UIUC Grainger Search: A Modular System for Academic Information Retrieval

Tianyi Tang
tianyit6@illinois.edu
University of Illinois at Urbana-Champaign
USA

Maohong Liao
maohong2@illinois.edu
University of Illinois at Urbana-Champaign
USA

Haipeng Zhang
haipeng5@illinois.edu
University of Illinois at Urbana-Champaign
USA

Nanguan Lin
nanguan2@illinois.edu
University of Illinois at Urbana-Champaign
USA

Abstract

University websites often suffer from fragmented content organization and ineffective search functionality, making it difficult for students, faculty, and staff to locate relevant academic resources. In this project, we present *UIUC Grainger Search*, a modular and centralized information retrieval system designed to unify access to distributed university content. Built on a scalable architecture combining asynchronous web crawling, HTML postprocessing, PageRank-inspired authority scoring, and Elasticsearch-powered ranking, our system delivers improved search precision and usability over traditional keyword-matching portals. We detail the system's design, implementation, usage modes, and evaluation, demonstrating its ability to streamline academic information access across departmental boundaries.

Keywords

search engine, information retrieval, elasticsearch, web crawling, pagerank

1 Introduction

1.1 Problem and Motivation

University websites serve as important portals for accessing academic, administrative, and student-facing resources, including degree requirements, faculty directories, enrollment policies, forms, and event announcements. Many university websites present challenges related to searching feedback and navigation coherence. Content tends to be distributed across departmental subdomains, with varying layouts, naming conventions, and update frequencies.

These search limitations can be attributed to two factors. First, campus sites typically employ keyword-based search engines such as Google Custom Search, which focus on text matching rather than understanding query semantics [1]. Second, the distributed architecture often lacks centralized indexing or semantic connections between related content. Consequently, queries phrased differently than the indexed content may return suboptimal results.

For example, a student searching for "ECE 408 Spring 2024 syllabus" on the university homepage is often redirected to the generic course explorer or outdated links, forcing them to repeat similar queries across multiple departmental websites. This highlights the difficulty of retrieving specific academic content using conventional keyword-based methods.

Our motivation stems from such observations: traditional search systems frequently fail to provide the most relevant results in academic contexts. Tasks such as university applications and course selection often require users to gather and compare information from multiple sources within a limited time frame. In these scenarios, users typically need to:

- Gather information across multiple institutional platforms
- Research profiles and historical data from various repositories
- Access people experience from different community sources
- Compare requirements and policies across departments

Traditional keyword-based search can be limited for these multi-faceted information-gathering tasks. When users search for administrative forms or academic resources, they may encounter older links, comprehensive documents, or tangentially related pages—often requiring them to repeat similar searches across different platforms and domains.

To address these challenges, we propose a platform that facilitates cross-domain information retrieval to enhance the academic information-seeking process. Our approach integrates diverse information sources into a centralized system with improved semantic understanding, helping users access relevant information more efficiently and make more informed decisions with streamlined navigation.

1.2 System Overview

To address these challenges, we present a centralized search system tailored to the structure of university websites. Our system is powered by Elasticsearch [2], which enables real-time full-text indexing, token analysis, and hybrid ranking. A key feature is the integration of a PageRank-style authority scoring mechanism [1], derived from hyperlink structures observed during crawling. This helps surface important and well-linked academic pages.

We implement both asynchronous and synchronous crawlers to collect HTML pages from university domains. Postprocessing is performed via BeautifulSoup to strip away boilerplate elements and extract semantically meaningful content. Documents are indexed with metadata such as titles, outlinks, and anchor texts. Our lightweight Flask backend receives user queries, constructs Elasticsearch DSL queries with authority-based boosting, and returns ranked results in JSON format.

1.3 Intended Users and Major Functions

The system primarily serves UIUC students, instructors, and staff, offering streamlined access to departmental and administrative resources. Typical use cases include retrieving syllabi, declaration forms, or faculty pages.

