Automating Textbook Requirements From Course Syllabi and Availability Tracking For Libraries

Parsa Mallik
Earlham College
Richmond, Indiana, USA
pmallik22@earlham.edu

ABSTRACT

This project focuses on developing a tool designed to automate extracting and tracking required and recommended readings from course syllabi at Earlham College. The tool utilizes zero-shot named entity recognition (NER) to parse syllabi and accurately identify textbook information, even when presented in varied and incomplete formats. Extracted data is stored in a database and further refined by web scraping to verify the availability of these textbooks in the Lilly Library's catalog. The tool distinguishes between physical books and eBooks and updates the database accordingly. By automating this process, the tool will enable the Lilly Library to proactively ensure that essential textbooks are available to students, potentially reducing their financial burden. This project not only aims to benefit Earlham College but also holds the potential to be adapted for use in other educational institutions globally, promoting broader access to academic resources.

KEYWORDS

Natural language processing (NLP), web scraping, user interface, and user experience (UI/UX), user-centered design (UCD), system usability scale (SUS), fine-tune, zero-shot named entity recognition (NER)

1 INTRODUCTION

Course syllabi are crucial for guiding students through their academic coursework, detailing the required and recommended readings that are essential for their success. However, at Earlham College, there is currently no automated system to ensure that these textbooks are available in the Lilly Library, leading to a situation where students often need to purchase books that might already be accessible through the college's resources. The existing process relies on professors to notify the library about necessary books, but this approach is inconsistent and can result in gaps in the library's offerings.

This project addresses this issue by developing a tool that automatically processes course syllabi to extract textbook information, even when the data is presented in varied or incomplete formats. The tool utilizes a pre-trained large language model (LLM) to perform zero-shot NER to identify and compile the necessary readings' in-publication data into a database accurately. It then uses web scraping technology to check the availability of these books in the library's catalog. By ensuring that all essential readings are easily accessible, this tool aims to reduce the financial burden on students and improve the efficiency of resource management at Earlham College. Additionally, the framework of this tool could be adapted for use in other academic institutions, offering a scalable solution to enhance library services and support student learning on a broader scale.

2 RELATED WORK

2.1 Fine-tuning large language models

Large Language Models (LLMs) are sophisticated algorithms capable of processing and generating human-like text by leveraging vast datasets and advanced architectures, such as transformers. Fine-tuning refers to adapting a pre-trained LLM to specific tasks or domains by training it on a smaller, task-specific dataset. This process is crucial as it enhances the performance and accuracy, enabling it to understand specialized terminology and context[Nav+23]. Several recent studies have explored different approaches to fine-tuning LLMs for various applications.

Xu et al. emphasize the importance of effective and generalizable fine-tuning strategies for LLMs in their work. They propose methods to enhance the adaptability of LLMs, ensuring that models can learn from limited domain-specific data while maintaining their general language capabilities[Xu+21]. Li et al. proposed a novel framework that aligns generated internal knowledge with external knowledge through in-context learning (ICL) to improve the accuracy and reliability of LLM outputs for automated clinical data extraction[Li+24]. Their approach employs a retriever to identify relevant units of internal or external knowledge and a grader to evaluate the truthfulness and helpfulness of the retrieved internal-knowledge rules to align and update the knowledge bases.

Patil and Gudivada provide a comprehensive review of LLMs, discussing various fine-tuning techniques tailored for specific downstream tasks[PG24]. They highlight methods such as discriminative fine-tuning, which adjusts model parameters based on task-specific datasets, and gradual unfreezing, which allows for selective training of model layers. Howard and Ruder introduced another general fine-tuning method called Universal Language Model Fine-tuning (ULMFiT) that achieves state-of-the-art performance on a wide range of text classification tasks[HR18]. ULMFiT uses discriminative fine-tuning, slanted triangular learning rates, and gradual unfreezing to adapt a language model to a target task. The authors demonstrate the effectiveness of their approach on six text classification tasks, achieving significant improvements over previous methods.

2.2 Zero-shot named entity recognition (NER)

NER is another approach to extracting specific information from large structured or unstructured text. NER is a natural language processing (NLP) technique that can identify entities such as

names of people, organizations, and locations, and numeric expressions, including time, date, money, and percent expressions[NS07]. Traditional methods of NER involve using handcrafted rule-based algorithms, which are limited to a predefined set of entity types. LLMs enhance the applicability of NER by allowing open-type recognition. Now, models can identify an arbitrary number of entity types based on natural language instructions. However, this process can be cost-heavy due to the computational resources required for large and complex models.

