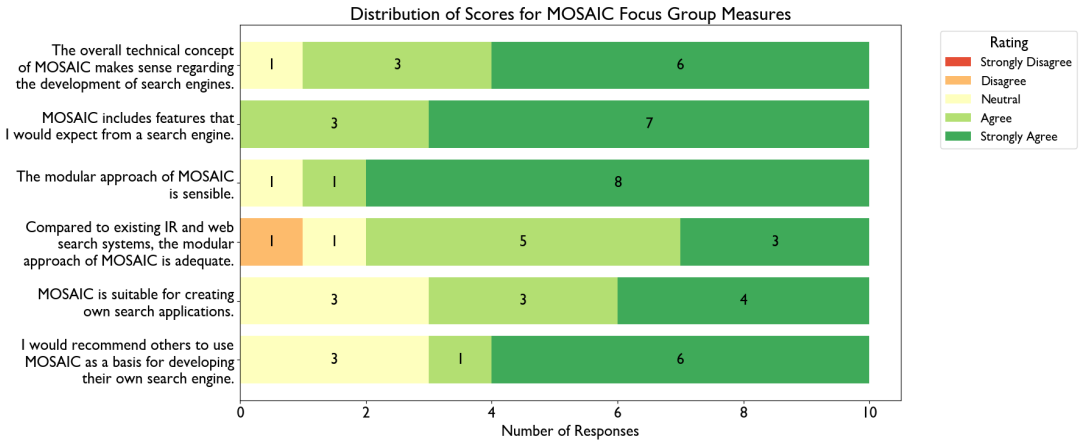


Figure 5.6.: Distribution of ratings for the focus group evaluation measures of MOSAIC's technical concept, modular architecture and applicability

The majority of participants rated the technical concept and expected features (items 1 and 2) positively, with most responses falling under the *Agree* and *Strongly Agree* categories. Items 3 and 4, which focus on the modular approach and its adequacy compared to existing systems, also received high ratings, indicating strong support for MOSAIC's modular framework. Items 5 and 6, which assess the suitability of MOSAIC for creating custom search applications and the likelihood of recommending it to others, were also rated favorably, reflecting participants' confidence in MOSAIC's practical applicability and utility. Table 5.8 provides a statistical summary of these six evaluation items.

5.4. Discussion

This discussion aims to address the three RQs posed in this thesis by utilizing insights from both the practical user study and the focus group discussions.

5.4.1. Technical Approach

> *RQ1: How can partitions of the OWI be effectively integrated and utilized to enhance web search engine capabilities?*

The integration and utilization of MOSAIC partitions were central to the MOSAIC framework's development. Through the dual use of CIFF files and Parquet files, MOSAIC demonstrated a straightforwad and effective method of importing and managing OpenWebSearch.eu index data. This approach was evaluated through a practical user study at a hackathon event, where participants were able to integrate index partitions seamlessly and use them

Item	n	$\bar{x}$	s
The overall technical concept of MOSAIC makes sense regarding the development of search engines.	10	4.50	0.71
MOSAIC includes features that I would expect from a search engine.	10	4.70	0.48
The modular approach of MOSAIC is sensible.	10	4.70	0.67
Compared to existing IR and web search systems, the modular approach of MOSAIC is adequate.	10	4.10	0.74
MOSAIC is suitable for creating own search applications.	10	4.10	0.88
I would recommend others to use MOSAIC as a basis for developing their own search engine.	10	4.30	0.95

Table 5.8.: Statistical analysis of evaluation items rated by experts in a questionnaire after the focus group discussions, sample size (n), mean ($\bar{x}$), and standard deviation (s)

for various search and analysis tasks. The integration process, supported by scripts for importing CIFF files into Lucene and Parquet files into DuckDB, showed that OWI partitions could be efficiently managed and queried within the MOSAIC framework. The results from the hackathon indicated that users found the system capable of handling comprehensive index data, thus the potential to enhance web search engine capabilities, especially for vertical search engines in demonstration and academic settings. Furthermore, the practical user study highlighted that the framework's ability to handle datasets in different sizes without significant performance degradation was crucial for practical applications.

Additionally, experts in the focus group discussions noted that the dual approach of using both CIFF and Parquet files provided flexibility in managing metadata and full-text search capabilities. They suggested that offering different approaches, such as using only CIFF where metadata is included, relying solely on DuckDB due to its support for full-text search, or even integrating alternative methods such as Solr, could further enhance the framework's adaptability. The experts appreciated the ease of integrating OWI partitions and recognized this flexibility as a significant advantage for various use cases and data management requirements. They also mentioned that a few more concrete examples showcasing the integration process would be beneficial. The core search func-

tionality was praised for its effectiveness, indicating that the integration of OWI partitions enhances search engine capabilities. Experts suggested exploring options in the direction of conversational search to further utilize these partitions effectively. Additionally, they emphasized the importance of considering appropriate ranking and re-ranking mechanisms in future developments, which would further refine the search results and maximize the benefits of utilizing OWI partitions.

The integration of OWI partitions through MOSAIC has demonstrated an enhancement in web search engine capabilities by providing scalable and flexible indexing solutions. The feedback from both the user study and focus group discussions confirms that MOSAIC effectively utilizes these partitions, thereby addressing diverse search and retrieval needs while also presenting opportunities for further improvements in areas such as the integration of generative AI features including conversational search and common features including proper ranking mechanisms.

5.4.2. Modular Design

RQ2: What are the advantages and limitations of incorporating a modular web search engine architecture, and how does it affect the system's flexibility and scalability?

The modular architecture of the MOSAIC framework provides significant advantages in terms of flexibility and scalability, as highlighted in both the practical user study and focus group discussions. This design supports the seamless integration of various components, such as metadata modules and optional system components, which can be enabled or disabled according to specific requirements. Both the practical user study and focus group discussions indicated that this modular approach enhances customization and adaptability to diverse use cases. Furthermore, the MOSAIC2go tool exemplifies this flexibility by allowing users to create personalized vertical search engines through the selection of specific modules and index partitions.

However, the focus group discussions also highlighted certain limitations. Experts pointed out that while the modular approach is beneficial, it requires careful management of dependencies and consistent updates to ensure compatibility across modules and metadata schema versions. Additionally, the potential complexity of configuring and maintaining a modular system could be a barrier for users with limited technical expertise. The discussions emphasized that providing more concrete examples such as blueprints and detailed documentation would help mitigate these challenges and make the modular components more accessible to a wider range of users.