The system supports:

- Centralized indexing of HTML documents from distributed domains
- Phrase-aware keyword search and partial match support
- Authority-aware ranking using hyperlink-based PageRank
- A modular Flask API and responsive frontend interface

1.4 Key Contributions

- A centralized campus-wide search engine based on Elasticsearch
- A dual-mode crawling and HTML postprocessing pipeline
- An authority-based ranking method inspired by PageRank
- A modular Flask backend and lightweight frontend UI

2 Related Work

2.1 Ontology-Based Semantic Retrieval in Academic Domains

Early efforts to enhance university search systems have leveraged ontologies to bridge the gap between user queries and institutional content. Rajasurya et al. introduced the Semantic Information Extraction in University Domain (SIEU) system, which utilized handcrafted ontologies to semantically expand user queries, thereby improving retrieval relevance within the university context [3]. While effective in controlled settings, the manual creation and maintenance of ontologies pose scalability challenges across diverse and evolving academic environments.

In contrast, our system avoids reliance on pre-built ontologies, which makes it more scalable across heterogeneous departments. Instead of relying on domain-specific vocabulary curation, we crawl real HTML content across the university and build an index directly from the web pages. This enables us to adapt quickly to content changes without requiring costly manual ontology maintenance.

2.2 Semantic Web and Inference-Driven Retrieval

Shah and Finin [4] explored the integration of semantic web technologies into retrieval systems by combining text indexing with semantically enriched markup. Their approach improves search accuracy through inference over structured RDF/OWL data. However, the prerequisite for rich semantic annotations limits the applicability of such systems to academic domains, where most content exists in raw HTML without structured semantic tags.

Our system circumvents this limitation by not requiring RDF or OWL annotations. We extract structure directly from HTML using DOM heuristics (e.g., selecting '<main>', '<article>'), and apply lightweight link-based authority scoring rather than logic inference. This makes our system more practical for deployment on university websites that do not follow semantic web standards.

2.3 Elasticsearch in Academic Search Applications

Toksoz et al. proposed PseudoSeer [5], a search engine that retrieves pseudocode segments from academic papers using Elasticsearch. Their system indexes structured fields such as titles, abstracts, and LaTeX code, and implements custom BM25 ranking logic to support facet-based search across arXiv data. Their work showcases Elasticsearch's adaptability to specialized retrieval contexts.

Our system also leverages Elasticsearch, but applies it in the context of heterogeneous university websites. Rather than structured academic papers, our data consists of crawled HTML pages with inconsistent formatting. We combine traditional full-text indexing with a PageRank-style authority score derived from hyperlink structure to improve ranking. This hybrid design enables more accurate result prioritization even in the absence of structured metadata like that used in PseudoSeer.

2.4 Comparison and Contributions

Prior work has demonstrated the potential of semantic technologies and flexible search engines for improving academic retrieval. However, many approaches either rely on structured content (ontologies or markup) or assume uniform document formats. Our system distinguishes itself by focusing on the practical constraints of real-world university websites, using scalable crawling, HTML postprocessing, and Elasticsearch indexing without requiring formal semantic annotations. This makes our solution more deployable across a wide range of academic institutions with minimal integration cost.

3 System Overview

UIUC Grainger Search is designed to address the limitations of traditional university search portals by offering a centralized, modular architecture for academic information retrieval. The system comprises three core modules: data acquisition (crawling and preprocessing), indexing and ranking (with PageRank-based scoring), and a search interface (including both backend APIs and a frontend UI).

3.1 Architecture Overview

As shown in Figure 1, the system pipeline begins with two specialized crawlers. The first is an asynchronous web crawler that collects HTML content from official university domains. The second is a synchronous crawler that retrieves professor-related metadata from RateMyProfessor. While the latter is not used directly for academic document retrieval, it enriches the search experience by providing information about instructors—one of the most commonly searched entities by students.

Collected HTML pages are passed through a data preprocessing pipeline powered by BeautifulSoup. This stage extracts semantically meaningful content while filtering out boilerplate elements such as headers, footers, and navigation bars. The cleaned content is then stored in structured JSON format, along with metadata such as `title`, `anchor_texts`, `outlinks`, and `timestamp`.