Zero-shot NER eliminates this cost by enabling models to recognize entities they have not been explicitly trained on. It uses the contextual understanding inherent in LLMs, which allows them to infer entity types without requiring additional labeled data[Zar+23]. In a zero-shot setting, smaller pre-trained models like GLiNER can outperform larger LLMs, making them efficient for resource-limited environments.

2.3 Web scraping

Web scraping is the process of extracting data from websites. The process is divided into three key stages: fetching, extracting, and transforming [Khd21]. The first step involves accessing the relevant website using the HTTP protocol. Important data is then extracted from the website using techniques such as regular expressions, HTML parsing libraries, and XPath queries. Finally, the extracted data is converted into a structured format for storage and further processing.

Glez-Pena et al. demonstrated the usefulness and simplicity of data scraping in addressing numerous practical needs, using a biomedical data extraction scenario as an example [Gle+14]. The authors highlighted the potential of web scraping in various domains, paving the way for its wider adoption. To enhance the efficiency of web scraping, researchers have explored advanced techniques and tools. OXPath, an extension of XPath based on the XML query language, enables the simulation of user interaction on web pages and facilitates data extraction and manipulation. Neumann et al. utilized OXPath to extract relevant papers from Google Scholar and other digital libraries, demonstrating the tool's versatility in navigating through search results and accessing specific content [NSS17].

2.4 Database interface

Traditional metadata formats like Machine Readable Cataloging (MARC) are insufficient for capturing the complex bibliographic relationships and semantics required for effective digital library operations. Creating ontological metadata by extracting and transforming MARC data allows advanced search, browsing, and reasoning capabilities [WB98].

Weinstein uses a set of control files to map each selected MARC field and code to one or more ontology concepts to convert binary MARC data to text tagged with ontology concept [Wei98]. The coded attributes and values from natural-language comments in the tagged text are then extracted using a lexicon and natural-language processing. The tagged text is then converted to Loom assertions to use its reasoning capabilities to unify matching instances and deduce relations between works. This process converts the relationships that are implicit in the MARC data into explicit relations that can be easily utilized by computers.

Data can also be directly extracted from textual documents and added to ontological metadata. Monica et al. used Application Programming Interfaces (APIs) to extract metadata on a corpus of HTML documents, including file name, author, and creation date, and effectively mapped them to ontology-based representations of the textual documents. An algorithm called "Extract-Align" was used to extract relevant terms from the HTML files and populate the ontology [CC06]. The ontology-based representation enabled improved organization, searching, and retrieval of the textual documents.

Ontological metadata mapping can be used to automate the database mediation process, where data is distributed across multiple heterogeneous databases. Nadkarni et al. described a rule-based framework for defining ontological metadata mapping rules. These rules rely on elements of a global vocabulary, which allows a query specified in one database to be automatically translated and executed in other databases [MWN09].

2.5 User-centered UI/UX

User-centered design (UCD) is a method used to alleviate system usage difficulties. It focuses on user habits and preferences in designing User Interface and User Experience (UI/UX) [Ind22]. UCD follows four key principles: user focus, integrated design, user testing, and interactive design [LR23].

Lumingkewas et al. applied UCD to improve the UI/UX of the Academic Information System (AIS) at their university [LR23]. They began by signing interview agreements with developers and experts, then identified the context of use through interviews. The System Usability Scale (SUS) was used to evaluate the UI/UX via a questionnaire of 330 student users. Based on the feedback, a design solution was developed using Figma, and the SUS was repeated, showing improved user satisfaction.

Similarly, Indah applied UCD to develop a library website for the Faculty of Computer Science at Sriwijaya University [Ind22]. Librarians had struggled with managing the book collection database using Excel, but the university required an accessible UI for approval of an online application. By creating user personas and gathering input through surveys and interviews, Indah developed an application that met user needs. The SUS method confirmed its success, and the application was approved for library use.

3 DESIGN

The initial design and implementation of the tool will be tailored for use in Lily Library in Earlham College. Earlham College stores all course syllabi in a Box Folder that is widely available to all students, faculty, and staff members on campus. Most of the files are in PDF format, and this tool will only focus on that format. To cater to Earlham College's system, the initial step of gathering all the PDF files from different directories into one directory is done manually before using the tool. The gathered PDF files are then fed to the tool which automatizes the next steps as detailed below.

