ETDs Knowledge Graph Building

Final Report
CS 4624: Multimedia, Hypertext, and Information Access
Virginia Tech, Blacksburg, Spring 2025

Mentor:
Satvik Chekuri

Prepared by:
Dashiell Budd
Sidney Fredericks
Samantha Zheng
Junsoo Kim

Submitted: 5/6/2025

Table of Contents

Table of Figures

1. Executive Summary

ETDs are Electronic Theses and Dissertation, and this project aims to enhance the storage, accessibility, and exploration of Virginia Tech's ETDs by transforming traditional metadata into a searchable knowledge graph. Recognizing the limitations of flat or relational storage for representing rich academic relationships, we developed a dual-database architecture using Virtuoso (RDF/SPARQL) and Neo4j (property graph/Cypher) to model key entities such as authors, advisors, departments, and disciplines. A Streamlit-based web interface provides an intuitive search experience across both databases, enabling users to explore semantic connections by keyword, year, and entity type. The backend includes a Python-based data pipeline that transforms flat CSV data into normalized graph structures, optimized for batch loading at scale. This framework demonstrates a scalable, future-proof approach to managing large volumes of academic content, supporting more meaningful discovery and long-term preservation of institutional knowledge.

2. Introduction

This project focuses on improving the storage, accessibility, and exploration of Electronic Theses and Dissertations (ETDs) produced by Virginia Tech graduate students. ETDs are important scholarly contributions that contain years of academic research, and it is critical to preserve them in a way that allows for scalable, meaningful, and long-term retrieval. As the volume of digital academic content grows, traditional storage and retrieval methods are increasingly limited in their ability to support rich, relationship-driven exploration.

To address this, we structured the ETD metadata into a searchable knowledge graph, enabling deeper insights and more intuitive exploration of connections between key academic entities. These entities include authors, advisors, departments, disciplines, universities, degree types, and more. By modeling ETDs as graphs rather than rows in a table, we enable users to explore semantic relationships which are difficult to uncover using standard relational databases. Our architecture leverages two complementary graph database systems:

- Virtuoso, which supports SPARQL for querying RDF data
- Neo4j, which uses Cypher, a user-friendly graph query language

Using both systems ensures query flexibility, compatibility with existing semantic web tools, and adaptability for future use cases. This dual-database setup allows us to experiment with different query patterns, performance models, and user preferences.

To support real-time exploration, we developed a Streamlit-based web interface that connects to both databases and provides users with a unified search experience. The web app enables metadata search by keyword, year, and entity type, and displays clean, scrollable results with links to full ETD records.

Our data pipeline begins with a large CSV dataset of ETDs. We designed and implemented Python scripts to transform this flat data into structured formats appropriate for both Virtuoso and Neo4j. This involved defining entity nodes, assigning predicate relationships, and ensuring consistent data normalization across records. We also focused heavily on batch loading optimization to enable the insertion of large datasets efficiently with the goal of loading thousands of records and preparing the system to scale for the full dataset of 500,000+ ETDs.

Overall, this project provides a framework for transforming institutional academic records into graph-based formats that are easier to search, explore, and scale.

3. Requirements

General requirements of our project are:

- Extraction and formatting of more than 20,000 ETDs, with scalability to 500,000 ETDs, to be ingested in a Knowledge Graph.
- Ingesting the data into both Neo4j and Virtuoso databases for the purposes of comparison and querying.
- Creating a query interface to the Knowledge Graph that allows questions spanning both databases.
- Use object detection on all ETD submissions to enable more effective metadata extraction.
- Ensuring appropriate conversion of ETD elements to RDF triples and Cypher-compatible data for Virtuoso and Neo4j, respectively.

The minimum requirements guarantee the adequate growth of the Knowledge Graph database and its availability. Information and protocols are elaborated upon in the following sections.

3.1 Data Extraction and Formatting

Data extraction is a crucial task in our project, a task that relies on extracting metadata from 20,000+ ETDs, with the ambition of expanding to 500,000 ETDs, which are kept in XML format. The metadata should encompass basic elements such as titles, authors, publication details, and abstracts. The data extracted needs to be put into RDF triples for Virtuoso and a suitable form for Neo4j. The process should ensure:

- All metadata fields are properly extracted without data loss.
- Virtuoso RDF triples are represented in N3 format, as suggested by current research, for the sake of consistency.
- The data structure in Neo4j is designed to adhere to the graph schema outlined in Section 3.3.
- The process needs to be scalable to accommodate 500,000 ETDs upon project completion.

3.2 Data Ingestion into Virtuoso and Neo4j

The data ingestion involves loading the formatted data into Virtuoso and Neo4j databases. In the case of Virtuoso, the RDF triples must be loaded to the provided server endpoint. In the case of Neo4j, the data must be loaded as nodes and relationships, following the schema created in Section 3.3. The process must guarantee:

- The number of RDF triples loaded into Virtuoso is equal to the data extracted, without any duplications or omissions.

- The information consumed by Neo4j precisely embodies the relations among ETD components. Both databases need to handle concurrent data ingestion for the first 20,000+ ETDs and scale up to 500,000 ETDs.
- The consumption process needs to be automated through a data pipeline for efficiency and reproducibility.

3.3 Creating Neo4j Schema and Data Pipeline

One of the main requirements of this semester is the design of a Neo4j schema and a set of data pipelines to facilitate the Knowledge Graph. The schema should define the shape of nodes and the relationships to capture the ETD metadata. Moreover, the data pipeline should make it easy to extract, transform, and import data into Neo4j. The requirements are:

- The schema is required to accommodate all metadata components derived from the ETDs, thereby guaranteeing that no information is omitted in the graphical representation.
- The pipeline needs to plug into the current Virtuoso ingestion procedure and enable simultaneous data loading to both databases.
- The pipeline should be scalable to process 500,000 ETDs without any loss in performance. Schema and pipeline documentation should be provided for future use and included in the Developer's Manual.

3.4 Object Detection for ETD Entries

Object detection is a key capacity in ETD entry metadata enrichment. It entails the detection and extraction of objects from ETD documents in a bid to enrich the Knowledge Graph. The requirements are:

- Object detection must be performed on all 500,000 ETD entries by the project's conclusion, starting with the initial 20,000+ ETDs.
- The detected objects need to be connected to their corresponding ETD entries in Virtuoso and Neo4j.
- The process must be at least 95% accurate at detecting objects to be dependable.
- The output of object detection must be quarriable via the user interface (refer to Section 3.5).

3.5 Interface for User Search Activities

The user interface (UI) is a key component for interaction with the Knowledge Graph. The UI, developed in Python and Streamlit, must allow users to query and retrieve data from Virtuoso and Neo4j databases. The requirements include:

- The interface should support SPARQL queries for Virtuoso and Cypher queries for Neo4j so that ETD metadata can be searched by users.
- The user interface should present search results in a useful manner, with hyperlinks to the original ETD documents where possible.
- The UI should enable live search, linking to the Virtuoso endpoint and Neo4j instance in real time.
- The UI must be fully functional to handle requests for 500,000 ETDs with less than 3 seconds of response time for routine searches.

