PaperWeb: Visual Representation Tool for ArXiv Papers

Akul Datta
akuld2@illinois.edu
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Mutma Adebayo
adebayo5@illinois.edu
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Nisha Prasad
nishap4@illinois.edu
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Abstract

PaperWeb is a standalone software tool/web-based application for visual representation of research papers on the ArXiv website, maintained locally on a user's device. Users can look up a paper and find its abstract, metadata, and all of its cited sources in a graph visualization. Unlike existing tools that often become outdated, PaperWeb uses API calls to ArXiv to regularly update its database, ensuring up-to-date information. This tool aims to expedite the research process by enabling researchers to efficiently find all the information and relevant papers needed for their work.

Keywords

Research Papers, ArXiv, Citation Network, Graph Visualization, Paper Discovery

ACM Reference Format:
Akul Datta, Mutma Adebayo, and Nisha Prasad. 2023. PaperWeb: Visual Representation Tool for ArXiv Papers. In *Proceedings of PaperWeb: Visual Representation Tool for ArXiv Papers (CS510).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Academic research often requires finding related papers, tracking citations, and understanding how different research connects. PaperWeb addresses this need by providing a visual representation tool for research papers on ArXiv. With the rapid pace of academic publishing, researchers face challenges in keeping up with new developments and finding relevant papers. Existing tools for visualizing research paper connections are often outdated and do not update their databases frequently enough.

PaperWeb solves this problem by utilizing API calls to ArXiv to regularly update its data, ensuring the tool always provides up-to-date information. This allows researchers to efficiently explore citation networks, discover related papers, and gain insights into their fields of interest.

2 Description

PaperWeb is a standalone software tool/web-based application for visual representation of research papers on the ArXiv website, and it is kept locally on a user's device. Users can look up a paper and find the paper's abstract and other data, along with all of its cited sources in a graph.

We need this tool because tools like it that already exist are often outdated and do not update their databases frequently enough to keep up with the rate at which papers are published. We use API calls on ArXiv to regularly update our data and ensure the tool is always able to provide up-to-date information. This tool can help to change the world by enabling researchers who do important work to find all the information and relevant papers needed to make further contributions to society in their work. It is important to address this need because expediting the research process will help the efficiency of research that could drive society forward.

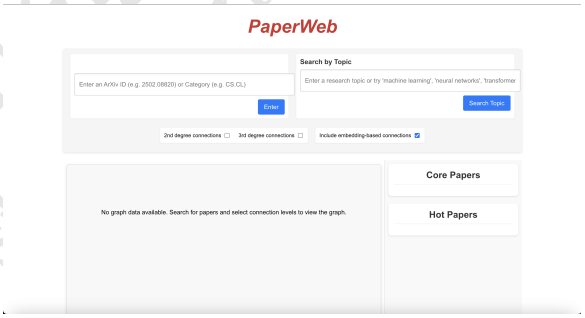


Figure 1: PaperWeb Main Interface - The landing page where users can search for papers by ID, title, or category

3 Users

The intended users of our tool will be researchers and professionals looking to catch up on new developments in their various fields, and potentially also students looking for papers relevant to their projects and assignments. The interface is designed to be intuitive for academics, with specialized search features including:

- Direct arXiv ID search for locating specific papers
- Title-based search for discovering papers by topic
- Category search (e.g., CS.AI, CS.CL) for browsing papers in specific fields
- Topic-based semantic search using vector embeddings

The tool accommodates both precise queries when researchers know exactly what they're looking for and exploratory searches when they need to discover papers in a particular field or topic area.

4 Major Functions

The major function of our software is to visualize the citation network of a searched paper up to the 3rd degree of connections. Additionally, a user will also find papers that are most similar to

CS510, University of Illinois at Urbana-Champaign

ACM ISBN 978-1-4503-XXXX-X/2023/06
https://doi.org/XXXXXXX.XXXXXXX

the searched papers (hot papers), and then papers that are big in the field they are based in (core papers).

The three key components of PaperWeb's functionality are:

(1) **Citation Network Visualization**: Shows papers are cited by the searched paper, visualized as an interactive graph with color-coded nodes representing different degrees of connection.
(2) **Hot Papers**: Identifies papers that are semantically similar to the searched paper based on their content, using text embeddings.
(3) **Core Papers**: Highlights influential papers in the same field or category, helping researchers identify fundamental work in that area.

Users can control the visualization by toggling which degrees of connections to display, allowing them to manage the complexity of the graph according to their needs.