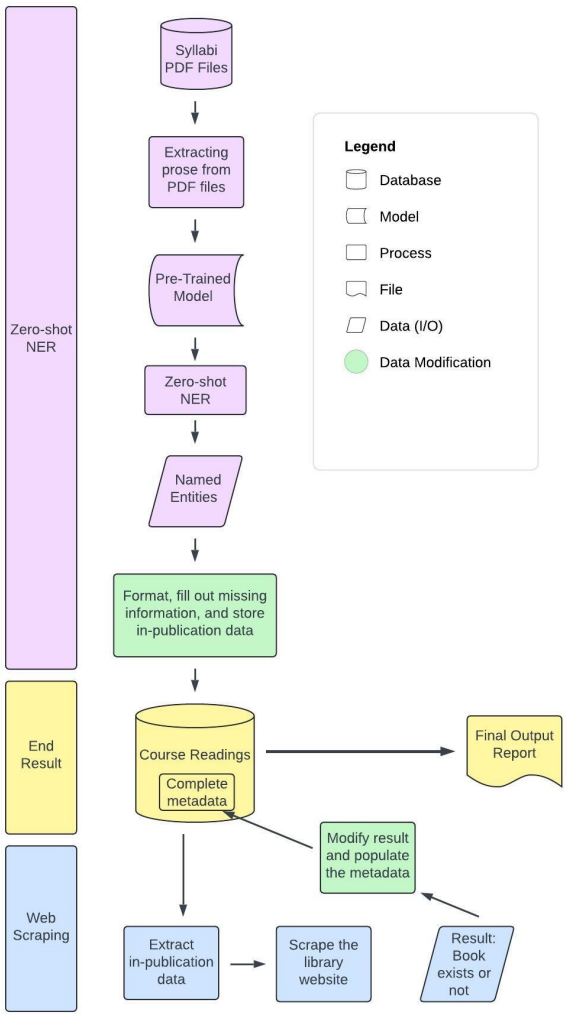


Figure 1: Data architecture diagram

The following data architecture diagram outlines a pipeline for processing course syllabi to automatically extract and track required and recommended readings. The workflow begins with PDF syllabi files, from which prose is extracted and fed into a pre-trained LLM that performs zero-shot NER to recognize and extract reading information. The extracted reading information is then formatted and stored in a central database. A web scraping module queries the library catalog to verify availability, updating the metadata with availability information. The final output is a report detailing course readings and their availability status, facilitating resource tracking for the library.

3.1 Fine-tuning LLM or using zero-shot NER for extracting reading information

The platform will initiate the process by accessing course syllabi in PDF format from the user. It will then extract prose from the PDF files and save them in subsequent text files in another directory. Depending on which method produces better results, the tool will use either fine-tuning or zero-shot NER on pre-trained LLMs to extract reading information.

3.1.1 Fine-tuning LLM. An appropriate pre-trained model and tokenizer will be loaded and saved to utilize the fine-tuning approach. The preprocessed text files will be fed to this model for fine-tuning. The tokenizer will transform the text into a format the model can understand. The pre-trained model will then be fine-tuned on these tokenized inputs. The fine-tuned model will be queried to extract required and recommended readings from the text files it has been trained on. The model will process this query and generate a file containing a list of readings for further processing, which is elaborated in section 3.3.

3.1.2 Zero-shot NER. The zero-shot NER approach will load a smaller pre-trained LLM. A list containing the labels of entities, like 'textbooks' and 'books', and the extracted text from the text files will be given to the model to predict the entities. The predicted entities would be a list of all the required readings in the syllabi, which will be used to populate the database explained in section 3.3.

3.2 web scraping to determine the availability of the books

The program will extract in-publication data from the course reading database and use the HTTP protocol to access the user-specified library website. It will employ a Python library called Selenium to search for book titles listed in the database, parsing the search results to identify matches. The program further verifies whether the books found on the website are eBooks or physical books and if they align with the complete publication information provided in the database. As it processes each book, the program generates results indicating whether a match has been found for each entry in the database.

