Information Storage and Retrieval - CS5604
Final Report
Team 1 (Knowledge Graphs)

Daniel Gaugler* Ashish Aggarwal † Instructor: Dr. Edward Fox‡

Mentor: Satvik Chekuri§

12-13-2023

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia, 24061

*Master of Engineering Candidate, Department of Computer Science, Virginia Tech, danielg19@vt.edu
†Master of Engineering Candidate, Department of Computer Science, Virginia Tech, ashish1702@vt.edu
‡Professor, Department of Computer Science, Virginia Tech, fox@vt.edu
§Doctoral Student, Department of Computer Science, Virginia Tech, satvikchekuri@vt.edu

Contents

1 Abstract **iii**

2 Tables and Figures **iv**

3 Overview **1**
3.1 Work Completed . . . 1
3.2 Scope of Further Work . . . 2

4 Literature Review **3**
4.1 Knowledge Graph . . . 3
4.2 Querying RDF Triple Databases . . . 3

5 Requirements **5**
5.1 MVP Requirements . . . 5
5.2 Stretch Goals . . . 5

6 Design **6**
6.1 Approach . . . 6
6.2 Knowledge Graph . . . 7
6.2.1 Resource Description Framework (RDF) . . . 7
6.2.2 RDF Triple . . . 7
6.2.3 ETD Graph Representation . . . 9
6.3 URI Resolution . . . 10
6.3.1 Overview . . . 10
6.3.2 URI Resolution Schema . . . 10
6.4 Querying the Knowledge Graph . . . 11
6.5 Data Schema . . . 12
6.5.1 ETD Metadata . . . 12
6.5.2 RDF Triple Schema . . . 13
6.6 Tools . . . 16

7 Implementation **17**
7.1 Overview . . . 17
7.2 List of Milestones and Deliverables . . . 19
7.3 Timeline . . . 20

8 Future Work **21**
8.1 Securing the SPARQL Endpoint . . . 21
8.2 Expanding the ETD Dataset . . . 21
8.3 Implementing Automated Data Pipelines . . . 21
8.4 Connecting the Search and SPARQL Interfaces . . . 21
8.5 Knowledge Graph of Entities from ETDs . . . 21

9 User Manual **23**
9.1 Accessing the Query Interface . . . 23
9.2 Querying for Records . . . 23
9.3 Viewing Details of a Record . . . 24

10 Developer Manual **26**
10.1 URI Resolution Service 26
10.2 ETD Processing and Triple Generation Script 27
10.2.1 Analysis Script 28
10.2.2 Usage 28
10.3 Virtuoso Database 29
10.4 CI/CD 29
10.4.1 URI Resolution Service 29

11 Bibliography **31**

1 Abstract

This project presents an ontological framework designed to enhance the utility of Electronic Theses and Dissertations (ETDs) by transforming them into a semantically enriched Knowledge Graph representation. The central technical approach leverages widely accepted standards such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL) to convert ETD metadata into machine-readable RDF triples, each associated with unique Uniform Resource Identifiers (URIs). RDF schema and OWL ontologies are used to explicitly define the classes, properties, and relationships of entities within the ETD domain. By constructing a structured ETD Knowledge Graph, we enable the encoding of rich semantics and interconnections between entities based on these predefined ontologies. A major innovative aspect of this framework is the integration of semantic search capabilities that allow for complex contextual queries by capitalizing on the inherent graph structure. This search functionality relies on the standards-based SPARQL Protocol and RDF Query Language (SPARQL). The query processing mechanism employs graph traversal algorithms, which empower users to perform in-depth exploratory searches, unveiling non-obvious insights and connections from the Knowledge Graph. Our project commenced by analyzing a corpus of roughly 200 ETDs to comprehend the structure and relationships within the constituent components of an ETD. Utilizing this study, we compiled 76 predicates, 7 objects, and 37 subjects which comprehensively encompass the relationships within the ETD in a top-down and bottom-up ontology. The total number of triples generated for the 200 ETDs was roughly 95,000. This brings the average number of triples per ETD to around 600. Based upon this ontology, we process the ETDs which we ingest in an XML format to generate and persist the RDF triples within the Virtuoso database. We provide a SPARQL query interface which is utilized to execute heterogeneous queries upon the repository of this ETD data. To empower our data pipeline and back-end system, we harnessed modern technological architectures including event-driven programming, Representational State Transfer (REST) APIs, version control, web technologies, and Agile methodologies. To harness ancillary metadata for the URIs stored within the Knowledge Graphs, we architected a URI resolution microservice which integrates seamlessly with the central PostgreSQL database. To ensure scalability, our framework utilizes dockerized methodologies, which are deployed within the cloud ecosystem of Virginia Tech that leverages distributed computing techniques. This configuration enables the processing of RDF graphs, even when dealing with very large ETD datasets. The cluster infrastructure is equipped with horizontal scalability, ensuring that it can efficiently handle growing Knowledge Graphs and increasing query workloads. In this context, the OpenLink Virtuoso graph database is utilized to efficiently store and index the ontology-based entities and their relationships. To automate the data pipeline and achieve scalability for data storage in Virtuoso, we aim to use Kafka's event based architecture. We have laid down the groundwork in terms of design but its implementation could not be completed this semester due to complications within other teams. We also discuss, as part of future work, extensions related to the doctoral plans of Satvik Chekuri. To effectively represent ETDs and capture semantic information, the KG should reflect both intra- and inter-document entity-entity relations. The KG structure, as a directed heterogeneous graph with domain-related semantics, addresses the limitations of transformer models in handling long documents. We propose building an entity-based KG for a document collection by extracting entity-related triples using improved OpenIE and NER methods, enriching the graph for improved coverage and performance in downstream tasks.

2 Tables and Figures

List of Tables

1 RDF Triple Table of SPARQL Query Result . . . 12
2 Definitions in the ETD Knowledge Graph . . . 13
3 RDF Top-Down Schema for the Knowledge Graph . . . 14
4 RDF Bottom-Up Schema in the ETD Knowledge Graph . . . 15
5 Task Assignments, Status and Due Dates . . . 20
6 Predicate Term Counts . . . 28

List of Figures

1 High Level Design for the Service . . . 6
2 Representation of an RDF Triple . . . 7
3 Example Representation of a Top-Down RDF Triple . . . 8
4 Example Representation of a Bottom-Up RDF Triple . . . 8
5 Top-Down Knowledge Graph Representation for an ETD . . . 9
6 RDF Triple URI Example for ETD's Chapter . . . 10
7 RDF Triple URI Example for Author . . . 11
8 SPARQL Query Example . . . 11
9 Default Virtuoso SPARQL Interface . . . 18
10 URI resolution service response for "abstract" predicate search of ETD with id 12345 19
11 Retrieve all triples related to a specific subject. . . . 23
12 Retrieve all triples related to a specific predicate. . . . 24
13 Retrieve all triples related to a specific subject and predicate. . . . 24
14 Results of the query that filters for both predicate and subject. . . . 25
15 Clicking on the Subject URI to view details. . . . 25
16 High-level view of XML Schema. . . . 27
17 Docker file for URI Resolution Service . . . 29
18 YAML file for URI Resolution Service . . . 30

3 Overview

The significance of theses and dissertations cannot be overstated, as they encapsulate invaluable research contributions. Electronic Theses and Dissertations (ETDs) have become a prominent mode of submission, with over 39,000 ETDs curated and preserved within VTechWorks. Initially, these documents existed solely in hard copy form, presenting challenges in preservation and accessibility. Virginia Tech pioneered electronic submission on January 1, 1997 by becoming the inaugural university to mandate electronic submission of theses and dissertations, marking a new era in digital archiving. This forward-looking policy allowed students to submit their research in diverse digital formats, including PDFs.

In the past two decades, universities across the nation have adopted digital repositories for Electronic Theses and Dissertations (ETDs). This widespread adoption has resulted in a vast and diverse collection of ETDs that are readily accessible in digital formats. This enhanced accessibility has ushered in a transformative era for researchers, graduate students, and knowledge seekers, breaking down geographical barriers. Our research initiative aims to extend the global reach of students' research efforts, overcoming the constraints of traditional archival practices. It is a paradigm shift that empowers academics to leverage the digital landscape for a revolutionary advancement in scholarly exploration and understanding. Semantically searchable ETDs will grant scholars the ability to not only access these documents but also to formulate and perform precise and targeted searches within this extensive ETD repository stored in the form of Knowledge Graphs that unlock unseen insights.

This endeavor seeks to democratize access to ETD data by constructing Knowledge Graphs that interconnect ETD metadata, preserving their intricate relationships with other elements within the ETD corpus. The integration of semantic enrichment for this largely textual data aims to facilitate seamless navigation for readers, users, researchers, and educators. For instance, the Knowledge Graph will establish links between figures and their corresponding captions, as well as between chapters and the figures, tables, and chapters nested within them.

3.1 Work Completed

We initiated our project by selecting Virtuoso as our database solution due to its extensive functionality and open-source nature. Concurrently, we explored various APIs for integration. Our Subject Matter Expert, Satvik Chekuri, compiled a list of relevant objects, subjects, and predicates, which laid the groundwork for our project.

We began by setting up the framework for uploading RDF triples into Virtuoso, enabling structured storage and retrieval of semantically enriched data. Simultaneously, we used a temporary JSON conversion to bridge the gap between the original XML format and Virtuoso. We then proceeded to convert this JSON data into RDF triples in the desired N3 Format [14]. This conversion was executed using the Python programming language and suitable public packages, like RDFLib [15], representing a step in the population of our Knowledge Graph.

With this architecture and leveraging the available 200 ETDs, we constructed an initial version of our Knowledge Graph, which we were able to query via SPARQL. This solution has been hosted on the Endeavour cluster, with an endpoint for ease of access. Additionally, we conducted research to determine a suitable structure for Unified Resource Identifiers (URIs). These URIs are integral to our efforts to make enriched ETD content accessible through web pages.