3.6 System and Deployment Requirements

To ensure the system functions appropriately, the system requirements listed below should be met:

- The system must be capable of running on a teaching cluster with at least 32GB RAM and 1TB disk space to process 500,000 ETDs.
- The system should be compatible with Virtuoso Open Source 7.2 and Neo4j 4.4 or newer.
- It should be compatible with Python 3.9 for scripting and Streamlit for the interface.
- The system will operate on a network of at least 100 Mbps operating bandwidth for live search capability.

4. Design

4.1 System Overview

Our system is composed of four major components.

1. Data Extraction: Structured information from raw ETDs PDFs (e.g. title, author, abstract, advisor, department, URI) is extracted.
2. Data Conversion: Extracted data is converted into formats compatible with Virtuoso and Neo4j
3. Graph Databases:
    - **Virtuoso** stores data as RDF triples and supports SPARQL queries.
    - **Neo4j** stores data as property graphs and supports Cypher queries
4. User Interface: A Python/Streamlit GUI that lets users search and explore ETDs by interacting with either graph database.

4.2 System Diagram

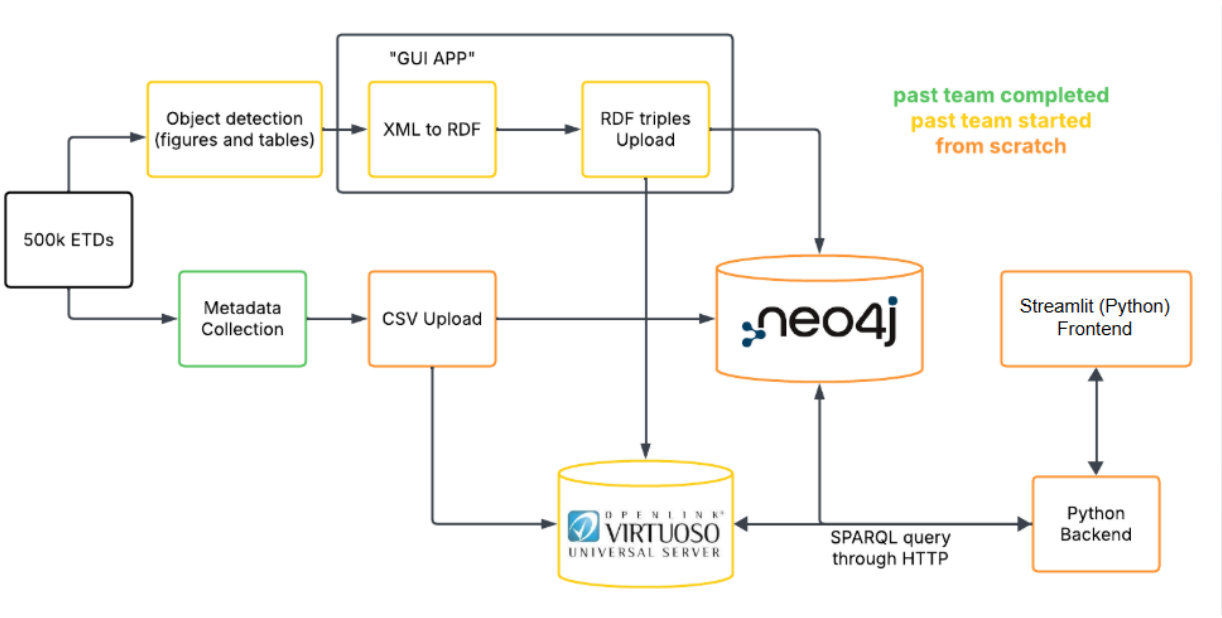

Figure 1: System Architecture

4.3 Data in Virtuoso

Virtuoso is a RDF triple store that uses the Resource Description Framework (RDF) model to store structured data. It is well-suited for representing metadata and linked data in a standardized format and supports querying using the SPARQL language. In our project, Virtuoso is used to store ETD information in the form of triples.

Each data point in Virtuoso is represented as a triple:

Subject → Predicate → Object

- Subject: The resource being described
- Predicate: The property or relationship
- Object: The value of the property

Example:

<http://etds.vt.edu/etd123> <http://localhost:8890/schemas/CSV/hasTitle> "ETD Knowledge Graphs"

These triples are inserted into Virtuoso using **SPARQL** INSERT queries and retrieved using **SPARQL** SELECT queries.

4.4 Data in Neo4j

In our Neo4j implementation, we represented ETDs using a flexible, modular graph schema centered around the Title node. Instead of bundling metadata as properties within a single node, we created separate nodes for each major field such as Author, Advisor, Abstract, Year, Department, Discipline, University, Degree, etc. This structure enables more granular queries and more natural traversal paths across metadata.

Each Title node includes the ETD's ID and URI as attributes and connects outward to all related metadata nodes through clearly defined relationships. For instance, to find all ETDs advised by a certain faculty member, one can traverse the ACADEMIC_ADVISOR edge from the Advisor node to connected Title node.

To populate the database, we designed a Python loader script that reads a JSON-formatted metadata file and creates the appropriate nodes and relationships in batches. We used Cypher MERGE statements to avoid duplicate nodes and made sure that each relationship was semantically meaningful. We also included optional logic to handle missing fields gracefully.

4.5 Data Flow

- We get raw PDF files from the ETD collection.
- We extract important fields using custom scripts.
- The data is formatted into:
    - Triples for Virtuoso
    - Node/Edge data for Neo4j
- Load:
    - Into Virtuoso using SPARQL
    - Into Neo4j using Cypher

