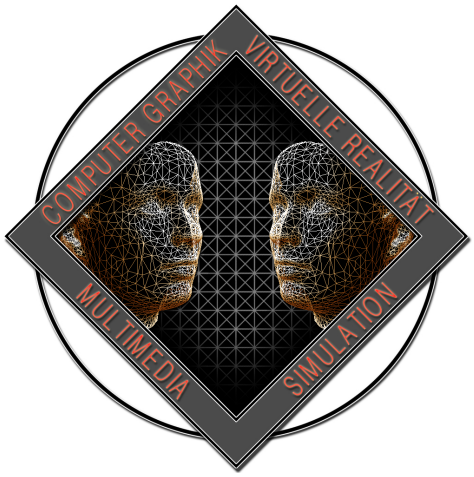
Bachelor's Thesis

AI-Powered Analysis from Diverse Research Systems for Efficient Data Management and Evaluation

Ahmed Badis Lakrach



Institute of Computer Science II – Visual Computing
University of Bonn

First Reviewer: Prof. Dr. Stephan Jonas
Second Reviewer: Prof. Dr. Reinhard Klein

January 30, 2025

Summary

In the rapidly evolving field of academic research, efficient collaboration between different disciplines is essential for the progress of scientific innovation. Unified metadata facilitates and empowers this synergy, as it can substantially improve the practicality of research data across various scientific areas.

But in practice, metadata is often fragmented and comes in different formats. It is often scattered across multiple databases and systems like PyRAT, RSpace, Zotero, and other research platforms. This makes the integration and analysis of the data a much more tedious process which leads to inefficiencies and research barriers.

This thesis will incorporate both imperative programming methods and AI-driven approaches, such as Large Language Models (LLMs) and Natural Language Processing (NLP) to automate and improve data management and evaluation. Furthermore, the capabilities of NFDI4Health—the National Research Data Infrastructure for Personal Health Data–will be leveraged to integrate and access streamlined metadata from diverse sources within the scientific research domain.

This thesis presents solutions with the help of the described systems and technologies and compares them in order to make research metadata more accessible to scientists, with the aim of optimizing the standardization of metrics from different formats through automation. Establishing an interoperable data infrastructure facilitates collaboration between researchers and lays the groundwork for larger projects.

Zusammenfassung

Im sich schnell entwickelnden Bereich der akademischen Forschung ist eine effiziente Zusammenarbeit zwischen verschiedenen Disziplinen von entscheidender Bedeutung für den Fortschritt wissenschaftlicher Innovationen. Vereinheitlichte Metadaten fördern und stärken diese Synergie, da sie den Zugriff und die Verwendbarkeit von Forschungsdaten in verschiedenen wissenschaftlichen Bereichen erheblich verbessern können.

In der Praxis sind Metadaten jedoch häufig fragmentiert und in unterschiedlichen Formaten vorhanden. Sie sind oft über mehrere Datenbanken und Systeme wie PyRAT, RSpace, Zotero und andere Forschungsplattformen verstreut. Dies erschwert die Integration und Analyse der Daten erheblich, was zu Ineffizienzen und Barrieren in der Forschung führt.

Diese Arbeit wird sowohl imperative Programmiermethoden als auch KI-basierte Ansätze wie Large Language Models (LLMs) und Natural Language Processing (NLP) integrieren, um das Datenmanagement und die -auswertung zu automatisieren und zu verbessern. Darüber hinaus wird NFDI4Health – die Nationale Forschungsdateninfrastruktur – genutzt, um vereinheitlichte Metadaten aus verschiedenen Quellen im Bereich der wissenschaftlichen Forschung zu integrieren und zugänglich zu machen.

Diese Arbeit implementiert Lösungen mithilfe der beschriebenen Systeme und Technologien und vergleicht diese, um Forschungsmetadaten für Wissenschaftler besser zugänglich zu machen, wobei die Standardisierung von Metriken unterschiedlicher Formate durch Automatisierung optimiert werden kann. Der Aufbau einer interoperablen Dateninfrastruktur erleichtert die Zusammenarbeit zwischen Forschern und legt die Grundlage für größere Projekte.

Acknowledgments

First, I would like to thank my advisors Laura Bresser and Marko Jovanović for their continuous guidance and valuable feedback throughout the course of writing this thesis. I greatly appreciate the time and effort you dedicated to reviewing my work, offering constructive insights, and helping me refine my ideas.

Second, I would like to thank Prof. Dr. Stephan Jonas and Prof. Dr. Reinhard Klein for providing me with the opportunity to write this thesis. I also extend my gratitude to Dr. Lara Marie Reimer and Leon Nissen for directing me to the Institute of Digital Medicine at the Universitätsklinikum Bonn, where I was provided with the necessary resources during the development phase. Special thanks go to Yifan Yang for granting me access to the GPU server.

Furthermore, I would like to express my sincere gratitude to my friends, family, as well as former teachers and professors for their unwavering support and encouragement.

Finally, I dedicate this thesis to my mother for believing in me every step of the way.

Disclosure of Generative AI and AI-assisted Technologies in the Writing Process
In developing this work, the author used generative AI to proofread the text and correct typographical and grammatical errors. Through manual review, any remaining spelling errors were verified and the proper use of the English language was ensured. After using the tool, the author edited the content accordingly, taking full responsibility for the final work.

Contents

1 Introduction

In today's scientific research field, integration and analysis across diverse datasets and disciplinary borders lie at the very core of advancing knowledge and innovation, particularly in areas such as data science. This creates the need for unified metadata—a standardized approach to describing and organizing research data in a manner that can be widely understood and used in different systems and fields. With access to unified metadata, researchers can explore existing studies, build on past research, and collaborate meaningfully with colleagues, ultimately achieving far greater prospects for breakthroughs and discoveries.

Scientific research has recently seen an increase in the demand for unified metadata. This is due to management challenges that lead to compatibility issues and significant fragmentation, which are further amplified by decentralization across databases and research platforms. This combination creates obstacles for scientists trying to perform a comprehensive analysis of the data, hindering their progress, and negatively affecting academic studies.

A survey indicates that data scientists spend up to 60% of their time on tasks related to cleaning, standardizing, and organizing data. Similarly, knowledge workers can use up to 50% of their time addressing issues caused by inaccurate and unclear metrics. End-users, or individuals who ultimately rely on the data for decision-making or analysis, must then invest additional time confirming the reliability of *dirty data*—data that is inaccurate, incomplete, or inconsistent (Couwenbergh, 2022; Lee et al., 2021)—further diminishing speed and productivity. Furthermore, manual procedures, such as manually sorting, entering, or verifying data lead to more errors and inconsistencies, especially as the number of dirty records rises. Beyond the immediate financial impacts, stale or outdated metrics can subtly affect organizations in various ways: Mismanaged data results in inefficient marketing expenditures, decreased productivity, ineffective internal and external communication, and wasted resources. According to market studies, inaccurate or incomplete information regarding clients and prospects can lead to a waste of approximately 27% of revenue. Consequently, dirty data hinders productivity and resource management across several sectors, including finance, electronic health records, and e-commerce (Lee et al., 2021).

This thesis aims to address some of these challenges by providing an approach that will improve the accessibility and analyzability of data for integration into a research data infrastructure. To

achieve this, it is essential to align with established frameworks that promote effective metadata management. One such framework is based on the FAIR principles—findable, accessible, interoperable, and reusable—which ensure that research output is structured within a single metadata framework. When research is organized according to these principles, it becomes significantly easier for other researchers to access and reuse the data, which in turn offers a very strong foundation for continuous development and provides an interconnected research environment able to adapt to the ever-changing demands of this field.

Imperative programming along with advanced Artificial Intelligence (AI) techniques, such as Natural Language Processing (NLP) and Large Language Models (LLMs), will be leveraged to automate the process of data extraction and structuring from scattered data sources and research platforms such as PyRAT, RSpace, and Zotero. This thesis demonstrates and compares the efficacy of these different approaches in optimizing metadata retrieval and management in scientific research. Following this process, the metrics thus processed will be integrated into the Local Data Hub (LDH) of the National Research Data Infrastructure for Personal Health Data (NFDI4Health), where the unified metadata will be accessible and reusable. Through efficient collaboration and data availability across scientific sectors, this integration will provide a framework for subsequent studies in interoperable research.

The objective of this thesis is to provide an in-depth study of research data management within the established framework. First, the foundation is laid by summarizing existing research and relevant information, defining key concepts, and discussing current challenges and practices in the field of metadata. The system architecture is then introduced, which thoroughly inspects the organized framework of the data pipeline and explains the roles of its various layers. The practical aspects are then addressed by examining the data sources, extraction techniques, processing methods, and integration efforts. Subsequently, the outcomes of the implemented approaches are summarized, followed by a comparative evaluation of these technologies and their significance for unified metadata management. Finally, the thesis concludes with important insights and suggestions for further research.

2 Background

This chapter provides a summary of current research and background information relevant to the subject of the thesis. It begins with a definition of metadata, followed by an exploration of the FAIR Guiding Principles and their role in effective data management, and then highlights the significance of metadata in ensuring reproducibility and supporting data lineage in research. The chapter then addresses current challenges in metadata management, such as fragmentation, standardization issues, and poor data quality, which are particularly prevalent in highly interdisciplinary domains. One such domain is digital medicine, where the rapid integration of diverse data sources has amplified these challenges. Finally, this chapter explores existing solutions that contribute to improving research data management and evaluation, including data management platforms, along with advanced technologies such as large language models and natural language processing, as well as frameworks developed by NFDI4Health.

2.1 Metadata and the FAIR Guiding Principles

Effective data management is highly dependent on proper metadata. This section defines metadata and introduces the FAIR principles, a set of guidelines that aim to improve metadata management and accessibility.

2.1.1 Defining Metadata

Metadata refers to structured information that describes, explains, locates, or facilitates the retrieval, use or management of an information source (Press, 2004). Historically, before the exact term "metadata" was used, similar concepts were first applied in bibliographic records about books, such as noting the book's author. Essentially, even the type of a data item itself can be metadata, helping to define its usability (Jeusfeld, 2009).

Metadata can be categorized into several types (Baca, 2016):

Administrative metadata manages and oversees collections and resources. Includes details on acquisition, rights and reproduction tracking, legal access requirements, location information, and criteria for digitization.

Descriptive metadata identifies and describes collections. Encompasses cataloging records, finding aids, version distinctions, specialized indexes, curatorial notes, linked relationships between resources, and annotations from creators and users.

Preservation metadata focuses on the management of data preservation. Documents the physical condition of resources, preservation actions for both physical and digital versions, and any changes during digitization or preservation.

Technical metadata relates to the functionality of systems and metadata. Includes documentation on hardware and software, technical details of digitization such as formats and compression ratios, system performance metrics, and security information such as encryption keys and passwords.

Use metadata tracks the use and interaction with collections and resources. Covers circulation records, exhibition details, user tracking, content reuse, multiversioning, search logs, and rights information.

The various categories of metadata each play a critical role in managing, describing, and ensuring the long-term usability of resources (Baca, 2016). However, for metadata to fulfill its purpose effectively, especially in the context of scientific research, it must adhere to established principles that promote efficient collaboration. One prominent framework for achieving this is the FAIR Guiding Principles.

2.1.2 The FAIR Guiding Principles

The FAIR Guiding Principles aim to optimize the management and reuse of scientific data by making it more accessible and usable for both humans and machines. In situations where a certain principle should be applied to both metadata and data, the distinction is drawn by the use of the term '(meta)data' (Wilkinson et al., 2016).

Findability guarantees that (meta)data is given globally distinct, persistent identifiers and is registered or indexed in searchable resources. Additionally, data has to be described with rich metadata, that is linked explicitly to the identifier of the data it describes.

Accessibility aims to allow retrieval of (meta)data via a standardized communications protocol that is free, open, universally implementable and supports required permission and authentication methods. Furthermore, metadata must remain accessible even if the data itself is unavailable.

Interoperability calls for (meta)data to employ shared, formal and accessible languages, in addition to vocabularies that align with FAIR principles and to include qualified references to other (meta)data.

Reusability emphasizes rich, well-documented metadata, clear licensing, detailed provenance, and adherence to domain-relevant community standards.

The FAIR Guiding Principles highlight the essential role of metadata in ensuring data is well-managed and reusable (Wilkinson et al., 2016). This importance is further evident in its growing

use across research and data sharing initiatives.

2.1.3 Metadata in Research

Metadata is playing an increasingly important role in research as it facilitates data organization, accessibility, and interoperability. Its significance rose with the development of digital information, open access, and data sharing policies, and has gained further momentum by employing the FAIR principles. Metadata may be defined as value added data, which provides the required context and documentation to the content it describes. This enriches metadata usability in Machine Learning (ML) and AI applications, as it becomes an integral feature for research data management across all domains, bringing transparency in handling metrics (Greenberg et al., 2023).

The growing interest in metadata is reflected across different communities, including academia, industry, and information system users. This significance is evident, for example, in a special issue of *Data Intelligence,* which features 14 original articles on metadata-related research and practice. In particular, contributions provided by more than 50 authors worldwide range from FAIR principles to metadata tools and policies. These contributions underline the importance of metadata in organizing research (Greenberg et al., 2023). For instance, it contributes to reproducible computational research by providing context and provenance to raw data and methods that facilitate discovery and validation. In fact, metadata plays a significant role in research integrity, which accelerates evaluation and reuse. In other words, it supports the research life cycle by embedding well-defined standards to promote computational reproducibility (Leipzig et al., 2021).

2.1.4 Metadata Applications in Data Lineage

Data lineage is another practical aspect of metadata applications, as it helps trace the information to its origin, showing how analysis results are derived. However, as the analysis grows more complex, particularly with technologies like AI and machine learning, traditional lineage methods may not provide enough traceability. For example, in ML model development, analysts are interested in finding out which part of the input is relevant or why specific decisions were made by the model. This is where metadata related to data provenance becomes essential. It describes the processing history of data and enables the traceability needed to interpret the results accurately. Recent research has focused on improving interpretability and explainability within AI/ML processes. To enhance this, the concept of *augmented lineage* has been proposed, which extends traditional lineage by integrating explainability techniques. This includes invocations of AI/ML models using user-defined functions within the data analysis flow, and efficient tracing with respect to how specific results were generated. Experimental results demonstrate the efficiency of this approach for improving traceability within complex data analysis (Yamada et al., 2023).