The hackathon supported these findings, with participants noting the ease

of adding or removing modules to suit specific needs, thus demonstrating the practical benefits of a modular architecture. Additionally, this is underlined by the fact that two participants in the practical user study added a new module, and three integrated MOSAIC with a LLM. However, they also encountered challenges related to managing the dependencies between modules and ensuring smooth integration. Despite these challenges, the participants in the practical user study and in the focus group discussions appreciated the modularity for prototyping and low-code development, which allowed for rapid iteration and testing of new features.

Regarding scalability, the modular design of MOSAIC supports extensive customization while maintaining performance. Users can tailor the system by enabling or disabling specific modules, allowing the framework to scale according to their needs. This flexibility ensures that MOSAIC can adapt to various applications and user requirements by providing a scalable solution for developing custom web search engines.

Despite some challenges, the modular architecture of MOSAIC offers significant benefits in terms of flexibility, customization, and scalability. Subsequently, it can be considered as a valuable framework for building tailored web search engines. Addressing the identified limitations through improved examples, documentation and user support can enhance its accessibility and effectiveness.

5.4.3. Usability and Applicability

> *RQ3: How applicable and user-friendly is a modular framework that utilizes partitions of the OWI for developing and deploying custom web search engines tailored to specific domains or needs?*

Participants in the hackathon event found the framework to be highly applicable for creating custom web search engines tailored to specific domains. The ability to create, select and integrate specific index partitions and modules allowed users to build search engines that met their unique requirements. The MOSAIC2go tool demonstrated how the framework could be used to easily create and deploy a personalised search engines, even for users with minimal technical expertise. This tool facilitated the customization process by providing a straightforward web user interface for selecting desired features, thus lowering the barrier to entry.

The focus group discussions reinforced these findings, with experts noting that the framework's user-friendly architectural design and comprehensive documentation made it accessible for both novice and experienced developers. They highlighted that MOSAIC's modular design allows for extensive customization without requiring in-depth technical knowledge, thereby making it suitable for a wide range of users. Experts also suggested that the inclusion of more

intuitive interfaces and additional support resources could further enhance usability.

The feedback from participants in the practical user study and focus group discussions indicated a strong willingness to recommend the MOSAIC framework to others. Many participants expressed positive experiences with MOSAIC, particularly highlighting its flexibility and ease of use. They appreciated the ability to tailor the framework to specific needs and found the process of creating custom search engines straightforward and efficient. This positive reception was reflected in their willingness to share their experiences with colleagues and peers.

Additionally, the feedback from practical user study participants emphasized that MOSAIC's straightforward installation process and clear instructions contributed to its user-friendliness. Focus group discussion participants highlighted the framework's potential in academic settings for teaching and research purposes, further demonstrating its wide applicability. Despite these strengths, there were suggestions for improving user support and simplifying the configuration process to enhance the overall user experience. Moreover, experts reported that they do not think that MOSAIC should be applied in production environments.

The MOSAIC framework's modular approach and user-friendly design demonstrate its applicability for developing and deploying custom web search engines tailored to specific domains or needs. The framework is accessible to users with varying levels of technical expertise. Implementing suggestions for enhanced user support and simplified configuration will further improve its usability and effectiveness.

5.5. Limitations

The evaluation of the MOSAIC framework, while thorough, has several limitations that must be acknowledged. These limitations identify areas where future research can enhance the assessment to provide a more comprehensive and accurate understanding of the framework's capabilities.

The relatively small sample size, with 13 participants in the practical user study and 10 experts in the focus group discussions, is a notable limitation of the evaluation. While these numbers provided valuable insights, a larger sample size would offer a more comprehensive and statistically significant assessment of the MOSAIC framework. Increasing the number of participants in future studies would enhance the reliability of the findings and ensure that a wider range of user experiences and expert opinions are considered.

Additionally, all focus group participants were partners in the research project. Consequently, their feedback might be more favorable or less critical compared to external, independent evaluators. To mitigate this, future studies should

include a more diverse group of participants, thereby encompassing a broader range of perspectives from outside the project consortium. This approach would provide a more balanced and comprehensive assessment of the framework's strengths and areas for improvement.

Another limitation is that participants in both the practical user study and the focus groups used the index partitions provided to them rather than their own data or data they were specifically interested in. This limitation may have affected their ability to fully evaluate the framework's capabilities and relevance to their unique needs. Future studies should allow participants to use their own datasets to provide a more thorough and personalized evaluation of the MOSAIC framework.

Moreover, the evaluation was conducted within a controlled environment, such as a hackathon event, which might not fully capture typical usage conditions. Participants were aware that they were part of a study, which could have influenced their behavior and feedback. Conducting evaluations in more varied and naturalistic settings would help in understanding how the framework performs under typical user conditions.

5.6. Summary

This chapter encompasses the evaluation of the MOSAIC framework through two distinct studies: a practical user study and focus group discussions. The practical user study was conducted within the context of a hackathon event, where participants actively engaged with the MOSAIC framework to develop custom search applications. The focus group discussions involved experts in the field to provide a deeper analysis and critical insights into the framework's design, functionality, and potential improvements.

The scope of these evaluations aimed to comprehensively assess the MOSAIC framework's technical approach, modularity and applicability. By addressing specific research questions, the studies should gather qualitative and quantitative data that would inform the overall effectiveness and user satisfaction with the framework. The evaluations also intended to identify areas for enhancement and potential challenges that users might face when interacting with the system.

The practical user study engaged participants of varying technical backgrounds, who utilized the MOSAIC framework during a hackathon event in the context of the OpenWebSearch.eu project. This study focused on user experiences, system performance, and the practical application of the framework in different scenarios. Participants were tasked with developing projects using MOSAIC, which allowed for a hands-on assessment of its capabilities and limitations. The focus groups comprised experts who provided critical feedback on the MOSAIC framework. These discussions highlighted strengths, weaknesses, and opportunities for improvement. Therefore, they offered a comprehensive

understanding of the framework's applicability in personal, academic and professional settings. In addition, the expert evaluations were instrumental in validating the technical approach and modular design of MOSAIC.