4.6 User Interface

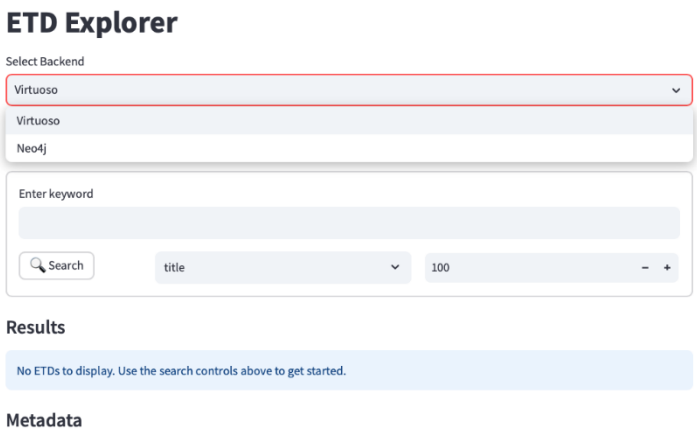

Figure 2: ETD Explorer Home Page

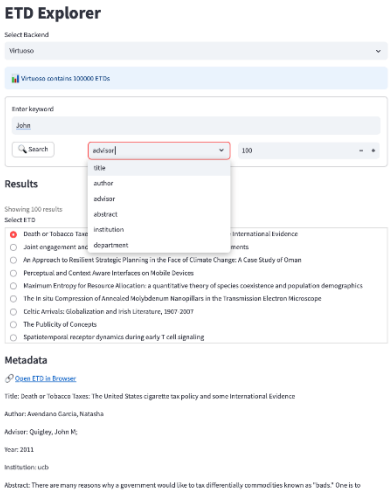

Figure 3: Searching Metadata

The user interface for this project was developed using **Streamlit**, a Python-based framework that enables fast prototyping of data-driven web apps. Our goal was to create an intuitive, responsive UI that could support both **Virtuoso** and **Neo4j** backends, making it easy for users to search Electronic Theses and Dissertations (ETDs) and view related metadata.

Key Features:

- **Backend Selection:** Users can choose between Virtuoso or Neo4j from a dropdown. The backend switch reloads the app with results from the selected database.
- **Keyword Search Bar:** Users can enter keywords and select a metadata field (e.g., title, author, advisor, department, abstract) to search within. The system supports partial matches and case-insensitive search.

- **Results Panel:** A scrollable list displays matching ETD titles. Selecting an ETD reveals detailed metadata about the document.
- **Metadata Viewer:** Below the results, the selected ETD's metadata (title, author, advisor, year, abstract, institution, department) is displayed in a readable format. A direct link to the full ETD (if available) is also shown.
- **Login System:** A simple authentication system was implemented using hashed credentials stored in a local JSON file. Users must log in to access the app, which provides a layer of access control.

5. Implementation

5.1 Implementation for Neo4j

To enhance the search and structure of Virginia Tech's Electronic Theses and Dissertations (ETDs), we built a knowledge graph using Neo4j, a native graph database. Neo4j allows us to represent metadata relationships—such as author, advisor, department, and university—as interconnected nodes, enabling more intuitive and flexible queries than traditional relational databases.

1. Graph Schema and Structure

Each ETD is centered on a Title node, which stores the title, ID, and URI. This node connects to others via relationships, including:

- HAS_AUTHOR → Author
- ACADEMIC_ADVISOR → Advisor
- PUBLISHED_IN → Year
- HAS_ABSTRACT → Abstract
- PUBLISHED_BY → University
- ACADEMIC_DEPARTMENT → Department
- DEGREE_TYPE → Degree
- ACADEMIC_DISCIPLINE → Discipline

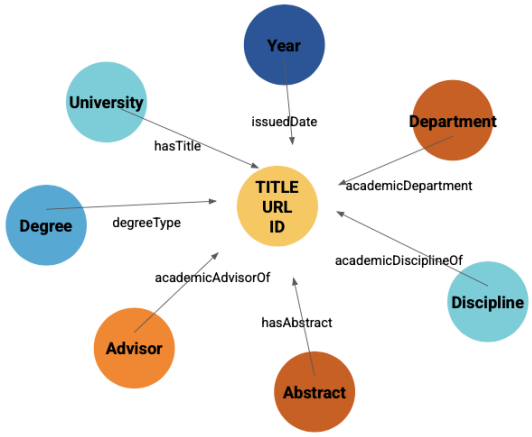

Figure 4: Neo4j Node Structure

This structure makes it easy to query individual metadata types.

2. Data Loading Process

We wrote a Python-based loader that:

- Connects to Neo4j using the official driver
- Parses JSON ETD data converted from the original CSV
- Uses MERGE to avoid duplicate nodes
- Connects all related metadata to the Title node with appropriate relationships

3. Streamlit app with Neo4j

Our Streamlit app incorporates Neo4j backend to:

- Search ETDs by keyword in fields like title, advisor, abstract, etc.
- Retrieve metadata and URI for a selected ETD
- Filter ETDs by year
- Count total ETDs loaded

These queries are powered by Cypher. For example, advisor searches use:

MATCH (t:Title)-[:ACADEMIC_ADVISOR]->(a:Advisor)
WHERE toLower(a.name) CONTAINS toLower($keyword)
RETURN t.uri AS s, t.value AS title

4. Performance and Scalability

We used a local instance for better load speeds and control. The loader is optimized to process thousands of records quickly and can be further tuned for bulk insertion or parallelization.

Overall, the Neo4j backend provides a scalable model for representing ETD metadata. Its graph structure supports deep queries and metadata exploration via a user-friendly Streamlit frontend, offering a powerful tool for maintaining and exploring Virginia Tech's ETD's

5.2 Implementation for Virtuoso

A virtuoso server on the VT endeavor cluster was already in place from previous groups work. The username and password were still the defaults 'dba' and 'admin'. Previous groups loaded triples into the server through python scripts utilizing the requests library for https requests. The endpoint URL that our scripts connect to is https://virtuoso.endeavour.cs.vt.edu/sparql-auth. A new graph was created to store triples at "http://etdkb.endeavour.cs.vt.edu/ETDs"

1. Graph Schema and Structure

The triples in Virtuoso can be divided into two categories; one where objects are URIs and one where objects are only literals. In SPARQL queries the literals are given in quotes like "object" while URIs are given in angle brackets like <object>. Some fields have two triples, one for each category.

This chart shows the different predicates and object type for different fields

Field	Predicate	Type
title	hasTitle	literal
author	Author	literal
author	hasAuthor	object
year	issuedDate	literal
URI	identifier	literal
abstract	hasAbstract	literal
department	academicDepartment	object
advisor	academicAdvisor	literal
discipline	academicDiscipline	object
university	publishedBy	object
keywords	hasKeyword	literal
keywords	hasChapter	object

2. Data Loading Process

The graph is loaded in bulk through virtuosoLoader.py from a json file of ETD metadata. The CSV to JSON script can be used to convert from CSV. The script runs multiple workers in parallel to load large amounts of data quickly. The virtuoso server seemed to slow down in handling requests significantly once it had received around 200,000 ETDs worth

3. Streamlit UI access

The Streamlit UI allows access to the virtuoso graph through VirtuosoQueries.py and the following capabilities:

- Keyword search different predicates such as author, title, abstract, etc.
- Retrieve the metadata and link for ETDs
- Limit number of ETD returned
- View how many ETDs are in the database

4. Performance and Scalability

The VirtuosoLoader.py can upload hundreds of ETDs per second, allowing it to handle large sets of metadata such as the 550,000 ETDs we were tasked with uploading. The queries used to search for ETDs are fairly simple so are quick and allow for a responsive UI. Because data is stored in a knowledge graph instead of a traditional relational database, new fields and relationships can easily be added. This allows the KG to be scaled in the future both horizontally and vertically.