While current metadata practices demonstrate progress in areas like research and data lineage,

significant challenges remain, particularly in fields like digital medicine.

2.2 Digital Medicine: Addressing Metadata Challenges

Digital medicine, which incorporates a wide range of technologies such as mobile apps, telemedicine, and AI (Thirumal and Monika, 2024), faces unique challenges in metadata management. Outlining these constraints is the first step towards finding a solution and identifying areas for improvement.

2.2.1 Fragmentation and Standardization Issues

Metadata fragmentation across databases and research information platforms significantly impedes interdisciplinary collaboration in digital medicine. Each of these different platforms, such as RSpace and PyRAT—to name a few—have their own unique structures, which often lead to inconsistencies across these platforms and complicate data integration and analysis.

A study has identified that interoperability—the sharing and integration of information across the health care system—remains hindered, despite active policy initiatives designed to promote this key capability. Stakeholder interviews revealed that technology mismatch and organizational barriers continue to impede efforts at data sharing. In fact, hospitals face difficulties due to the growing variety of applications, including electronic health records, health information exchanges, and population health platforms that hold critical medical information. These applications are often difficult and costly to maintain, and their lack of common data structures or storage formats results in fragmented data spread across multiple platforms. This makes it harder to locate useful information and creates a significant barrier to streamlined data sharing (Walker et al., 2023).

2.2.2 Poor Data Quality in Health Research

Poor data quality in health research significantly undermines the reliability of findings and the effectiveness of decision-making in health care organizations. This issue is further compounded by variability in assessing data quality domains and metrics, as well as a lack of consensus on standardized approaches. Moreover, barriers—such as technical limitations, organizational inefficiencies, and methodological flaws—affect every stage of the data quality process. From precollection and gathering to postcollection and analysis, these obstacles collectively diminish the accuracy and reliability of health data. Consequently, the outcomes of studies and the quality of health care service delivery are adversely impacted. Therefore, addressing these challenges requires standardized practices and collaborative efforts, as well as context-specific strategies to enhance data quality (Bernardi et al., 2023).

2.3 Research Systems, Data Processing Methods, and Data Integration

Various existing organizations and technologies play a crucial role in dealing with fragmented and unstructured metadata in the field of digital medicine. This section investigates important infrastructures and advanced frameworks that improve metadata management and promote interdisciplinary collaboration. Examining research data management platforms, including PyRAT, RSpace, and Zotero, alongside advanced technologies, such as large language models and natural language processing, as well as national infrastructures like NFDI4Health, provides a better understanding of innovative approaches to improve metadata access and integration in the field of scientific research.

2.3.1 PyRAT: A Comprehensive Overview

PyRAT (Python-based Relational Animal Tracking) is a web-based lab animal colony management software designed to make animal facilities more efficient. Since its launch in 2003, PyRAT has established itself as a reliable software solution. Over time, it has grown to become the leading animal facility management platform in Europe and is now trusted by users worldwide. By providing central access to data for researchers, facility managers, and staff, PyRAT ensures its continued relevance and effectiveness as animal research environments evolve (Scionics, 2024c).

One of the main benefits of PyRAT is the ability to thoroughly track communication and authorizations, so that all operations and procedures in the facility are compliant with regulations. In addition, users can parametrize their access to data, employing custom views and filters, in order to access information in a suitable manner. By simplifying and automating many processes, PyRAT makes operations more efficient, reduces costs and improves data reliability. This saves valuable time and resources, while offering a streamlined workflow with less operational stress. Moreover, PyRAT offers easy data handling for single or multiple animals and cages within their database (Scionics, 2024c), which will later be utilized for metadata extraction in Section 4.1.

2.3.2 RSpace: Innovations in Research Data Management

RSpace is a rich content management platform designed to support organizing research data through its electronic lab notebook and sample management system. In addition to enabling researchers to record sample and experimental data, RSpace allows for detailed documentation that encompasses all aspects of the research process, from initial experiments to final results (RSpace, 2024b).

One of the key features of RSpace is its connections to many research resources, serving as an intermediary throughout the study's lifecycle. By linking the active research phase to planning,

archiving and storage, RSpace supports the seamless flow of data and metadata, and promotes FAIR principles. This connectivity improves data management and promotes collaboration among users, in such a manner that critical information is shared and accessible when needed (RSpace, 2024b).

Additionally, RSpace offers various features within the electronic lab notebook, where researchers can create professional documents using a built-in editor, link samples, equipment and materials to collaborate through automatic document sharing and auditing. This open source platform is deployed across various institutions, including the University of Bonn, advancing innovation in research data management worldwide (RSpace, 2024b). RSpace will be used in this thesis to extract complex metadata.

2.3.3 Zotero: An Open-Source Research Platform

Zotero is an open-source research tool that facilitates data extraction by automatically capturing bibliographic information from various online resources. It automatically recognizes web pages containing articles, books, or other types of content. When Zotero detects such material, it extracts detailed references that include important information like authorship, titles, and publication details. This process enhances the efficiency of collecting data, allowing researchers to easily organize relevant content. Moreover, Zotero is compatible with several web services and apps, making it an essential tool for researchers seeking to keep organized and extensive bibliographic references (Corporation for Digital Scholarship and Media, 2024). Zotero will serve as the third research data source in this thesis, specifically for providing unstructured text data.

Building on platforms like Zotero, this thesis employs natural language processing and large language models to semantically analyze text and extract relevant metadata, offering advanced methods for organizing and interpreting complex information.

2.3.4 AI and Natural Language Processing in Modern Applications

Artificial intelligence refers to technologies that enable computers and machines to replicate human capabilities, such as learning, problem solving, decision making, creativity, and autonomy (Stryker and Kavlakoglu, 2024). A crucial subset of AI, machine learning, focuses on allowing computers to learn from data, perform tasks independently, and improve their performance over time through experience (IBM, 2024). One area where these technologies are particularly impactful is in natural language processing, a branch of computer science and AI that employs machine learning to allow computers to interpret and communicate with human language (Stryker and Holdsworth, 2024). With advancements in machine learning and deep learning, NLP has shown tremendous promise in a variety of fields, including health care, education, and agriculture. Its uses include voice-automated health care systems, diabetes prediction, and crop detecting techniques in agriculture (Priya et al., 2021).

2.3.5 LLMs in Healthcare Data Processing

Large language models are advanced machine learning models that have been trained on vast text datasets and can have tens to hundreds of billions of parameters (Cloudflare, 2024; Xie et al., 2024). These models excel at generating human-like responses to prompts and solving complex tasks, showing remarkable potential in specific domains, including biomedicine and clinical settings. However, the closed-source design of powerful LLMs such as Med-PaLM 2 and GPT-4, where their code and architecture are not publicly accessible, is a significant drawback. This hinders their customization, which is needed for medical applications that depend on accurate and context-specific solutions (Xie et al., 2024). Consequently, this thesis will leverage an open-source LLM to facilitate customization in processing research data.

Given the need for customization, large language models have been deployed in the healthcare sector, enabling more efficient interaction with vast amounts of complex medical data. By using advanced natural language understanding and generation, LLMs can automate tasks such as question-answering, relation extraction and document classification. These capabilities optimize clinical workflows so healthcare professionals can quickly and accurately extract insights from vast volumes of medical literature and electronic health records, thereby improving patient care and bolstering decision-making (Nazi and Peng, 2024).

The deployment of LLMs in healthcare highlights the broader trend of advancing data management and utilization across various sectors. In this context, this thesis will leverage the NFDI4Health Local Data Hub to integrate the processed data.

2.3.6 National Research Data Infrastructure for Personal Health Data

NFDI4Health is a consortium supported by the German Research Foundation, which aims to make structured health data findable and accessible internationally, following the FAIR principles. Its goal is to connect data users and Data Holding Organizations (DHOs), particularly those involved in epidemiological, public health, or clinical trial research. To facilitate this, local data hubs are provided to DHOs, enabling them to manage decentralized data and share metadata via centralized NFDI4Health services, like the German Health Study Hub. The LDH platform offers flexible, locally controlled data management. Furthermore, a customized metadata schema has been developed and integrated with LDH software, supporting metadata transfer to other NFDI4Health services (Hu et al., 2024).

Leveraging platforms like PyRAT, RSpace, and Zotero alongside advanced technologies such as NLP and LLMs is reshaping metadata management. These systems and technologies pave the way for large-scale infrastructures like NFDI4Health, enabling efficient data integration, and collaboration between research communities.

3 System Architecture and Design

This chapter outlines the architecture of the data pipeline created for the research framework. It describes the essential components and their interactions, from data extraction to processing and integration.

As shown in Figure 3.1, the system architecture contains several layers, each with specific functionalities and components that contribute to the overall process. The data pipeline depicts the flow of data within the established framework.

The extraction layer begins with collecting unprocessed data from multiple sources: PyRAT, RSpace, and Zotero. These research platforms provide various types of data, including animal research metadata in JavaScript Object Notation (JSON) format, collaborative research documents in HyperText Markup Language (HTML) format, and research papers as plain text.

The serialized metadata from PyRAT is passed to the processing layer, where it undergoes basic operations such as validating the response and saving the JSON output to a specified file.

The complex RSpace data flows into the Transformation Layer, where it undergoes necessary data parsing and conversion. The transformation phase ensures that the data is properly organized and formatted for integration. Here, imperative programming techniques—also known as traditional programming—as well as large language models are used to convert HTML documents to a structured format, namely JSON.

The unstructured textual data from Zotero is transferred to the Analysis Layer, where advanced techniques such as natural language processing and LLMs are used. While NLP tools analyze textual data in order to extract relationships, entities, and insights, LLMs perform high-level processing tasks, such as summarizing or generating structured metadata. Finally, the analyzed metadata is serialized into a unified format—JSON.

Finally, the processed metadata is saved in the Integration Layer, where it is organized and accessible for later use. The Local Data Hub (LDH) provided by NFDI4Health serves as the central repository, ensuring that the unified metadata is persistent and can be easily queried.

These layers enable the reusability of components across different contexts and ensure optimization,

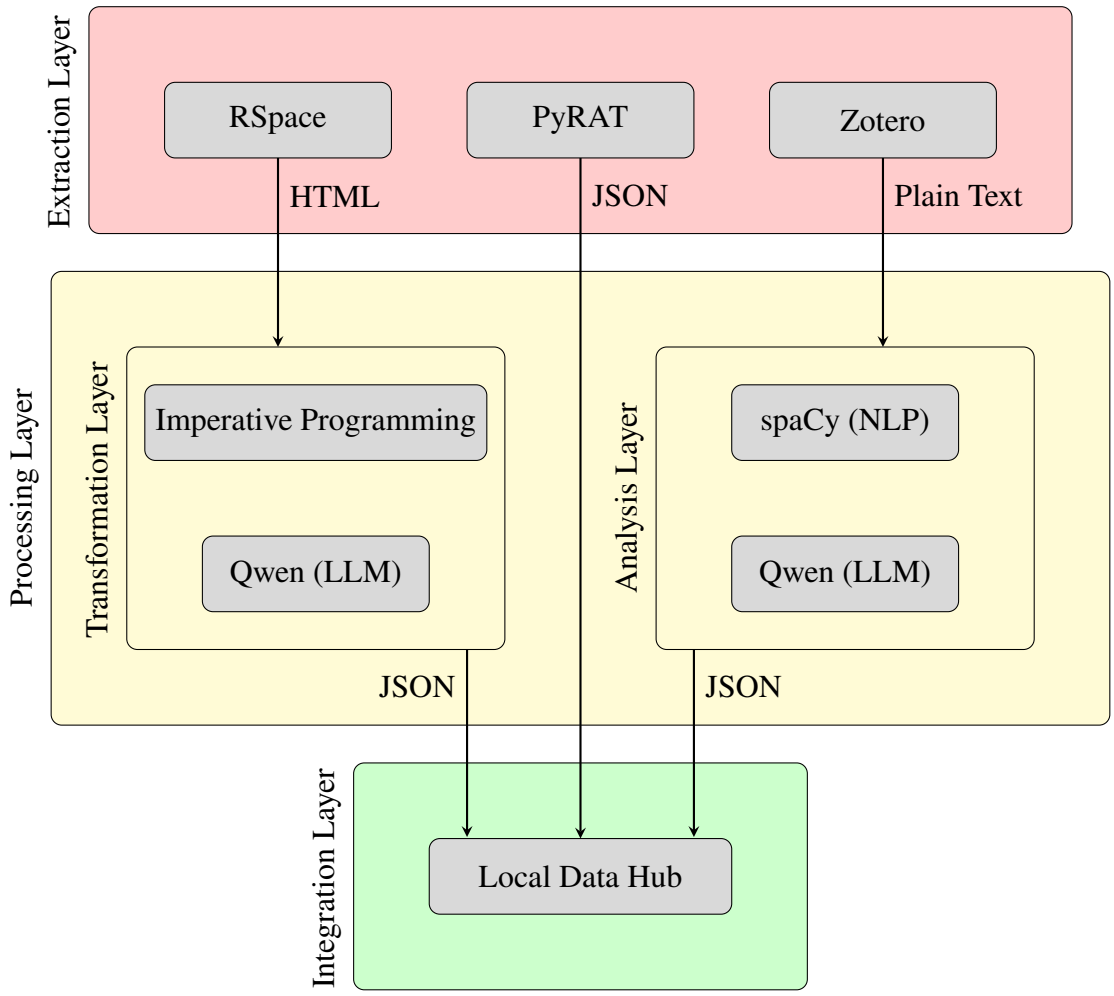


Figure 3.1: Visualization of the data pipeline

allowing each layer to be independently refined for better performance without affecting others.