3.3 Database interface

	data extraction from syllabi			web scraping	
ID	**ISBN**	**Book Title**	**Author**	**Availabity**	**Type**
1	1234567891011	Physics for Scientists and Engineers	John W. Jewett	yes	eBook
2	...	...	...	yes	physical
3	...	....	...	no	N/A

Figure 2: Database structure plan

Throughout the various stages of its operation, the program repeatedly modifies data to prepare it for further processing, to store it in specific formats, or to present it to the user in a human-readable form.

The result generated by the fine-tuned LLM or zero-shot NER, which contains the course reading descriptions, is first used to populate a database of course readings. Given the variability in how each syllabus lists its readings, many entries lack complete

publication details. To address this, the program utilizes a Python library, isbntools, to retrieve full in-publication data for the books. This retrieved information is then stored in a structured, well-formatted manner, as shown in figure 2.

The database is populated and properly formatted to ensure that the necessary information for web scraping the library's website is easily accessible. After scraping is complete, the results indicating the availability of each book are updated in the course readings database, indicating whether it is an eBook or physical copy.

In the final step, the complete course reading database, now containing both in-publication data and library availability, is converted into a human-readable report. The report is formatted as a spreadsheet, enabling librarians to review the course readings and easily check whether each book is available in the library.

3.4 User-centered UI/UX

The UI will allow librarians to provide the directory containing the course syllabi and a link to the website that they want the system to scour.

To develop a UI that is easily navigable by most librarians, the five-step UCD will be employed [LR23]. Librarians at Lilly Library will be consulted to understand their needs. They will also be surveyed with specified questions that will help design the UI of the platform. The survey results will be used to design the platform on Figma, which will then be shown to the primary users of the platform. They will be surveyed again, where they will be evaluating the usability of the system. The SUS method will be used to evaluate the survey results to determine how suitable the design is for librarians.

4 METHODS

4.1 Pre-processing

The initial step in preparing data for data extraction involves extracting text from syllabi provided in PDF format. Using the PDFReader function from Python's PyPDF2 library, I developed a script that systematically reads each PDF file provided by the user, extracts its textual content, and saves the output as a plain text (.txt) file. Each text file is stored in a designated directory, simplifying access and organization for later processing stages. This step ensures that the text is readily available for the subsequent information extraction phase without needing to access or process the original PDFs.

4.2 Fine-tuning

For the language model fine-tuning stage, I selected EleutherAI's GPT-J-6B, a pre-trained LLM known for its performance in natural language understanding tasks. To streamline future fine-tuning and avoid repeated downloads, the model and its associated AutoTokenizer were downloaded and saved locally. This setup allows for efficient reuse and quick access during multiple fine-tuning sessions, ensuring that the model can be further tailored to the project's task of extracting reading lists from text data. To check if the pre-trained model was correctly loaded, I gave it random proses to see if it generated responses, which worked in an appropriate way.

I ran a Python script that interacts with the model and reads the preprocessed text files. The AutoTokenizer processed the text by converting it into tokenized IDs, transforming it into a format the model can understand. These tokenized IDs were then organized into tensors and loaded into an iterable dataloader for training. The pretrained GPT-J model was fine-tuned on these tokenized inputs to adjust its parameters and optimize its performance for the target task.

Once the model was fine-tuned, I gave it a query instructing extraction of all required and recommended readings from the input text files. The fine-tuned language model processed this query, and I stored the generated response in a text file. The response text file was evaluated to see if it correctly identified the course reading requirements from the syllabi.

Evaluating the response generated by the fine-tuned model, I saw that it did not list any required readings. I concluded that this process would not be an appropriate method for data extraction and resolved to use zero-shot NER, described in the next section.

4.3 Zero-shot NER

I chose a smaller pre-trained model for zero-shot NER called GLiNER. This is a NER model capable of identifying any entity type using a bidirectional transformer encoder (BERT-like). It provides a practical alternative to traditional NER models, limited to predefined entities and LLMs.

I developed a Python script that takes the directory containing the syllabi text files and combines them into a single string before splitting them into several smaller chunks of strings in a list. I defined a list of labels that contains 'textbooks' and 'books'. The script then utilized GLiNER to perform zero-shot NER on the list of strings using the labels I defined. This process predicted entities from the strings based on the specified labels and generated a list of labeled entities.

To check if this process produced accurate results, I ran the script with one syllabus and printed out the entities list on a text file. I then verified it by manually checking the syllabus for book titles. This method produced better results than the fine-tuning method so, the rest of the project was built with the result generated by zero-shot NER.