6. Testing and Evaluation

6.1 Database access

To test the frontend, 20 ETDs were loaded manually into virtuoso using conductors import CSV function. The 20 ETDs loaded into Virtuoso were used to confirm that queries to the database return the right information and that the front end displays this data appropriately.

6.2 Database loading speed

The main metric used to evaluate the current database loading pipeline is speed. The method that was used to load a sample of 20 ETD’s metadata is limited to batches of 1 mb or around 200 ETDs. Because this is done through UI it takes roughly a minute. This indicates that loading all the ETDs metadata will take over a day of manually adding small csv files, in addition to the time it takes to split up the csv file. Future methods for uploading data will be timed for a small batch then the predicted total time can be calculated and decided if acceptable. Future work may also include more complex searching methods and queries that better exploit the interconnected nature of KGs.

7. Users' Manual

7.1 Overview

The current graphical user interface is built with Streamlit which allows users to:

- Select which backend (Virtuoso or Neo4j) they want to use
- Fetch and display a list of ETD titles
- Click on title to view metadata (author, advisor, year, etc.)
- Open the ETD document in a new browser

The tool supports two graph-based systems:

- Virtuoso: uses SPARQL queries
- Neo4j: uses Cypher queries

7.2 Installation Requirements

Specific instructions for downloading, installing, and launching from scratch are in the developer's manual. If all code and packages have previously been installed follow instructions below.

- These instructions assume that the desired data has already been loaded into the respective databases, if not follow starting at step 8 or 9 in Developer's Manual.
- Open the Neo4j database and hover over the name of the DBMS to select start.

Figure 5: Neo4j Start Button

- When prompted select 'Continue Anyways'

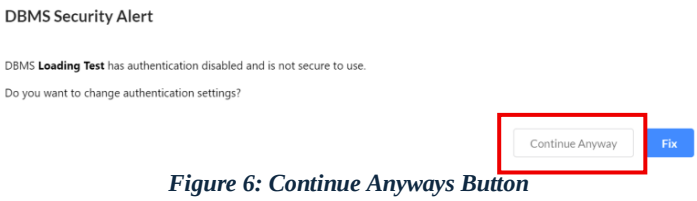

Figure 6: Continue Anyways Button

- Launch the Streamlit site
  - streamlit run StreamUI.py

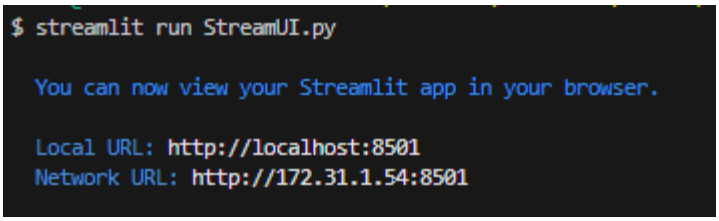

Figure 7: Command to Run Streamlit

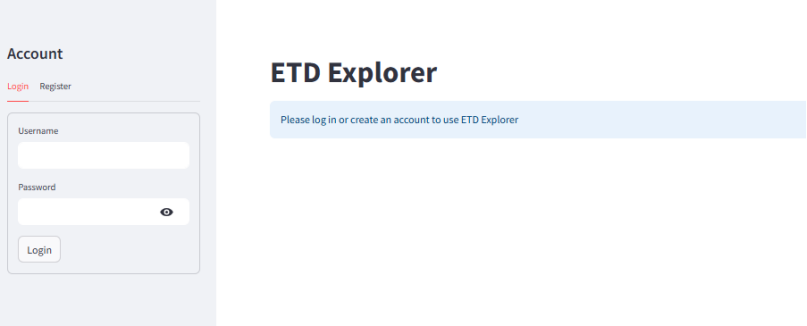

Figure 8: ETD Explorer Login Page

7.3 Using Application

- The user has the option to create an account if they have not before.

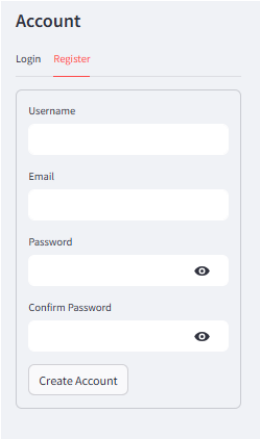

Figure 9: Register Account

- If an account has been created before, then the user should login with a previously set username and password.

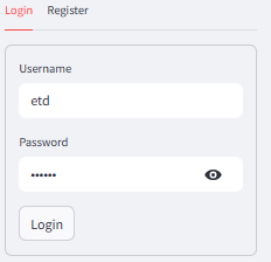

Figure 10: User Login

- Upon login the user the page is represented in Figure 7.

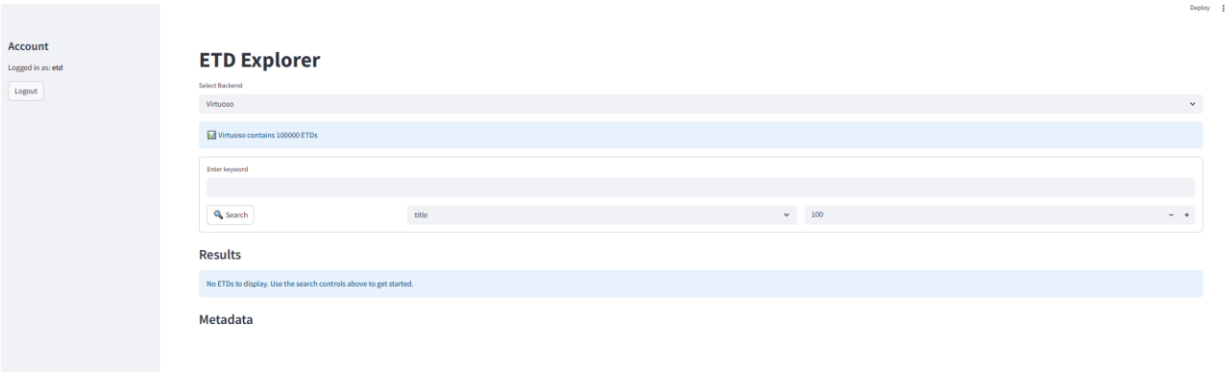

Figure 11: ETD Explorer Original State

- Virtuoso is the database that is automatically selected, Title is search category, and the limit to the number of ETDs found in a search is 100. The database and search category can be changed via a drop down and the limit can be changed either by using the '+' and '-' buttons or typing in the number.
- To search for ETDs, type desired field into the search bar and hit the search button or enter.