3.1 Extraction Layer

The extraction layer forms the first stage of the data workflow and consists of three external systems: PyRAT, RSpace, and Zotero.

- **PyRAT** is used to interact with and fetch metadata from the animal facilities to collect structured information that is essential to manage animal research data (Scionics, 2024c), such as animal species, experimental conditions, and other study-specific details.
- **RSpace** serves as a collaborative platform for research data, which comes in various formats, notebooks, documents, and third-party integrations (RSpace, 2024a). These

files often require processing, such as data cleaning or transformation, before they can be integrated with other systems.

- **Zotero** is a reference management tool providing bibliographic, as well as document access (Corporation for Digital Scholarship and Media, 2024), that will add value to the dataset by introducing unstructured textual data. Such documents include research papers and project reports.

This metadata is often extracted directly through the API functionalities provided by the respective organization, although it is possible to do so using more sophisticated techniques, such as web scraping.

3.2 Processing Layer

After the extraction layer lies the processing layer, which consists of two subcomponents: transformation and analysis.

3.2.1 Transformation Layer

This layer is a critical component of the data processing pipeline, as it converts complex data from RSpace into a simple and unified format, preparing it for subsequent integration. The two data transformation approaches, imperative programming and large language models, will be compared in terms of their methods for converting HTML to JSON.

- **Imperative Programming** allows for writing structured code that can manipulate and process data in a very effective way. This approach ensures clear and direct control over data transformations and is well-suited for simpler or more defined tasks.
- **LLMs** are relatively versatile and capable of advanced transformation techniques. In this context, the `Qwen2.5-1.5B-Instruct` open-source large language model is utilized to parse and transform metadata with complex structures, a task where traditional programming approaches may be less effective due to their limited flexibility in dealing with diverse and complex data formats.

  The `Qwen2.5-1.5B-Instruct` model was chosen for this task since it can handle structured data and process complex formats such as RSpace's HTML data files. Its ability to recognize and organize nested data structures and convert them to simpler formats, such as JSON, makes it ideal for metadata transformation tasks. Furthermore, the model's instruction-following tuning improves its precision to generate structured outputs, and its support for long context lengths enables it to process large documents while maintaining data relationships and hierarchical integrity (Qwen, 2023).

3.2.2 Analysis Layer

The analysis layer focuses on extracting patterns from unstructured data, specifically plain text from research papers on Zotero.

- **NLP** approaches with the help of the `spaCy` library include analyzing the syntactic structure of textual data using methods such as dependency parsing, lemmatization, part-of-speech tagging, and named entity recognition. These techniques allow us to identify relevant relationships within the text, including grammatical connections between words, and to extract entities (names of individuals, organizations, or technologies). NLP additionally aids in identifying phrases and noun chunks that represent meaningful units, highlighting the text's main topics. The Python library `spaCy` was specifically chosen for its optimization in large-scale text processing, offering both high efficiency and performance. Furthermore, `spaCy` provides pre-trained models for multiple languages, making it a versatile tool for various text analysis tasks.
- **LLMs**, such as the `Qwen2.5-1.5B-Instruct` model, can process unstructured, plain text by applying pre-trained models to large datasets. They are capable of summarizing plain text from research papers and project reports, such as those stored in Zotero, and generating structured JSON metadata by extracting important information and organizing it into a simpler, unified format.

Although not a standalone layer, the final step of data processing involves saving the refined metadata as a JSON file. This ensures that the data is in a standardized structure before being integrated. Once saved, the metadata is subsequently passed to the integration layer.

3.3 Integration Layer

The final layer is the integration layer, which comprises the NFDI4Health Local Data Hub. This layer retains the integrated, processed metadata in a central resource for future access and research.

The integration layer is built upon the LDH, a MySQL database system configured within Docker containers. This structure enables the persistence of data across container restarts, allowing for seamless access and collaboration, as researchers can interact with the database through a web interface. This ensures that all entries are findable, accessible, interoperable, and reusable.

4 Implementation

The practical implementation of metadata extraction, transformation, analysis, and integration procedures within the framework of the thesis are covered in this chapter. The following sections outline the tools and approaches used to manage data from diverse sources and describe the methods of processing and integrating data into a cohesive framework. In this thesis, research management platforms such as PyRAT, RSpace, and Zotero are employed to extract metadata, alongside advanced techniques including LLMs and NLP for transforming and analyzing research data. Furthermore, the integration of metadata into the Local Data Hub is examined, to depict the process necessary for efficient metadata retrieval and management.

This research mainly draws on metadata through three complementary platforms: PyRAT is a software that can be used to extract research metadata in JSON format. In contrast, RSpace provides a rich environment for data organization and integration, thus offering multiple formats (e.g., notebooks, documents, and third-party integrations) for the findings that researchers are working on (RSpace, 2024a). Zotero complements these tools by providing large textual datasets, which facilitates the process of retrieving plain text for the extraction layer. These tools become important in metadata collection, embedded with different features that help enrich the research process.

Using their capabilities—such as metadata management and extraction—the retrieval process is facilitated to ensure detailed and reliable metadata, serving as the basis for further analysis and structuring. This shall enrich not only the dataset in itself but also contribute to the general reliability of the research methodology. The code developed for the framework of this thesis is available in a GitHub repository[1].

4.1 Extracting and Processing Serialized PyRAT Metadata

PyRAT is a Python-based software built to store and manage animal-related data. The immediate benefit of PyRAT's web-based architecture is the centralization of data access, which offers security and scalability in database management, along with the easily readable and lightweight

[1] https://github.com/digital-medicine/ba-25-alakrach-code

JSON format (json.org, 2024; Scionics, 2024c) its database uses. PyRAT will serve as an extraction tool for metadata that is gathered primarily from experiments conducted on rodents. To access and manage this metadata, PyRAT supports API calls through its web interface and third-party clients, offering efficient data retrieval (Scionics, 2024a). The collected data will then be validated and saved as JSON, to be subsequently integrated into the NFDI4Health LDH.

4.1.1 Authentication Methods

Authentication with the PyRAT API requires the Authorization header using Basic Authentication. Omitted or invalid headers will result in a 401 (Unauthorized) error. To ensure secure interaction with the API, two primary authentication methods are supported (Scionics, 2024a).

- **API-Client-Token**
  The client authenticates into the system with the help of a combination of an `API-Client-Token` as the username and an `API-User-Token` to act as the password. This is the standard procedure for third-party API clients to authenticate with this web server in a simple manner while maintaining security.
- **Cookie & Session-Id**
  The client provides a valid `Session-Id` with an associated `Cookie` to authenticate. Here, the `cookie-session` is regarded as the username, the password as the `Session-Id`. This method is mainly used internally from the web interface, as the aforementioned credentials cannot be obtained through third-party clients.

The PyRAT API facilitates authentication, allowing users to interact securely with it. This provides an avenue for different users who have been authorized with a secure communication channel for retrieving and manipulating data, without compromising the integrity of such a system (Scionics, 2024a).

4.1.2 API Interaction

PyRAT provides a formal way of interacting with the system by enabling users to perform various operations in terms of standard HTTP methods. The PyRAT API is designed to provide the most efficient ways for data retrieval and manipulation, having essential operations available for resources, like GET, POST, PUT, and DELETE. Each operation will have an equivalent resource it operates on for clarity and conciseness. The PyRAT API follows the principle of RESTful architecture and thus allows intuitive and predictable interaction (Scionics, 2024a), which is key in frameworks that promote the FAIR principles.

The PyRAT API is versioned, with the software currently at version 3, while the documentation is still based on version 2. Although PyRAT provides a variety of internal endpoints for

communication between frontend and backend code, these are not publicly documented, change frequently, and require an active user session. On the other hand, the API outlined here is stable for the entire major release and is officially supported for use by PyRAT customers and external software, ensuring safety and reliability (Scionics, 2024a).

4.1.2.1 API Endpoints Overview

The PyRAT API supports multiple endpoints that allow interaction with stored metadata. Each of these endpoints serves a different purpose and can be used to retrieve various datasets from the system. However, the endpoints may vary with each minor version and are not documented publicly. This overview examines four important endpoints: `/docs`, `/animals`, `/cages`, and `/locations` (Scionics, 2024b):

- **Endpoint: `/docs`**
  The `/docs` endpoint offers an extensive catalog of all available API calls for interacting with the PyRAT system (Scionics, 2024b).
- **Endpoint: `/animals`**
  The `/animals` endpoint allows users to access detailed data on animals in the database. Information queried from this endpoint typically includes the species, age, and health status of the animals. For researchers seeking precise metadata for their studies, the data provided by this endpoint is essential, especially regarding its structure and reliability.
- **Endpoint: `/cages`**
  Similar to the `/animals` endpoint, the `/cages` endpoint describes the cages where animals are or have been placed in the past. Data ranges from cage number and type to the name of the responsible personnel. This information helps track animal housing conditions and ensure compliance with animal welfare standards.
- **Endpoint: `/locations`**
  The `/locations` endpoint provides information about the facilities where animals are housed. A data structure of this kind includes attributes such as the location name, type, and building ID. Access to this endpoint enables researchers to correlate animal data with housing conditions, improving the accuracy and depth of their analytical frameworks.

4.1.2.2 Querying API Endpoints

Listing 4.1 demonstrates how to authenticate and send a query to the PyRAT API using the `requests` Python library. It shows how to start a session, authenticate using the provided credentials, and send a request to the `'/animals'` endpoint, which retrieves animal data based on custom query parameters.

Listing 4.1: Python code for logging into the PyRAT API

```python
import requests
from requests.auth import HTTPBasicAuth

# Login credentials
base_url = 'https://pyrat.ukbonn.de/pyrat-test/api/v3'
payload = HTTPBasicAuth('Some-API-Client-Token',
                        'Some-API-User-Token')

# Define the query parameters
params = {
    'k': [
        "animalid",  # Example return parameter
        "eartag_or_id",
        ...
        "strain_name"
    ],
    's': 'species_name:asc',  # Example sort parameter
    'o': 0, # Number of items to skip from the beginning
    'l': 100, # Maximum amount of returned items
    'status': 'active' # Animal status
}

# Start a session
session = requests.Session()

# Perform login and request data with query parameters
response = session.get(base_url + '/animals', auth=payload, params=params)
```

The code hereby establishes a session with the PyRAT API using the credentials provided to ensure that subsequent requests within this context are authenticated. The script uses the `HTTPBasicAuth` class to manage the login credentials, ensuring proper authorization of requests. The `params` dictionary defines the query parameters for an API request related to animal data. It includes the following components:

- `k`: A list of attribute names to retrieve, such as `animalid`, `species_name`, and `age_days`. This allows the user to specify which data fields are of interest. In the actual code, a total of 17 fields were requested.
- `s`: A sorting parameter that indicates the results should be ordered by a specific attribute. Here, `species_name` in ascending order was chosen.
- `o`: An offset value set to 0, which specifies the starting point for the results, useful for pagination.
- `l`: A limit set to 100, defining the maximum number of records to return in the response.
- `status`: A filter for the results, set to `active`, which restricts the query to only include active animals.

This structured approach enables efficient retrieval of relevant data from the API while allowing flexibility in sorting and filtering the results.

4.1.2.3 JSON Output

The following JSON structure in Listing 4.2 provides a detailed example response from the `/animals` endpoint, showcasing various attributes related to animal records.

Listing 4.2: Example JSON response from the `/animals` endpoint

```
[
   {
    "animalid": 536051,
    "eartag_or_id": "240-0565",
    "state": "live",
    "origin_name": "Externe Einrichtung",
    "sex": "f",
    "species_name": "mouse",
    "sacrifice_reason_name": null,
    "sacrifice_comment": null,
    "datesacrificed": null,
    "cagenumber": "B240-00001",
    "room_name": "U10",
    "building_name": "Anatomie N10",
    "age_days": 384,
    "dateborn": "2023-10-08T00:00:00",
    "comments": [],
    "strain_id": 2371,
    "strain_name": "Lcn2-/-"
  }, ...
]
```

The above example illustrates only one item from the array that contains 100 animals that the PyRAT API returned in the response body. Naturally, this limit can be changed by modifying the `'l'` parameter in the HTTP request, which further proves the flexibility of the PyRAT API. However, the API may not always be directly accessible.

4.1.3 Web Interface

This section explains how an alternative approach is required to extract metadata from PyRAT when a direct API call is impossible. For instance, when API endpoints have changed without proper documentation or in case the `API-Client-Token` is unavailable, it is still possible to access animal metadata. The HTML content returned from the web interface can be scraped and saved locally, to be processed and integrated into the LDH. Though less efficient compared to direct API calls, this method is useful for extracting data in case endpoints undergo changes in naming conventions or the base URL structure is altered following an API update. To this end, a test instance of the website[2] has been provided to support this thesis.

[2] `https://pyrat.ukbonn.de/pyrat-test/cgi-bin/login.py`

4.1.3.1 Manual Web Scraping

Once the authentication is successfully established, users can access the tables and data displayed on the web page, providing an overview of the stored metadata. This includes all the different filters and sorting options, as well as user entries and modifications. However, to integrate this data into the LDH, it becomes necessary to have it stored locally. To achieve this, users can navigate to the network tab of the browser's developer tools—as shown in Figure 4.1—to inspect the various types of data transmitted between the web interface and the server.