5 Implementation Details

We first created our project on a small sample of papers, which was approximately 10,000 in size. We now use a database of over 700,000 papers. We made two SQL databases, one that stores metadata about ArXiv papers and their connections to papers that they cite, and another that contains embedding information that helps us determine paper similarity.

5.1 System Architecture

PaperWeb follows a modern web application architecture with the following components:

- **Frontend**: Built with React.js, handling the user interface and data visualization
- **Backend API**: Implemented in Python with Flask, providing endpoints for search, paper details, and citation connections
- **Databases**: Two SQLite databases:
  - `papers.db` - Stores paper metadata and citation relationships
  - `embeddings.db` - Contains vector embeddings of paper abstracts for semantic search
- **Data Collection**: Scripts to fetch and process papers from arXiv's API
- **Visualization**: D3.js library for rendering the interactive citation network graph

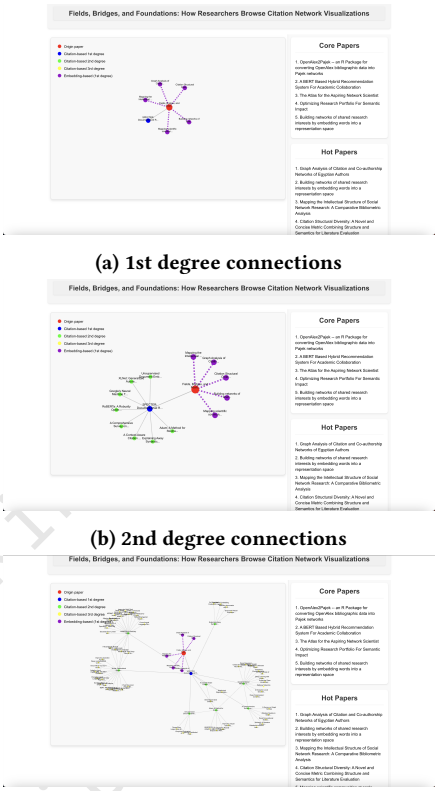
We are storing the data on Google Drive with a SQLite file. For the embeddings, we used an existing embedding model (GIST-Embedding-v0) and ran it locally using Python. For our web application, we used React and Flask. All of our backend code is written in Python. For displaying the graphs, we used the D3 library.

The papers in the database start from 2007 because it was simply technically unreasonable to fetch the papers from before that. Thus, more recent papers will generate better graphed results.

6 Key Algorithms and Code

6.1 ArXiv Data Collection

Our data collection Python script fetches recent Computer Science (CS) papers from arXiv's API and saves their metadata to a CSV file. It processes papers in weekly chunks between a specified start date



(a) 1st degree connections

(b) 2nd degree connections

(c) 3rd degree connections

Figure 2: Citation Network Visualization showing the progressive expansion of the citation graph from 1st to 3rd degree connections

(either from command-line input or a last_updated.txt file) and the current date. For each paper, it extracts and formats key details including arXiv ID, title, authors, abstract, categories, and publication date. The script handles arXiv's API rate limits by sleeping between requests and properly formats both modern and legacy arXiv IDs. Results are written to arxiv_cs_recent.csv with appropriate headers, while the completion date is recorded in last_updated.txt for future runs. The script includes robust error handling and progress reporting, showing the number of processed papers and date ranges being fetched. It's designed as a maintainable tool for regularly updating a local repository of recent CS papers from arXiv.

The system uses regular expressions to identify arXiv references in paper content, with support for multiple formats:

```
ARXIV_PATTERNS = [
    # Standard arXiv formats
    r'cs/\d{7}',                          # cs/1234567
    r'\d{4}\.\d{4,5}',                    # 1234.5678
    r'arXiv:\d{4}\.\d{4,5}',              # arXiv:1234.5678
    r'\[arXiv:\d{4}\.\d{4,5}\]',          # [arXiv
     :1234.5678]

    # Computer science specific formats
    r'cs\.[A-Za-z]{2}/\d{7}',             # cs.CL/1234567
    r'cs\s[A-Za-z]{2}/\d{7}'              # cs CL/1234567
    # ... additional patterns ...
```

```
]
```

```
127.0.0.1 - - [08/May/2025 22:44:43] "GET /api/paper/2004.071
80 HTTP/1.1" 200 -
DEBUG: /api/search received query: 2405.07267
DEBUG: Searching database for: 2405.07267
DEBUG: Found paper: 2405.07267
DEBUG: /api/connections received request for 2405.07267, degr
ee 1
DEBUG: Found paper by search: 2405.07267
DEBUG: Connection structure format - first_degree: <class 'di
ct'>, second_degree: <class 'dict'>, third_degree: <class 'di
ct'>
DEBUG: Second degree keys sample: ['2004.07180']
DEBUG: Returning connections for 2405.07267
```