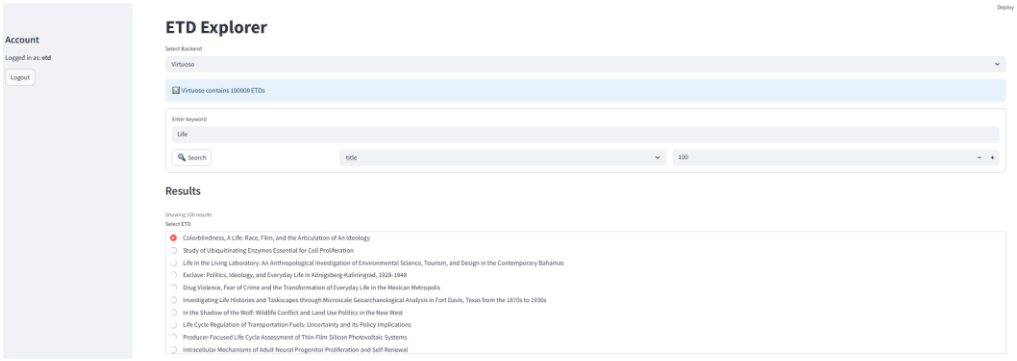

Figure 12: Searching for ETDs

- To see the metadata of an ETD select the circle next to the title of ETD desired. The metadata is then displayed below.

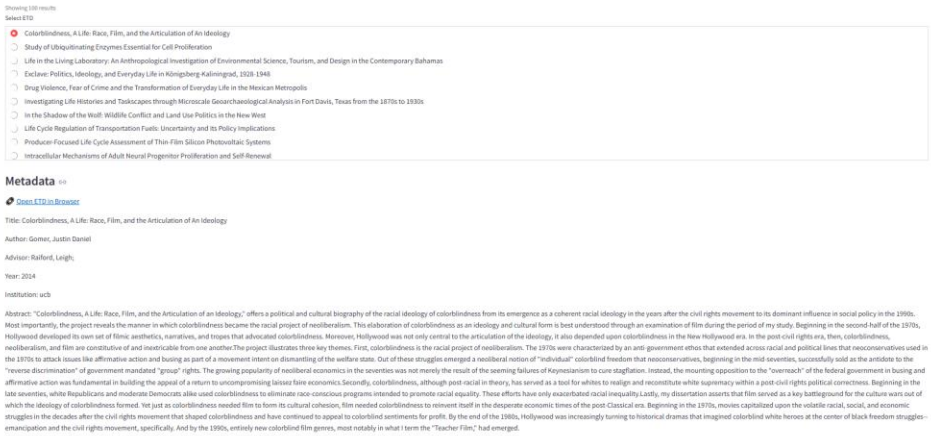

Figure 13: Metadata of an ETD

- To read the ETD select ‘Open ETD in Browser’ and the ETD will open in another window.

Figure 14: Open ETD in Browser Link

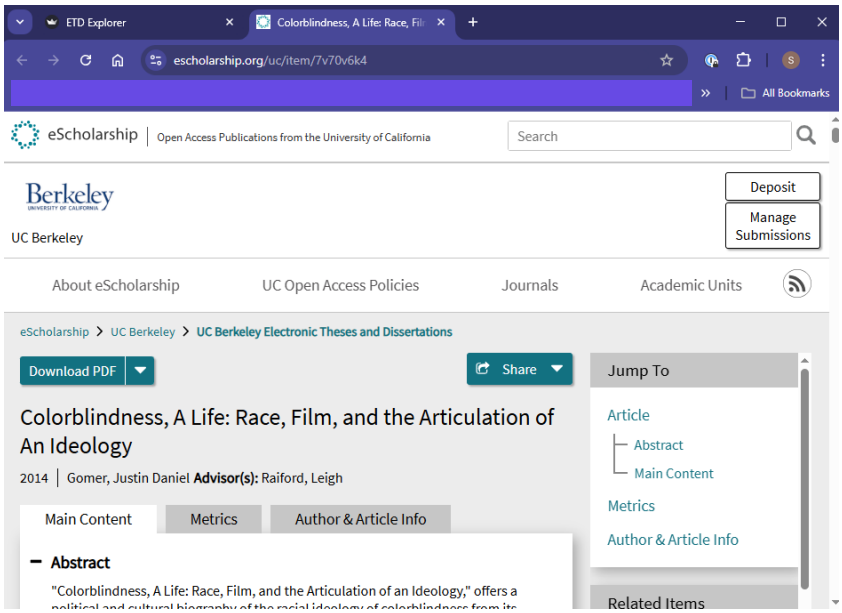

Figure 15: ETD Open in Another Browser

The search process mentioned can be down for both Neo4j and Virtuoso and for all the search categories: Title, Author, Advisor, Abstract, Institution, and Department. The user can then log out when finished with their session.

8. Developer's Manual

8.1 Overview

This manual serves as a guide for developers working with Virtuoso for ETD storage. It provides information on each system, their architecture, diagrams, installation, and troubleshooting to ensure a smooth transition for new developers.

System Overview

- Virtuoso is a widely used, multi-model database that supports relational, graph-based, and linked data storage. This database was used because it supports SPARQL queries for semantic searches, it efficiently handles linked data and RDF storage, and it is easily scalable and suitable for large datasets
- Neo4j is a graph database that represents and stores data as nodes, relationships, and properties rather than traditional relational tables. This database was used because it has efficient querying of relationships between different entities, it supports complex search patterns using Cypher query language, and it provides visualization capabilities for relationships between entities.

Architecture

Virtuoso stores data in a triple-store model, which consists of:

- Subjects – Which store ETDs, authors, topics, etc.
- Predicates – Which demonstrate the relationships between entities, such as "written by" or "references")
- Objects – Which represented the metadata or linked entities

Neo4j uses a property graph model, which consists of:

- Nodes – Which represent ETDs, authors, topics, etc.
- Relationships – Which define how nodes are connected, such as "written by" or "cites"
- Properties – Which includes metadata such as title, year, or department