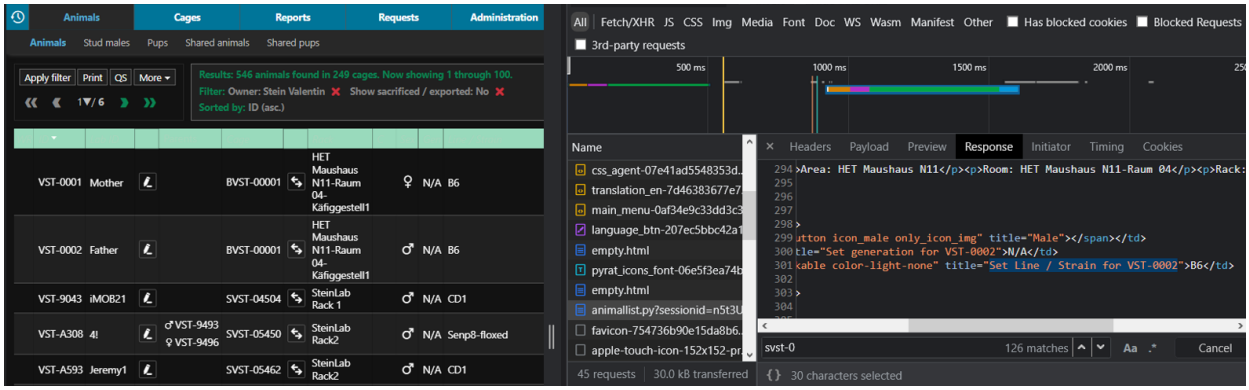


Figure 4.1: Manual web scraping of the PyRAT web interface

By examining the network activity, it is possible to identify relevant files, which contain essential metadata to be integrated. This manual inspection process thus provides a valuable alternative in acquiring research information, especially when direct access to the API is limited, which was the case at the beginning of the timeline of this thesis, but more on the technical challenges in Section 6.4.

After successful authentication and getting access to the database, the inspect tool highlights the content of `animallist.py?sessionid=some_session_id.html` among other response elements. This element details the list of animals and their attributes, showing essential metadata that is available for integration. The user can copy the displayed data and save it locally for later use, as shown in the `Imports/animals.html` file. However, this method can be tedious and time-consuming, especially when dealing with multiple requests. Another alternative to speed up the process is to use automatic web scraping.

4.1.3.2 Automated Web Scraping

Web scraping automation eliminates the need to manually browse any website for certain data, can process massive volumes of information, and perform repetitive tasks. Methods include the use of available libraries in Python, such as `requests` and `BeautifulSoup`, which log into the web interface, identify the specific location of the required information, and scrape the data in order to save it for later purposes. In `deprecated/scraper.py` is an example script of this

particular process. Specifically, the session object is instantiated via **requests.Session()** and maintains the cookies and headers throughout this session. The user then initiates the login process by submitting the required credentials to the designated authentication URL. After authentication has occurred, the script checks for the existence of a session ID in the URL, which will be required to access protected content. It then forms the URL of the page containing the data table to be scraped and retrieves the HTML content using the authenticated session. Using `BeautifulSoup`, it parses the HTML document by finding the table with the given class name. Then, the code goes through the table rows, cell by cell, and extracts text, storing the data in a list. Finally, the collected data is written to a JSON file named `table_data.json` for further data manipulation and analysis. This automated approach was implemented to improve efficiency, especially when the volume of data is large or repetition of the same task is required.

Although promising, this automated web scraping method was unable to deliver the required results. Upon executing the scraping script, the system redirected to the welcome page, as illustrated in Figure 4.2. This page included the standard header, body content, and footer, it failed however to return the intended metadata. In particular, the requested data would require access with a valid `Session-Id` cookie, which was not obtainable using third-party clients, according to the documentation. This design constraint considerably reduced the potential of the automated scraping method, as this authentication method is intended for internal use from the web interface (Scionics, 2024a). This limitation highlighted the need for a different approach to extracting HTML metadata. One such alternative is RSpace.

```
<!-- Footer with PyRAT complete name, copyright scionics and scionics logo -->
<div class="login_footer">
    <b>PyRAT</b> Python based Relational Animal Tracking   |   Copyright © Scionics Computer Innovation </br>
    <div class="scionics_logo_container">
        <a href="https://www.scionics.com/">
            <img src="../static/dist/scionics_logo_main_page-41f70af2d3fad02616a3.png"  alt="Scionics Computer Innovation" />
        </a>
```

Figure 4.2: Automatic web scraping of the PyRAT web interface

4.2 Extracting and Processing Complex RSpace Metadata

In this thesis, RSpace will be used as a platform to retrieve research data within the electronic lab notebook (ELN). The ELN provides an operating system that preserves the integrity of the scientific process, with notes and protocols stored as documents that can be flexibly organized and collaborated on, ensuring that research data remains structured and accessible for further evaluation (RSpace, 2024a). This data will later be parsed and transformed using two distinct methodologies: imperative programming and LLMs. The imperative programming approach will provide precision and control over data manipulation. In contrast, advanced LLM approaches will be used for fast and versatile data transformation techniques. These methods will then be

compared to evaluate their effectiveness in transforming metadata.

RSpace allows for the export of selected items or entire collections from both Workspace and Gallery, providing flexible export options for research data, which will be explained in the next section. Users can easily select which files they want to export, choose one of the formats, and export to third party repositories, using an intuitive interface. These exports preserve the document structure and metadata integrity, thus facilitating the archiving or sharing of research outputs. Links, attachments, and external files are preserved in the export bundle, depending on the selected format, to ensure that the exported data could be seamlessly integrated into other workflows (RSpace, 2024d).

4.2.1 Export Formats

RSpace offers multiple export formats, which are necessary for various use cases and research needs. Each of the following formats serves a specific purpose and supports different workflows in order to simplify use, sharing, and archiving of documents by different researchers (RSpace, 2024c):

- **Hypertext Markup Language (HTML)**: The HTML export creates a self-contained bundle of files, ideal for offline viewing and editing. The data structure remains intact, and it can be accessed via any web browser. This format is very suitable for sharing content with collaborators and archiving.
- **Extensible Markup Language (XML)**: This is suggested for long-term archiving or re-importing into RSpace or other systems, particularly when data needs to be transferred between institutions or when repositories require machine-readable formats.
- **Portable Document Format (PDF)**: This format is best for creating non-editable, viewable files that are easy to share. When exporting multiple documents, RSpace consolidates them into a single PDF with a footer detailing document links.
- **Microsoft Word Document (DOC)**: Best for users who prefer to edit their work in Microsoft Word. DOC exports are editable content which at later stages can be imported back into RSpace.

4.2.2 Exporting RSpace Documents as HTML

In this example, an RSpace document, of which a portion is depicted in Figure 4.3 on cytokine analysis is exported as HTML, including an outline of the experimental setup and procedures. The document provides an overview of cytokine data, highlighting variables such as volumes and concentrations.

Samples

SJHP3

1-29 PEC

1-9 PLEC (no8 missing)

Cytokines

IL-13

Volumes and concentrations

Cytokine	Coat	Standard	Detect
IL-13	20?l IN 5ml	1?l in 500?l	5?l in 5ml

ELISA Concentrations

Cytokine	Start (ug/ml)	Final (ug/ml)	Dilution	Buffer	Manufacture	Catologue	Location	Stocks	
Coating									
CCL3 (MIP 1a)	72	0.2	1 in 360	PBS	DuoSet ELISA	DY450	-20 oC Top Drawer below Steve's bench		
GM-CSF	360	2	1 IN 180	PBS	DuoSet ELISA	DY415	-80oC GMCSF ELISA Box		
IFNy			1 in 315	0.1M NaHCO3 pH9.6	Home Grown	NA	Fridge ELISA BOX	-80oC Antibody Box 1	
IL-4	1200	1.2	1 in 1000	PBS	Home Grown	NA	Fridge ELISA BOX	-80oC Antibody Box 1	
IL-5	800	1.6	1 in 500	PBS	Home Grown	NA	Fridge ELISA BOX	-80oC Antibody Box 3	
IL-6	500	1	1 in 500	0.1M Na2HPO4 pH12	BD Bioscience	554400	Fridge ELISA BOX		
IL-10	1000	4	1 in 250	0.2M NaHPO4 pH6.5	Pharmingen	551215	Fridge ELISA BOX		
IL-12p40	1000	2	1 in 500	0.2M Na2HPO4 pH6.5	BD Bioscience	551219	Fridge ELISA BOX		

Figure 4.3: Excerpt from the RSpace document "SJHP3 ELISA"

Although XML is typically recognized for its interoperability, HTML proved to be better suited to the specific needs of this thesis. While both formats are widely accessible and preserve the structure of the data, the XML export of this particular document contains HTML elements, such as tables and text, which would require two separate conversion processes for the export. These factors influenced the decision to export the RSpace document in HTML format.

The HTML text extracted from the 'SJHP3 ELISA' can be found in the `RSpace/Imports/` directory, as it was exported from RSpace and imported into the project. The document features a complex and disorganized format that complicates human readability. The prevalence of HTML tags—such as angle brackets (< and >)—obscures the content, while excessive spacing with ` ` disrupts the flow of information. For example, the use of a table defined with `<table style="border-collapse: collapse;" border="1">` adds another layer of complexity, as it may not be clear how data is organized within it. This lack of a standardized structure undermines data interoperability in addition to hindering immediate understanding, which highlights the need for effective metadata transformation.

4.2.3 Metadata Transformation

Metadata transformation is a crucial part of any data-driven evaluation, encompassing various techniques to ensure that the data is clean, consistent, and ready for meaningful interpretation. Furthermore, data may need to be transformed so that it fits the requirements of any particular study. This effective transformation improves the quality and reliability of the observations, leading to more accurate and insightful outcomes. In this context, two distinct approaches to metadata transformation will be explored: imperative programming and large language models.

4.2.3.1 Imperative Programming

This section explores a practical application of imperative programming for metadata transformation. The aim is to convert the previously exported HTML file from RSpace. The Python code in `RSpace/SJHP3_ELISA.py` utilizes the `BeautifulSoup` library to extract structured data from the HTML document located in `RSpace/Imports/`. The code's main functionalities are as follows:

1. **Imports Required Libraries:**
    - `BeautifulSoup` from the `bs4` package for parsing HTML content.
    - `json` for converting the extracted data into JSON format.
2. **Loads the HTML File:**
    - Opens the HTML file named `doc_SJHP3-ELISA-250555.html` in read mode with UTF-8 encoding, using `BeautifulSoup` to parse the content.
3. **Initializes a Data Dictionary:**
    - Creates an empty dictionary named `data` to store the extracted information, where each section title (from the <h3> elements) will be the key, and the corresponding content will be the value.
4. **Extracts Data from <h3> Elements:**
    - Iterates through all <h3> elements found in the HTML, which often represent section headings (e.g., *'Samples'* or *'Cytokines'*).
    - Retrieves the text of each <h3> tag as the section title and initializes an empty list named `content` to hold the section's extracted data.
5. **Processes Sibling Elements:**

- For each <h3> element, the script proceeds to check its following siblings (elements that appear after the heading).
- If another <h3> element is encountered, the loop breaks, indicating the end of the current section.
- If a element is found, the script looks for <table> tags within the and processes the rows and cells, appending non-empty rows to the `content` list.
- If the sibling is not a table, it checks for <p> (paragraph) and <li> (list item) elements and extracts any non-empty text content, which is also appended to `content`.

6. **Stores Extracted Content:**
   - After processing all siblings for a given <h3> element, the collected content is added to the `data` dictionary, with the section title as the key, provided the content is not empty.
7. **Converts the Data to JSON:**
   - Uses **json.dumps()** to convert the `data` dictionary into a properly formatted JSON string with indentation for readability.
8. **Saves the JSON Output:**
   - Prompts the user to enter a name for the output file (without extension).
   - Constructs the file path by appending `.json` to the user-provided name.
   - Saves the JSON output to the specified file in the `RSpace/Exports/` directory.

4.2.3.1.1 Metadata Storage and Serialization Before integrating the transformed metadata into the NFDI4Health LDH, it is important to store and organize it in a structured and manageable format. JSON, a widely used data format, is known for its lightweight, human-readable, and machine-parsable structure. The goal is to ensure interoperability across various platforms and programming languages by storing the gathered data in JSON files, making it easier to process and integrate at later stages (json.org, 2024).

JSON data structures consist of two main types: objects and arrays. Objects are collections of name/value pairs, while arrays are ordered lists of values, both of which are universally supported by most modern programming languages (json.org, 2024). As such, JSON is used as a storage format to build an intermediate layer from the raw data collection toward the integration of processed and refined metadata into the LDH. Since the metadata queried from PyRAT is already in JSON format, no conversion is needed, and thus the output will be directly saved to the corresponding `Export/` directory. For RSpace and Zotero the serialization process is necessary and will be carried out either through imperative programming using **json.dumps()**—as demonstrated in the code analysis from Section 4.2.3.1—or by means of a large language model.

4.2.3.1.2 JSON Output The following JSON output, which has been truncated for the sake of brevity, represents the structured data extracted from the document presented in Figure 4.3, which includes relevant information on the volumes and concentrations, plate plan, protocol steps, and changes made during the experiment.

Listing 4.3: Extracted "SJHP3 ELISA" data in JSON format

```
{
    "Samples": [
        "SJHP3",
        "1-29 PEC",
        "1-9 PLEC (no8 missing)"
    ],
    "Volumes and concentrations": [
        [
            "Cytokine",
            "Coat",
            "Standard",
            "Detect"
        ],
        ...
    ],
    "Plate Plan": [
        [
            "1", "9", "17", "25", "4"
        ],
        [
            "2", "10", "18", "26", "5"
        ],
        ...
    ],
    "Protocol": [
        "Coat with primary antibody 50ul/ well (ON at 4oC)",
        "Wash x3 PBS-T",
        "200ul/well Block added (2hrs/0N at RT/4oC)",
        ...
    ],
    "Changes": [
        "Plate coated on the 10.01.12",
        "Samples and standards 12.01.12",
        "Finished 13.01.12"
    ]
}
```

This structured format makes data manipulation and retrieval much easier and more suitable for further integration.

4.2.3.2 Large Language Models

This section illustrates how LLMs can be used to automate metadata transformation, contrasting this approach with imperative programming techniques. `Qwen2.5-1.5B-Instruct`, a state-of-the-art LLM tailored for instruction-based tasks, was utilized on the same RSpace document, 'SJHP3 ELISA' from Figure 4.3.

4.2.3.2.1 Overview of the Utilized Model Automated metadata transformation requires a model with the ability to parse HTML tags and deduce a logical hierarchy that can be transformed into JSON format while maintaining the data's relationships. Main attributes of the model include proficiency in recognizing and organizing nested data structures, handling complex formats like tables, and converting these elements into a simpler representation. Given these requirements, `Qwen2.5-1.5B-Instruct` was a fitting choice. Its strength in producing structured output, particularly in formats like JSON, made it suitable for this task (Qwen, 2023).