4.4 Data modification

In this phase, I used the result produced by the previous step to create a structured database of course reading materials. This database serves as the foundation for performing web scraping on the library's website to verify the availability of the listed readings.

I utilized SQLite to create a database for structured management and querying. I added functions to the Python script that performs zero-shot NER that takes the labeled entities and stores them into an SQLite3 database in the format outlined in figure 2. While populating the database with the book titles from the labeled entities, the script retrieves the ISBN of the books using a Python library called isbntools and stores them in the database.

The script also utilizes a text file provided by the user, which contains all course names, school-specific acronyms, or other common names that might be mistaken for book titles by the NER model, and filters them out while populating the database. At this

stage, the database had the 'data extraction from syllabi' portion labeled in figure 2 mostly filled out for web scraping.

4.5 Web scraping

This step loops through every entry in the database, gets the book title, and checks if they are available on the library website. The tool takes the URL of the library website from the user used in this part of the project.

I developed a Python script that takes the book titles from the database and creates a dictionary with the titles as the keys. It modifies the titles that match the format in which they appear in the HTML file of the library website and stores them as the subsequent values. The script then loops through the elements in the dictionary and uses Selenium to search for the book titles on the provided URL. It does so by attaching the names of the book titles, replacing spaces with '%20', with the website URL and using the WebDriver function in Selenium to navigate to the webpage. It finds the class name in the HTML script that contains book titles and compares it to the dictionary value it is looping through. If it finds a match, the function returns True and otherwise False. This return statement is used in the next step to record the result of the book availability search.

4.6 Metadata population

In this step, the result from the web scraping process is taken and used to populate the database's availability column. If the returned value of the web scraping function for an entry is True, the function updates the availability value from 'N/A to 'yes,' and if it is False, it is updated to 'no.'

After this process, the database structure looks as shown in figure 3. The orange part represents the current structure of the database. The blue part denotes things that could be added in future work, which is elaborated on in section 7.

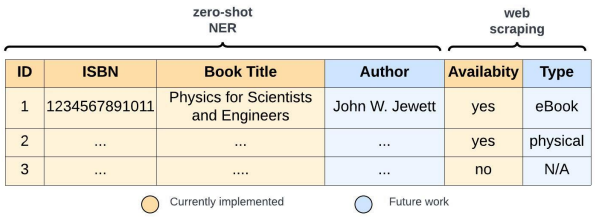


Figure 3: Database structure

4.7 Designing the UI and back-end

I used HTML and CSS files to develop the UI pages and used Flask to host the webpage and link the front end to the back end. I used Flask to write the Python app.py script, which starts the program and used HTML and CSS to design the home page, as shown in figure 4. On this page, the user uploads the syllabi from which they want to extract information in PDF format and the acronym file in text format. The user also provides the URL of the library website. When the user clicks the submit button, the Flask script takes all the PDF files and saves them in the upload/ directory in the back end. This directory is then used to extract prose and convert the PDF files into text files in another directory called extracted-files/ as described in section 4.1.

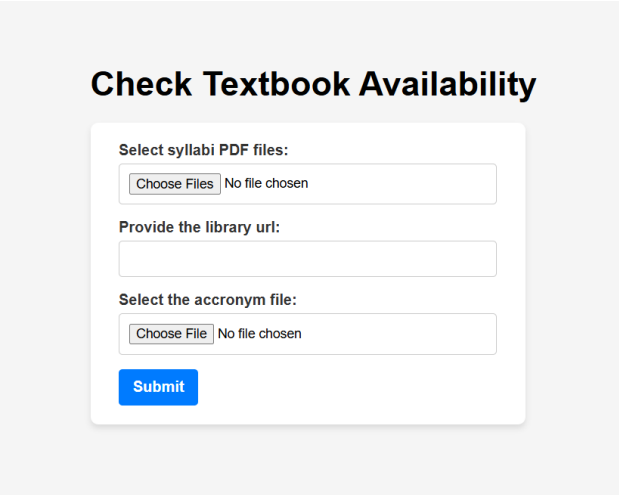


Figure 4: UI home page

Then, the script refers to all the other scripts I described in previous sections to run the functions that perform zero-shot NER, create and populate a database, and execute web scraping to populate the availability of the books. After these steps, the script runs a function that converts the SQLite database into an Excel sheet and returns it to the user on the page shown in figure 5. The user can then download the report, which looks like the sheet in figure 6.