During our discussions on potential use cases and queries our system may encounter, we refined our ontology. This refinement involved updating existing values and introducing new predicates, particularly focusing on addressing the bottom-up hierarchy and relationships within the ETD dataset.

These milestones represent significant progress in our goal of transforming ETD data into a structured and semantically enriched format.

3.2 Scope of Further Work

With the above work completed, we set out a list of future work that is necessary to achieve a fully functional service.

- **Triple Insertion**
  - Automate the population of the Virtuoso database utilizing OpenLink APIs.
- **Knowledge Graph Construction**
  - Populate the graph with final persistent Object IDs.
- **Semantic Search**
  - Identify more example SPARQL queries.
- **URI resolution Service**
  - Integrate with the PostgreSQL database.
  - Finalise UI for displaying the content.

4 Literature Review

4.1 Knowledge Graph

Knowledge Graphs, with their representation of information as nodes denoting entities and edges signifying relationships, have evolved as a pivotal paradigm in information organization and retrieval. This approach has garnered widespread applications across various domains, including semantic search, data integration, and knowledge representation [2]. The creation and maintenance of Google's Knowledge Graph, one of the largest and most comprehensive Knowledge Graphs to date, has not only revolutionized web search but has also laid the foundation for numerous applications that capitalize on this rich source of interconnected information [3].

Wikipedia, a cornerstone in the digital dissemination of knowledge, has played a pioneering role in the development of Knowledge Graphs. The Wikimedia Foundation's Wikidata project, an integral part of the Wikimedia ecosystem, serves as a centralized repository of structured data that powers Knowledge Graphs for Wikipedia and beyond. Wikidata encapsulates a wealth of information, including entities, their attributes, and interrelationships. This structured data forms the backbone of Wikipedia's Knowledge Graph, allowing for the seamless integration of diverse information across a multitude of topics [7]. The Knowledge Graph enriches the browsing experience for Wikipedia users by providing contextual information, connections between topics, and additional insights into related subjects. This resource has not only enhanced the depth and breadth of information available on Wikipedia but has also become a valuable asset for a myriad of applications across the academic, research, and information retrieval domains [8].

In addition to its querying capabilities, Virtuoso incorporates a full-text search engine, which enables efficient text-based searches within RDF data. This feature is particularly crucial for applications reliant on text search functionality. The database's automatic metadata extraction streamlines data organization and indexing, enhancing search capabilities and enabling users to extract insights from a wide range of diverse data sources. Virtuoso's support for standard APIs such as JDBC, ODBC, and OLE DB ensures its compatibility with a broad spectrum of applications and development environments. Lastly, its built-in support for geospatial data provides advanced spatial query and analysis capabilities, a key asset for applications that rely on location-based data for decision-making and analysis [2].

Furthermore, research efforts have focused on Knowledge Graph embeddings and representation learning. These approaches aim to embed entities and relationships into continuous vector spaces, facilitating machine learning applications on Knowledge Graphs. Techniques like TransE, DistMult, and ComplEx have shown promise in tasks such as link prediction and entity classification [9]. The database's data virtualization capabilities are another strong feature, allowing users to access and query data from diverse sources, including external RDF datasets and web services, as though they were part of a unified database. This capability simplifies and streamlines data integration processes, particularly in contexts where multiple data sources need to be harmonized.

In previous years of this project the team planned to implement the knowledge graph as we have. It's important to note the entire structure of the overall system as well. Our Knowledge Graph and its application only comprises a small part of a much larger digital library system for storing and retrieving ETDs. The services in this digital library cover ingesting, storing, curating, analyzing, classifying, summarizing, topic modeling, searching, recommending, and interacting with ETDs and other elements/objects [19]. It's also worth noting that the knowledge graph has been worked on in years prior to the start of our project [20].

4.2 Querying RDF Triple Databases

RDF serves as the foundational data model for Knowledge Graphs, representing information in the form of subject-predicate-object triples. The subject-predicate-object triples represent two entities

and the relationship they have with one another. Concurrently, the querying of RDF (Resource Description Framework) triple databases has garnered significant attention. RDF databases, exemplified by Virtuoso and Blazegraph, have played instrumental roles in efficiently storing and querying such interconnected data [1].

In this context, SPARQL (SPARQL Protocol and RDF Query Language) standards have played a pivotal role. SPARQL provides a powerful and expressive query language tailored for RDF data, enabling precise retrieval of information from Knowledge Graphs [4]. Moreover, studies have focused on optimizing SPARQL query performance, exploring techniques such as query rewriting, caching, and indexing strategies to enhance efficiency [5].

In summary, the interplay between Knowledge Graphs and querying RDF triple databases constitutes a rich area of research with extensive applications across various domains. The development of large-scale Knowledge Graphs, advancements in querying techniques, and the integration with natural language processing and machine learning have collectively contributed to the growth and impact of this field.

5 Requirements

5.1 MVP Requirements

The following gives baseline requirements for our system, also referred to as the minimum viable product:

- Convert the ETD XMLs received from Team 3 into RDF triples. Store each relationship as a top down triple and a bottom up triple.
  - Store triples top down: higher elements point down to objects contained within them.
    - ETD → has chapter → chapter.
    - Chapter → has figure → figure.
  - Store triples bottom up: elements point up to the objects that contain them.
    - Chapter → chapter of → ETD.
    - Figure → figure of → Chapter.
- Insert RDF Triples into RDF database - Virtuoso.
- Search the digital objects in the Knowledge Graph
  - Perform a search using SPARQL to retrieve all records matching the subject and predicate of the query.
  - Perform a SPARQL search that retrieves all records related to the subject in any way.

The goal of the MVP is to enable users to view the semantic location of an object in an ETD. It is the function of the Knowledge Graph to ensure that this feature is possible. By storing the RDF triples of every objects within every ETD we can display all closely related objects to a given subject by performing a simple search. Because of the nature of Knowledge Graphs and RDF triples we will be able to show the user the nature and relationship of each related object.

5.2 Stretch Goals

The following is a list of the stretch goals associated with the project. These are features or functionality that we wished to add but were unable to complete based on the available bandwidth.

- Provide a UI for directly querying the Knowledge Graph.
- Provide support for Question Answering.
- Complement Elasticsearch by enabling the use of more specialized queries.
- Connect ETD Knowledge Graphs to one another based off reference and citation to form an interconnected Knowledge Graph of all ETDs.
- Extend the Knowledge Graphs with triples that arise from running Named Entity Recognition (NER) on the text within an ETD.

6 Design

6.1 Approach

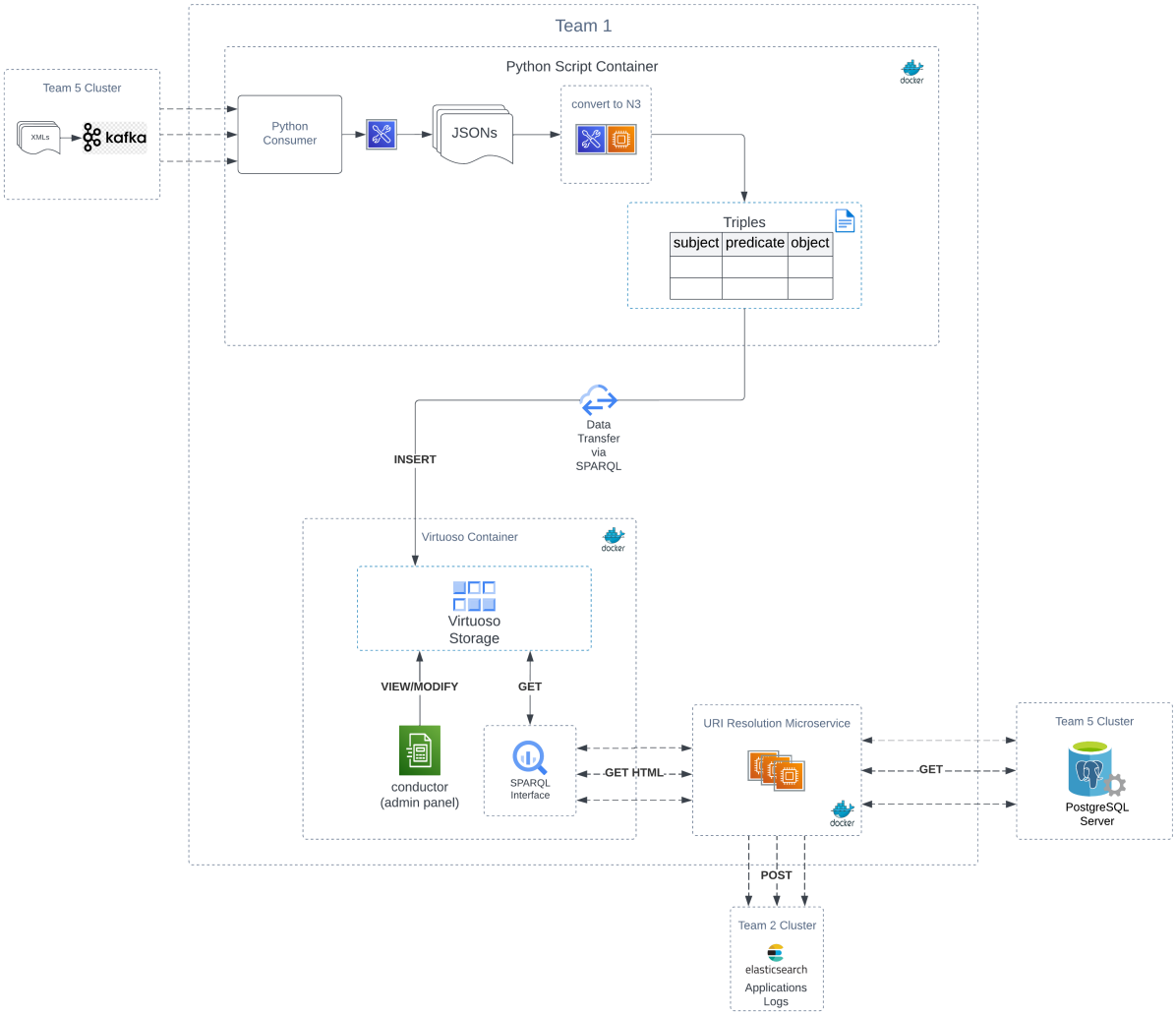

Figure 1: High Level Design for the Service

Overall our approach as displayed in Figure 1 has five distinct parts: RDF Transformation, Triple Insertion, Knowledge Graph Construction, SPARQL Querying, and Deployment. The process begins with RDF transformation, where extracted metadata undergoes conversion into RDF format, establishing a structured representation. Every node representing Electronic Thesis or Dissertation (ETD) metadata is assigned a unique Uniform Resource Identifier (URI), enabling identification and retrieval within the RDF Knowledge Graph. In addition to this, every predicate, which is a value representing the relationship between objects, is also represented by a URI, helping to add structure and uniformity to relationships between nodes. Subsequently, the RDF database is populated with ETD Knowledge Graph triples, forming the foundational structure. These triples establish meaningful connections between nodes, culminating in the construction of a comprehensive Knowledge Graph.