Within the `Qwen2.5` model family, the `Qwen2.5-1.5B-Instruct` model is an advanced, instruction-tuned version created primarily for tasks requiring precise and structured outputs. This model series, available in a range of parameter sizes from 0.5 to 72 billion, integrates notable enhancements from its predecessor, `Qwen2`, especially in knowledge retention, coding proficiency, and mathematical reasoning. `Qwen2.5-1.5B-Instruct`, with 1.5 billion parameters, is tuned for instruction-following, making it well-suited for applications requiring detailed metadata extraction and cleaning. One of the model's core strengths lies in its ability to manage structured data efficiently. With support for context lengths up to 32,768 tokens and the ability to generate up to 8192 tokens, it is capable of processing long documents while producing structured outputs in formats like JSON. This capability greatly facilitates metadata manipulation tasks, as it allows the model to maintain structured formatting and hierarchical data integrity. Multilingual support covering more than 29 languages increases its usability for worldwide research and metadata extraction from multilingual sources (Qwen, 2023).

4.2.3.2.2 Code Analysis Inspired by a previous group project on AI-powered early detection of dementia, the code framework in `LLM/model.py` has been significantly adapted and improved to meet the unique requirements of this task. This system establishes a structured interaction pipeline between users and the aforementioned LLM, specifically tailored for data cleaning, transformation, and analysis. With a focus on structured outputs, the code creates a user session utilizing `chainlit`, along with a chatbot using the `Qwen2.5-1.5B-Instruct` model, and Hugging Face's `transformers` library.

First, the script employs a global variable to load the model only once, reducing resource demands across sessions. When **load_model()** is called, it authenticates with the Hugging Face Hub and loads the `Qwen2.5-1.5B-Instruct` model onto the GPU, if available, in order to generate faster responses. The tokenizer and model are then wrapped in a text generation pipeline that manages response formatting to create coherent, dynamic responses.

The main chatbot functions are defined in **start()** and **main()**, which handle user session initialization and message processing. When a user connects, **start()** sends a welcome message, loads the model if needed, and sets session variables to track the context of the conversation. Each new message triggers **main()**, which formats the message with system-specific prompts that instruct the model to transform and analyze data given by the user, outputting it in the required

format. Finally, exception handling in **send_query_to_model()** guarantees resilience, properly managing potential model or connectivity problems to improve reliability and offer clear feedback for troubleshooting.

The utility functions **create_new_prompt()** and **format_model_response()** streamline interaction handling. Specifically, **create_new_prompt()** constructs queries with embedded context from prior messages, ensuring continuity in conversation history, which is essential for coherent, interactive dialogues. Meanwhile, **format_model_response()** processes the model's response by eliminating repeated prompts, improving clarity in the replies.

4.2.3.2.3 JSON Output Using the framework detailed in the previous section, the chatbot was provided with an instruction to convert structured data from HTML to JSON format. The request was as follows: *"I want you to convert this data from HTML to JSON format:"* followed by the HTML content of 'SJHP3 ELISA.' As shown in Figure 4.4, the model's response displayed a structured JSON output that accurately represented the protocol's hierarchical structure.

Figure 4.4: LLM response converting HTML data into structured JSON

The response included information on sample names, concentrations, and procedural steps. The JSON output was organized into categories such as *'Samples,' 'Plate Plan,' 'Protocol,'* and *'Changes,'* demonstrating the model's effective parsing and transformation of nested data. Building on this approach, the next step is to extract and process unstructured textual data.

4.3 Extracting and Processing Textual Zotero Data

Zotero is a research tool that is widely utilized for the management of bibliographic metadata and research material. Beyond its function as a reference manager (Corporation for Digital Scholarship and Media, 2024), it simplifies data extraction from various formats, including PDFs. In this thesis, Zotero will be used to download scientific reports through its API. Once the documents are retrieved, their plain text will be parsed to extract specific sections such as abstracts and introductions. Subsequently, the parsed text can be semantically analyzed to generate metadata.

4.3.1 Retrieving Documents with the Zotero API

This section studies the data extraction process by exploring the methodology for retrieving full-text articles and their underlying content. First, the process begins by importing the `Pyzotero` library, which allows users to programmatically interact with the Zotero API to access items and attachments within certain libraries like conference papers and journal articles. For instance, one can access the API to download a scientific report directly to a local system. Below is an analysis of a code snippet that executes this particular function.

The Python script `Zotero/chatbot_paper.py` calls the API client `Pyzotero` to download a PDF attachment from a Zotero library. It initializes the `api_key`–a unique credential used to authenticate requests–and `user_id`, the identifier of the particular Zotero user trying to retrieve the document. The `library_type` variable indicates whether the script is interacting with the library of a personal user or a group. In this case, it is set to *'user'*. Then, the code checks whether the specified PDF file already exists locally. If it is not found, the script fetches the attachment from Zotero by using the `attachment_key` to identify the file. The attachment is then saved locally under the specified path.

The integration of programming with `Pyzotero` improves efficiency through the automation of tasks that would otherwise be time-consuming. Further text parsing and isolation will be thoroughly explored in the upcoming section.

4.3.2 Text Extraction Process

Once downloaded, text can be extracted from the PDFs using the `PyPDF2` library. This tool enables the extraction of textual content, allowing researchers to isolate specific sections such as abstracts or introductions, to quickly and efficiently understand the context of academic papers. For example, a group project report[3] contains 29 pages but by parsing the text between "Abstract" and "Introduction," it is possible to summarize essential findings without needing to read through

[3] `https://www.zotero.org/groups/296328/digital_health_group/collections/WEEFJBJW/items/T7XRWJQQ/item-list`

the entire document, as demonstrated in Listing 4.4—a code snippet from `chatbot_paper.py`:

Listing 4.4: Python script for extracting a specific section from a PDF file

```
# Open and read the content of the PDF
with open(pdf_file_path, 'rb') as f:
    reader = PyPDF2.PdfReader(f)
    number_of_pages = len(reader.pages)

    # Define lists of synonyms for 'Abstract' and 'Introduction' in multiple languages
    abstract_keywords = [
        "Abstract",          # English
        "Summary",           # English
        ...
        "Zusammenfassung",   # German
        ...
    ]
    introduction_keywords = [
        "Introduction",      # English, French
        "Motivation",        # English
        ...
        "Einleitung",        # German
        ...
    ]

    # Extract text and find the Abstract
    found_abstract = False
    extracted_text = ""
    for page_num in range(number_of_pages):
        page = reader.pages[page_num]
        text = page.extract_text()

        # Split the text into lines for better control
        lines = text.split('\n')

        for line in lines:
            # Check for the start of the Abstract section
            if any(keyword in line for keyword in abstract_keywords):
                found_abstract = True
                extracted_text += line.strip() + " "  # Include the line with the keyword
                continue  # Move to the next line

            # If in the abstract, append the line
            if found_abstract:
                # Stop if we reach any of the Introduction section keywords
                if any(keyword in line for keyword in introduction_keywords):
                    found_abstract = False  # Stop capturing
                    break

                extracted_text += line.strip() + " "  # Append line to the result

    extracted_text = extracted_text.strip()
```

The implementation supports 9 languages, including English, German, Spanish, Catalan, Portuguese, Arabic, Japanese, Chinese, and French. This multilingual support allows the script to recognize various synonyms for the keywords "Abstract" and "Introduction" across different languages, enabling researchers to extract the summaries from different documents and

overcoming language barriers, which aligns with the FAIR principle of interoperability. For example, in German, the term "Zusammenfassung" is commonly used to refer to the abstract, while the word "Einleitung" is often used for the Introduction section. The script can be utilized with a range of research materials due to the adaptability of these synonyms.

Furthermore, direct data extraction and manipulation from Zotero support advanced techniques like qualitative coding and content analysis. The data retrieved can be methodically analyzed by researchers to find trends, topics, or patterns that are pertinent to their research.

4.3.2.1 Text Output

After using the Zotero API to download the PDF document, the extraction procedure was able to locate and retrieve the abstract of the project report. The extracted abstract, for instance, reads: *"Abstract. With the rising prominence of artificial intelligence (AI) and generative pre-trained transformers (GPTs) in various domains, our project taps into these technologies to address a specific need in healthcare..."* This extraction facilitates quicker access to relevant information for analysis.

4.3.3 Data Analysis

Data analysis enables researchers to organize and interpret attributes. In this section, two approaches are examined: NLP techniques and LLM methodologies, each offering unique insights into the generation of metadata by semantically analyzing unstructured textual content.

4.3.3.1 Natural Language Processing

Natural language processing is often used to extract meaningful insights from raw textual data. Advanced tools such as `spaCy` enable researchers to automatically analyze grammatical structures of sentences with the purpose of studying the relations between words and phrases, and key patterns in linguistic data. A code snippet is shown in Listing 4.5, demonstrating how the textual data is analyzed using `spaCy` and then serialized into JSON format.

Listing 4.5: Code for tokenization, entity recognition, and noun chunk extraction with `spaCy`, followed by JSON serialization

```
# Initialize SpaCy
nlp = spacy.load("en_core_web_sm")

# Process the extracted text with SpaCy
doc = nlp(extracted_text)

# Prepare the output structure
nlp_output = {
    "tokens": [],
    "entities": [],
    "sentences": [],
```

```
    "noun_chunks": []
}
# Process tokens and sentences
for sent in doc.sents:
    sentence = sent.text
    nlp_output["sentences"].append({"sentence": sentence, "tokens": []})
    for token in sent:
        nlp_output["tokens"].append({"token": token.text, "pos": token.pos_})
        nlp_output["sentences"][-1]["tokens"].append({
            "token": token.text,
            "pos": token.pos_,
            "dependency": token.dep_,
            "head": token.head.text
        })
# Process entities
for ent in doc.ents:
    nlp_output["entities"].append({"text": ent.text, "type": ent.label_})

# Process noun chunks
for chunk in doc.noun_chunks:
    nlp_output["noun_chunks"].append(chunk.text)

# Convert to JSON
json_output = json.dumps(nlp_output, indent=4)
```

In `chatbot_paper.py`, `spaCy`—a natural language processing library—is used to analyze the extracted text. First, the script loads the `small` English model of `spaCy` and creates a document from the isolated text data that was previously extracted from the project report, namely the abstract. The code then performs a number of analyses: it tokenizes the text, outputting each token along with its lemma and part-of-speech (POS) tag. Then, it labels the entities that are present in the text. The code subsequently analyzes sentences, providing details on each token's POS tag, dependency relation, and syntactic head. Additionally, it extracts noun chunks, which are useful for identifying relevant phrases. The section on dependency parsing demonstrates the relations between words within the text. The resulting file `nlp_output.json` can be found in the `Zotero/Exports/` directory. Finally, `displaCy`, the dependency visualizer of `spaCy` is used to illustrate the dependency graph for a clear examination of the syntactic structure of the text. The results of this analysis are studied in depth in Section 5.3.1.

4.3.3.2 Large Language Models

As mentioned earlier, the `Qwen2.5-1.5B-Instruct` model was first used for metadata transformation, making use of its ability to parse and convert complex HTML data. Here, the same model is used to analyze textual data, demonstrating the versatility of LLMs in managing a variety of tasks.

To guarantee a fair and consistent comparison between NLP tools and LLMs for data analysis in the Discussion chapter, the same project report from Section 4.3.2 will be used as an example. Specifically, the content of the document will be copied from the title to the end of the abstract,

encompassing everything in between as plain text. This plain text will then be provided to the LLM, which will be prompted to extract pertinent metadata attributes, such as keywords, author information, title, and other essential elements, all while outputting the results in JSON format. This procedure aims to convert unstructured textual data into easily identifiable and editable structured metadata.

4.3.3.2.1 LLM Output The `Qwen` model successfully analyzed the text provided in the title and abstract of the project report and transformed it into JSON metadata, demonstrating its ability to recognize and arrange important components. The JSON output generated by the model is shown in Listing 4.6.

Listing 4.6: Project report JSON metadata generated using the Qwen model

```
{
  "summary": "The study explores how language modeling and keyboard dynamics can be used to
      detect neurodegenerative diseases through patient interactions and data collection.",
  "authors": ["Ahmed Badis Lakrach", "Rami Kallel", "Rami Baffoun", "Frederik Hasenbach", "Ali
       Reda", "Malte Steingass"],
  "conference": "",
  "copyright": "",
  "keywords": ["neurodegenerative diseases", "language modeling", "keyboard dynamics", "early
      detection", "chatbots", "data collection"],
  "paper_id": "N/A",
  "project_name": "Chatbot for Neurodegenerative Disease Detection",
  "title": "Exploring the Efficacy of Language Modeling and Keyboard Dynamics in the Early
      Detection of Neurodegenerative Diseases"
}
```

4.4 Integrating Metadata into the Local Data Hub

This section discusses the process of integrating processed metadata into the LDH—the platform provided by NFDI4Health. First, it covers the deployment setup of the Local Data Hub including the Docker installation and volume creation. This section then details the integration of extracted and processed metadata into the LDH. The integration guarantees seamless access to structured metadata for future analysis and interdisciplinary collaboration.

4.4.1 LDH Deployment Setup

This section describes the steps involved in deploying the Local Data Hub on a Windows machine to integrate the metadata processed so far. The setup involves containerization using Docker, which facilitates the management of services and guarantees a streamlined environment for interacting with the processed metadata.

4.4.1.1 Installing Docker

This section describes the steps taken to set up the LDH locally on a Windows machine. The LDH deployment relies on containerized services managed by Docker. The modern method of service orchestration uses **docker compose** and offers a robust mechanism for handling many containers efficiently. This project was set up to allow local testing and interaction with the LDH, which will be used to integrate unified metadata derived from the diverse research systems.