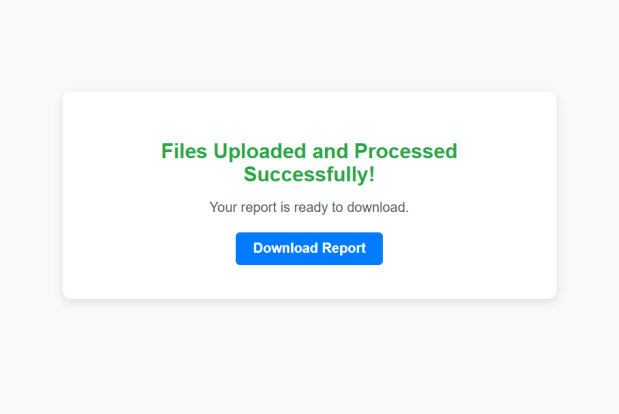
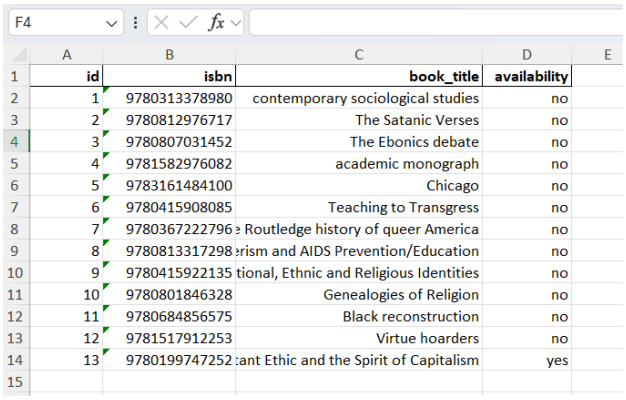


Figure 5: UI result page



	A	B	C	D	E
1	id	isbn	book_title	availability	
2	1	9780313378980	contemporary sociological studies	no	
3	2	9780812976717	The Satanic Verses	no	
4	3	9780807031452	The Ebonics debate	no	
5	4	9781582976082	academic monograph	no	
6	5	9783161484100	Chicago	no	
7	6	9780415908085	Teaching to Transgress	no	
8	7	9780367222796	Routledge history of queer America	no	
9	8	9780813317298	rism and AIDS Prevention/Education	no	
10	9	9780415922135	tional, Ethnic and Religious Identities	no	
11	10	9780801846328	Genealogies of Religion	no	
12	11	9780684856575	Black reconstruction	no	
13	12	9781517912253	Virtue hoarders	no	
14	13	9780199747252	ant Ethic and the Spirit of Capitalism	yes	
15					

Figure 6: Output excel report

Finally, the script cleans up the upload/ and extracted-files/ directories and removes the database so that the tool is ready for use again.

Since the UI design turned out to be simple, I skipped the step of using SUS method to evaluate the usability of the tool.

5 RESULTS

I analyzed the tool's functionality by providing one syllabus of a course at Earlham College and a text file containing acronyms or words common at this college. As the tool extracted information, parsed the library website, and generated the report, I simultaneously manually looked through the same syllabus, listed all the textbook requirements, and checked the library website for their availability. I compared the results of the two processes to verify the tool's reliability.

The tool correctly identified 50% of the textbooks mentioned in the syllabus. Out of the 13 books that the tool identified, 69% of them were accurate, as shown in the table below:

Category	Value	Description
Total books in syllabus	18	Books listed in the syllabus.
Books identified by the tool	13	Total books identified by the tool.
True Positives	9	Books correctly identified by the tool.
False Positives	4	Books identified by the tool but not in the syllabus.
False Negatives	9	Books in the syllabus but missed by the tool.

Table 1: Performance of the tool.

On the other hand, the web scraping part of the tool generated 100% accurate results. The tool correctly identified the availability of all the books in the syllabus on the library website.

Although the above analysis gives some representation of how well the tool functions, testing the tool on more syllabi and manually verifying the results would accurately illustrate the tool's reliability.

6 CONCLUSION

This project aimed to develop a tool for automating the extraction and tracking of textbook information from course syllabi at Earlham College. By leveraging NLP techniques, integrating zero-shot NER, and utilizing web scraping, the tool successfully parsed syllabi, extracted relevant textbook data, checked the availability of these textbooks in the library's catalog, and produced a human-readable report. Implementing an SQLite database allowed for efficient storage and querying of extracted data, while the web scraping module enabled real-time verification of book availability. By improving the accuracy of the data extraction component of the tool and making it adaptable for other institutions, this tool can aid many educational institutions in keeping track of reading requirements and their availability. The institutions can use this information to make the textbooks they have more available to the students and work towards buying or reserving the unavailable textbooks to reduce the students' financial liability.