Figure 3: Terminal output showing the paper fetching process and database operations

6.2 Citation Network Extraction

The main(paper_id) function in get_connections.py performs a hierarchical extraction of academic references for a given paper, building a three-level deep citation graph. It begins by logging the start time in an error log file, then retrieves the first-degree references (direct citations) of the input paper using get_paper_connections(). If no references are found, it processes the paper directly using process_paper() and filters out any existing references. Next, it gathers second-degree references by extracting the citations of the first-degree references using process_connections(). The function then flattens these second-degree references and retrieves their citations to obtain third-degree references. Finally, it structures the results into a nested dictionary containing the formatted first-degree references (including the source paper's title), second-degree references, and third-degree references, providing a comprehensive three-tiered citation network for the original paper.

The citation extraction process follows these steps:

(1) **PDF Retrieval**: Downloads the paper PDF from arXiv
(2) **Text Extraction**: Uses PyMuPDF to extract the full text
(3) **Reference Parsing**: Applies regex patterns to identify arXiv references
(4) **Graph Building**: Constructs a hierarchical graph structure for visualization

Our hierarchical approach processes papers through multiple layers:

```
# First-degree references (direct citations)
first_degree_refs = get_paper_connections(paper_id)
if not first_degree_refs:
    # Process paper directly if no cached connections
    first_degree_refs = process_paper(paper_id)

# Second-degree references (papers cited by first-degree)
second_degree_refs = process_connections(
    first_degree_refs)

# Third-degree references (papers cited by second-degree)
flattened_second_degree = [ref for sublist in
    second_degree_refs.values()
                           for ref in sublist]
third_degree_refs = process_connections(
    flattened_second_degree)
```

6.3 Semantic Search

The fuzzy_search_get_all_related_papers(query_text: str) function in embed.py performs a semantic search to retrieve academic papers relevant to a given query by leveraging vector embeddings. It first converts the query text into a numerical embedding using generate_embeddings(), then connects to an SQLite database containing paper records with their respective embeddings, titles, abstracts, and metadata. For each paper in the database, the function computes the cosine similarity between the query's embedding and the paper's stored embedding to determine relevance. After compiling all papers with their similarity scores, it sorts them in descending order based on similarity and returns the ranked list, enabling a fuzzy, content-based search that goes beyond simple keyword matching to find semantically related research papers.

Our semantic search system uses the GIST-Embedding model with the following implementation:

```
def generate_embeddings(texts: List[str]) -> np.ndarray:
    """Generate embeddings for texts using GIST-Embedding
     model."""
    load_model()

    max_hf_batch_size = 16
    all_embeddings = []

    for i in range(0, len(texts), max_hf_batch_size):
        batch_texts = texts[i:i+max_hf_batch_size]

        inputs = tokenizer(batch_texts, padding=True,
                           truncation=True, max_length
    =512,
                           return_tensors="pt")

        # GPU acceleration when available
        if torch.backends.mps.is_available():
            inputs = {k: v.to("mps") for k, v in inputs.
    items()}
        elif torch.cuda.is_available():
            inputs = {k: v.to("cuda") for k, v in inputs.
    items()}

        with torch.no_grad():
            outputs = model(**inputs)
            batch_embeddings = outputs.last_hidden_state
    [:, 0, :].cpu().numpy()

        all_embeddings.append(batch_embeddings)

    return np.vstack(all_embeddings)
```

We calculate similarity between papers using cosine similarity:

```
def cosine_similarity(a: np.ndarray, b: np.ndarray) ->
    float:
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.
    norm(b))
```

6.4 API Layer

The Flask-based API provides several endpoints for the frontend to interact with the database:

- `/api/search` - Search papers by title, ID or general query
- `/api/paper/<paper_id>` - Get detailed information about a specific paper