To enable this, Docker Desktop was installed on a Windows machine. Although by default Docker recommends Windows Subsystem for Linux (WSL) 2, which offers the dynamic memory allocation feature to reduce resource consumption (Docker, 2024), Docker Desktop was instead setup with Hyper-V, a hypervisor native to Windows. This decision was based on personal preference, particularly due to prior experience with using Hyper-V. This Windows hypervisor provides a solution for running Docker containers by creating a lightweight virtual machine (VM) to host a Linux environment, in which the containers can operate. This approach can help manage resource usage by isolating containers in a VM. Hyper-V can also mitigate certain compatibility issues by offering a stable and isolated virtualization environment.

4.4.1.2 Creating Volumes

With Docker installed, the LDH repository needed to be cloned from GitHub, containing the necessary configuration and environment files. In the `.env` file, a number of project parameters had to be set up, such as port configuration and project name prefix. The `docker-compose.yml` file provides further specification for the database and other service configurations, and defines how the MySQL database and other containers are built and linked (Meineke and Wagner, 2024). The MySQL database was configured within a shell file `start.sh` using a volume to ensure persistence of data across container restarts. Once the containers were successfully deployed, the web interface could be accessed through the `localhost`, providing the means of interaction with the LDH. This setup effectively emulates the environment needed to integrate the processed research metadata.

4.4.2 Metadata Integration

To be able to run Docker, a prior connection to a virtualization server is required. After launching Hyper-V Manager on Windows and selecting "Connect to Server" followed by "Local computer," the container can be started with the `start.sh` script. This grants access to the NFDI4Health LDH via `http://localhost:3000`. Upon access, the page redirects to a sign-up form to create an account within the platform. Registration is necessary for the first container startup. After the registration is complete, logging in with the newly created credentials grants access to the main interface, where projects can be created or joined, as demonstrated in Figure 4.5.

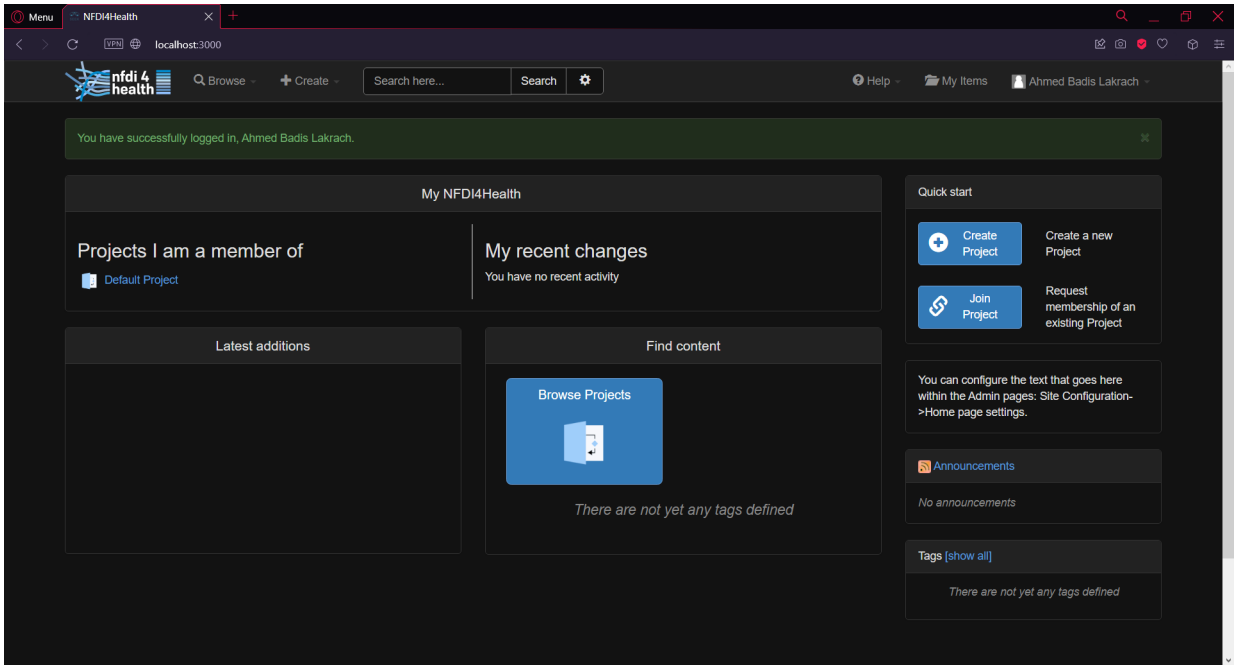


Figure 4.5: Local Data Hub main interface

Under each project, new data files can be imported and created. At this point, the metadata processed so far can be integrated into the platform. For instance, JSON files such as `nlp_output.json`—the output provided by the code in Listing 4.5—can be uploaded and accessed. This data integration allows the unified storage of metadata from various sources within the LDH for further analyses and collaborations between projects. As illustrated in Figure 4.6, the previously imported file `nlp_output.json` can be found, saved within the NFDI4Health Local Data Hub.

Even after pausing, shutting down, and subsequently restarting the container, the integrated metadata remains stored and accessible. This allows it to be read, modified, and extracted as needed. The persistent storage mechanisms ensure that no data is lost during these operations, preserving continuity and facilitating future access for further analysis and updates.

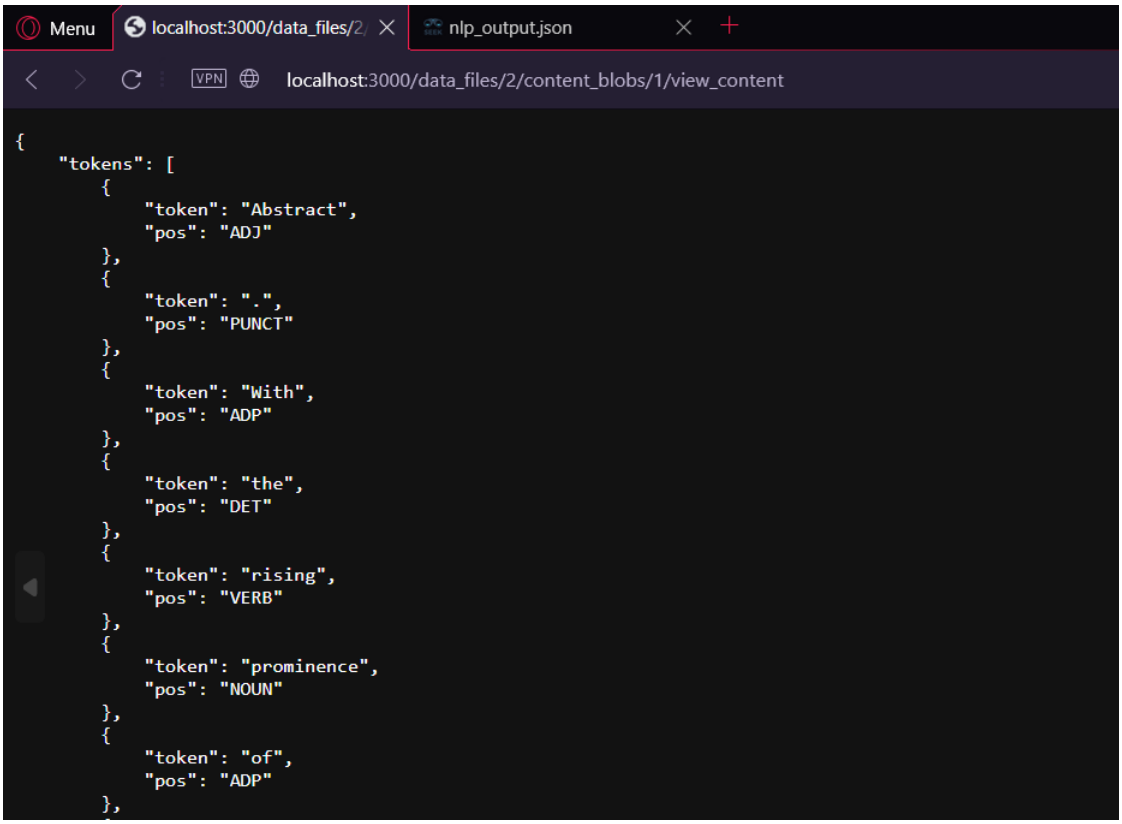


Figure 4.6: `nlp_output.json` accessed from the Local Data Hub

5 Results and Findings

This chapter encapsulates the findings derived from the extensive metadata extraction and processing activities undertaken throughout the implementation part of this thesis. In the following sections, the results of several metadata management approaches will be presented in order to evaluate their effectiveness and identify which methods are best suited to this specific scenario.

5.1 Metadata Extraction

Serialized metadata is collected using PyRAT, a system designed to manage and retrieve animal-related data through an API using endpoints such as `/animals`. This facilitates the integration of animal attributes, with three methods available for metadata retrieval: direct API access, manual, and automated web scraping. Each of these methods offers unique advantages and challenges, and their effectiveness is evaluated based on criteria such as performance and reliability. In this section, the outcomes of these methods are discussed in detail.

5.1.1 Metadata Retrieval through API Endpoints

Direct API calls were successful, providing a robust and efficient means of retrieving animal-related data. The following results were observed by querying the `/animals` endpoint. The authentication process, using the `requests` library and `HTTPBasicAuth`, was effective in securing authorized access to the API. This ensured that only authenticated users could query the endpoints. Third-party clients allow for customized queries by defining specific query parameters, such as `k`, `s`, `o`, `l`, and `status`. This enabled tailored requests for specific animal attributes and ensured that the returned data was relevant to research needs. The API returned metadata in a structured `JSON` format, which simplified integration into downstream processes. The data was clean, consistent, and ready for integration. Using third-party clients for metadata extraction proved to be efficient, allowing large-scale querying and integration without manual intervention, thus improving workflow and reducing errors.

5.1.1.1 Measuring API Performance

The following describes the methods used to measure the performance of the API request in terms of **memory usage** and **response time**, followed by the results obtained from these measurements.

5.1.1.1.1 Memory Usage Measurement To assess the memory consumption during the API request, the `psutil` Python library was utilized to monitor the memory usage of the process executing the script. Specifically, the **resident set size (RSS)**—which represents the amount of memory allocated by the operating system to the process—was measured. The memory usage was captured just before making the API request, allowing the measurement of the baseline memory consumption of the script, then it was measured again after the request was completed to determine the additional memory used by the process during the execution. The tests were run on an `11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz` CPU.

The memory usage was recorded by capturing the `RSS` value in bytes and converting it into **megabytes (MB)**. The difference between memory usage before and after the request reflects the amount of memory consumed by the API request, including overhead associated with the `requests` library and the metadata returned by the API.

To measure the response time of the API request, the `elapsed` attribute from the `requests` library was used. This attribute returns a `timedelta` object, from which the total number of seconds taken to complete the request is extracted. The response time reflects the time from sending the request to receiving the response, including network latency, server processing time, and the time it takes for the response data to be returned. The response time was measured in **seconds**.

The memory usage before the request was sent to the `/animals` endpoint was **44.06 MB**, while after the request, it increased to **47.62 MB**. This resulted in a memory difference of **3.56 MB**. Additionally, the total response time for the API request was measured at **0.23 seconds**.

The methods used to measure memory usage and response time provide insight into the efficiency of the API request. The memory usage difference of **3.56 MB** suggests a moderate increase in memory consumption, which indicates that the metadata returned by the API and the associated request handling added some overhead. The response time of **0.23 seconds** is reasonable, reflecting the time required to retrieve and process the data for 100 animal objects, each with 17 parameters.

Overall, directly querying the API proved to be reliable, scalable, and efficient for extracting metadata, making it the preferred method for future data retrieval tasks.

5.1.2 Manual Web Scraping

Manual web scraping was used as a workaround when direct API access was unavailable at the start of this thesis. The following results were observed from the manual scraping process.

By inspecting the network activity in the browser's developer tools, it was possible to identify and extract relevant data from the web interface, which could then be saved locally for further integration. The metadata extracted manually via web scraping contained the same information as that returned by the API, albeit in a different format (HTML).

The manual process required extensive effort to identify the correct resources, copy the metadata, and save it locally. This time-consuming method is not scalable for large datasets or when dealing with frequent data updates.

While manual scraping provided a viable solution in the absence of direct API access, it was inefficient and not scalable for large-scale metadata extraction tasks.

5.1.3 Automated Web Scraping

To improve efficiency, an automated web scraping approach using Python libraries like `requests` and `BeautifulSoup` was implemented. However, several limitations were encountered during this process.

After authentication, the automated scraping script retrieved HTML content from the target page and extracted some data from it. This approach was designed to replace manual intervention, making it faster and more scalable for large volumes of data.

Despite the automation script being able to log in and scrape some data, it encountered issues with session management. Specifically, it was impossible to retrieve the necessary `session-id` cookie required to access protected resources, due to a design decision related to session handling. Failing to manage the session ID, the automated script was unable to retrieve the relevant metadata. Instead, the system redirected the request to the default welcome page, which included generic content but lacked the targeted animal-related data.

The automated web scraping approach showed promise in improving efficiency but was ultimately hindered by issues related to session management. Without the ability to persist a valid session ID, the automated method could not replace direct API access for consistent metadata retrieval.

5.2 Metadata Transformation

Metadata transformation techniques include various methods intended to convert, organize, and unify data attributes to promote the FAIR principles. This section will cover the findings derived

from the transformation approaches employed throughout the thesis, each with unique benefits for managing and structuring complex RSpace metadata, from imperative programming methods to large language models.

5.2.1 Imperative Programming

The imperative programming approach described in Section 4.2.3.1 demonstrates the utility of traditional methods for processing complex HTML documents, namely 'SJHP3 ELISA' from RSpace. The following is a summary of the results derived from the output.

The Python script successfully parsed and organized document sections, such as *'Samples,' 'Volumes and Concentrations,'* and *'Plate Plan'* into a structured JSON format. Relationships between headings (<h3>) and their corresponding content were preserved, providing a logical and readable structure. Furthermore, the script is scalable, handling arbitrary numbers of <h3> and sibling elements, making it robust for similar documents. Finally, the output in JSON format facilitates integration with analysis tools and workflows for biological experiments.