7 FUTURE WORK

There are several areas where the tool can be improved to enhance its functionality and adaptability. As highlighted in the results section, the tool accurately identifies only 50% of textbooks listed in the syllabus and identifies 31% false positive books. Additionally, the syllabi data extraction process currently focuses on identifying textbook names but cannot associate them with specific authors or editions. This limitation may lead to inaccurate or incomplete results. Future improvements could incorporate NLP techniques to improve the accuracy of book identification and to extract complete in-publication information.

The web scraping component currently verifies the availability of books but does not differentiate between eBooks and physical copies. Adding functionality to distinguish between these formats will provide more useful availability information for users. This component of the tool is also currently designed specifically for the Earlham College library website. To broaden the tool's applicability, future iterations should include the ability to adapt to various library websites by supporting dynamic web scraping configurations.

By addressing these areas, the tool can become more reliable and adaptable, ultimately providing a more comprehensive solution for tracking and managing course reading materials across diverse educational institutions.

8 ACKNOWLEDGEMENT

I would like to thank Charlie Peck for his guidance and assistance in building this tool, and the librarians Amy Bryant and Karla Fribley for their cooperation.

REFERENCES

[CC06] Marc Caubet and Mònica Cifuentes. *Extracting metadata from textual documents and utilizing metadata for adding textual documents to an ontology.* 2006.

[Gle+14] Daniel Glez-Peña et al. "Web Scraping Technologies in an API World". In: *Briefings in Bioinformatics* 15.5 (2014), pp. 788–797.

[HR18] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[Ind22] Dwi Rosa Indah. "Implementation of User-Centered Design Method in Planning User Interface Application at Library Faculty of Computer Science Sriwijaya University". In: *INFOKUM* 10.5 (2022), pp. 263–273.

[Khd21] Moaiad Ahmad Khder. "Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application". In: *International Journal of Advances in Soft Computing & Its Applications* 13.3 (2021).

[Li+24] Diya Li et al. "Automated Clinical Data Extraction with Knowledge Conditioned LLMs". In: *arXiv preprint arXiv:2406.18027* (2024).

[LR23] Cindy Sandra Lumingkewas and Agus Rofi'i. "The Implementation of User Centered Design Method in Developing UI/UX". In: *Journal of Information System, Technology and Engineering* 1.2 (2023), pp. 26–31.

[MWN09] Luis Marenco, Rixin Wang, and Prakash Nadkarni. "Automated database mediation using ontological metadata mappings". In: *Journal of the American Medical Informatics Association* 16.5 (2009), pp. 723–737.

[Nav+23] Humza Naveed et al. "A comprehensive overview of large language models". In: *arXiv preprint arXiv:2307.06435* (2023).

[NS07] David Nadeau and Satoshi Sekine. "A survey of named entity recognition and classification". In: *Lingvisticae Investigationes* 30.1 (2007), pp. 3–26.

[NSS17] Mandy Neumann, Jan Steinberg, and Philipp Schaer. "Web-Scraping for nonprogrammers: Introducing OXPath for digital library metadata harvesting". In: *Code4Lib Journal* 38 (2017).

[PG24] Rajvardhan Patil and Venkat Gudivada. "A review of current trends, techniques, and challenges in large language models (llms)". In: *Applied Sciences* 14.5 (2024), p. 2074.

[WB98] Peter C Weinstein and William P Birmingham. "Creating ontological metadata for digital library content and services". In: *International Journal on Digital Libraries* 2 (1998), pp. 20–37.

[Wei98] Peter C Weinstein. "Ontology-based metadata: transforming the MARC legacy". In: *Proceedings of the Third ACM Conference on Digital libraries*. 1998, pp. 254–263.

[Xu+21] Runxin Xu et al. "Raise a child in large language model: Towards effective and generalizable fine-tuning". In: *arXiv preprint arXiv:2109.05687* (2021).

[Zar+23] Urchade Zaratiana et al. "Gliner: Generalist model for named entity recognition using bidirectional transformer". In: *arXiv preprint arXiv:2311.08526* (2023).