Diagrams

Top-Down			Bottom-Up		
Subject	**Predicate**	**Object**	**Subject**	**Predicate**	**Object**
ETD	hasTitle	Title	Title	titleOf	ETD
ETD	hasAuthor	Author	Author	authorOf	ETD
ETD	issuedDate	Date			
ETD	publishedBy	University	University	hasPublished	ETD
University	location	University Location			
ETD	academicAdvisor	Committee Chair	Committee Chair	academicAdvisorOf	ETD
ETD	committeeMember	Committee Member	Committee Member	committeeMemberOf	ETD
ETD	degreeType	Degree			
ETD	academicDiscipline	Department	Department	academicDisciplineOf	ETD
ETD	hasKeyword	Subject/Keywords	Subject/Keywords	keywordOf	ETD
ETD	identifier	URI			
ETD	writtenIn	Language			
ETD	hasRights	Rights			
ETD	supportedBy	Sponsoring Agency			
ETD	hasAbstract	Abstract			
ETD	hasAbstract	Abstract General			
ETD	hasContents	List of Contents Heading			
List of Contents Heading	hasPart	List of Contents Text			
ETD	hasChapter	Chapter Title	Chapter Title	chapterOf	ETD
Chapter Title	hasSummary	Chapter Summary			
Chapter Title	hasPart	Section	Section	partOf	Chapter Title
Chapter Title	hasPart	Paragraph			
Chapter Title	hasFigure	Figure	Figure	figureOf	Chapter Title
Figure	hasCaption	Figure Caption	Figure Caption	captionOf	Figure
Chapter Title	hasTable	Table	Table	tableOf	Chapter Title
Table	hasCaption	Table Caption	Table Caption	captionOf	Table
Chapter Title	hasEquation	Equation	Equation	equationOf	Chapter Title
Equation	hasPart	Equation Number			
Chapter Title	hasAlgorithm	Algorithm	Algorithm	algorithmOf	Chapter Title
Chapter Title	hasPart	Footnote			
ETD	hasCited	Reference Text	Reference Text	citedBy	ETD
ETD	citedAuthor	Cited Author	Cited Author	citedAuthorOf	ETD
Chapter Title	classifiedAs	Classification Label	Classification Label	classificationOf	Chapter Title
Chapter Title	hasTopic	Topic Label	Topic Label	topicOf	Chapter Title
ETD	hasTopic	Topic Label	Topic Label	topicOf	ETD
ETD	hasPart	Page Number			
ETD	dedicatedTo	Dedication text			
ETD	hasAcknowledgement	Acknowledgement			

Figure 16: ETD Node Relationships Used by Both Virtuoso and Neo4j Databases

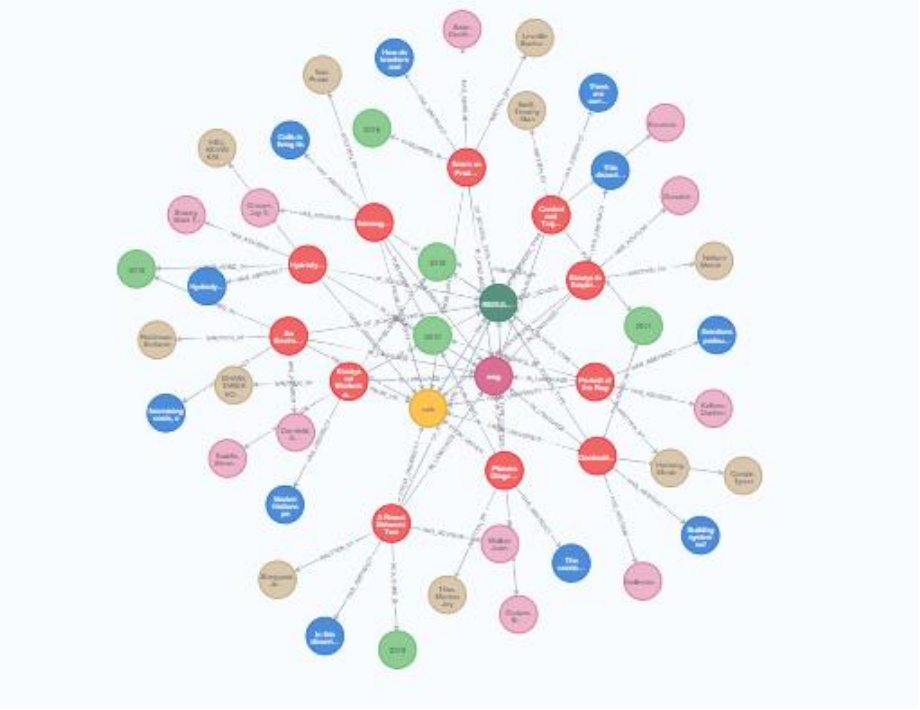

Figure 17: Neo4j Graph Relationships Example

8.2 Generic Manual Installation via GUI

1. Download code editing software, VS Code is recommended
    a) Link: https://code.visualstudio.com/
2. Packages to install:
    a) Example: pip install package_name==version
    b) pip install streamlit==1.44.1
    c) pip install requests==2.32.3
    d) pip install tqdm==4.67.1
3. Download Neo4j local database
    a) Link: https://neo4j.com/download/
    b) Recommended system requirements: MacOS 10.10 (Yosemite)+, Windows 8.1+ with Powershell 5.0+, Ubuntu 12.04+, Fedora 21, Debian 8.
    c) Fill out form as requested

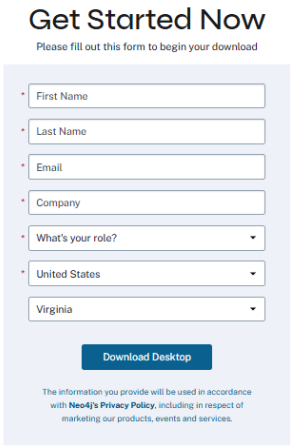

Figure 18: Form to Download Neo4j Local Database

    d) Copy Neo4j Desktop Activation Key

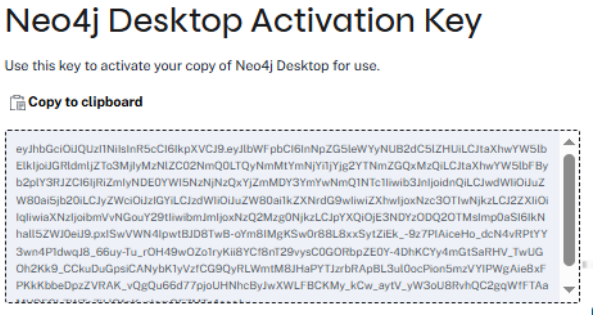

Figure 19: Neo4j Desktop Activation Key

    e) Once the download is complete open the Neo4j Desktop Installer and follow on screen directions to complete installation
    f) To launch the database Copy and Paste the activation at the top of this page in the "Activation Key" box in the Neo4j Desktop app. Alternatively, you can also

generate a key from within the app by filling out the form on the right side of the app screen.