As examined later, relationships between nodes are precisely defined and structured, emphasizing important associations, such as those between documents and their respective chapters or figures, providing a contextual framework.

The development of SPARQL queries tailored to the specific RDF-based Knowledge Graph constitutes the next phase, facilitating precise information retrieval. A query is implemented to interpret SPARQL and RDF-based queries, ensuring a seamless search experience. This search leverages the Knowledge Graph to deliver contextually relevant search results, employing techniques for meaningful extraction. It allows users to search over the contents of every ETD to find objects by their object ID. Additionally, APIs are designed to enable seamless integration with external systems, facilitating easy interaction with the system. Finally, the solution is deployed to Endeavour, a cluster environment provided by the Department of Computer Science at Virginia Tech. This approach combines RDF transformation, Knowledge Graph construction, and SPARQL querying to provide information retrieval from intricate data, providing a tool-set for extracting insights from the ETDs.

6.2 Knowledge Graph

6.2.1 Resource Description Framework (RDF)

RDF serves as a comprehensive framework for representing interconnected data. At its core, it employs URIs to denote nodes and edges within a graph. In the context of this project, RDF statements are utilized to describe and exchange metadata, specifically pertaining to Electronic Theses or Dissertations (ETDs). This framework is particularly valuable in scenarios involving large amounts of complex data, providing a means to parse it into a semantically searchable form. By modeling the data within an ETD – encompassing both the document's metadata and its intricate relationships with chapters, tables, figures, and more – as a Knowledge Graph, we can subsequently store it in the RDF database. This knowledge-driven approach enhances search functionality by yielding results with heightened relevance compared to conventional text-based searches. Moreover, this methodology helps us to effectively conduct searches over digital objects.

6.2.2 RDF Triple

RDF triples are the basic building blocks of data in the Semantic Web [17]. They provide a simple way to represent information in a machine-readable format.

Figure 2: Representation of an RDF Triple

Each RDF triple consists of three parts (as demonstrated by Figure 2):

- Subject: This represents the resource being described. In other words the subject is what the information is about. It is stored as a URI (Uniform Resource Identifier) in our system.
- Predicate: This represents the property or relationship between the subject and the object. Like the subject, it is represented by a URI within our system. The predicate defines the kind of information being expressed such as *title_of*, *chapter_of*, *cited_by*, etc.

- Object: This represents the value of the property or relationship. It can be a URI or a literal value (like a string or a number). The object is what the subject's property is related to.

For our project we will process each ETD and generate RDF triples for all the data contained within it. This includes storing the chapter, figures, tables, abstracts, etc. of an ETD along with their relationships to other elements. In our approach to generating these RDF triples, we will build two sets of triples for each ETD we process. The basic reason for this is that the Knowledge Graph is directed, with each arc having a single direction.

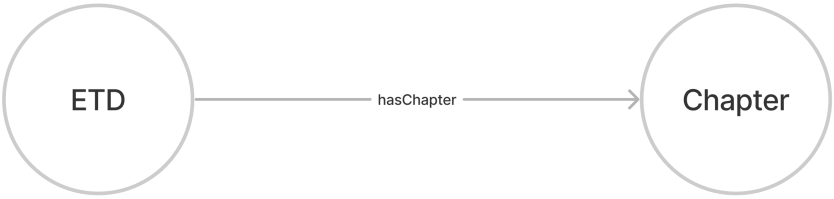

Figure 3: Example Representation of a Top-Down RDF Triple

The first set of triples will follow a top-down hierarchy with respect to how an ETD is organised. The ETD node will be the parent node and all subsections will be related child nodes. This means objects like chapter, author, and other entities will be connected to an ETD by the predicates "hasChapter" and "hasAuthor" (as demonstrated in Figure 3). It also means that because objects like a chapter have other objects inside it, then there must also be triples generated for these objects with the chapter as the subject of the triple.

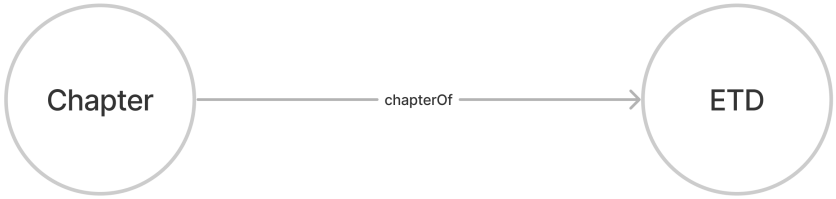

Figure 4: Example Representation of a Bottom-Up RDF Triple

The second set of triples will follow a bottom-up approach (see Figure 4). This approach lets figures, sections, paragraphs and other leaf nodes point back to their chapters and ETDs. This means that objects like chapter and abstract will now be in the subject position of an RDF triple and would connect to the ETD by the predicates "ChapterOf" and "AbstractOf". The objects contained within chapters and other objects would also be stored as the subject in the RDF triples connecting to the chapter or other object that contains them by the appropriate predicate. It is also important to note that because all subjects are stored as URIs and the bottom-up approach to the RDF triples essentially flips the triples to now store the objects of the top-down approach as subjects, then we must store objects as URIs as well. This only applies to objects where we wish for the leaf-node to be query-able and thus need a bottom-up triple as well. Certain objects, that are only stored as objects in the triples, are still stored as text and are only present in the top-down approach because it is not necessary to support the reverse query on these objects.

The reason we must build the Knowledge Graph with two types of RDF triples is to support additional SPARQL queries. Generally, SPARQL queries can search two ways: subject only or subject

plus predicate. Subject only means the query will return all triples where the given subject is present regardless of predicate. If this query were performed on a chapter for instance, the query would return all sections, paragraphs, figures, etc. where the given chapter was the subject of the triple. The subject plus predicate query would be used to search for specific objects located within chapters and ETDs. For example a query could be created to return all chapters of a given ETD by searching with the ETD as the subject and the predicate as "hasChapter". With these two queries we can search for general connections for each subject as well as specific relationships it has.

The problem arises when we try to query leaf nodes. For example say you wanted to see the chapter that a given paragraph is a part of. With only a top-down approach to the Knowledge Graph, this would be impossible to query. Since the paragraph only appears in the top-down triples as an object, we cannot search for it by subject only or subject and predicate. Thus the solution was to build a second set of RDF triple schema to help perform every query we could possibly need to. With the bottom-up approach to triples the leaf nodes are now stored in the subject column and can point to the objects they are contained within. This makes all possible connections query-able. In essence, we are adding inverse arcs so we are able to go in the reverse direction for a directed arc.