The discussion of the findings addresses the RQs posed in this thesis, thereby integrating insights from both the practical user study and focus group discussions. For *RQ1*, which investigates the technical approach to utilizing OWI partitions, feedback indicated that the dual approach of CIFF and Parquet files is beneficial for flexibility and performance. Experts suggested additional examples showcasing different configurations and pointed out the core search functionality's effectiveness. *RQ2* explores the modular aspect of MOSAIC. Both the hackathon participants and focus group experts highlighted the framework's flexibility and customization capabilities. However, they also noted challenges such as the need for better documentation and user support to facilitate easier integration and usage. *RQ3* examines the applicability and usability of the framework for developing custom search engines. The practical user study demonstrated a high level of user satisfaction, with participants successfully implementing new modules and integrating MOSAIC with LLMs. The focus groups supported these findings and outlined the framework's potential for wide applicability and stressing the need for ongoing enhancements to improve the user experience.

The evaluation of the MOSAIC framework has several limitations that must be acknowledged. The relatively small sample size, with 13 participants in the practical user study and 10 experts in the focus group discussions, limits the comprehensiveness and statistical significance of the findings. Additionally, all focus group participants were partners in the research project, which may have influenced their feedback. Including a more diverse group of participants in future studies would provide a more balanced assessment. Furthermore, the evaluation was conducted within a controlled environment, which might not fully capture typical usage conditions. Future studies should allow participants to use their datasets and conduct evaluations in varied settings to provide a more thorough and personalized assessment.

6. Lessons Learned

This chapter shares the significant learnings and perspectives gained across the design and development phases of the MOSAIC framework, as well as during the composition of this thesis. It examines crucial insights from the review of literature and the processes of development and evaluation. Additionally, personal reflections and observations are included.

6.1. Literature

The landscape of web search engines is vast and encompasses a multitude of platforms tailored for diverse purposes. From general search engines such as Google and Bing to specialized ones such as Google Scholar and PubMed[1], each serves a unique function. Despite this variety, it remains challenging to find an open-source search engine framework that encourages the utilization of an openly available index, one that is feasible for covering substantial portions of the web. This gap highlights the need for a solution that not only democratizes access to web data but also provides the technical robustness required to handle at least parts of the web's expansive and dynamic nature. Nevertheless, creating a web search engine architecture from the ground up would be redundant, as the fundamental structures and methodologies are already well-established and have been extensively studied.

One of the core insights from the literature is the importance of a robust technical pipeline for efficient data handling. This involves the use of advanced indexing techniques, such as the combination of CIFF and Parquet files, to manage large-scale datasets effectively. The dual approach allows for flexibility and performance optimization, which are crucial for handling the diverse and extensive datasets typical of modern web search engines. Integrating these formats into a modular architecture, as seen in the MOSAIC framework, enhances scalability and customizability, which allows developers to tailor the system to specific needs and data sources.

In the modern era, the integration of AI, particularly generative AI, has become a crucial consideration in the design of web search engines. As outlined in Chapter 2, traditional IR systems have evolved significantly with the emergence of neural networks and deep learning models. These advancements enhance the semantic understanding of queries and documents, thereby improving the

[1] https://pubmed.ncbi.nlm.nih.gov

relevance and accuracy of search results. Generative AI, such as LLMs, offers capabilities beyond pure retrieval. The literature particularly highlights the potential of LLMs to improve query understanding, provide semantic search capabilities, and even generate summaries of search results, which enrich the user experience. Although the MOSAIC framework does not leverage these technologies directly, this was considered in the design of the framework as it provides easily accessible entry points and interfaces to integrate such advanced AI technologies.

Moreover, the modular design of the MOSAIC framework, as mainly derived from the literature, allows for seamless integration of various components, including LLMs, through well-defined APIs and other interfaces. This architecture ensures that new features can be added or existing ones can be modified without disrupting the overall system. The literature underscores that such a design is essential for maintaining flexibility and adaptability in a rapidly evolving technological landscape.

6.2. Development

The development phase of the MOSAIC framework provided several critical insights into the challenges and strategies associated with building a scalable and modular web search engine. One key lesson was the importance of adopting a robust yet flexible architecture that could accommodate various components and functionalities without compromising performance. The use of a modular design allowed for the straightforward integration of new features and improvements, which facilitates ongoing development and innovation.

Another significant lesson learned during the development was the necessity of optimizing the codebase when processing large-scale datasets typical of web search engines. Given the substantial volumes of data involved, even minor inefficiencies can lead to significant performance issues. Performance was continually improved throughout the development phase, thanks to regular benchmarking and the valuable input of experts. Their insights were instrumental in identifying bottlenecks and optimizing critical components of the system.

Flexibility to adapt to changes is essential when utilizing data from an ongoing research project. Research projects are dynamic and often involve updates to methodologies, data sets, and findings. This necessitates a development approach that can quickly integrate new information and adjust to evolving requirements. The MOSAIC framework's modular design supported this need, which allows for updates and modifications without disrupting the overall system. By remaining adaptable and responsive to changes, the framework could continuously utilize the most current and relevant data.

6.3. Evaluation

The practical user study provided valuable insights into the usability and effectiveness of the MOSAIC framework. Participants from diverse backgrounds interacted with the system, thus offering a broad perspective on its strengths and areas for improvement. One key lesson was the importance of a user-friendly web interface. Users consistently highlighted the need for intuitive navigation and clear visual cues, which are crucial for enhancing the overall user experience. Feedback from the study emphasized the necessity of simplifying complex features and providing comprehensive help resources to support users with varying levels of technical expertise. This insight underscored the importance of prioritizing user-centric design principles in future iterations of the framework. Additionally, the project outcomes from the hackathon event were impressive, with all participants successfully integrating MOSAIC with AI technologies. This demonstrated the framework's versatility and potential for innovative applications.

The experts in the focus group discussions provided a detailed evaluation of the technical aspects of the MOSAIC framework, thereby complementing the findings from the practical user study. Experts recognized the strengths of the modular design and the easy integration of various components, which enhanced flexibility and scalability. However, they identified a lack of index management options, such as index updates, additions, or removals within index partitions, as well as for merging index partitions. Additionally, they suggested improvements in more detailed guidelines and concrete examples to facilitate developer use and adoption. The acceptance of the system by experts, indicating their willingness to use it in their work, reflects confidence in the framework's design and functionality.