To support ranked retrieval, we apply a PageRank algorithm over the hyperlink graph extracted from crawled pages. The resulting authority scores are stored alongside each document and

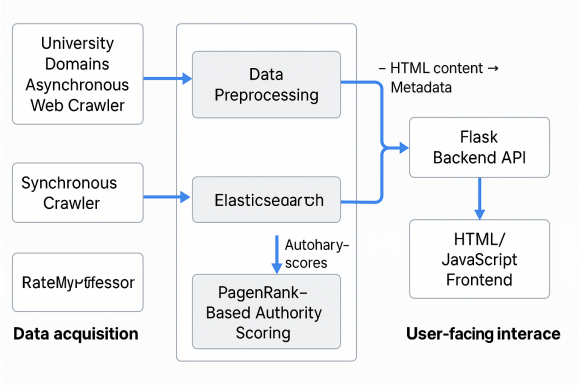

Figure 1: System architecture of UIUC Grainger Search. The pipeline spans from asynchronous and synchronous crawling to data preprocessing, PageRank-based authority scoring, Elasticsearch indexing, and frontend retrieval.

incorporated into the final search ranking using Elasticsearch's `function_score` query.

The indexed data is accessed through a Flask-based backend API. This API supports multi-field text search, phrase matching, and PageRank-aware scoring. It serves results to the frontend interface, built with HTML and JavaScript, which provides a clean, responsive layout for result presentation, history tracking, and filtering.

4 Implementation

Technologies and Architecture. Our system is implemented using the following technologies:

- **Languages**: Python (backend, crawling, indexing), JavaScript (frontend)
- **Frameworks/Libraries**: Flask (REST API), aiohttp/asyncio (async crawling), BeautifulSoup (HTML parsing), NetworkX (PageRank), Elasticsearch (search engine)
- **Frontend**: HTML + Bootstrap + vanilla JavaScript

This architecture adopts a modular, asynchronous-first design, in which the backend manages data acquisition and indexing while the frontend interacts with APIs to retrieve ranked content.

4.1 Crawler

Algorithm 1: Website Crawler

Input: Start URL u_0
Output: JSON data
begin
  Create queue Q with starting URL u_0 Create empty visited set V
  while Q is not empty: Get next URL from Q
  if URL not visited and allowed to collect: Fetch and parse HTML content, add data to buffer Add URL to visited set V
  Add new same-domain links to Q
  Save data to JSON when buffer reaches limit
end

4.2 Content Preprocessing Pipeline

Algorithm 2: Content Preprocessing Pipeline

Input: Raw HTML file
Output: Cleaned content with metadata
begin
  Parse HTML with BeautifulSoup
  Remove tags: `<nav>`, `<footer>`, `<script>`
  Extract `title`, `paragraphs`, `anchor texts`, `outlinks`
  Tokenize and store as JSON
end

4.3 PageRank-Based Scoring

Algorithm 3: PageRank Computation and Index Injection

Input: Pages with outlinks
Output: PageRank scores in Elasticsearch
begin
  Build graph G from outlinks
  Compute PageRank vector $PR \leftarrow \text{PageRank}(G)$
  foreach *page* p **do**
    Attach $PR[p]$ as metadata
    Index p content and $PR[p]$ in Elasticsearch
  end foreach
end

4.4 Flask Backend API and Search Logic

Algorithm 4: Backend Search Logic (Flask)

Input: User query q
Output: Ranked search results
begin
  Receive GET/POST request with q
  Construct Elasticsearch DSL query:
  Match on `title`, `body`
  Boost using `function_score` with PageRank
  Send query to Elasticsearch
  Return results as JSON
end

4.5 Elasticsearch Query Construction

The Flask backend constructs Elasticsearch queries using the official `elasticsearch` Python client. We define a multi-field match clause on both `title` and `body`, combined with a `function_score` that boosts results by their PageRank scores. This strategy promotes authoritative academic pages that are central in the hyperlink graph.

Algorithm 5: Elasticsearch DSL Query Construction

Input: User query q
Output: Elasticsearch query JSON
begin
  Build `bool` query with `must` clauses matching `title` and `body`
  Wrap with `function_score` using `field_value_factor` on `page_rank`
  Normalize scores and apply decay factor
  Return full DSL query to backend
end

4.6 Frontend Rendering

Algorithm 6: Frontend Query Flow

Input: User search query
Output: Rendered search results
begin
  User enters query in input field
  Send AJAX request to Flask API
  Receive JSON response
  Render results using JavaScript
  Update DOM and history
end