6.2.3 ETD Graph Representation

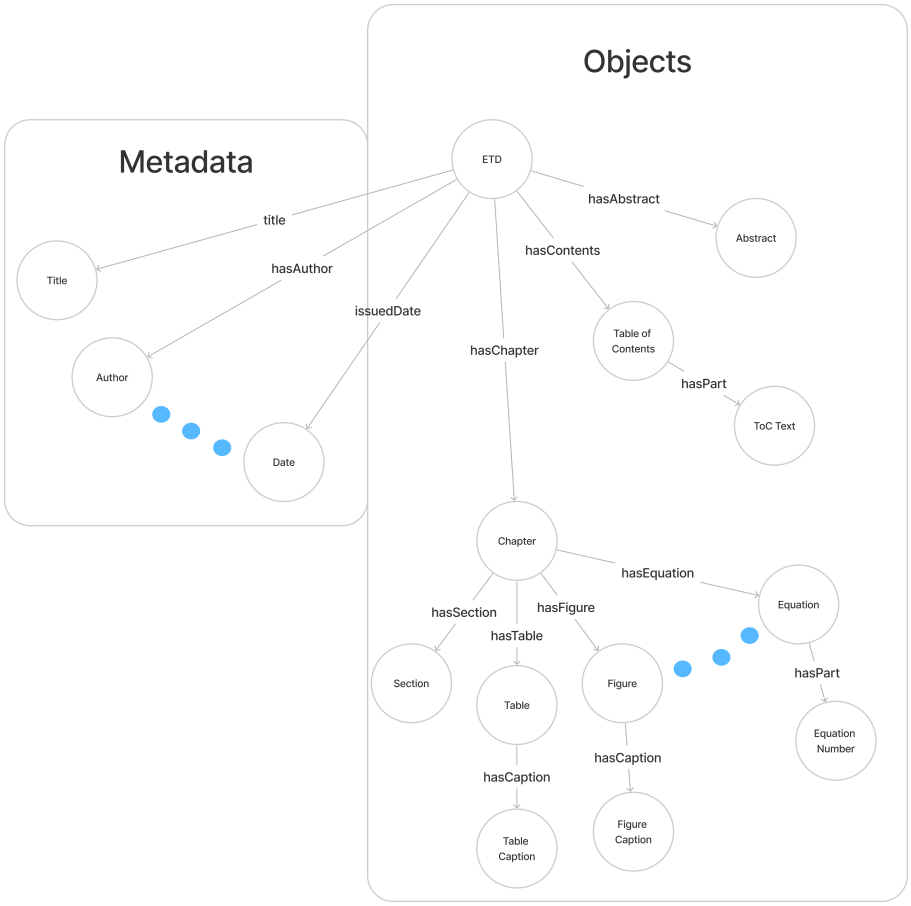

Figure 5: Top-Down Knowledge Graph Representation for an ETD

In the context of metadata-based triples, represented by Figure 5, the subject pertains to any element within the Electronic Theses or Dissertations (ETDs) framework, encompassing entities such as equations, chapters, sections, tables, and more. The object, in turn, encompasses attributes or properties of the subject. For instance, a figure might possess objects like a file path and a corresponding caption. The predicate serves to establish the relationship between the subject and its attributes, denoted by expressions like "hasSection/hasTable/hasChapter". When it comes to querying RDF data, SPARQL stands as the preeminent language. It necessitates the specification of a series of triple patterns, against which the data must align in one or more graphs. Each constituent of the triple is uniquely identified by a Uniform Resource Identifier (URI), which, in this scenario, references the generated URI that when clicked will open the browser to display the selected entity by retrieving the digital object from where it is stored.

6.3 URI Resolution

6.3.1 Overview

As mentioned, we seek to provide the user a capability to select any URI in the Knowledge Graph to see the entity it represents displayed in the browser. Since the Knowledge Graph design for our project uses URIs to store the subject, predicate, and, in some cases, the object, we must create a service to resolve the URIs that are stored as triples. When the user queries our Knowledge Graph the query will return with the triples that matched the desired search. The user should then be able to click on any URI to view it. This means that when a URI is clicked by a user it will be a valid URI in the browser and the page will show the element that the URI represents. If the user selects a URI for a figure, then the browser should show the image of the figure. If the user selects the URI of an entire ETD then the entire ETD should be displayed in the browser. In order to achieve this we must query the PostgreSQL database to retrieve the contents of the element. This feat requires the URIs to retain information that allows the URI resolver to understand what table must be queried and the key to query it with. Thus a complex schema for the URIs of the RDF triples becomes necessary because of the many different cases to handle.

6.3.2 URI Resolution Schema

The first problem we encounter is that in order to query the PostgreSQL database we need a valid key for an object stored there. The current solution, as this is a working schema, is to store the object ID directly in the URI so it can be retrieved by the URI resolver when querying the PostgreSQL database. This solution requires the URI to store both the appropriate table for an entity and the appropriate key.

subject	predicate	object
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384265

Figure 6: RDF Triple URI Example for ETD's Chapter

As shown by Figure 6, both the subject and object columns contain the database ID for the entries they represent. It is also worth noting that the predicate is also stored as a URI. When this URI is clicked by the user it will display simple information about the predicate and what relationship it portrays.

This solution works great for every object that has its own key in the PostgreSQL database such as chapters, paragraphs, and ETDs. However there are metadata values in the database such as author, title, and other metadata elements that do not contain their own key. Rather they are stored in a separate table called ETD metadata. The working solution we have to retrieve these from the PostgreSQL database is to add a path in the URI.

subject	predicate	object
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasAuthor	http://etdkb.endeavour.cs.vt.edu/v1/etd/6384228/meta/author

Figure 7: RDF Triple URI Example for Author

Figure 7 shows an RDF triple that represents the relationship between an ETD and its author. The difference between the author URI and a normal object URI is that the author of an ETD is not stored as its own object, but rather is stored in the metadata table. Thus from looking at the author URI the URI resolver must be able to determine that this is a metadata value and not an object. This is accomplished by including the "/etd/" and "/meta/" paths in the URI. After this the URI resolver knows that it is working with metadata, so it looks for the ID of the ETD and the metadata value we seek to query for. This retrieval technique would require a query to retrieve the correct metadata table and a query to retrieve title from this metadata table, with the title being eventually displayed on the browser.

6.4 Querying the Knowledge Graph

Originally, we had intended to work with Team 6 to create a UI page for the Knowledge Graph or at least incorporate its components into other pages. However due to the small size and limited scope of both of our teams we have pivoted to building a system to generate the full Knowledge Graph for all 500,000 ETDs, and allowing the user to directly query the graph. To query the Knowledge Graph there is a Docker image of Virtuoso running on the Endeavour cluster. This allows users to directly query the contents of the Knowledge Graph. The queries use SPARQL, which as previously stated is a query language for querying and manipulating data stored in Resource Description Framework (RDF) format. The user will simply add filters to refine the query in order to retrieve the triples and URIs the user seeks.

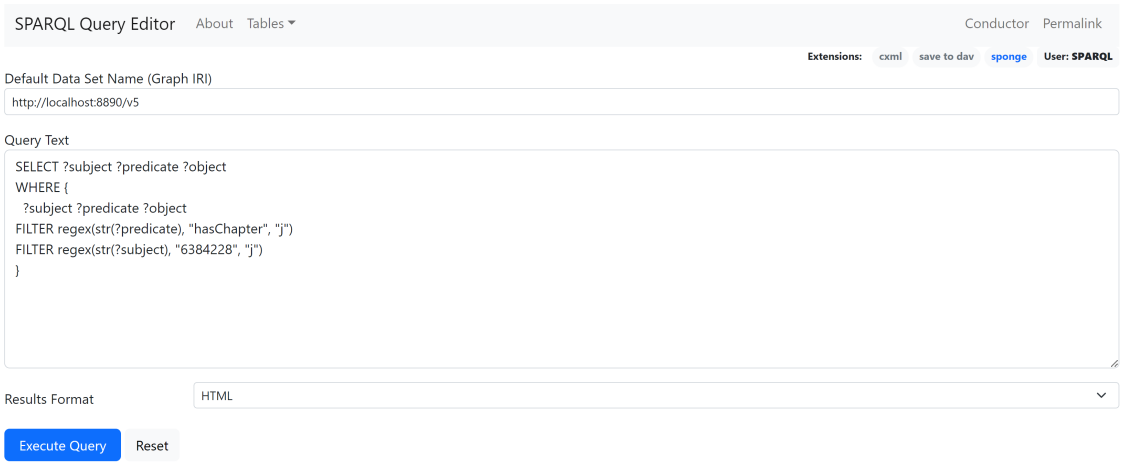

Figure 8: SPARQL Query Example

Figure 8 is an example of how the querying on the Knowledge Graph works. The user first selects all the elements they want to display. If they want to see subject, predicate, and object then they should add "SELECT ?subject ?predicate ?object" as is demonstrated in Figure 8. If the user only seeks to see the URI of the object, then they would simply add "SELECT ?object". Secondly, the user is able to focus on filtering the query to find the information they seek. In Figure 8, the query is filtering for the subject and predicate. The values of the subject are being filtered by the ID of a specific ETD, and the value of the predicate is being filtered by the relationship "hasChapter". Thus this query will return all of the chapters of the specific ETD represented by the ID that was provided.

subject	predicate	object
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384265
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384267
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384312
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384340
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384411
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384456
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384552
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384562
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384624
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384631
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384633
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384666

Table 1: RDF Triple Table of SPARQL Query Result

Table 1 shows the result of the query. The subjects are all equal to the URI of the ETD that was provided, and all the predicates are equal to the URI of the "hasChapter" predicate. Thus all the objects in this example are the URIs of the chapters contained within this ETD. Consequently, the user is able to view every chapter of the ETD.

6.5 Data Schema

6.5.1 ETD Metadata

Table 2 includes a list of all possible value types for the metadata and objects of an ETD.

Word	**Definition**	**Database**
Abstract	The abstract text for the ETD.	Metadata Table
Abstract General	The general abstract text for the ETD. This represents the abstract in terms so that the general public could understand.	Objects Table
Acknowledgement	The text of the acknowledgments given by the author in the ETD.	Metadata Table
Algorithm	An algorithm that appears in the ETD.	Objects Table
Author	The name of the author of the ETD.	Metadata Table
Chapter Summary	The summary of a chapter in an ETD. This is a value generated by the summarization model developed by Team 4.	Objects Table
Chapter Title	The title of a chapter in an ETD.	Objects Table
Cited Author	Any cited authors that appear in a given ETD. This means authors who are included in the bibliography section of an ETD.	Metadata Table
Committee Chair	The name of the chair of the committee.	Metadata Table
Committee Member	The list of committee members and their respective positions (advisor, co-advisor, etc.) that review the ETD.	Metadata Table
Date	The date that the ETD was published.	Metadata Table
Dedication text	The text of the dedications given by the ETD author.	Metadata Table
Degree	The degree of the author of the ETD.	Metadata Table
Department	The department to which the ETD was submitted.	Metadata Table
Equation	An equation that appears in an ETD.	Objects Table

Word	Definition	Database
Equation Number	The number assigned to a given equation in an ETD based on the sequential order.	Objects Table
Figure	A figure that appears in an ETD.	Objects Table
Figure Caption	The caption of a figure that appears in the ETD; it often provides background that describes the meaning of the figure.	Objects Table
Footnote	A footnote included in the ETD.	Metadata Table
Language	The language of the ETD.	Metadata Table
List of Contents Heading	The list of contents heading (e.g., "Table of Contents" or "List of Figures") of the ETD.	Objects Table
List of Contents Text	The actual text stored in the list of contents in the ETD.	Objects Table
Page Number	The page number of a page within the ETD. This will be used to relate figures, chapters, and other objects based on what page they appear on in the ETD.	Not Stored
Paragraph	The text of a paragraph within an ETD.	Objects Table
Reference Text	The text form of the references that were referenced in the ETD.	Metadata Table
Rights	The copyright and access rights of an ETD.	Metadata Table
Section	The text of a section within an ETD.	Objects Table
Sponsoring Agency	An agency or entity which is providing or intends to provide financing, e.g., in the form of grants, for a given ETD.	Metadata Table
Subject/Keywords	The keywords from the ETD, useful to get an idea of what the ETD is about.	Objects Table
Table	A table that appears in the ETD.	Objects Table
Table Caption	The caption of the table that appears in an ETD; it often provides background that describes the contents of the table.	Objects Table
Title	The title of the ETD.	Metadata Table
Topic Label	The topic label is a generated label that represents the topic the object relates to. The topic label will be generated for both chapters and overall ETDs, by Team 3.	Objects Table
University	The university to which the ETD was submitted.	Metadata Table
University Location	The location of the university where the ETD was submitted/uploaded to.	Not Stored
URI	The Uniform Resource Identifier of the ETD.	Metadata Table

Table 2: Definitions in the ETD Knowledge Graph

6.5.2 RDF Triple Schema

With all these possible value types for an ETD object there are a large number of possible relationships between different objects. Tables 3 and 4 list all possible Subject, Predicate, and Object types that can be stored as an RDF triple within our system for both the top-down approach and the bottom-up approach, respectively.