6.4. Personal

Collaborating closely with experts throughout the development of the MOSAIC framework was highly valuable. The insights gained from their extensive knowledge and experience provided a deeper understanding of the complexities involved in creating a web search engine framework. This collaboration not only enhanced the technical aspects of the project but also contributed significantly to understanding the standards and methodologies involved in advanced research.

Discussing web search technologies with real technical and non-technical end users was also very insightful. These interactions highlighted the practical needs and challenges faced by users, thereby providing a grounded perspective that is often missing in purely theoretical work. Listening to their feedback and observing how they interacted with the system emphasized the importance of

user-centric design and the need for intuitive, accessible interfaces.

7. Conclusion and Future Work

In this chapter, the thesis is concluded by summarizing the efforts made in developing and evaluating the MOSAIC framework as a contribution to web search technologies using index partitions from the OWI. Furthermore, it will discuss potential future work aimed at enhancing and extending the capabilities of the MOSAIC framework.

7.1. Conclusion

Web search engines play a crucial role in modern society as the primary means for retrieving information from the vast expanse of the internet. They have become indispensable tools for academic research, business decision-making, and everyday information needs. The effectiveness and efficiency of web search engines significantly impact how users interact with digital information, which makes the development of advanced and transparent search technologies essential.

This thesis aims to contribute to the field of web search technologies by developing the MOSAIC framework, which leverages the OWI from the OpenWebSearch.eu initiative. The objective was to create a scalable, flexible, and transparent search engine framework that addresses the limitations of existing proprietary systems. By focusing on modularity and the utilization of openly available data, this work seeks to enhance the accessibility, applicability and customizability of web search technologies.

Based on a review of literature and existing IR and web search tools, the design of the MOSAIC framework was informed by the need for flexibility, usability and transparency in web search technologies. The review revealed that existing proprietary systems frequently lack transparency and customization options, which results in both user and developer dissatisfaction as well as limited adaptability.

The design of the MOSAIC framework was shaped by these insights, thereby leading to several key decisions to address the identified gaps. Modularity was prioritized to ensure that different components could be developed, tested, and integrated independently. The integration of OWI partitions from the OpenWebSearch.eu initiative, consisting of CIFF and Parquet files, was chosen to support developers in creating their own search engines utilizing the OWI, but also to promote openness and collaboration. Additionally, interfaces to core application

components, such as metadata enrichment, and optional components, such as the integration of emerging AI technologies, were designed to offer extensive customization options, so users can tailor the search engine to their specific needs.

To realize the design principles of the MOSAIC framework, Lucene was used as index component to import CIFF indices, which enhances their programmatic accessibility. Therefore, the core application of the framework was developed using Java, and consequently, Maven modules were used to ensure a modular architecture for the core application and additional components. To provide an efficient and flexible solution for handling the structured metadata of OWI partitions, DuckDB was utilized.

With this, one significant finding is that MOSAIC successfully demonstrates an application how to make OWI partitions practical and usable for search engines in a straightforward manner, particularly in personal and academic contexts. This was evidenced by its practical application during the hackathon and was also highlighted by experts in the field. Additionally, both the practical user study and focus group discussions revealed that the modular architecture of the system contributes to creating a customized search engine that is not only easy to use as a service but also simple to configure and extend, thereby offering flexibility and adaptability to meet diverse user needs.

Clearly, the approach of the MOSAIC framework shows considerable potential to support developers creating customized search engines utilizing the OWI, though it requires further investigation. By addressing deficiencies in existing systems and integrating insights from literature and practical evaluations, the framework offers a feasible approach to developing open, customizable, and efficient web search solutions.

7.2. Future Work

Despite receiving positive feedback during the evaluation phase, the practical user study and the focus groups also revealed several areas for improvement and additions.

Currently, the MOSAIC framework supports a dual approach to index management, utilizing both CIFF files and DuckDB for data storage. As highlighted by experts in the focus group discussions, future enhancements should provide more flexibility by allowing users to choose their preferred index management and access method. Options could include using a single CIFF file for an index partition that contains all necessary data, importing all the index partition data into DuckDB, or implementing alternative indexing approaches. Additionally, introducing capabilities for index operations such as updating, merging, and deleting partitions without a lot of effort would significantly enhance the framework's functionality. Expanding these capabilities would not

only potentially improve performance and efficiency in certain cases but also offer greater customization to meet diverse user needs and preferences.

Moreover, obtaining index partitions is a challenging task, particularly for external users. To address this issue, future work should focus on integrating the *owilix*[1] tool, which facilitates straightforward access and downloads of index partitions. This integration will simplify the process for users to create a tailored search engine with their desired index data.

Another significant area of development involves the implementation of an effective ranking algorithm or ranking framework that considers multiple factors in a scoring pipeline for indexed documents. The design of this scoring pipeline should be inspired by existing solutions to leverage proven techniques and best practices. Particularly highlighted by experts and also mentioned in the practical user study, the ranking algorithm should be capable of functioning not only within a single index partition but also across different index partitions to ensure comprehensive and consistent search results.

Future enhancements of the MOSAIC framework should focus on integrating advanced AI technologies. This includes a direct incorporation of LLMs to improve query understanding and provide more relevant search results. Additionally, AI-driven features such as automatic query expansion, relevance feedback, and personalized search experiences can significantly enhance the system's functionality. By leveraging these advanced AI capabilities, MOSAIC can offer a more intelligent and user-centric search experience.

Additionally, to encourage a user-centric search experience, it will be essential to develop a comprehensive web user interface for the MOSAIC framework. This interface should not only demonstrate the capabilities of MOSAIC but also provide an interactive platform for end users to experience and evaluate its features. By offering a user-friendly web user interface, MOSAIC can reach a broader audience and facilitate more widespread adoption.

Expanding the use cases and applications of the MOSAIC framework is essential for identifying additional requirements that should be integrated into the system. By exploring diverse domains and real-world scenarios, the framework can be adapted to meet the specific needs of various industries and research fields. This process will uncover new functionalities and features that are necessary for broader applicability. Consequently, such expansion efforts will drive the continuous improvement and relevance of the MOSAIC framework.

