Science Management Design Journal

Journal Homepage: www.smdjournal.com

ISSN: 2583-925X
Volume: 2
Issue: 1
Pages: 48-65

Google Scholar Citation Creation Tool

Sanidhya Chimurkar[1*]

Vishwakarma University, Pune

*Corresponding Author:202001368@vupune.ac.in

Article history: Received: 25/05/2024, Revised: 29/05/2024, Accepted: 30/05/2024, Published Online: 31/05/2024

Abstract

The Google Scholar Citation Creation Tool is a Python-based application designed to streamline the process of searching for and retrieving academic citations from Google Scholar, a widely used and respected academic search engine. The tool leverages web scraping techniques to fetch and parse search results from Google Scholar. By integrating the Beautiful Soup library for HTML parsing and the requests library for sending HTTP requests, the application extracts relevant citation information, including titles, authors, and publication details.

Keywords: User Interface, Web scraping, Citation Formatting, Google Scholar

1. INTRODUCTION

In the realm of academic research, the process of generating and managing citations is crucial. Proper citation not only gives credit to the original authors but also enhances the credibility and traceability of scholarly work. However, manually formatting citations according to different styles such as APA, MLA, Chicago, Harvard, and IEEE can be a tedious and error-prone task. To address this challenge, we have developed a Google Scholar Citation Search Tool using Python, which integrates web scraping and a graphical user interface (GUI) to facilitate the automated generation of citations in multiple formats. The tool leverages the capabilities of web scraping to retrieve academic articles from Google Scholar, extracts relevant bibliographic information, and formats the citations according to the selected style. This user-friendly application is built using Tkinter for the GUI, BeautifulSoup for HTML parsing, and the requests library for handling HTTP requests [1].

By providing an easy-to-use interface, our tool allows users to input their search queries, select their desired citation style, and browse through the search results with pagination controls[5-55]. This functionality not only saves time for researchers but also ensures consistency and accuracy in citation formatting. In this report, we will delve into the design and implementation details of the Google Scholar Citation Search Tool. We will delve into the methodology for web scraping, the structure of the Tkinter-based user interface, and the citation formatting logic. Additionally, we will explore potential improvements and future enhancements to the tool, aiming to make it an indispensable resource for academic researchers and students [2].

2. Structure

The structure of the Google Scholar Citation Creation Tool is divided into several key components, each serving a distinct function to ensure the smooth operation and user-friendly interface of the application. This section provides an overview of the main components and their roles.

2.1. User Interface (UI) : The User Interface is the front-end part of the tool that interacts directly with the user. It is built using the tkinter library in Python, which provides a simple way to create graphical user interfaces.

- Main Window: The main window of the application where all the components are placed.
- Entry Fields: Input fields for the search query, like author name, publication date range, and subject area.
- Buttons: Buttons for initiating the search, navigating through search results (Next and Previous), and selecting citation styles.
- Dropdown Menu: A dropdown menu for selecting the preferred citation style (APA, MLA, Chicago, Harvard, IEEE).
- Text Widget: A text widget to display the formatted citations and pagination information [3].

2.2.Citation Formatting Functions

This component includes functions that format the extracted citation data according to different citation styles. Each function takes a dictionary containing citation details and returns a formatted string.

- APA Style: Formats the citation in APA style.
- MLA Style: Formats the citation in MLA style.
- Chicago Style: Formats the citation in Chicago style.
- Harvard Style: Formats the citation in Harvard style.
- IEEE Style Formats the citation in IEEE style.

2.3. Web Scraping Component

This component is responsible for fetching and parsing search results from Google Scholar.

-Search Function: Constructs the Google Scholar search URL based on user input and sends an HTTP GET request to retrieve search results.

- Parsing Function: Uses BeautifulSoup to parse the HTML content of the search results and extract relevant information (title, authors, publication).

2.4. Pagination Handling

To manage large sets of search results, the tool includes functionality for pagination.

- Pagination Logic Keeps track of the current page and start index to fetch the next or previous set of results.

Pagination Buttons : Buttons in the UI that allow users to navigate through different pages of search results.