Subject	Predicate	Object
Chapter	classifiedAs	Classification Label
Chapter	hasAlgorithm	Algorithm
Chapter	hasEquation	Equation
Chapter	hasFigure	Figure
Chapter	hasPart	Footnote
Chapter	hasPart	Paragraph
Chapter	hasPart	Section
Chapter	hasSummary	Chapter Summary
Chapter	hasTable	Table
Chapter	hasTopic	Topic Label
Equation	hasPart	Equation Number
ETD	academicAdvisor	Committee Chair
ETD	academicDiscipline	Department
ETD	citedAuthor	Cited Author
ETD	dedicatedTo	Dedication Text
ETD	degreeType	Degree
ETD	hasAbstract	Abstract
ETD	hasAbstractGeneral	Abstract General
ETD	hasAcknowledgement	Acknowledgement
ETD	hasAuthor	Author
ETD	hasChapter	Chapter
ETD	hasCited	Reference Text
ETD	hasContents	List of Contents Heading
ETD	hasEquation	Equation
ETD	hasKeyword	Subject/Keywords
ETD	hasRights	Rights
ETD	hasTitle	Title
ETD	hasTopic	Topic Label
ETD	identifier	URI
ETD	issuedDate	Date
ETD	publishedBy	University
ETD	supportedBy	Sponsoring Agency
ETD	writtenIn	Language
Figure	hasCaption	Figure Caption
List of Contents Heading	hasPart	List of Contents Text
Table	hasCaption	Table Caption
University	location	University Location

Table 3: RDF Top-Down Schema for the Knowledge Graph

Subject	**Predicate**	**Object**
Abstract	abstractOf	ETD
Abstract General	abstractOf	ETD
Acknowledgement	acknowledgementOf	ETD
Algorithm	algorithmOf	Chapter
Author	authorOf	ETD
Chapter Summary	summaryOf	Chapter
Chapter	partOf	ETD
Classification Label	classificationOf	Chapter
Committee Chair	advisorOf	ETD
Committee Member	committeeOf	ETD
Dedication Text	dedicatedBy	ETD
Equation	equationOf	Chapter
Equation Number	partOf	Equation
Figure	figureOf	Chapter
Figure Caption	captionOf	Figure
Footnote	partOf	Chapter
List of Contents Heading	contentsOf	ETD
List of Contents Text	partOf	List of Contents Heading
Paragraph	partOf	Chapter
Rights	rightsOf	ETD
Section	partOf	Chapter
Table	tableOf	Chapter
Table Caption	captionOf	Table
Title	titleOf	ETD
Topic Label	topicOf	Chapter
Topic Label	topicOf	ETD
University	hasPublished	ETD

Table 4: RDF Bottom-Up Schema in the ETD Knowledge Graph

6.6 Tools

Below is a list of the tools we use in this project:

- Programming Languages - We use Python and JavaScript to write our services. The XML to RDF triple conversion service is developed in Python and the URI resolution service is developed in JavaScript's server-side runtime environment, Node.js.
- CI/CD - Docker [10] is used to containerize our services and we utilise GitLab's DevOps platform to develop integration and deployment pipelines. It is intended to modularize our work and provide an easy interface for future work to build off of.
- Inter-Service Communication - Kafka [11] can be used for fetching XML of any new processed ETD, and lead to use of RESTful APIs for experimenters of other microservices to consume them.
- Database - Apart from Virtuoso, PostgreSQL [12] is used as the database to store the ETD metadata and objects. For storing figures and images each row has a column that points to a location in the file system where the digital format is stored.
- Graph DB - Virtuoso [13] is used to store and query the generated RDF triples. This database would store the relations between the subjects and objects that the object detection team, Team 3, identifies.

7 Implementation

7.1 Overview

Our project's technical implementation centers around the creation of an extensive Knowledge Graph and the generation of RDF triples to represent a comprehensive collection of ETD metadata, facilitating queries via the SPARQL language. Our approach adopts an event-driven architecture with inter-service communication.

The initial step in our (Team 1) workflow involves accessing processed XMLs containing ETD metadata, which are provided by Team 3. This process was initially envisioned to be initiated by listening to a Kafka Consumer that subscribes to Team 5's Kafka Topic responsible for an XML file publication, generated through ETD Object Detection algorithms maintained by Team 3. The Kafka cluster was to be centrally deployed by Team 5 and Team 3 would've sent data to the respective Kafka Topic whenever a new ETD was processed. However, due to complications in other teams, this architecture could not be implemented within the desired time-frame. We had to manually ingest XML files received from Team 3. This means Team 3 would email or otherwise send us a zipped folder containing the XMLs. We would place them in a directory that the script would then read from. The XML files produced contain the complete ETD information along with unique object IDs that Team 3 generates using the PostgreSQL database. We lay the groundwork for future researchers to automate this asynchronous data pipeline.

Once we acquire an XML file, a Python script is employed for systematic conversion into JSON format and the subsequent generation of RDF triples for the specified ETD. These triples are encoded in the N3 RDF format utilizing Python packages, and encompass both the top-down and bottom-up predicate ontology. These RDF triples are then uploaded into our RDF database using SPARQL's INSERT functionality, accessible through RESTful APIs via SPARQLWrapper. SPARQLWrapper is a simple Python package around the SPARQL service that helps in RESTfully executing queries.

Upon successful insertion of RDF triples, Virtuoso internally orchestrates the creation and update of graphs, abstracting this process from the end user. It ultimately furnishes a SPARQL endpoint, enabling the retrieval of specific relationship types associated with a given subject entity. This query interface is indispensable for supporting the visualization component in the front end, assisting users in locating objects within an ETD.

Our efforts align with those of Team 2, responsible for core search functionality utilizing Elasticsearch in the backend. By persisting semantic linked data, we aim to augment users' search capabilities, facilitating specialized analysis, insights, and visualizations. Within the scope of the current implementation, our Knowledge Graph system is not directly invoked by end users via the frontend interface that Team 6 has constructed. Instead, it is utilized through a SPARQL interface. Through this default Virtuoso interface (see Figure 9), users can query the Knowledge Graph with flexibility.

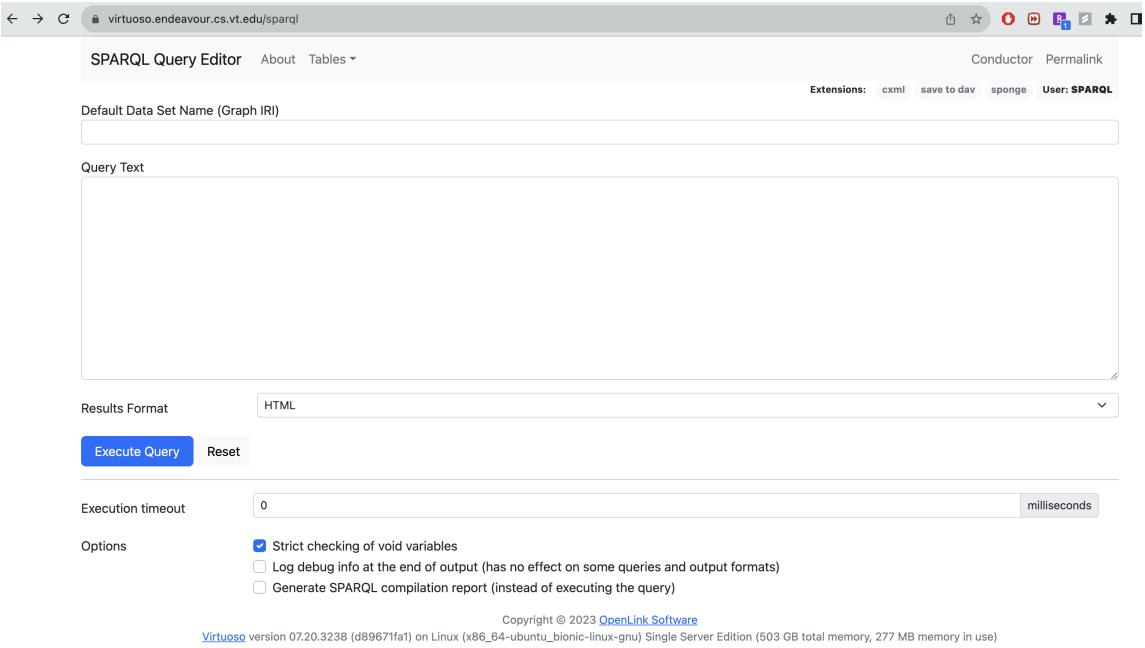

Figure 9: Default Virtuoso SPARQL Interface

The responses of SPARQL queries contain de-referenceable URIs, see Table 1, that users can click to retrieve additional related information. Our system furnishes these linked, clickable URIs within the SPARQL interface by implementing a URI resolver microservice. This is developed using a representational state transfer (REST) architecture and acts as an independent backend service. It utilizes configurable business logic to resolve URIs based on extracted path parameters from the incoming API request. The microservice is containerized using Docker to enable portability and scalability. To enrich the response payload, the microservice interacts with various relational tables in Team 5's PostgreSQL database. The microservice response consists of static hypertext markup language (HTML) and cascading style sheets (CSS) content rendered as web pages. See Figure 10, giving the result as seen in the user's browser. The HTML is configured for accessibility using semantic tags.