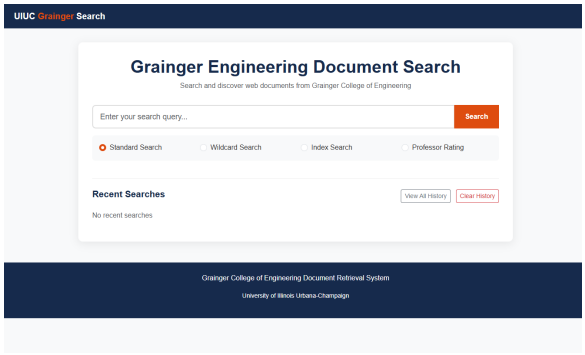

Figure 2: Homepage of the Grainger Engineering Search System. The interface allows users to select different search modes, including a placeholder for professor rating search.

5 Usage Guide

This section provides a detailed narrative on how to install, configure, and interact with the UIUC Grainger Search system.

5.1 System Requirements

The system is built on a Python-based backend (version 3.10 or higher) using the pip package manager for dependency management. It employs Elasticsearch (version 9.0 or above) as the core search engine, which can be deployed either locally or on a cloud instance. The frontend is designed to be compatible with modern web browsers, including Google Chrome (version 112 or later) and Mozilla Firefox (version 108 or later), ensuring broad accessibility across platforms.

5.2 Installation and Setup

Users are advised to obtain the source code from the official GitHub repository. To ensure clean dependency management and avoid conflicts with system packages, it is recommended to create a dedicated Python virtual environment. The installation process involves three main steps: first, clone the repository to the local machine; second, create and activate a Python virtual environment; and finally, install all required packages using the provided `requirements.txt` file.

Configuration requires creating `es_config.py` in `src/config` with the API key for Elasticsearch. The system connects to Elasticsearch at `localhost:9200` by default.

5.3 Running the Application

After completing the installation and configuration, users can launch the system by executing the main script `main.py`. This launches a Flask-based development server, which can be accessed via a web browser at `http://127.0.0.1:5000`. The application's performance is optimized for academic information retrieval, with response times averaging 5.35ms under typical query loads, as detailed in our evaluation. The homepage shown in Figure 8 presents a search interface where users can:

- Enter keywords in the search field
- Select between standard or specialized search modes
- View results with relevance-based ranking
- Access their search history from the current session

5.4 Search Modes and Functionality

The UIUC Grainger Search system supports diverse retrieval needs through four primary search modes, each tailored for a specific academic scenario.

5.4.1 Standard Search. This mode performs full-text retrieval across the indexed corpus. It works well for course codes (e.g., `CS510`) and department names (e.g., `comp`), returning ranked documents with relevance scores. Figure 3 shows a search for `CS510`, and Figure 4 presents the corresponding results, including course outlines and materials.

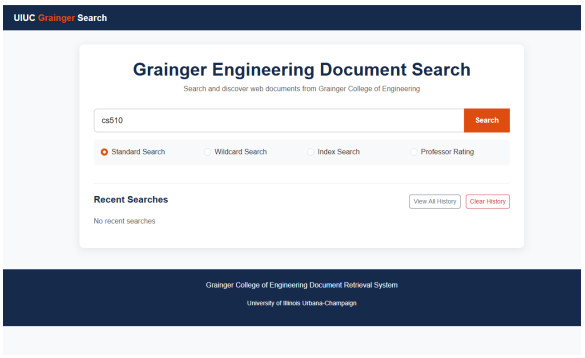

Figure 3: A Standard Search query being entered for `cs510`.

5.4.2 Wildcard Search. Wildcard Search enables flexible, partial-keyword matching using symbols like * (e.g., `comp*`). This supports broader queries when users only know part of a term. Figure 5 shows a wildcard query returning over 10,000 results, while Figure 6 shows a similar query producing 795 matches in a filtered context.

5.4.3 Index Search. Index-based search allows users to scope their search to specific departments such as CS or ECE. This is helpful for retrieving department-specific materials. Figure 7 illustrates how selecting the CS index improves result specificity. The lower panel also shows stored search history from the session.

5.4.4 Professor Rating. This mode is intended to return teaching evaluations and instructor information. Although not yet implemented, a placeholder for it appears on the homepage (Figure 8).