Eventually, advancing MOSAIC2go should aim to minimize the technical expertise required for users to build their own web search engines. The goal is to create a comprehensive user-friendly tool where users can easily click and select index partitions and modules, configuring everything directly within the web interface. This enhancement will make the framework more accessible to a broader audience and allow individuals without extensive technical knowledge

[1] https://opencode.it4i.eu/openwebsearcheu-public/owi-cli

to leverage the capabilities of the OWI and MOSAIC. By focusing on this development, MOSAIC2go can become an intuitive, all-in-one solution for creating customized and tailored web search engines.

Bibliography

Adnan, K., & Akbar, R. (2019). An analytical study of information extraction from unstructured and multidimensional big data. *Journal of Big Data*, *6*(1), 1–38 (cit. on pp. 20, 24).

Ali, A. (2011). *Sphinx search beginner's guide*. Packt Publishing Ltd. (Cit. on p. 36).

Alonso, O., Gertz, M., & Baeza-Yates, R. (2009). Clustering and exploring search results using timeline constructions. *Proceedings of the 18th ACM conference on Information and knowledge management*, 97–106 (cit. on p. 21).

Bacciu, A., Palumbo, E., Damianou, A., Tonellotto, N., & Silvestri, F. (2024). Generating query recommendations via llms. *arXiv preprint arXiv:2405.19749* (cit. on p. 22).

Baeza-Yates, R., & Ribeiro-Neto, B. (2010, December). *Modern information retrieval* (2nd ed.). Addison-Wesley Educational. (Cit. on pp. xxi, 5, 6, 8–10, 12).

Barceló Baeza, P. (2013). Querying graph databases. *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, 175–188 (cit. on p. 14).

Bevendorff, J., Wiegmann, M., Potthast, M., & Stein, B. (2024). Is google getting worse? a longitudinal investigation of seo spam in search engines. *European Conference on Information Retrieval*, 56–71 (cit. on p. 42).

Białecki, A., Muir, R., Ingersoll, G., & Imagination, L. (2012). Apache lucene 4. *SIGIR 2012 workshop on open source information retrieval*, 17 (cit. on p. 32).

Bobić, A., Cheong, C., Filippou, J., Cheong, F., & Guetl, C. (2020). Infret: Enhancing a tool for explorative learning of information retrieval concepts. *The Impact of the 4th Industrial Revolution on Engineering Education: Proceedings of the 22nd International Conference on Interactive Collaborative Learning (ICL2019)–Volume 1 22*, 67–78 (cit. on p. 35).

Bobić, A., Gütl, C., & Cheong, C. (2021). Infret: Preliminary findings of a tool for explorative learning of information retrieval concepts. *Cross Reality and Data Science in Engineering: Proceedings of the 17th International Conference on Remote Engineering and Virtual Instrumentation 17*, 849–865 (cit. on p. 35).

Bostoen, F. (2018). Online platforms and vertical integration: The return of margin squeeze? *Journal of antitrust enforcement*, *6*(3), 355–381 (cit. on p. 26).

Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U., & Schwartz, M. F. (1995). The harvest information discovery and access system. *Computer networks and ISDN Systems*, *28*(1-2), 119–125 (cit. on pp. xxi, 15, 33, 34).

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, *30*(1-7), 107–117 (cit. on pp. 16, 18, 24).

Buttcher, S., Clarke, C. L., & Cormack, G. V. (2016). *Information retrieval: Implementing and evaluating search engines*. Mit Press. (Cit. on pp. 8, 9).

Cambazoglu, B. B., & Baeza-Yates, R. (2022). *Scalability challenges in web search engines*. Springer Nature. (Cit. on p. 15).

Castellà-Roca, J., Viejo, A., & Herrera-Joancomartí, J. (2009). Preserving user's privacy in web search engines. *Computer Communications*, *32*(13-14), 1541–1551 (cit. on p. 27).

Chau, M., Fang, X., & Liu Sheng, O. R. (2005). Analysis of the query logs of a web site search engine. *Journal of the American Society for Information Science and Technology*, *56*(13), 1363–1376 (cit. on p. 26).

Chen, J., Mao, J., Liu, Y., Zhang, F., Zhang, M., & Ma, S. (2021). Towards a better understanding of query reformulation behavior in web search. *Proceedings of the web conference 2021*, 743–755 (cit. on p. 23).

Chirita, P.-A., Firan, C. S., & Nejdl, W. (2007). Personalized query expansion for the web. *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 7–14 (cit. on p. 23).

Croft, W. B., Metzler, D., & Strohman, T. (2010). *Search engines: Information retrieval in practice* (Vol. 520). Addison-Wesley Reading. (Cit. on pp. 7, 8).

Croft, W. B., Turtle, H. R., & Lewis, D. D. (1991). The use of phrases and structured queries in information retrieval. *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, 32–45 (cit. on p. 13).

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *psychometrika*, *16*(3), 297–334 (cit. on p. 88).

Ding, Z., Gao, X., Guo, L., & Yang, Q. (2012). A hybrid search engine framework for the internet of things based on spatial-temporal, value-based, and keyword-based conditions. *2012 IEEE International Conference on Green Computing and Communications*, 17–25 (cit. on p. 27).

Dinzinger, M., Zerhoudi, S., Al-Maamari, M., Istaiti, M., Mitrović, J., & Granitzer, M. (2024). Owler: Preliminary results for building a collaborative open web crawler (cit. on pp. 29, 30).

Efthimiadis, E. N. (1996). Query expansion. *Annual review of information science and technology (ARIST)*, *31*, 121–87 (cit. on p. 8).

Ellis, D. (1989). A behavioural approach to information retrieval system design. *Journal of documentation*, *45*(3), 171–212 (cit. on p. 6).

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems (TOIS)*, *23*(2), 147–168 (cit. on p. 21).

Franceschet, M. (2011). Pagerank: Standing on the shoulders of giants. *Communications of the ACM*, *54*(6), 92–101 (cit. on p. 25).

Gao, J., Xiong, C., Bennett, P., & Craswell, N. (2023). *Neural approaches to conversational information retrieval* (Vol. 44). Springer Nature. (Cit. on p. 13).

Garg, D., & Sharma, D. (2012). Information retrieval on the web and its evaluation. *International Journal of Computer Applications*, *40*(3), 26–31 (cit. on p. 18).