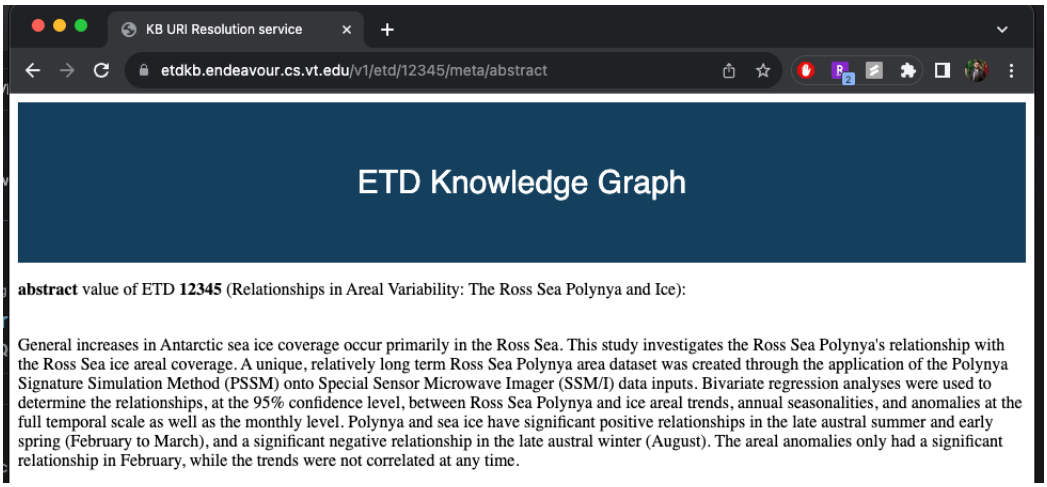

Figure 10: URI resolution service response for "abstract" predicate search of ETD with id 12345

Python and JavaScript are our primary programming languages for this project. Each microservice and their modules will be leveraging both standard libraries and external packages. Docker will be used for containerization to ensure consistent deployment across various environments. We planned to harness Kafka for efficient inter-service communication, optimizing data retrieval and integration processes. This implementation strategy aims to create a robust system capable of efficiently processing and extracting meaningful insights from ETD metadata, contributing to the overall success of the project.

7.2 List of Milestones and Deliverables

The following is a comprehensive list of milestones and deliverables in the project.

- Virtuoso Image Running
  - Find Virtuoso's open source image on Dockerhub.
  - Instantiate this image on our local system and the Endeavour cluster and achieve Virtuoso's functionality.
- Script to convert JSON to RDF Triples.
  - Finalize RDF Schema.
- Upload/Insert RDF Triples into Virtuoso.
  - Integrate with SPARQLWrapper to INSERT data in Virtuoso programmatically.
- Receive data
  - Plan to integrate with Kafka Topic of Team 3 and consume data in an asynchronous manner.
  - Process a sample dataset provided by Team 3.
- Define example queries for the Knowledge Graph
  - Determine SPARQL examples for querying RDF triples.
  - Perform searches retrieving relevant objects related to a subject.
  - Perform semantic search query to retrieve specific relationship types and objects.

7.3 Timeline

Table 5 gives the assignments of each task involved in the implementation as well as the timeline for the deliverables.

Tasks	**Assignee**	**Status**	**Due Date**
Virtuoso Image Running	Ashish Aggarwal	Done	Oct 3
Manual Script to convert JSON to RDF Triples	Daniel Gaugler	Done	Oct 3
Predicate Ontology	Satvik Chekuri	Done	Oct 10
Manual Upload/Insert RDF Triples into Virtuoso	Whole Team	Done	Oct 19
Automate Script to convert JSON to RDF Triples	Daniel Gaugler	Called Off	Oct 25
Automate Upload/Insert RDF Triples into Virtuoso	Whole Team	Done	Nov 1
Design URI resolution service	Whole Team	Done	Nov 1
Integrate with Kafka instance	Whole Team	Called Off	Nov 5
Implement URI resolution service	Whole Team	Done	Nov 15
Write test cases for both services	Whole Team	Done	Nov 20
Implement CI/CD to both services	Whole Team	Done	Nov 20

Table 5: Task Assignments, Status and Due Dates

8 Future Work

8.1 Securing the SPARQL Endpoint

The current SPARQL endpoint allows open, public access without authentication or usage restrictions. This poses risks of denial-of-service attacks or unauthorized data mining. It is urgent that future work should implement access controls and query analysis to secure the endpoint. Possible security measures include VPN-restricted or SSO enabled access, allowing only authorized users and processes to query. The end goal should be an enterprise-grade endpoint with authentication, optimized querying, and protections against threats. This robustness will prove critical as dataset scale and user loads grow over time.

8.2 Expanding the ETD Dataset

The Virtuoso database contains 200 XML records serving as dummy data for testing purposes. The next critical step is integrating the full dataset of 500,000 enriched ETD records that will enable more robust testing and analysis. Structuring and mapping this data properly within Virtuoso is an essential task for future researchers. Establishing standardized processes for ongoing data integration as new ETDs are published should also be a priority.

8.3 Implementing Automated Data Pipelines

While the current process relies on manual uploading of XML files, introducing automated pipelines using Kafka for data transfer would improve reliability and efficiency. The initial design had been laid, but full Kafka integration proved infeasible due to delays with other system components. Future efforts should focus on completing this integration to allow seamless transfer of new ETD records from upstream processing into the Virtuoso database. Creating this pipeline is key for scaling up the system.

8.4 Connecting the Search and SPARQL Interfaces

To improve the overall user experience, the semantic SPARQL query features would ideally integrate with the search and discovery interface via the unified web interface that Team 6 developed. This would empower users with a single portal to access the full capabilities of the system. The design process should emphasize usability and leverage SPARQL to underpin advanced search functionalities. Thorough user testing will help ensure the interface design is intuitive for researchers while providing the full richness of the underlying Linked Data.

8.5 Knowledge Graph of Entities from ETDs

The future work outlined here closely follows the methodology in [16].

We have built a proof-of-concept of a Knowledge Graph capturing the structure of ETDs and their elements using 200 ETD XMLs. To efficiently aid the ETDs with downstream tasks such as summarization, search, and QA, the KG must build a suitable representation of the ETDs, capturing semantic information. The representation should include a structure reflecting the intra- and inter-document entity-entity relations. The Knowledge Graph structure, as a directed heterogeneous graph with domain related semantics, must accommodate what has been extracted from ETDs. This is essential, since transformer based models mostly excel in short, single documents, not dealing with long documents (due to attention limitations). The ETD KG should fill this gap, aiding downstream tasks through a graph representation of the entities and relations extracted from the ETDs, with both in-document and distant document relations connecting entities.

We propose to build the entity-based KG for a collection of documents by first extracting entity-related triples using improved methods for OpenIE and NER [18] for the different entities and relations. Triples will yield node-edge-node additions to the graph, with entities as nodes. This graph will be extended and enriched using entities from all of the ETDs. This will have an impact on the coverage and performance of KGs in downstream tasks such as search and question-answering by including entity-related triples extracted from various text-based elements in the ETDs.

9 User Manual

This user manual will guide you through the process of querying our Knowledge Graph using the SPARQL query language. The application allows you to search for records based on the subject or predicate and retrieve detailed information about the matches.

9.1 Accessing the Query Interface

- Open the application in your preferred web browser.
- Navigate to *https://virtuoso.endeavour.cs.vt.edu/sparql*
- The page should display a large text box for users to enter their queries.

9.2 Querying for Records

In the query input box, enter your SPARQL query. Example queries are provided for reference. Figure 11 displays an example query that retrieves all triples related to a specific subject. In this case the subject is an ETD with the ID "6384228". Figure 12 displays a query for all triples with a given predicate. The predicate in question is "hasChapter", so the query will return every triple for every chapter of every ETD. Figure 13 displays a query that filters by both subject and predicate.

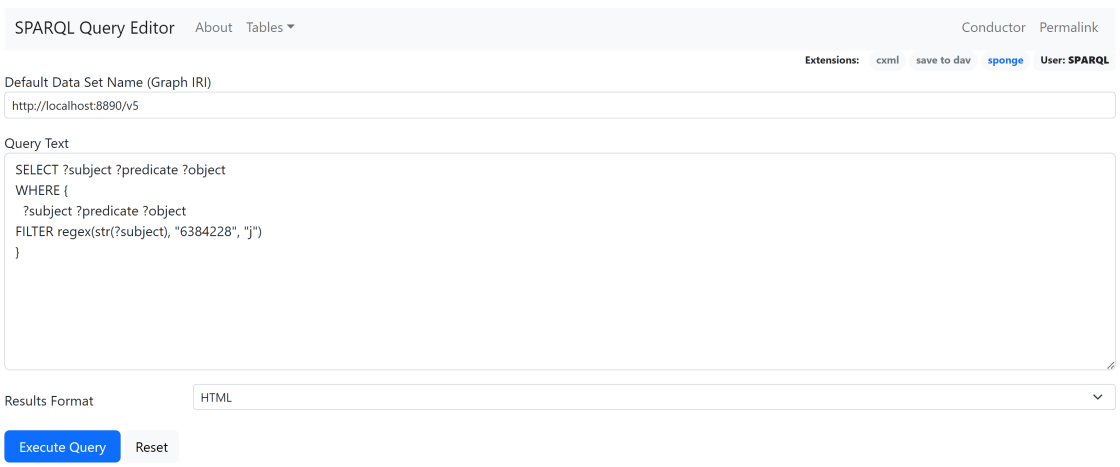

Figure 11: Retrieve all triples related to a specific subject.