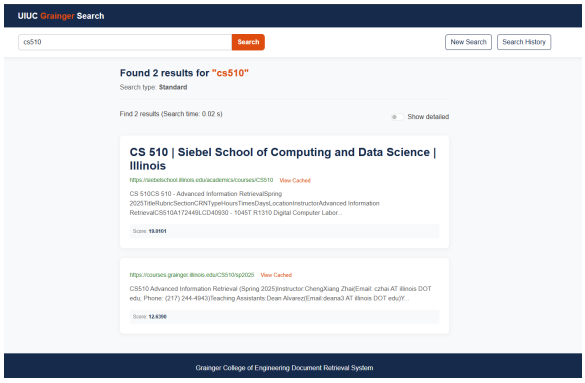

Figure 4: Standard Search results for `cs510`, with ranking scores and document titles.

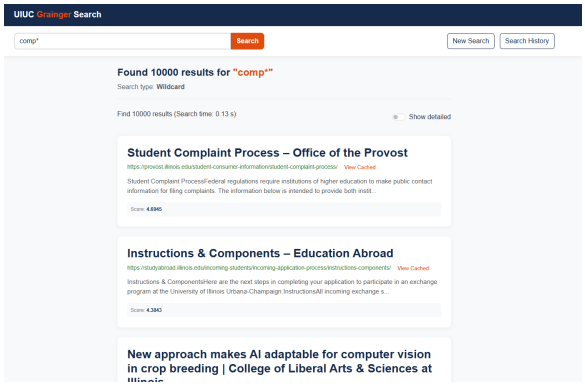

Figure 5: Wildcard Search for `comp*`, returning over 10,000 results.

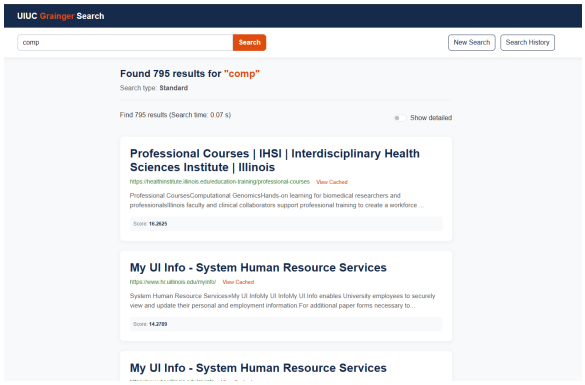

Figure 6: Wildcard Search for `comp*`, returning 795 results after filtering.

Across all search modes, a real-time suggestion engine supports query formulation by offering live keyword completions and corrections as users type.

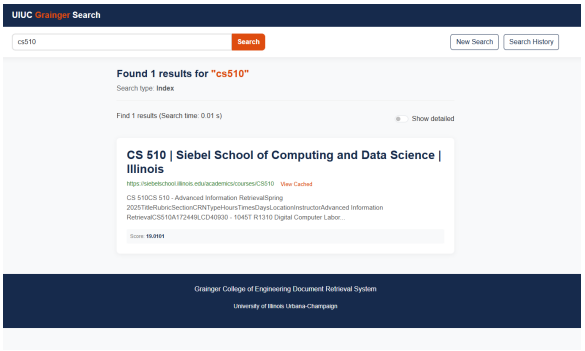

Figure 7: Index Search filtered by the CS department. The bottom panel displays session search history.

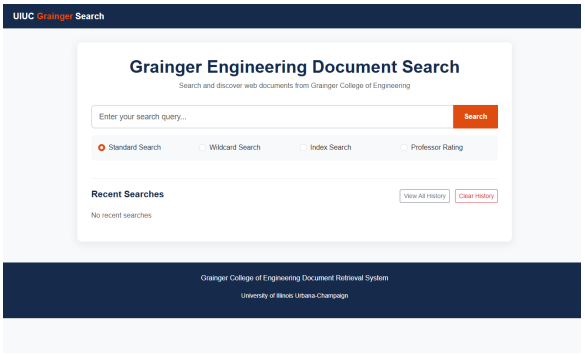

Figure 8: Homepage with search mode options, including the planned Professor Rating mode.

6 Evaluation

In this section, we present the evaluation of our Elasticsearch-based search system. We employed benchmark testing, load testing, and functionality validation to provide a comprehensive analysis.