Geyik, S. C., Ambler, S., & Kenthapadi, K. (2019). Fairness-aware ranking in search & recommendation systems with application to linkedin talent search. *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*, 2221–2231 (cit. on p. 25).

Gollub, T., Stein, B., Burrows, S., & Hoppe, D. (2012). Tira: Configuring, executing, and disseminating information retrieval experiments. *2012 23rd International Workshop on Database and Expert Systems Applications*, 151–155 (cit. on p. 31).

Granitzer, M., Voigt, S., Fathima, N. A., Golasowski, M., Guetl, C., Hecking, T., Hendriksen, G., Hiemstra, D., Martinovič, J., Mitrović, J., et al. (2024). Impact and development of an open web index for open web search. *Journal of the Association for Information Science and Technology*, *75*(5), 512–520 (cit. on pp. xxi, 28, 29, 31, 33, 41).

Gudivada, V. N., Rao, D., & Paris, J. (2015). Understanding search-engine optimization. *Computer*, *48*(10), 43–52 (cit. on p. 16).

Guo, J., Fan, Y., Pang, L., Yang, L., Ai, Q., Zamani, H., Wu, C., Croft, W. B., & Cheng, X. (2020). A deep look into neural ranking models for information retrieval. *Information Processing & Management*, *57*(6), 102067 (cit. on p. 26).

Hawking, D. (2004). Challenges in enterprise search. *ADC*, *4*, 15–24 (cit. on p. 27).

Hearst, M. (2009). *Search user interfaces*. Cambridge university press. (Cit. on pp. 9, 21, 22).

Hendriksen, G., Dinzinger, M., Farzana, S. M., Fathima, N. A., Fröbe, M., Schmidt, S., Zerhoudi, S., Granitzer, M., Hagen, M., Hiemstra, D., et al. (2024). The open web index: Crawling and indexing the web for public use. *European Conference on Information Retrieval*, 130–143 (cit. on pp. 30, 31, 33).

Hiemstra, D. (2009). Information retrieval models. *Information Retrieval: searching in the 21st Century*, 1–19 (cit. on p. 11).

Hyusein, B., & Patel, A. (2003). Web document indexing and retrieval. *Computational Linguistics and Intelligent Text Processing: 4th International Conference, CICLing 2003 Mexico City, Mexico, February 16–22, 2003 Proceedings 4*, 573–579 (cit. on p. 20).

Jones, C. B., Abdelmoty, A. I., Finch, D., Fu, G., & Vaid, S. (2004). The spirit spatial search engine: Architecture, ontologies and spatial indexing. *Geographic Information Science: Third International Conference, GIScience 2004, Adelphi, MD, USA, October 20-23, 2004. Proceedings 3*, 125–139 (cit. on pp. xxi, 17).

Kathuria, M., Nagpal, C., & Duhan, N. (2016). Journey of web search engines: Milestones, challenges & innovations. *International Journal of Information Technology and Computer Science*, *12*, 47–58 (cit. on p. 15).

Kleinberg, J. M. (1999). Hubs, authorities, and communities. *ACM computing surveys (CSUR)*, *31*(4es), 5–es (cit. on p. 24).

Kononenko, O., Baysal, O., Holmes, R., & Godfrey, M. W. (2014). Mining modern repositories with elasticsearch. *Proceedings of the 11th working conference on mining software repositories*, 328–331 (cit. on p. 36).

Kopliku, A., Pinel-Sauvagnat, K., & Boughanem, M. (2014). Aggregated search: A new information retrieval paradigm. *ACM Computing Surveys (CSUR)*, *46*(3), 1–31 (cit. on pp. 16, 17).

Kusner, M., Sun, Y., Kolkin, N., & Weinberger, K. (2015). From word embeddings to document distances. *International conference on machine learning*, 957–966 (cit. on pp. 13, 22).

Kvanvig, J. L. (2003). *The value of knowledge and the pursuit of understanding*. Cambridge university press. (Cit. on p. 1).

Lewandowski, D. (2015). Evaluating the retrieval effectiveness of web search engines using a representative query sample. *Journal of the Association for Information Science and Technology*, *66*(9), 1763–1775 (cit. on p. 14).

Lewis, D. D., & Jones, K. S. (1996). Natural language processing for information retrieval. *Communications of the ACM*, *39*(1), 92–101 (cit. on p. 13).

Lin, J., Mackenzie, J., Kamphuis, C., Macdonald, C., Mallia, A., Siedlaczek, M., Trotman, A., & de Vries, A. (2020). Supporting interoperability between open-source search engines with the common index file format. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2149–2152 (cit. on pp. 31, 32).

Liu, J., & Mozafari, B. (2024). Query rewriting via large language models. *arXiv preprint arXiv:2403.09060* (cit. on p. 23).

Liu, T.-Y., et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, *3*(3), 225–331 (cit. on p. 26).

Malki, Z. (2016). Comprehensive study and comparison of information retrieval indexing techniques. *International Journal of Advanced Computer Science and Applications*, *7*(1) (cit. on p. 20).

Manning, C., & Schutze, H. (1999). *Foundations of statistical natural language processing*. MIT press. (Cit. on p. 8).

Manning, C. D. (2008). *Introduction to information retrieval*. Syngress Publishing, (cit. on pp. 7, 8, 10, 12, 14–16, 24).

Meng, W., Yu, C., & Liu, K.-L. (2002). Building efficient and effective metasearch engines. *ACM Computing Surveys (CSUR)*, *34*(1), 48–89 (cit. on p. 26).

Mintzberg, H. (1998). Strategy safari: A guided tour through the wilds of strategic management (cit. on p. 85).

Najork, M. A., Zaragoza, H., & Taylor, M. J. (2007). Hits on the web: How does it compare? *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 471–478 (cit. on p. 24).

Naseer, S., Rashid, U., Saddal, M., Khan, A. R., Abbas, Q., & Daadaa, Y. (2023). Wsreb mechanism: Web search results exploration mechanism for blind users. *Applied Sciences*, *13*(19), 11007 (cit. on p. 22).

Ning, L., Liu, L., Wu, J., Wu, N., Berlowitz, D., Prakash, S., Green, B., O'Banion, S., & Xie, J. (2024). User-llm: Efficient llm contextualization with user embeddings. *arXiv preprint arXiv:2402.13598* (cit. on p. 23).