SPARQL Query Editor About Tables Conductor Permalink

Extensions: cxml save to dav sponge User: SPARQL

Default Data Set Name (Graph IRI)

http://localhost:8890/v5

Query Text

```
SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object
FILTER regex(str(?predicate), "hasChapter", "j")
}
```

Results Format HTML

Execute Query Reset

Figure 12: Retrieve all triples related to a specific predicate.

SPARQL Query Editor About Tables Conductor Permalink

Extensions: cxml save to dav sponge User: SPARQL

Default Data Set Name (Graph IRI)

http://localhost:8890/v5

Query Text

```
SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object
FILTER regex(str(?predicate), "hasChapter", "j")
FILTER regex(str(?subject), "6384228", "j")
}
```

Results Format HTML

Execute Query Reset

Figure 13: Retrieve all triples related to a specific subject and predicate.

Click the "Execute Query" button to submit your query. The application will return a list of RDF triples that match your query.

9.3 Viewing Details of a Record

In the query results, each triple is represented with Subject, Predicate, and Object URIs. After pressing the execute query button the user will automatically be taken to a page displaying all the triples that match the query. An example of this is given in Figure 14. This figure shows the results of the query shown in Figure 13.

From here the user is able to click on the URI of a record to view detailed information about that object. Figure 15 displays an example of the URI resolution service displaying an abstract of an ETD.

subject	predicate	object
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384265
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384267
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384312
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384340
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384411
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384456
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384552
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384562
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384624
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384631
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384633
http://etdkb.endeavour.cs.vt.edu/v1/objects/6384228	http://etdkb.endeavour.cs.vt.edu/v1/predicate/hasChapter	http://etdkb.endeavour.cs.vt.edu/v1/objects/6384666

Figure 14: Results of the query that filters for both predicate and subject.

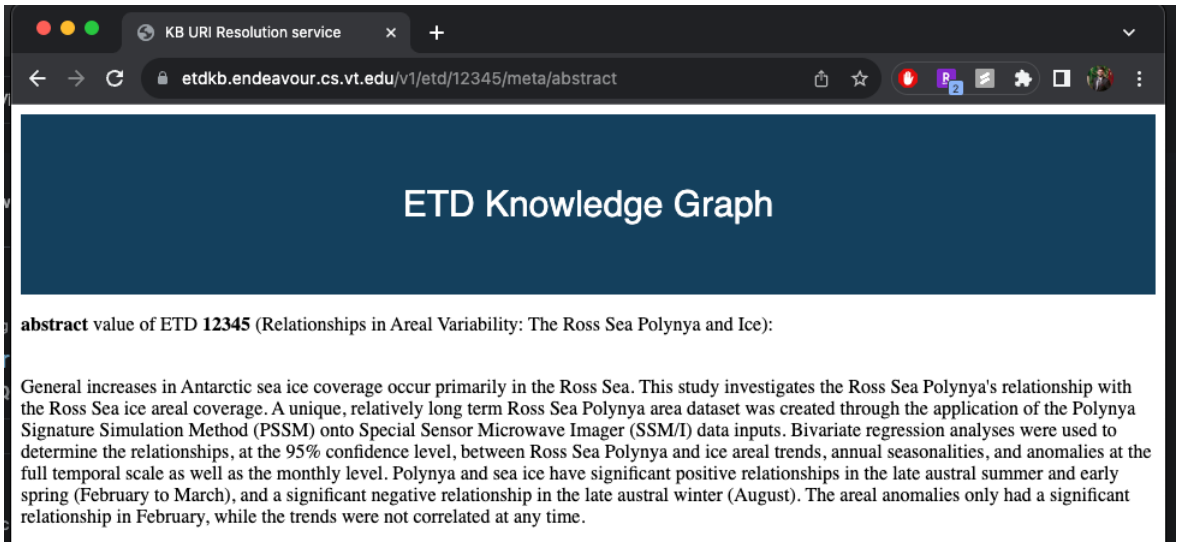

Figure 15: Clicking on the Subject URI to view details.

10 Developer Manual

10.1 URI Resolution Service

This is a Node.js and Express.js based Web service hosted on the Endeavour cluster. The domain name for the service is `https://etdkb.endeavour.cs.vt.edu`. There are 3 routes in this service.

1. v1/etd/:etdId/meta/:metaProp
   - Params: ETD ID and metadata value to fetch.
   - This route fetches data based on the ETD ID from the metadata table of PostgreSQL database hosted by Team 5.
2. v1/objects/:objectId
   - Params: Object ID of the object
   - This route fetches data based on the Object ID from the Objects table of the PostgreSQL database hosted by Team 5.
3. v1/predicate/:predicateName
   - Params: Bottom-up or Top-down predicate name.
   - This route fetches data based on the predicate name from a JSON file stored in this service's server.

To run it locally,

- make sure you have Node.js and NPM installed on your computer,
- *git clone* the repo,
- run *npm i* from the terminal, and
- run *npm start* from the terminal. This would start the local server on port 3000, *localhost:3000*.

Using a tool like Postman, the above mentioned routes can be invoked. To run test cases, use the command *npm test* from the terminal.

The errors for this service are logged into Team 2's Elasticsearch index. To search the logs in their index, have the *User* parameter as *kg-uri-app-logs*. The index.js file of our URI resolution service has the implementation for POSTing logs. Currently, only the global error logs are sent to the Elaticsearch index, but this functionality can be extended to other log types too. More details on accessing logs, searching logs, and Kibana access can be found in the report of Team 2.

10.2 ETD Processing and Triple Generation Script

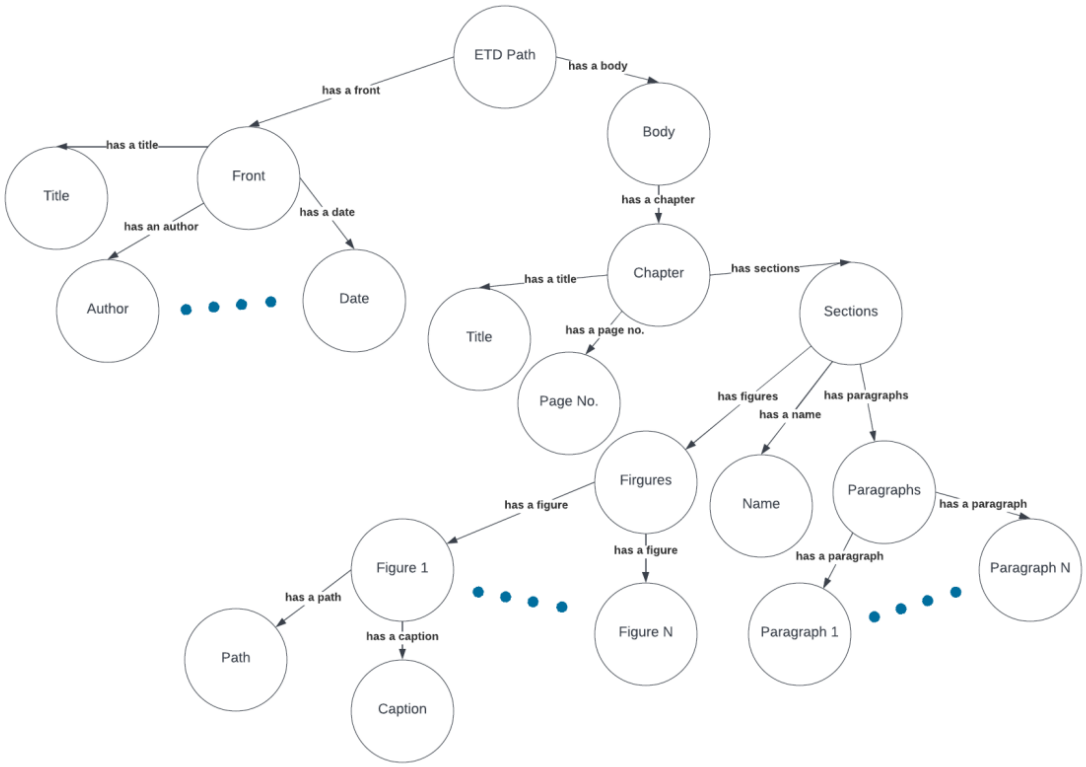

Figure 16: High-level view of XML Schema.

The ETD processing and triple generation script is designed to convert XML files representing Electronic Theses and Dissertations (ETDs) into dictionaries and generate triples from these dictionaries. The schema for the XML files is displayed in Figure 16. Every XML has a front, which contains metadata values like Title, Author, and Date. The XML files also contain a body section. This body section contains all the chapters of an ETD. Within the chapters are the multiple sections that make up a chapter. Within these sections are objects like figures, tables, and paragraphs. All these objects must be extracted from where they are contained and have triples generated for their relationships. It is currently designed for local processing, with future plans to host it on the Endeavour cluster and read ETDs from a Kafka topic. The script follows these key steps:

1. **XML to Dictionary Conversion:** The script parses through each XML file in the provided collection of ETDs, converting the XML structure into Python dictionaries to capture the hierarchical relationships and attributes of each element.

2. **Triple Generation Methods:** For each object encountered in the XML files, the script generates triples using specific methods tailored to each object type. These methods accommodate variations in triple structures for different types of ETD-related entities, such as chapters, sections, figures, tables, and metadata.