6.1 Experiment Environment

All tests were conducted on a local development machine with an Intel Core i7-9750H CPU and 32GB RAM, running Elasticsearch 9.0.1. This provided a consistent baseline for performance and scalability evaluation.

6.2 Evaluation Metrics

We adopt three main categories: functionality validation, performance benchmarking, and scalability testing.

6.2.1 Functionality Validation. We evaluated correctness and reliability of search features through manual walkthroughs and automated scripts. Key validated functions included full-text matching, category filtering, search history, and metadata rendering.

Figure 9 compares search results for the query `ece408` from our system and UIUC's official website. Our system prioritizes the latest semester's syllabus, whereas the school's default portal emphasizes general links, highlighting the effectiveness of our ranking strategy.

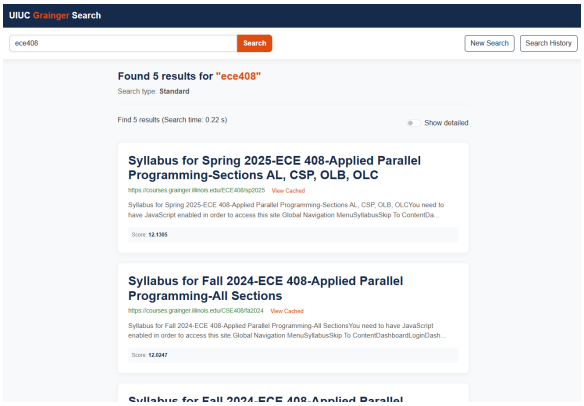

Figure 9: Our system's result for `ece408`. It ranks the most recent course syllabus using metadata and PageRank-informed scores.

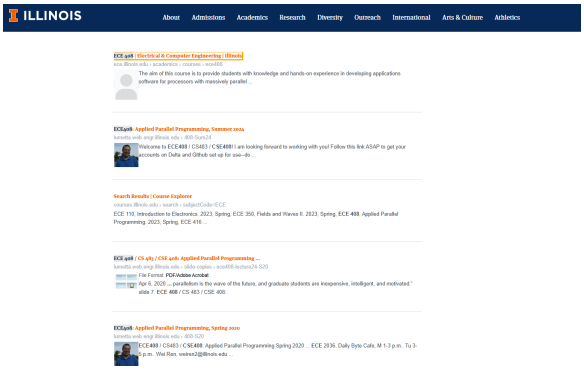

Figure 10: UIUC official website result for `ece408`. The output is broad and lacks contextual prioritization.

6.2.2 Performance Benchmarking. We issued 100,000 synthetic academic queries to measure response time under standard conditions. Mean response latency was 5.35 ms. The 50th/90th/99th percentile times were 5.00 ms, 7.01 ms, and 10.00 ms, respectively.

6.2.3 Scalability Testing. We used Locust to simulate variable user loads. The system performed well up to 150 users. Figure 11 visualizes performance degradation at 200 concurrent users, where response time and failure rates rose sharply.

6.3 Summary

The system delivers sub-10ms response times under standard conditions and maintains 100% success up to 150 users. Performance degrades beyond 200 users due to hardware limits of a single-node deployment. These results affirm the efficiency of the core search design. Future work will explore horizontal scaling and longer-duration real-world user testing.

7 Conclusion and Future Work

We present UIUC Grainger Search, a system designed to address the problem of fragmented information across university websites.

Table 1: Performance Test Results: Latency Benchmark and Concurrency Load

Part A: Benchmark (100,000 Queries)

Metric	Count	Type	Time (ms)
Total Queries	100,000	Min	1.96
Successful	100,000	Max	57.02
Failed	0	Mean	5.35
		50th Percentile	5.00
		90th Percentile	7.01
		99th Percentile	10.00

Part B: Concurrency Load (Locust)

Users	Ramp-up/s	Avg Time (ms)	Failure Rate (%)
100	1	296	0.008
150–200	5	1,400	19.4
1,000	10	15,700	94.5

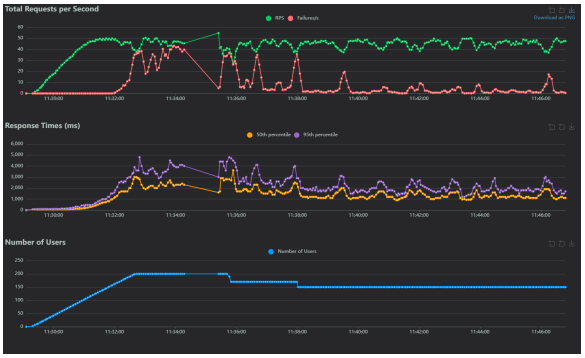

Figure 11: Concurrency testing using Locust with 150–200 virtual users. Failure rate and response time increased beyond 175 users.