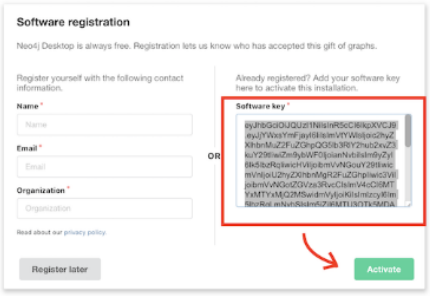

Figure 20: Neo4j Activation Key

g) Once activated and Neo4j is running locally, create a new project

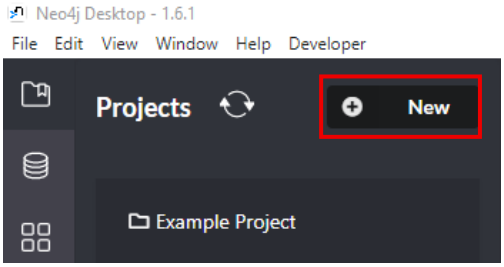

Figure 21: Create new project

h) Add a “Local DBMS”

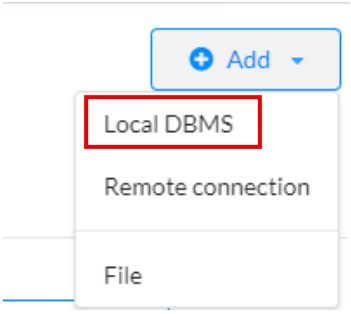

Figure 22: Create local DBMS

i) Make a password and hit create

Figure 23: Create button

j) For this project authentication needs to be removed from the database. Hover over the name of the DBMS and select the three dots on the right and click on settings.

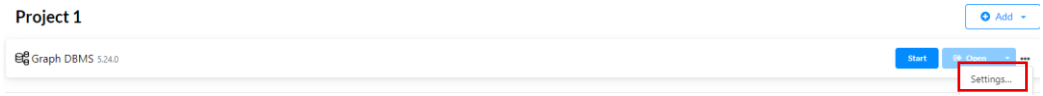

Figure 24: Begin editing authentication settings

k) Change “dbms.security.auth_enabled=true” to false and click Apply

```
# Whether requests to Neo4j are authenticated.
# To disable authentication, uncomment this line
dbms.security.auth_enabled=true
```

Figure 25: Authentication line to change

l) To load data into Neo4j the database needs to be running, so hover over the name of the DBMS and select start.

Figure 26: DBMS start button

m) When prompted select 'Continue Anyways'

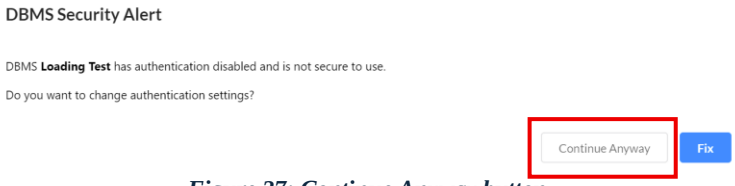

Figure 27: Continue Anyway button

4. There is nothing required to install to run the Virtuoso database because it is running on a Virginia Tech teaching cluster.
5. Download the code for this project from the repository linked below, by using the command below
    a) https://github.com/dashiellbuddVT/CS-Capstone-Group-20.git
    b) git clone https://github.com/dashiellbuddVT/CS-Capstone-Group-20.git
6. Change directories to be in CS-Capstone-Group-20
    a) cd CS-Capstone-Group-20/
7. File overview:
    a) CSVtoJSON.py – Used to change CSV data into Json form (both Virtuoso and Neo4j ingest data from this format)
    b) Neo4jQueries.py – Frontend uses this to search metadata in Neo4j
    c) Neo4j_loader_v2.py – Used to ingest data into Neo4j local database
    d) StreamUI.py – Frontend, displays all information from both databases
    e) VirtuosoLoader.py – Used to ingest data in Virtuoso cloud database
    f) VirtuosoQueries.py – Frontend uses this to search metadata in Virtuoso
    g) users.json – Json that stores all usernames and passwords that could be created
8. Load data into Virtuoso:
    a) Convert your CSV data to JSON format using CSVtoJSON.py
        - Python CSVtoJSON.py input.csv output.json
    b) Once you have your data in JSON format, use VirtuosoLoader.py to load it into Virtuoso
        - Optional parameters for VirtuosoLoader.py
            - `--max-batches N`: Limit the number of batches to load (each batch contains up to 100 ETD records)

- `--workers N`: Set the number of parallel workers (default: 4)
- `--clean`: Clear the existing graph before loading new data
- `--force`: Force loading even if write permission check fails

c) The loader will:
- Connect to the Virtuoso endpoint at https://virtuoso.endeavour.cs.vt.edu/sparql-auth
- Split your data into batches (100 ETDs per batch)
- Process batches in parallel using multiple workers
- Report progress and statistics during and after loading

d) Example command for loading all ETDs with 8 parallel workers after clearing the existing data:
- python VirtuosoLoader.py output.json --workers 8 --clean

9. Load data into Neo4j:
   a) Convert CSV data into a Json using CSVtoJSON.py
      - python CSVtoJSON.py Test_ETD_10.csv output_file_10.json
   b) Load Json into Neo4j using Neo4j_loader_v2.py
      - python Neo4j_loader_v2.py output_file_10.json
      - Neo4j needs to be running locally, Step 3l
   c) To verify loaded data, click the 'Open' button in Neo4j window, move to the window that gets opened, run the command 'MATCH (n) RETURN n LIMIT 50'