While the imperative programming approach produces accurate and structured metadata, it falls short of the ideal output in a few aspects. For instance, the section *'Volumes and Concentrations'* shows no clear indication of a parent-child relationship. Instead, each label is grouped in an array without direct association with their respective values, as demonstrated in Listing 5.1. This structure requires additional effort to map labels to values, increasing parsing complexity and error potential.

Listing 5.1: Volumes and concentrations converted to JSON using imperative programming

```
"Volumes and concentrations": [
    [
        "Cytokine",
        "Coat",
        "Standard",
        "Detect"
    ], ...
],
```

5.2.2 Large Language Models

Here, the `Qwen` LLM was used to convert HTML data into a structured JSON format, specifically applied to the same RSpace document, 'SJHP3 ELISA'. The following results were observed.

The JSON output provided by the LLM effectively transforms the HTML metadata into a hierarchical structured format, enhancing clarity and accessibility. It even solved the grouping problem encountered with imperative programming by pairing each key (e.g., *'Cytokine'*) directly with its corresponding value (e.g., *'IL-13'*). Moreover, `Qwen2.5-1.5B-Instruct`'s response includes contextual descriptions that improve the readability and interpretability of the research

data. Each section is arranged with a *'Description'* field, providing brief explanations of the purpose or content of the data, allowing users unfamiliar with the original document to understand and interact with it effectively.

5.3 Data Analysis

Metadata is needed for organizing, evaluating, and deriving insights from textual information. By utilizing advanced NLP tools and LLMs, this section shows how metadata from a project report can be inferred by analyzing plain text. Such findings help identify important elements and structural insights that can guide further research and development.

5.3.1 Natural Language Processing

This section explores how dependency parsing, part-of-speech (POS) tagging, and named entity recognition (NER) were applied in a Python script called `chatbot_paper.py` using `spaCy` to analyze the abstract of the project report mentioned in Section 4.3.2. This abstract discusses the implementation of a chatbot employing AI and large language models for data collection among users, focusing on early detection of cognitive impairments.

5.3.1.1 Syntactic Analysis

By performing dependency parsing, `spaCy` allows examining all the different grammatical relationships between words in the text. It assigns to every token a head, along with its dependency label and POS tag, which serve to identify the role that particular word plays in the sentence structure. Listing 5.2 shows the dependency tree for a portion of the abstract.

Listing 5.2: Dependency relations and POS tags of tokens

```
Token: With, POS: ADP, Dependency: prep, Head: taps
Token: the, POS: DET, Dependency: det, Head: prominence
Token: rising, POS: VERB, Dependency: amod, Head: prominence
Token: prominence, POS: NOUN, Dependency: pobj, Head: With
Token: of, POS: ADP, Dependency: prep, Head: prominence
Token: artificial, POS: ADJ, Dependency: amod, Head: intelligence
```

The token *'rising'* is labeled as a `'VERB'` in the output above, despite the fact that it actually serves as an adjective. This is due to the lemmatization that NLP tools undertake for the contextual analysis of words, which is shown in Listing 5.3.

Listing 5.3: Lemmatization of "rising" to "rise"

```
Tokens:
rising -> rise (VERB)
```

The token *'rising'* being reduced to *'rise'* occurs due to how lemmatization works in NLP. Lemmatization is a transformation that reduces a word to its base or dictionary form, which is called a lemma (Liu et al., 2012). In this case, *'rising'* is the present participle of the verb *'to rise,'* and the lemmatizer properly identified its base form *'rise'*-the infinitive. In this sentence, *'rising'* is an adjective modifying *'prominence'* as indicated by the `amod` (adjective modifier) dependency label. Therefore, it is still treated in its base form by `spaCy`.

Similarly, other tokens are lemmatized and combined to form the prepositional phrase *'With the rising prominence'*. This level of syntactic detail reveals how each element in the sentence relates to the others.

These dependency parsing results provide a deep understanding of the text's structure beyond simple word order.

5.3.1.2 Visualization the of Dependency Graph

The dependency graph can also be visualized for easier interpretation of these relationships. Figure 5.1 shows a simplified graph representation of the syntactic structure.

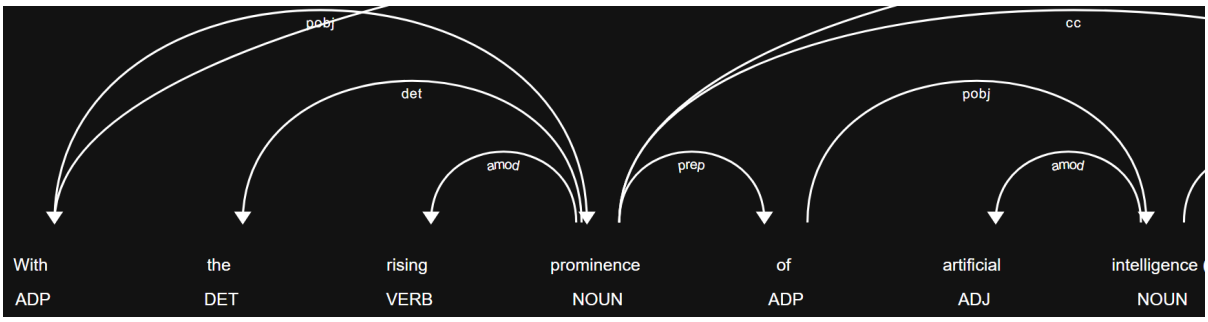


Figure 5.1: Dependency graph for a phrase in the abstract

This syntactic structure helps break down the core components of the abstract, making for a faster and better understanding of how the linguistic elements interact to create meaning. It also allows for a more refined extraction of key concepts from the text for subsequent analysis.

5.3.1.3 Named Entity Recognition

In addition to syntactic parsing, `spaCy`'s NER capabilities were used to identify important entities within the text. The entities recognized in the abstract are shown in Listing 5.4.

Listing 5.4: Named entities extracted using SpaCy

```
AI (ORG), Llama 2 (PRODUCT), healthcare (ORG),
agile methodology (METHODOLOGY), GPTs (PRODUCT)
```

These entities represent important ideas and technological advancements mentioned in the text, specifically:

- *'AI'* and *'healthcare'* are identified as organizations (`ORG`); this reflects the wide range of industries in which the discussed technologies are being used.
- *'Llama 2'* and *'GPTs'* are categorized as products (`PRODUCT`), which emphasizes these cutting-edge AI models and their applicability across a range of industries.
- *'Agile methodology'* is correctly recognized as a methodology (`METHODOLOGY`), showing its importance in iterative, flexible development processes.

By organizing the main ideas in the text, this named entity recognition helps contextualize how these ideas and technologies were applied in the project report and how they relate to each other. Naturally, this concept can be applied to any kind of textual content to identify its central themes.

5.3.1.4 Noun Chunks

A noun chunk is a continuous group of words that together represent a noun in a sentence, usually consisting of a noun and its modifiers. These modifiers refer to meaningful units, such as adjectives or articles (Cloud, 2024). In this NLP analysis, the noun chunks extracted using `spaCy` from the text are shown in Listing 5.5.

Listing 5.5: Noun chunks extracted using SpaCy

```
the rising prominence
artificial intelligence
AI
generative pre-trained transformers
GPTs
various domains
our project
these technologies
a specific need
```

These noun chunks highlight important discourse topics and help identify the main points of emphasis within the text. For example, the technologies at the center of the project are characterized by terms such as *'generative pre-trained transformers'* and *'artificial intelligence'*. Similarly, *'our project'* and *'a specific need'* pinpoint the project's relevance to healthcare, while *'the rising prominence'* and *'various domains'* situate these technologies in a broader context.

Noun chunk extraction provides clear illustrations of important concepts and simplifies other processing activities including entity extraction and summarization.

5.3.1.5 Measuring NLP Performance

Using the same method to measure memory usage and response time as with the PyRAT API request from Section 5.1.1.1, the results are as follows: Memory usage before querying the Zotero API was at **90.72 MB**, while after applying the NLP techniques, it increased to **140.45 MB**. This resulted in a memory difference of **49.73 MB**. Furthermore, the total execution time for this approach was measured at **5.89 seconds**.

5.3.2 Large Language Models

Here, the `Qwen2.5-1.5B-Instruct` LLM was used to generate metadata by analyzing textual content, specifically the title and abstract of the project report on AI-powered early detection of dementia. The following is a summary of the findings that were observed.

The derived metadata, as shown in Listing 4.6, demonstrates that the `Qwen` model effectively recognized and categorized various components of the document, including the *'abstract,' 'title,' 'authors,'* and *'keywords,'* all in a structured JSON format. Each element was accurately identified, suggesting the model's strong capability in analyzing and organizing complex textual data.

While it identified and provided an accurate abstract of the project report, it was not identical to the actual abstract; instead, it was a condensed version that retained the essential information. Additionally, the model added some unprompted attributes, such as *'conference,' 'copyright,'* and *'paper_id,'* which were not requested but appeared as empty strings or marked as N/A. These fields could potentially be filled with relevant data in the future, promoting reusability.

5.3.2.1 Measuring LLM Performance

Due to the heavy requirements of an LLM, the model was run on a different setup compared to the previous performance tests. Specifically, it was deployed on an `NVIDIA GeForce RTX 4090` GPU. Based on the input and conversation length, both memory usage and response time varied. On average, the memory usage, as measured by `nvidia-smi`, was around **3.4 GB**, and the response time, which includes the time to load the model and the time between the first query and the first response, typically ranged between **1.5 and 2 minutes**.

In conclusion, both NLP techniques using `spaCy` and LLMs like `Qwen2.5-1.5B-Instruct` are effective for analyzing and inferring metadata from plain text. While `spaCy` offers detailed syntactic and semantic analysis, the `Qwen` model excels at identifying and structuring relevant document components. These approaches contribute significantly to the efficient processing and analysis of textual data, with distinct trade-offs in terms of computational resources and processing time.

6 Discussion

This chapter reflects on the outcomes of the metadata extraction, transformation, and analysis techniques, evaluating the challenges, limitations, and broader implications encountered during the process. The relative advantages and constraints of imperative programming, natural language processing, and large language models are compared, offering valuable insights for future developments.

6.1 Comparing Metadata Extraction Techniques

This section compares different methods of metadata extraction in PyRAT, including the use of API endpoints versus web scraping. It also examines the trade-offs between manual and automated scraping. The goal is to highlight the most effective approach for extracting metadata in this context.

The endpoints provided a more reliable and consistent method for metadata extraction compared to web scraping. They enabled programmatic retrieval of structured metadata without relying on the session state or user interaction. Web scraping, on the other hand, presented significant challenges, which ultimately limited its usefulness for large-scale metadata extraction.

While automated scraping using libraries such as `requests` and `BeautifulSoup` to parse HTML content had the potential to improve metadata retrieval efficiency, session handling issues meant that manual scraping was the more effective option in this particular case, albeit time consuming.

Direct API calls were ultimately more reliable, efficient, and scalable for metadata extraction, offering a structured format (JSON) that was easy to work with. While manual web scraping could serve as a workaround when direct API access is unavailable, it is harder to scale for large datasets.

6.2 Comparing Metadata Transformation Techniques

The following compares the outcomes of converting HTML metadata into JSON format using conventional imperative programming and a large language model. Regarding data structure and suitability for various cases, both approaches exhibit unique features.

The imperative programming approach offers precise control, while preserving relationships between headings and content. However, the absence of inherent hierarchical organization in the input requires additional effort to accurately identify and construct a suitable structure. This means that when the input structure changes, this method can lead to suboptimal results.

In contrast, the LLM-generated output resolves these issues by directly pairing keys with values, grouping related elements logically, and providing descriptions that promote interpretability.

From the gathered results, it can be concluded that while both imperative programming and large language models are theoretically capable of converting HTML metadata into structured JSON formats, in practice, the LLM approach offers significant advantages in terms of reliability and flexibility, as it demonstrates a more user-friendly and insightful approach to structuring complex metadata.

6.3 Comparing Data Analysis Techniques

As follows, Table 6.1 compares natural language processing techniques with large language models with regard to data analysis. It directly contrasts both methods, revealing their benefits and drawbacks in different scenarios.

NLP tools such as `spaCy` are best for simple and structured tasks with clear linguistic features. They offer interpretability and lower resource requirements, making them appropriate for specific applications in data analysis, such as processing text documents.

LLMs like `Qwen2.5-1.5B-Instruct` are excellent in terms of flexibility and performance, solving a wide array of tasks without additional task-specific training. They are especially powerful for complex and unstructured data analysis, though they demand significantly more computational resources.

6.4 Challenges and Limitations

While the methodologies and tools described in this work offer significant advantages in unifying metadata, numerous challenges emerged during their implementation. These problems emphasize the complexities of integrating several technologies, as well as the trade-offs between precision and scalability. This section discusses the restrictions experienced while developing and using

Aspect	NLP tools (e.g., SpaCy)	LLMs (e.g., Qwen)
Resource Allocation	Typically require less computational power and memory compared to LLMs. Can often run efficiently on CPUs.	Require significant computational resources depending on the model. Larger models often require GPUs.
Flexibility	Highly task-specific; excel in dependency parsing, NER, and POS tagging but need custom programming for new tasks.	Inherently flexible; adapt to a wide variety of tasks such as summarization, translation, and data extraction without task-specific retraining.
Performance	Perform well for predefined linguistic tasks (e.g., syntactic analysis, NER) and deliver predictable outputs for defined workflows. Faster response times.	Perform better in understanding and generating human-like text, especially in nuanced contexts and unusual formats. Slower response times.
Ease of Use	Often require more programming knowledge to set up and optimize for specific tasks.	Usually simpler to use via APIs or interfaces, allowing non-experts to make use of their features.
Analytical Approach	Rely on rule-based syntactic and semantic analysis; structured data extraction is possible but requires predefined workflows.	Utilize deep learning to understand and generate insights; seamlessly extract structured and unstructured data from complex documents.
Use Cases	Ideal for basic linguistic analysis, dependency parsing, and noun chunk extraction.	Suitable for advanced applications, such as metadata generation, summarization, and structuring text.