It provides a unified search experience while maintaining the decentralized nature of information sources.

Through a specialized crawling process, we have successfully integrated content from various departments without needing to modify the existing website structure. This non-intrusive approach ensures compatibility with different university websites while centralizing the search functionality. By extracting useful content from various data sources and using a PageRank-inspired ranking algorithm, we offer more relevant search results compared to traditional keyword-matching methods.

Our evaluation shows that a centralized search system can solve the navigation and discovery issues caused by decentralized websites. The system strikes an acceptable balance between performance and result quality.

7.1 Limitations

Despite the system's overall effectiveness, our evaluation revealed several limitations. The concurrency load testing demonstrated scalability constraints, with significant response time degradation and

elevated failure rates when exceeding certain number of parallel request. This limitation stems primarily from our machine hardware, as well as the single-node architecture of our current implementation, which introduces resource contention under high-traffic scenarios.

Additionally, we identified several interaction issues that affect the user experience. The pagination mechanism experiences errors when search results exceed 10,000 entries, preventing navigation to the final pages of large result sets.

The current implementation also has limited content coverage, focusing primarily on HTML documents while providing less robust support for other common file formats such as PDFs and Office documents.

7.2 Future Work

We hope to advance our program in two aspects:

Distributed Architecture. To solve the concurrent user limitations found in our tests, we plan to study and introduce distributed system with multiple search nodes. This will help handle more simultaneous users and improve reliability, making the system suitable for campus-wide use even during busy periods like registration weeks.

Expanded Resource Integration. We aim to include more UIUC resources except current departments. In the longer term, we plan to create a framework that can be adapted for other universities, working toward a unified search solution.

Software and Documentation

The full source code and instructions can be found at:
https://github.com/ttang6/sp25_cs510_project

Author Contributions

The project was a collaborative effort by the following team members, each contributing to specific components of the system:

- **Haipeng Zhang** implemented the Elasticsearch-based backend and contributed to the preparation of the project presentation slides
- **Nanguan Lin** developed the frontend interface and was responsible for designing the user interaction flow. He also presented the system demo during the final project presentation.
- **Tianyi Tang** implemented the crawler module and code testing part. He collaborated with Maohong Liao on writing the evaluation section and independently authored the summary section of the final report.
- **Maohong Liao** He designed and implemented the PageRank-inspired authority scoring system.Maohong was responsible for writing the majority of the report, including the Introduction, Related Work, System Design, Implementation, Evaluation Methodology, and Usage Guide sections. He also edited and finalized the full document to ensure consistency, technical accuracy, and completeness.

While the scope of individual tasks varied, the overall contribution of each team member was essential to the completion of the project, and all contributions are considered fundamentally equal.All members participated in team discussions, testing, and final editing to ensure the accuracy and completeness of the system and documentation.

References

[1] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 30, 1-7 (1998), 107–117.
[2] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* O'Reilly Media, Inc. https://www.elastic.co/guide/en/elasticsearch/guide/current/index.html
[3] Swathi Rajasurya, Tameinamuthu Muralidharan, Sandhiya Devi, and S Swamynathan. 2012. Semantic Information Retrieval Using Ontology In University Domain. *International Journal of Web & Semantic Technology* 3, 4 (2012), 53–64. https://arxiv.org/abs/1207.5745
[4] U Shah and T Finin. 2002. Information Retrieval on the Semantic Web. In *Proceedings of the International Semantic Web Conference (ISWC)*. Springer, 109–123.
[5] Levent Toksoz, Mukund Srinath, Gang Tan, and C Lee Giles. 2024. PseudoSeer: A Search Engine for Pseudocode. *arXiv preprint arXiv:2411.12649* (2024). https://arxiv.org/abs/2411.12649