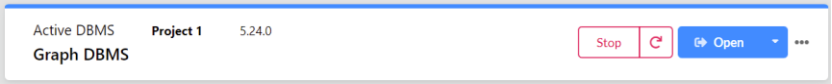

Figure 28: Open Button

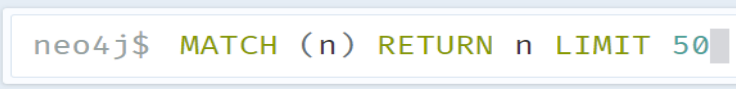

Figure 29: Validate data in Neo4j command

10. Launch the frontend
    a) streamlit run StreamUI.py

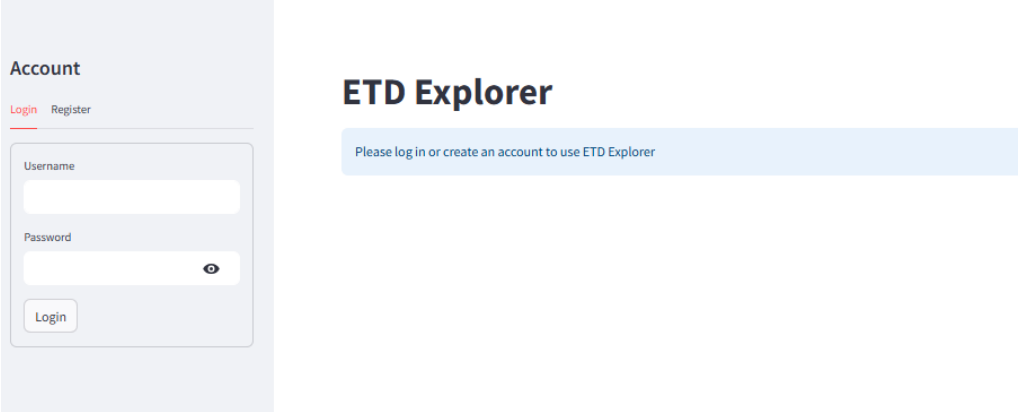

Figure 30: ETD Explorer Login Page

8.3 Generic Manual Installation via CLI

1. Packages to install:
    a) Example: pip install package_name==version
    b) pip install streamlit==1.44.1
    c) pip install requests==2.32.3
    d) pip install tqdm==4.67.1
2. Install Neo4j
    a) sudo apt install neo4j
3. Change authentication settings
    a) Command ‘cd /usr/share/neo4j/conf’
    b) Command ‘sudo vim neo4j.conf’
    c) Press letter I on keyboard to enter INSERT mode
    d) Press enter to make edits
    e) Uncomment #dbms.security.auth_enabled=false, delete ‘#’
    f) Press escape and type ‘:wq’ to save and quit vim
4. Start Neo4j
    a) sudo neo4j start
5. Get starter code
    a) Link: https://github.com/dashiellbuddVT/CS-Capstone-Group-20.git
    b) git clone https://github.com/dashiellbuddVT/CS-Capstone-Group-20.git
6. Load Neo4j
    a) Convert CSV to Json
        - python CSVtoJSON.py input.csv output.json
    b) Load Neo4j
        - python Neo4j_loader_v2.py output.json
7. Launch frontend
    a) ‘streamlit run StreamUI.py’
    b) If error command not found run ‘~/.local/bin/streamlit run StreamUI.py’

8.4 Troubleshooting

System	Issue	Possible Cause	Solution
Virtuoso	Server not starting	Configuration error	Check virtuoso.ini settings
Virtuoso	Query is slow	Missing indexes	Create indexes on frequently search properties
Neo4j	Database not starting	Port conflict	Change port in neo4j.conf
Neo4j	Query is slow	Missing indexes	Create indexes on frequently search properties

9. Lessons Learned

9.1 History

This project builds on the work of a prior team that began the Knowledge Graph Building effort. Their task was to transform XML-formatted ETD metadata into RDF triples and load them into a Virtuoso database. They successfully set up a local Virtuoso instance, tested data upload for a sample of records, and developed SPARQL queries for searching. However, they faced challenges in scaling their pipeline to support all 500,000 ETDs and in implementing large-scale URI resolution.

In this semester's continuation, we expanded the project to use JSON-formatted metadata instead of XML and included Neo4j as a second database backend to improve query flexibility. We cleaned and formatted metadata from 20,000+ ETDs, wrote Python-based batch loaders, and connected a Streamlit frontend to both Virtuoso and Neo4j. We optimized our Neo4j schema to remove reliance on a central ETD node, implemented efficient batch loading, and deployed the UI with login features. Our additions addressed previous limitations by building a scalable dual-database system with modular backend querying and a unified web interface.

9.2 Timeline/Schedule

The project timeline was structured to ensure steady progress toward our goals. Below is a table summarizing the key milestones, their planned dates, and their status as of March 26, 2025.

Task	Start Date	End Date	Status	Notes

Virtuoso server setup	M01	M15	Completed	Endpoint URL established: https://virtuoso.endeavour.cs.vt.edu/sparql-auth.
SPARQL queries for titles, links, metadata	M01	M15	Completed	Queries successfully retrieve metadata from Virtuoso.
GUI built with Python + Streamlit	M01	M15	Completed	UI connected to Virtuoso for live search; basic functionality implemented.
Extract + format data from 20,000+ ETDs	M25	A04	Completed	Cleaned malformed fields and structured JSON for both databases
Load data into Virtuoso + Neo4j	A04	A11	Completed	ETDs inserted using SPARQL and Cypher batch loaders
Build Neo4j schema + data pipeline	A11	A18	Completed	Removed central ETD node, improved relationship granularity
Full UI with backend switching	A18	M01	Completed	Users can toggle between Neo4j and Virtuoso
Scale to 500,000 ETDs + finalize UI	M01	M08	Could be improved	Prepped for full 500k ETDs; added filters, improved load time

9.3 Problems

Reflecting on our progress, several areas remain for future development:

- **Slow load speeds in Neo4j Aura:** Initially used AuraDB but switched to a local instance for better speed and memory control.

- **No central ETD node in new schema:** Adapting the frontend to work without a central node required restructuring metadata queries.

- **Dual backend testing:** Maintaining functional parity across Virtuoso and Neo4j was time-consuming and required parallel implementations.

9.4 Solutions

- Designed a simplified and scalable schema for Neo4j using the Title node as the anchor.
- Switched to local Neo4j for efficient insertions and batch processing.
- Refactored backend modules for streamlined search and metadata display.

9.5 Future work for Subsequent Teams

Although this project established a foundation for scalable ETD knowledge graphs using Virtuoso and Neo4j, there are opportunities for development for future teams

- **Full dataset loading:** Scale up to 500,000+ ETDs in both databases. Improve the loader script with multi-threading or batch optimization for faster ingestion.
- **Deploy Neo4j to the cloud:** Move the Neo4j database to a scalable cloud platform like AuraDB or Neo4j Enterprise on AWS to support larger datasets and concurrent users.

- **Graph visualization tools:** Add dynamic graph visualization inside the UI to help users explore metadata relationships in an intuitive way.

- **Streamlit deployment:** Host the Streamlit app on a production-grade server or cloud platform to allow public or institutional access.

- **Improved user experience:** Enhance UI responsiveness and add features like bookmarkable search results or exportable metadata views.

10. Acknowledgements

We want to acknowledge Mohamed Farag for his invaluable guidance and support throughout the project, as well as Satvik Chekuri for his technical expertise and insights, which significantly advanced our work.

Contact Information:

Mohamed Farag
mmagdy@vt.edu
Department of Computer Science, Virginia Tech

Satvik Chekuri
satvikchekuri@vt.edu
Department of Computer Science, Virginia Tech

11. References

1. IBM Knowledge Graph: IBM. (n.d.). *Knowledge Graph*. IBM. Retrieved March 27, 2025, from https://www.ibm.com/think/topics/knowledge-graph

2. Neo4j Desktop Download & Installation Guide: Neo4j. (n.d.). *Neo4j Desktop download & installation guide*. Neo4j. Retrieved March 27, 2025, from https://neo4j.com/download-thanks-desktop/?edition=desktop&flavour=winstall64&release=1.6.1&offline=true#installation-guide

3. Neo4j Documentation: Neo4j. (n.d.). *Neo4j documentation*. Neo4j. Retrieved March 27, 2025, from https://neo4j.com/docs/

4. Virtuoso: OpenLink Software. (n.d.). *Virtuoso*. OpenLink Software. Retrieved March 27, 2025, from https://virtuoso.openlinksw.com/

5. VTechWorks (Virginia Tech Digital Repository): Virginia Tech. (n.d.). *VTechWorks*. Virginia Tech. Retrieved March 27, 2025, from https://vtechworks.lib.vt.edu/