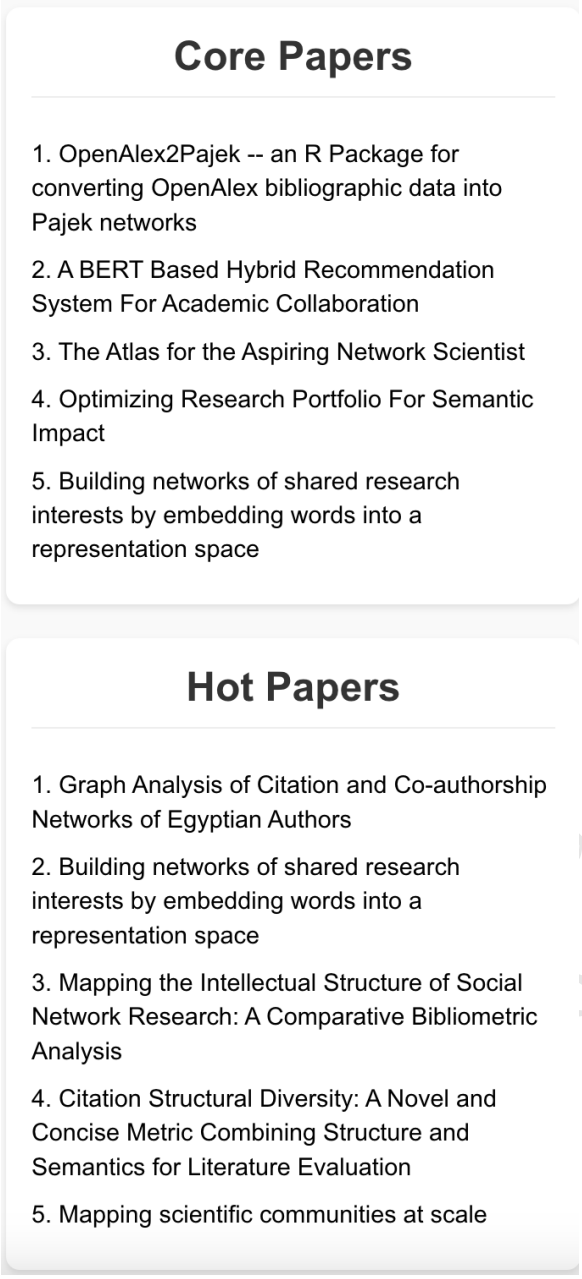


Figure 4: Display of Core and Hot papers related to the searched paper, showing the most influential and similar papers in the field

- `/api/connections/<paper_id>/<degree>` - Get citation network for a paper
- `/api/category-search` - Search papers by arXiv category (e.g., CS.AI)
- `/api/topic-search` - Semantic search for papers related to a topic
- `/api/update` - Trigger a database update to fetch new papers

The API includes robust error handling and rate limiting to prevent overloading the arXiv API:

```python
@app.route('/api/search', methods=['GET'])
def search_papers():
    query = request.args.get('q')

    if not query:
        return jsonify({"success": False,
                        "error": "No query provided"}),
     400

    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        # First try exact ID match
        cursor.execute("""
            SELECT id, title, authors, abstract,
     categories,
                   year, month, day
            FROM papers
            WHERE id = ?
        """, (query,))

        # If not found, try title search
        # ...

        # Process results
        # ...

        return jsonify({"success": True, "results":
     results})
    except Exception as e:
        return jsonify({"success": False,
                        "error": str(e)}), 500
```

6.5 Frontend Search Component

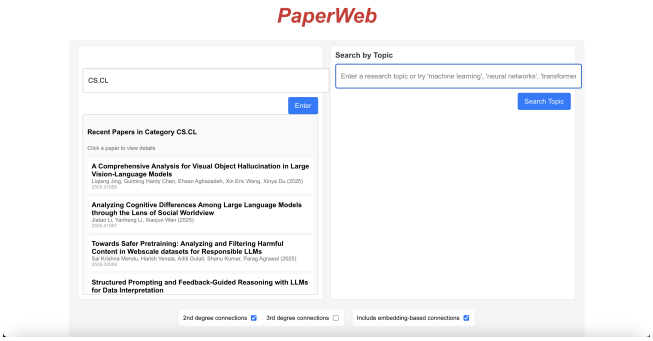
The SearchBar component in the client/src/components folder is a React functional component that provides a user interface for searching academic papers by title, arXiv ID, or category. It maintains state for the search query, loading status, and potential errors. When a user initiates a search (either by clicking the button or pressing Enter), the component first checks and formats the input as a potential arXiv ID if it matches certain patterns. It then sends a request to a backend API endpoint (/api/search) with the query. If no results are found, it attempts a direct lookup using the arXiv ID. Upon receiving results, it updates the UI by displaying the paper title and any related "hot" or "core" papers in designated sections, while also notifying other components (via a custom connectionsLoaded event) if connection data is available. The component handles errors gracefully, showing appropriate messages to the user, and includes loading states during API requests. The search bar features a clean design with an input field, a search button, and error display capabilities.