Nioche, J. (2019). Stormcrawler: A collection of resources for building low-latency, scalable web crawlers on apache storm. *DigitalPebble Ltd* (cit. on p. 29).

Nussbaumer, A., Kaushik, R., Hendriksen, G., Gürtl, S., & Gütl, C. (2023). Conceptual design and implementation of a prototype search application using the open web search index (cit. on pp. 41, 49).

Olston, C., Najork, M., et al. (2010). Web crawling. *Foundations and Trends® in Information Retrieval*, *4*(3), 175–246 (cit. on pp. xxi, 18, 19).

Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., & Johnson, D. (2005). Terrier information retrieval platform. *Advances in Information Retrieval: 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005. Proceedings 27*, 517–519 (cit. on p. 35).

Padaki, R., Dai, Z., & Callan, J. (2020). Rethinking query expansion for bert reranking. *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42*, 297–304 (cit. on p. 15).

Page, L., Brin, S., Motwani, R., Winograd, T., et al. (1999). The pagerank citation ranking: Bringing order to the web (cit. on pp. 16, 24).

Papadopoulos, S., Saiz, P., Schwickerath, U., & Kleszcz, E. (2024). Architecting the opensearch service at cern. *EPJ Web of Conferences*, *295*, 07006 (cit. on p. 36).

Potthast, M., Gollub, T., Wiegmann, M., & Stein, B. (2019). Tira integrated research architecture. *Information Retrieval Evaluation in a Changing World: Lessons Learned from 20 Years of CLEF*, 123–160 (cit. on p. 31).

Qvarfordt, P., Golovchinsky, G., Dunnigan, T., & Agapie, E. (2013). Looking ahead: Query preview in exploratory search. *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, 243–252 (cit. on p. 21).

Raasveldt, M., & Mühleisen, H. (2019). Duckdb: An embeddable analytical database. *Proceedings of the 2019 International Conference on Management of Data*, 1981–1984 (cit. on p. 32).

Robertson, S. E. (1977). The probability ranking principle in ir. *Journal of documentation*, *33*(4), 294–304 (cit. on p. 12).

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M., et al. (1995). Okapi at trec-3. *Nist Special Publication Sp*, *109*, 109 (cit. on p. 9).

Rose, D. E., & Levinson, D. (2004). Understanding user goals in web search. *Proceedings of the 13th international conference on World Wide Web*, 13–19 (cit. on p. 23).

Salton, G. (1983). Introduction to modern information retrieval. *McGraw-Hill* (cit. on p. 11).

Salton, G., Fox, E. A., & Wu, H. (1983). Extended boolean information retrieval. *Communications of the ACM*, *26*(11), 1022–1036 (cit. on p. 11).

Schalkwyk, J., Beeferman, D., Beaufays, F., Byrne, B., Chelba, C., Cohen, M., Kamvar, M., & Strope, B. (2010). "your word is my command": Google search by voice: A case study. *Advances in Speech Recognition: Mobile Environments, Call Centers and Clinics*, 61–90 (cit. on p. 21).

Seymour, T., Frantsvog, D., Kumar, S., et al. (2011). History of search engines. *International Journal of Management & Information Systems (IJMIS)*, *15*(4), 47–58 (cit. on p. 15).

Shahi, D. (2016). *Apache solr*. Springer. (Cit. on pp. 34, 35).

Shamaeva, I., & Galley, D. M. (2021). *Custom search-discover more:: A complete guide to google programmable search engines*. Chapman; Hall/CRC. (Cit. on p. 37).

Shou, L., Bai, H., Chen, K., & Chen, G. (2012). Supporting privacy protection in personalized web search. *IEEE transactions on knowledge and data engineering*, *26*(2), 453–467 (cit. on p. 22).

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter*, *19*(1), 22–36 (cit. on p. 26).

Smith, J. R., & Chang, S.-F. (1997). Visually searching the web for content. *IEEE multimedia*, *4*(3), 12–20 (cit. on p. 21).

Sommerville, I. (2011). *Software engineering, 9/e*. Pearson. (Cit. on p. 42).

Stansfield, C., Dickson, K., & Bangpan, M. (2016). Exploring issues in the conduct of website searching and other online sources for systematic reviews: How can we be systematic? *Systematic reviews*, *5*, 1–9 (cit. on p. 22).

Strohman, T., Metzler, D., Turtle, H., & Croft, W. B. (2005). Indri: A language model-based search engine for complex queries. *Proceedings of the international conference on intelligent analysis*, *2*(6), 2–6 (cit. on p. 36).

Tang, Q., Chen, J., Yu, B., Lu, Y., Fu, C., Yu, H., Lin, H., Huang, F., He, B., Han, X., et al. (2024). Self-retrieval: Building an information retrieval system with one large language model. *arXiv preprint arXiv:2403.00801* (cit. on p. 13).

Vohra, D., & Vohra, D. (2016). Apache parquet. *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*, 325–335 (cit. on p. 32).

Wachsmuth, H., Potthast, M., Al Khatib, K., Ajjour, Y., Puschmann, J., Qu, J., Dorsch, J., Morari, V., Bevendorff, J., & Stein, B. (2017). Building an argument search engine for the web. *Proceedings of the 4th Workshop on Argument Mining*, 49–59 (cit. on pp. xxi, 17).

Wang, L., Yang, N., & Wei, F. (2023). Query2doc: Query expansion with large language models. *arXiv preprint arXiv:2303.07678* (cit. on p. 23).

Wilson, M. L., Kules, B., Shneiderman, B., et al. (2010). From keyword search to exploration: Designing future search interfaces for the web. *Foundations and Trends® in Web Science*, *2*(1), 1–97 (cit. on p. 21).

Xing, W., & Ghorbani, A. (2004). Weighted pagerank algorithm. *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, 305–314 (cit. on p. 25).

Xu, S., Pang, L., Xu, J., Shen, H., & Cheng, X. (2024). List-aware reranking-truncation joint model for search and retrieval-augmented generation. *arXiv preprint arXiv:2402.02764* (cit. on p. 22).

Zhu, Y., Yuan, H., Wang, S., Liu, J., Liu, W., Deng, C., Dou, Z., & Wen, J.-R. (2023). Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107* (cit. on pp. 14, 22).