Table 6.1: Comparison of NLP and LLMs

imperative programming, integrating third-party tools including PyRAT, and managing system dependencies, such as dealing with compatibility issues in software like Docker Desktop. These findings provide important lessons that will help refine current approaches and direct future research in this field.

The challenges surrounding PyRAT demonstrate the complexity of integrating third-party tools into metadata operations. For the web interface, although the automated web scraping method seemed promising, it was ultimately unable to deliver the intended results. When executing the scraping script, the system redirected to a default welcome page instead of the expected resource. Although the returned page included the anticipated header, body, and footer content, it was missing the necessary particular metadata. As mentioned in Section 5.1.3, this failure was attributed to the requirement for a valid `'session-id'` cookie, which could not be programmatically obtained. Consequently, access to metadata was restricted to manual interaction through a browser session. This limitation severely reduced the potential of the scraping method for automatically retrieving metadata. As a result, metadata from the web interface could only be accessed manually through a browser session. Access to the test version of the PyRAT API presented further challenges. The base URL and endpoints of the API were frequently updated

as it evolved, making previous scripts obsolete and requiring frequent adjustments.

In data extraction workflows, imperative programming offers precise control over text retrieval but presents significant limitations in terms of flexibility and scalability. For example, the text extraction script in `chatbot_paper.py` is tailored to specific languages and keywords. Although it currently supports 9 languages, the script can face challenges with varying terminology, as different publishers may use synonyms or alternative terms that are not included in the list. Furthermore, it may stop extracting the abstract prematurely if words like "Introduction," "Background," or "Motivation" appear within the Abstract section, mistakenly identifying them as the start of the introduction, even before the actual introduction begins. Similarly, transforming HTML documents to JSON requires documents with a consistent structure, which limits adaptability to diverse formats. Because of these restrictions, imperative programming can be effective for narrowly defined tasks but more challenging for broader applications. This further promotes the use of LLMs as they provide a more versatile alternative and can adapt to varying input structures and languages without requiring language-specific hard coding.

For Docker—the platform used to containerize and manage the Local Data Hub—, the compatibility issues with the latest version presented a substantial hurdle. Docker Desktop required a minimum of Windows 10 Pro or Enterprise build 199044, but the system in use was capped at build 119042. Manual registry adjustments were required in order to bypass this restriction. Specifically, the process involved accessing the Registry Editor (`regedit`) and navigating to the following path: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion`. The `CurrentBuild` and `CurrentBuildNumber` entries were manually updated to 199044 to meet Docker's requirements (Jlasssi, 2024). This workaround eventually enabled installation, but it highlighted the more general difficulties in managing infrastructure dependencies, especially when using programs that have stringent system requirements.

These challenges and limitations not only highlight the intricacies of the current methods but also pave the way for future developments to refine existing methods, improve system compatibility, and provide more reliable solutions.

7 Conclusion and Future Work

In conclusion, this thesis explored numerous opportunities and methodologies associated with metadata management and evaluation in research workflows.

Utilizing PyRAT, RSpace, and Zotero, the extraction pipeline provides the foundation that is needed for retrieving metadata and structuring it from diverse research sources. While traditional imperative programming guarantees precise control over data transformations, it is limited by language-specific dependencies and document structure limitations. The introduction of large language models, specifically the `Qwen2.5-1.5B-Instruct` model, offers greater flexibility, allowing for the processing of complex metadata and enhancing transformation techniques such as converting HTML—commonly parsed from most websites—to the simple and universal JSON format. The analysis layer employs natural language processing and LLMs, thus enabling the extraction of valuable insights from textual data, while the integration layer, enables efficient storage of the processed metadata in repositories like the Local Data Hub for further research purposes.

Throughout the stages of metadata extraction, processing, and integration, particular attention has been given to ensuring that the resulting metadata adheres to the FAIR principles—findable, accessible, interoperable, and reusable. The integration of metadata from multiple sources into the NFDI4Health LDH creates a centralized repository where the research data is unified and serialized into a standardized JSON format. This integration ensures that the metadata meets the following FAIR criteria:

Findable: Metadata from diverse research systems is unified into a structured format, ensuring that it can be effectively indexed within the Local Data Hub. Each data point, whether it's animal metadata from PyRAT, experimental data from RSpace, or academic papers from Zotero, is tagged with clear identifiers and categorized attributes. This structure supports easy discovery through search tools that allow users to quickly locate the relevant metadata.

Accessible: The deployment of the LDH is straightforward, utilizing a user-friendly interface that simplifies access. Additionally, the platform allows for the seamless publication of metadata, enabling researchers to share their data efficiently.

Interoperable: By converting the metadata into a universal format (JSON), it becomes

easily interoperable with other systems. This conversion enables the effective integration and application of metadata from PyRAT, RSpace, and Zotero, regardless of their different formats, within various platforms and research tools. Furthermore, the multilingual support in both Zotero's text extraction and `Qwen` helps overcome linguistic barriers and promote interoperability.

Reusable: The structured format and clear categorization enable the reuse of metadata in a variety of research contexts. This unification process allows researchers to repurpose data for subsequent analyses without the need for extensive reformatting. In the LDH, integrated metadata remains usable even after shutting down, and restarting containers, ensuring data continuity.

However, obstacles such as availability issues with the PyRAT API and limitations of web scraping methods emphasize the need for further refinement in this metadata unification framework. Ultimately, the potential for future advancements is highlighted by these difficulties, particularly in improving the reliability and adaptability of both extraction and integration procedures.

7.1 Directions for Future Work

While this thesis has made significant progress in managing and evaluating metadata in research workflows, particularly in ways that adhere to the FAIR principles, some areas remain open for further development and exploration.

First, the extraction method has to be made more streamlined. The files that were unified still required some degree of manual access, either to retrieve the file itself or to obtain its identifier. This might be improved by developing the system to allow for the automated extraction of several files and documents, similar to importing an entire Zotero library, for instance. This would allow for more efficient processing by enabling the retrieval of multiple files and datasets, without any manual intervention. Implementing this functionality would increase the system's scalability, ensuring that it can handle large volumes of metadata more effectively.

Regarding the integration layer, future work could focus on implementing API functions—similar to those used in PyRAT—within the NFDI4Health LDH, to directly integrate and manage metadata through third-party clients. Such API features would simplify access to the research database, eliminate the need for manual interventions, and provide a more automated approach for data integration. By offering a straightforward method for importing and integrating metadata, this improvement would streamline the process and result in faster and easier data management.

Addressing these limitations and expanding the functionality of the existing system will further advance the metadata unification techniques in the field of science, enabling AI-powered analysis from diverse research systems for efficient data management and evaluation.

List of Figures

Listings

List of Tables

Bibliography

Baca, M. (2016). *Introduction to Metadata*. Getty Publications. URL: https://books.google.com/books?hl=en&lr=&id=xgZVDQAAQBAJ&oi=fnd&pg=PP1&dq=introduction+to+metadata&ots=bbPscN2iO1&sig=jwqYNWleSvDIxLG3XyKZr9bJliY.

Bernardi, F. A., D. Alves, N. Crepaldi, D. B. Yamada, V. C. Lima, and R. Rijo (2023). "Data quality in health research: integrative literature review". In: *Journal of Medical Internet Research* 25, e41446.

Cloud, N. (2024). *NLP Noun Chunks - Noun Phrase Extraction API*. Copyright: NLP Cloud. URL: https://nlpcloud.com/nlp-noun-chunks-noun-phrase-extraction-api.html (visited on 10/26/2024).

Cloudflare, I. (2024). *What is a Large Language Model?* URL: https://www.cloudflare.com/learning/ai/what-is-large-language-model/ (visited on 11/17/2024).

Corporation for Digital Scholarship, R. R. C. f. H. and N. Media (2024). *Zotero*. URL: https://www.zotero.org/about/ (visited on 11/16/2024).

Couwenbergh, S. (Aug. 2022). *The Importance of Cleaning Dirty Data for Improved Operations and Customer Success*. Accessed: 2024-12-04. URL: https://www.validity.com/blog/dirty-data/.

Docker (2024). *WSL 2 Backend for Docker Desktop*. Copyright: © 2013-2024 Docker Inc. All rights reserved. URL: https://docs.docker.com/desktop/wsl/ (visited on 11/10/2024).

Greenberg, J., M. Wu, W. Liu, and F. Liu (Mar. 2023). "Metadata as Data Intelligence". In: *Data Intelligence* 5.1, pp. 1–5.

Hu, X., H. Abaza, R. Hänsel, M. Abedi, M. Golebiewski, W. Müller, and F. Meineke (2024). "NFDI4Health Local Data Hubs Implementing a Tailored Metadata Schema for Health Data". In: *German Medical Data Sciences 2024*. IOS Press, pp. 115–122.

IBM (2024). *What is Machine Learning?* Copyright: IBM. URL: https://www.ibm.com/think/topics/machine-learning (visited on 12/23/2024).

Jeusfeld, M. A. (2009). "Metadata". In: *Encyclopedia of Database Systems*. Ed. by L. LIU and M. T. ÖZSU. Boston, MA: Springer US, pp. 1723–1724. URL: https://doi.org/10.1007/978-0-387-39940-9_893.

Jlasssi, N. (2024). *Answer to: "Can't install Docker: Docker Desktop requires Windows 10 Pro or Enterprise version"*. Stack Overflow. URL: https://stackoverflow.com/questions/60809467 (visited on 11/25/2024).

json.org (2024). *Introducing JSON*. URL: https://www.json.org/json-en.html (visited on 11/10/2024).

Lee, G. Y., L. Alzamil, B. Doskenov, and A. Termehchy (2021). "A survey on data cleaning methods for improved machine learning model performance". In: *arXiv preprint arXiv:2109.07127*.

Leipzig, J., D. Nüst, C. T. Hoyt, K. Ram, and J. Greenberg (2021). "The role of metadata in reproducible computational research". In: *Patterns* 2.9, p. 100322. URL: https://www.sciencedirect.com/science/article/pii/S2666389921001707.

Liu, H., T. Christiansen, W. A. Baumgartner, and K. Verspoor (2012). "BioLemmatizer: a lemmatization tool for morphological processing of biomedical text". In: *Journal of biomedical semantics* 3, pp. 1–29.

Meineke, F. and J. Wagner (2024). *LDH Deployment*. GitHub repository. URL: https://github.com/nfdi4health/ldh-deployment (visited on 10/26/2024).

Nazi, Z. A. and W. Peng (2024). "Large language models in healthcare and medical domain: A review". In: *Informatics*. Vol. 11. 3. MDPI, p. 57.

Press, N. (2004). *Understanding metadata. National Information Standards Organization*. URL: https://www.lter.uaf.edu/metadata_files/understandingmetadata.pdf.

Priya, B., J. Nandhini, and T. Gnanasekaran (2021). "An analysis of the applications of natural language processing in various sectors". In: *Smart Intelligent Computing and Communication Technology*. IOS Press, pp. 598–602.

Qwen (2023). *Qwen2.5-1.5B-Instruct*. Copyright: Qwen Team. URL: https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct (visited on 11/08/2024).

RSpace (2024a). *Getting Started with RSpace ELN*. Copyright: Research Space. URL: https://documentation.researchspace.com/article/pfsj1e1u7j-getting-started (visited on 11/17/2024).

— (2024b). *RSpace: Innovations in Research Data Management*. Copyright: © 2024 Research Space. URL: https://www.researchspace.com (visited on 10/25/2024).

— (2024c). *Export Formats in RSpace*. Copyright: Research Space. URL: https://documentation.researchspace.com/article/s3j29if453-export-formats (visited on 11/17/2024).

— (2024d). *Export Options in RSpace*. Copyright: Research Space. URL: https://documentation.researchspace.com/article/25mt56kamf-export-options (visited on 11/17/2024).

Scionics (2024a). *PyRAT API-v2 Documentation*. Copyright: Scionics Computer Innovation GmbH. URL: https://www.zotero.org/groups/296328/digital_health_group/collections/WEEFJBJW/items/ZBBNSUEP/item-details (visited on 10/17/2024).

— (2024b). *PyRAT API-v3 Endpoints*. Copyright: Scionics Computer Innovation GmbH. URL: https://pyrat.ukbonn.de/pyrat-test/api/v3/docs (visited on 11/17/2024).

— (2024c). *PyRAT: Animal Facility Software*. Copyright: Scionics Computer Innovation GmbH. URL: https://www.scionics.com/pyrat.html (visited on 11/16/2024).

Stryker, C. and J. Holdsworth (2024). *Natural Language Processing*. Copyright: IBM, Updated: 2024-08-11. URL: https://www.ibm.com/topics/natural-language-processing (visited on 11/17/2024).

Stryker, C. and E. Kavlakoglu (2024). *What is AI?* Copyright: IBM. URL: https://www.ibm.com/think/topics/artificial-intelligence (visited on 12/23/2024).

Thirumal, M. and S. Monika (2024). "Transforming Healthcare: The power and potential of digital medicine". In: *Future Science OA* 10.1, p. 2430357.

Walker, D. M., W. L. Tarver, P. Jonnalagadda, L. Ranbom, E. W. Ford, and S. Rahurkar (2023). "Perspectives on challenges and opportunities for interoperability: findings from key informant interviews with stakeholders in Ohio". In: *JMIR Medical Informatics* 11.1, e43848.

Wilkinson, M. D., M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. (2016). "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific data* 3.1, pp. 1–9.

Xie, Q., Q. Chen, A. Chen, C. Peng, Y. Hu, F. Lin, X. Peng, J. Huang, J. Zhang, V. Keloth, X. Zhou, H. He, L. Ohno-Machado, Y. Wu, W. Qi, and J. Bian (May 2024). "Me-LLaMA: Foundation Large Language Models for Medical Applications". In: *Research square*.

Yamada, M., H. Kitagawa, T. Amagasa, and A. Matono (2023). "Augmented lineage: traceability of data analysis including complex UDF processing". In: *The VLDB Journal* 32.5, pp. 963–983.