The frontend is built with React and implements several key components:

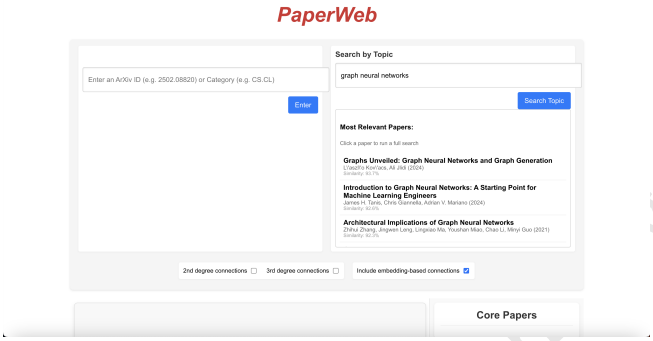
- `SearchBar` - Handles user search input and API requests
- `Graph` - Visualizes the citation network using D3.js
- `DegreeCheckbox` - Controls which levels of connections to display
- `TopicSearch` - Specialized interface for semantic search

- `SideBar` - Navigation and display of additional paper information

The search component automatically detects the input format and adjusts accordingly:



(a) Category Search dropdown



(b) Topic Search dropdown

Figure 5: Search interface dropdowns showing category and topic search options

6.6 Graph Visualization

The visualization component uses D3.js to create an interactive force-directed graph that accurately represents the citation relationships. The graph implements several advanced features:

- **Color-coding**: Different colors for different degrees of connection
  - Red for the source paper
  - Blue for 1st degree connections
  - Green for 2nd degree connections
  - Yellow for 3rd degree connections
  - Purple for semantically related papers
- **Interactive elements**: Zoom, pan, and click interactions
- **Tooltips**: Displaying paper titles on hover
- **Force simulation**: Realistic physics-based layout for the network
- **Responsive design**: Adapts to different screen sizes

The D3.js implementation creates dynamic force-directed graph layouts:

```
const simulation = d3.forceSimulation(nodes)
    .force("link", d3.forceLink(links)
        .id(d => d.id)
        .distance(l => l.isFuzzy ? 200 : 100))
    .force("charge", d3.forceManyBody()
        .strength(d => d.level === 0 ? -500 : -200))
    .force("center", d3.forceCenter(width / 2, height / 2))
    .force("collide", d3.forceCollide(d => d.val * 1.5));
```

7 Challenges and Solutions

During development, we faced several technical challenges:

(1) **PDF Text Extraction**:
- *Challenge*: Extracting clean text from PDFs with different formats
- *Solution*: Used PyMuPDF with fallback to PyPDF2, with robust error handling

(2) **Citation Pattern Recognition**:
- *Challenge*: Identifying arXiv citations in diverse formats
- *Solution*: Developed an extensive set of regex patterns covering major citation styles

(3) **Database Performance**:
- *Challenge*: Embedding database grew to over 2GB, slowing search
- *Solution*: Implemented batch processing and optimized SQLite queries

(4) **Graph Visualization Performance**:
- *Challenge*: Large citation networks (3rd degree) overwhelmed the browser
- *Solution*: Added degree toggles and optimized D3.js rendering

8 Conclusion

PaperWeb provides researchers with a powerful tool for visualizing citation networks and discovering related papers on ArXiv. By maintaining up-to-date information through regular API calls, PaperWeb addresses the limitations of existing tools and offers a more efficient research experience. The combination of direct citation visualization up to the third degree, along with the identification of hot and core papers, enables users to gain comprehensive insights into their fields of interest.

Our implementation leverages modern web technologies and natural language processing techniques to deliver a responsive and intuitive interface. The hierarchical approach to citation extraction, combined with semantic search capabilities, provides a more complete picture of the research landscape than traditional citation databases.

Future work could include expanding the database to cover papers before 2007, adding more source repositories beyond ArXiv, and implementing additional visualization features to enhance the user experience. We also plan to add collaborative features allowing researchers to share their discovered citation networks and annotations with colleagues.

Acknowledgments

We would like to thank our instructor and teaching assistants for their guidance throughout this project.