3. **Metadata Triple Generation:** In addition to object-specific triple generation methods, there is a general method to handle the metadata of the ETD. This method extracts relevant in-

formation, such as title, author, date, university, and other metadata, and generates triples accordingly.

4. **Local Execution:** Currently, the script is designed to run locally, processing the entire collection of XML files in a batch, providing a quick way to generate triples for analysis.

5. **Future Work: Kafka Integration:** The script has future plans to be hosted on the Endeavour cluster and read ETDs one at a time from a Kafka topic. This enhancement would allow for real-time processing and integration into a larger data pipeline.

10.2.1 Analysis Script

In addition to the ETD processing script, there is an analysis script designed to examine the generated triples and derive insights from the ETD data. The results of the analysis script include information about the count of certain predicates and patterns within the ETDs, facilitating a deeper understanding of the content and structure of the academic documents. The results of the analysis script are displayed in Table 6.

Term	**Count**
abstractOf	145
algorithmOf	143
authorOf	158
captionOf	6777
chapterOf	1403
committeeOf	138
degreeType	156
equationOf	5891
figureOf	4895
hasAbstract	145
hasAuthor	158
hasCaption	6777
hasChapter	1403
hasCommittee	138
hasContents	475
hasEquation	5891
hasAlgorithm	143
hasFigure	4895
hasPart	42609
hasPublished	133
hasTable	1882
issuedDate	146
partOf	8534
publishedBy	133
hasTitle	156
titleOf	156
tableOf	1882

Table 6: Predicate Term Counts

10.2.2 Usage

1. **Local Execution:**

   (a) Ensure Python is installed on your machine.

(b) Install the necessary dependencies by running `pip install -r requirements.txt`.

(c) Execute the script using the command `python etd_processing_script.py`.

2. **Future Deployment:** For future deployment on the Endeavour cluster, additional configuration steps and dependencies may be required. Consult the deployment documentation for the specific environment.

The ETD processing and triple generation script, along with the analysis script, provides a foundation for extracting valuable information from ETDs. Future development efforts should focus on hosting the code on the Endeavour cluster and integrating it with Kafka for a scalable and real-time processing solution.

10.3 Virtuoso Database

The Virtuoso image used for this project is available at `https://hub.docker.com/r/openlink/virtuoso-opensource-7`. The configuration of this image required a username and password which is available with our SME, Satvik Chekuri. We were helped with this configuration by Team 5 when they deployed the image on Endeavour. The admin console is hosted at `https://virtuoso.endeavour.cs.vt.edu` and uses the same set of username and password as above.

10.4 CI/CD

To enable continuous integration and deployment we are utilising the DevOps functionality of our version control tool, GitLab. Team 5 has spearheaded this setup for our team.

10.4.1 URI Resolution Service

The first step in enabling CI/CD is Dockerizing the application. The Docker file is added in the root of the code directory. Figure 17 shows the Docker configuration used for this service.

```
# Use the official Node.js image as the base image
FROM node:18-alpine
# Set the working directory in the container
WORKDIR /app
# Copy the package.json and package-lock.json files to the container
COPY package*.json ./
# Install application dependencies
RUN npm install
# Copy the rest of your application code to the container
COPY . .
# Expose the port your application will run on
EXPOSE 3000
# Start your Node.js application
CMD ["npm", "start"]
```

Figure 17: Docker file for URI Resolution Service

The second step is to write the GitLab *.gitlab-ci.yml* file. This file is automatically detected by the GitLab Agents responsible for running CI/CD. The stages in this file are run sequentially and the pipeline auto-triggers whenever a commit is pushed to the *master* branch. Figure 18 shows the YAML configuration for this service.

```yaml
stages:
  - test
  - build
  - deploy

test-job:
  stage: test
  image: node:18-alpine
  script:
    - echo "Running tests..."
    - npm install
    - npm test
    - echo "Tests completed successfully"

build-job:
  stage: build
  image: docker:stable
  rules:
    - if: $CI_COMMIT_BRANCH != 'master'
      variables:
        SHOULD_UPDATE_REGISTRY: "false"
    - if: $CI_COMMIT_BRANCH == 'master'
      variables:
        SHOULD_UPDATE_REGISTRY: "true"
  script:
    - echo "Building Docker image..."
    - docker build -f ./Dockerfile -t code.vt.edu:5005/cs5604-f2023/team1-kg/kg-uri-resolution/
    t1f23-kg-uri:"$CI_COMMIT_SHORT_SHA" .
    - echo "Build complete"
  after_script:
    - >
      if [ $SHOULD_UPDATE_REGISTRY == "true" ]; then
        docker login -u "$DOCKER_USER" -p "$DOCKER_TOKEN" code.vt.edu:5005
        echo "Login Complete."
        docker push code.vt.edu:5005/cs5604-f2023/team1-kg/kg-uri-resolution/
        t1f23-kg-uri:"$CI_COMMIT_SHORT_SHA"
        docker logout
      else
        echo 'Registry does not need to be updated'
      fi
    - docker image prune -a -f

deploy-job:
  stage: deploy
  image:
    name: bitnami/kubectl:latest
    entrypoint: [""]
  rules:
    - if: $CI_COMMIT_BRANCH == 'master'
  script:
    - echo "Setting the image..."
    - kubectl -n etd set image deployment/kg-uri-resolution
        kg-uri-resolution=code.vt.edu:5005/cs5604-f2023/team1-kg/kg-uri-resolution/
        t1f23-kg-uri:"$CI_COMMIT_SHORT_SHA"
```

Figure 18: YAML file for URI Resolution Service

11 Bibliography

References

[1] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). *DBpedia: A nucleus for a web of open data.* In The Semantic Web (pp. 722-735). Springer, Berlin, Heidelberg.

[2] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). *Freebase: A collaboratively created graph database for structuring human knowledge.* In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 1247-1250).

[3] Google. (2012). *Introducing the Knowledge Graph: Things, not Strings [Blog post].* Retrieved from `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html`, accessed on October 1, 2023.

[4] Harris, S., & Lamb, A. (2013). *SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013.* Retrieved from `https://www.w3.org/TR/sparql11-query/`, accessed on October 1, 2023.

[5] Hartig, O. (2017). *Foundations of an alternative approach to reification in RDF.* arXiv preprint arXiv:1708.01894.

[6] Huang, Z., Xu, W., & Yu, K. (2016). *Bidirectional LSTM-CRF models for sequence tagging.* arXiv preprint arXiv:1508.01991.

[7] Vrandečić, D., & Krötzsch, M. (2014). *Wikidata: A free collaborative knowledgebase.* Communications of the ACM, 57(10), 78-85.

[8] Wikimedia Foundation. (2023). *Wikidata: The Free Knowledge Base.* Retrieved from `https://www.wikidata.org/`, accessed on October 1, 2023.

[9] Nicolas Hubert, Pierre Monnin, Armelle Brun, and Davy Monticolo. *Knowledge Graph Embeddings for Link Prediction: Beware of Semantics!* In *DL4KG@ISWC 2022: Workshop on Deep Learning for Knowledge Graphs, held as part of ISWC 2022: the 21st International Semantic Web Conference*, Oct. 2022, Virtual, China. `https://hal.archives-ouvertes.fr/hal-03787512`

[10] Docker Inc. 2023. *Docker.* Retrieved from `https://www.docker.com/`, accessed on November 7, 2023.

[11] Apache Software Foundation. 2023. *Apache Kafka.* Retrieved from `https://kafka.apache.org/`, accessed on November 7, 2023.

[12] The PostgreSQL Global Development Group. 2023. *PostgreSQL.* Retrieved from `https://www.postgresql.org/`, accessed on November 7, 2023.

[13] OpenLink Virtuoso Universal Server Documentation Team. 2023. *Virtuoso.* Retrieved from `https://virtuoso.openlinksw.com/`, accessed on November 7, 2023.

[14] Tim Berners-Lee, Dan Connolly. 2023. *N3 RDF Format.* Retrieved from `https://www.w3.org/TeamSubmission/n3/#:~:text=The%20Resource%20Description%20Framework%20(%20RDF,is%20a%20superset%20of%20RDF`, accessed on November 7, 2023.

[15] RDFLib Team. 2023. *RDFLib. RDFLib Documentation.* Retrieved from `https://rdflib.readthedocs.io/en/stable/`, accessed on November 7, 2023.

[16] Satvik Chekuri. *Hierarchical Knowledge Graph for Long Documents and their Elements.* In *Doctoral Consortium held June 2023 as part of ACM/IEEE JCDL, Santa Fe, New Mexico, USA.* To appear, 2023 presentation.

[17] Pascal Hitzler. (2021). *A Review of the Semantic Web Field.* In *Proceedings of ACM, New York, NY, USA.* *https://doi.org/10.1145/3397512*

[18] Basra Jehangir, Saravanan Radhakrishnan, Rahul Agarwal. *A survey on Named Entity Recognition — datasets, tools, and methodologies. Natural Language Processing Journal*, Volume 3, 2023, 100017. ISSN 2949-7191. https://doi.org/10.1016/j.nlp.2023.100017. (https://www.sciencedirect.com/science/article/pii/S2949719123000146).

[19] Satvik Chekuri, Prashant Chandrasekar, Bipasha Banerjee, Sung Hee Park, Nila Masrourisaadat, Aman Ahuja, William A. Ingram, and Edward Fox. *Integrated Digital Library System for Long Documents and their Elements.* In *Proceedings of the 23rd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2023)*, nominated for Best Student Paper, pages 13-24, DOI: 10.1109/JCDL57899.2023.00012, 2023.

[20] Daniel B. Weedon, Daniel Olsen. 2021. *Knowledge Graphs.* In *CS 4624: Multimedia, Hypertext, and Information Access*, Virginia Tech team project, Blacksburg, VA, http://hdl.handle.net/10919/103271