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM computing surveys (CSUR)*, *38*(2), 6–es (cit. on p. 7).

Appendix

Appendix A.

Metadata Module Creation Guide

1. **Create New Maven Module:** Create a new Maven module with <MODULE_NAME> as name (replace <MODULE_NAME> with the actual name of the module) and search-service as parent module. By default, the newly created Maven module should have the same folder structure as the existing modules. You can either use an integrated development environment (IDE) to create a new Maven module or use the following command in the directory search-service:

```
mvn archetype:generate -DgroupId=eu.ows.mosaic
                       -DartifactId=<MODULE_NAME>
                       -DinteractiveMode=false
```

2. **Add Dependency in New Module:** Add a dependency for the shared module in the file pom.xml of the newly created module:

```
<dependencies>
  <dependency>
    <groupId>eu.ows.mosaic</groupId>
    <artifactId>shared</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

3. **Add Dependency in Core Module:** Add a dependency for the newly created module in the file pom.xml of the core module:

```
<dependency>
  <groupId>eu.ows.mosaic</groupId>
  <artifactId>MODULE_NAME</artifactId>
  <version>1.0-SNAPSHOT</version>
  <optional>true</optional>
</dependency>
```

4. **Register Module in Parent Module:** If not done automatically, register your new module as such in the file `pom.xml` in the parent `search-service` by appending it to the existing modules:

   ```
   <module>MODULE_NAME</module>
   ```

5. **Create Java File:** Create a new Java file in `search-service/<MODULE_NAME>/src/main/java/eu/ows/mosaic/` that contains a class which extends `MetadataModule`[1]. For example, name this Java file and class `<MODULE_NAME>Metadata`.

6. **Override Methods:** Override methods in the newly created class as you like. Particularly, it is recommended to override `getMetadataColumns()` and `getFilterColumns()` which are responsible for retrieving additional metadata columns and defining metadata filter columns respectively.

7. **Enable New Module in Config:** Add an entry in `search-service/core/src/main/resources/config.json` in the `plugins` object to enable the module for MOSAIC.

8. If you build and start the framework, MOSAIC should now load and use the newly created module.

[1] https://opencode.it4i.eu/openwebsearcheu-public/mosaic/-/blob/main/search-service/shared/src/main/java/eu/ows/mosaic/MetadataModule.java

Appendix B.

Practical User Study Questionnaire

Personal Information

1. **What is your age?**
   - ○ Under 18
   - ○ 18-24
   - ○ 25-29
   - ○ 30-34
   - ○ 35-39
   - ○ 40-44
   - ○ 45-49
   - ○ 50-54
   - ○ 55-59
   - ○ 60-64
   - ○ Above 64
   - ○ Prefer not to say
2. **Which gender do you identify with?**
   - ○ Female
   - ○ Male
   - ○ Inter
   - ○ Diverse
   - ○ Open
   - ○ Prefer not to say
3. **What is the highest degree or level of education you have completed?**
   - ○ No degree
   - ○ Compulsory school
   - ○ Trade school
   - ○ Apprenticeship
   - ○ A-levels / High school
   - ○ Bachelor's degree
   - ○ Master's degree
   - ○ Doctorate degree
   - ○ Prefer not to say

4. **What is your current employment status?**
   - ○ Full-time employed
   - ○ Part-time employed
   - ○ Student
   - ○ Full-time employed & student
   - ○ Part-time employed & student
   - ○ Seeking opportunities
   - ○ Retired
   - ○ Prefer not to say
5. **What is your profession?**

   _____
6. **What is your personal field of work/study/research?**

   _____

   _____
7. **What best describes your level of education in computer science?**
   - ○ I have no education in computer science
   - ○ I am currently enrolled in a computer science (or similar) bachelor's programme
   - ○ I am currently enrolled in a computer science (or similar) master's programme
   - ○ I have completed my study in computer science (bachelor's programme or master's programme) and I am currently not enrolled in a computer science study programme
8. **How would you rate your level of experience in computer science?**
   no experience ○—○—○—○—○ expert level
9. **How would you rate your knowledge of web search systems/engines?**
   no knowledge ○—○—○—○—○ expert knowledge

Experience with MOSAIC

10. **How did you get in touch with MOSAIC?**
    - ○ OpenWebSearch.eu Hackathon (May 2024)
    - ○ OpenWebSearch.eu Training Event
    - ○ Expert Survey
    - ○ Other: _____

Your opinion on MOSAIC

11. **I appreciate the concept of OpenWebSearch.eu to create vertical search engines.**
    Strongly Disagree ○—○—○—○—○ Strongly Agree

12. **The technical concept and design of MOSAIC is useful to create own applications.**
    Strongly Disagree ○—○—○—○—○ Strongly Agree
13. **The installation of MOSAIC is easy.**
    Strongly Disagree ○—○—○—○—○ Strongly Agree
14. **Using MOSAIC in the development process is easy.**
    Strongly Disagree ○—○—○—○—○ Strongly Agree
15. **When using and integrating MOSAIC, no problems occurred.**
    Strongly Disagree ○—○—○—○—○ Strongly Agree
16. **What did you do with MOSAIC?**
    - □ I used it as service (i.e., I used the REST API to send queries and get responses)
    - □ I changed something in the source code
    - □ I added and used another/a new index
    - □ I modified an existing module
    - □ I added a new module
    - □ I modified the existing front-end
    - □ I created a new front-end
    - □ I integrated MOSAIC with a LLM
    - □ None
    - □ Other: _____
17. **From a developer's point of view, to what extent would you be willing to work and share the experience with MOSAIC?**
    - □ I would be willing to use MOSAIC as a service in a private/non-professional environment
    - □ I would be willing to use MOSAIC as a service in a professional environment
    - □ MOSAIC can be helpful to complete current or future tasks
    - □ I would be willing to modify or improve MOSAIC in the future
    - □ I would be willing to recommend other developers to use MOSAIC
    - □ None
18. **In your opinion, what are MOSAIC's strengths and/or weaknesses?**
    _____
    _____
    _____
    _____
19. **What features do you think are missing in MOSAIC?**
    _____
    _____
    _____
    _____

20. What suggestions for improvement do you have?