3. User Interface Explanation

The user interface (UI) of the Google Scholar Citation Search Tool is designed to be user-friendly and intuitive, allowing users to easily search for and retrieve citations in various formatting styles. Here's a detailed explanation of each component of the UI:

3.1 Main Window

The main window serves as the primary container for all the interface elements. It is created using the tkinter.Tk() method and is titled "Google Scholar Citation Search."

```
python
root = tk.Tk()
root.title("Google Scholar Citation Search")
```

Search Query Entry Field

This entry field allows users to input their search query. It is a single-line text box where users can type the keywords they want to search for on Google Scholar.

```
python
entry_query = tk.Entry(root, width=50)
entry_query.grid(row=0, column=0, padx=10, pady=10)
```

- Attributes:
- width=50: Sets the width of the entry field.
- padx and pady: Adds padding around the entry field for better spacing.

ISSN: 2583-925X
Volume: 2
Issue: 1
Pages: 48-65

3.2 Search Button

The search button initiates the search operation when clicked. It triggers the search_and_display_citations function, which handles the search query and displays the results.

```
python
button_search = tk.Button(root, text="Search", command=search_and_display_citations)
button_search.grid(row=0, column=1, padx=10, pady=10)
```

- Attributes:
 - text="Search": Sets the text displayed on the button.
 - command=search_and_display_citations: Links the button click event to the search_and_display_citations function.

3.3 Citation Style Dropdown Menu

This dropdown menu allows users to select the citation style they prefer (e.g., APA, MLA, Chicago, Harvard, IEEE). The selected style determines how the search results will be formatted.

```
python
style_var = tk.StringVar(root)
style_var.set("APA")  Default citation style
dropdown_style = tk.OptionMenu(root, style_var, citation_styles.keys())
dropdown_style.grid(row=0, column=2, padx=10, pady=10)
```

- Attributes:
- StringVar: Holds the value of the selected option.
- OptionMenu: Provides a list of citation styles to choose from.
- default="APA": Sets APA as the default citation style.

3.4.Citations Display Text Widget

This multi-line text widget displays the formatted citations returned from the search. Users can view the search results in the selected citation style.

```
python
text_citations = tk.Text(root, width=80, height=20)
text_citations.grid(row=1, column=0, columnspan=3, padx=10, pady=10)
```

- Attributes:

- width=80: Sets the width of the text widget.
- height=20: Sets the height of the text widget.
- columnspan=3: Spans the text widget across three columns for a better layout.

3.5 Pagination Buttons

These buttons allow users to navigate through multiple pages of search results. The "Prev" button loads the previous set of results, and the "Next" button loads the next set of results.

```
python
button_prev = tk.Button(root, text="Prev", command=lambda: handle_pagination("prev"))
button_prev.grid(row=2, column=0, padx=10, pady=10)
button_next = tk.Button(root, text="Next", command=lambda: handle_pagination("next"))
button_next.grid(row=2, column=2, padx=10, pady=10)
```

- Attributes:
- text="Prev" and text="Next": Sets the text displayed on the buttons.
- command=lambda: handle_pagination("prev") and command=lambda: handle_pagination("next"): Links the button click events to the handle_pagination function with respective directions.

3.6 Initialization of Pagination Variables

These variables manage the pagination state, keeping track of the current start index and the number of results per page.

```
python
start_index = 0
num_results = 10
```

3.7 Main Event Loop

The main event loop keeps the application running, listening for user interactions and updating the UI accordingly.

```
python
root.mainloop()
```

4. Web Scraping Process

Web scraping is a technique used to automatically extract data from websites. In the context of this project, web scraping is utilized to gather citation information from Google Scholar, a popular academic search engine. This section outlines the web scraping process, including the tools and libraries used, the step-by-step methodology, and the considerations taken into account to ensure an ethical and robust implementation.

Tools and Libraries

The following Python libraries are used for the web scraping process:

- Requests: A library that allows sending HTTP requests to websites.
- BeautifulSoup: A library used for parsing HTML and XML documents and extracting data from them.

Methodology

1. Sending an HTTP Request

The process begins with sending an HTTP GET request to the Google Scholar search URL using the requests library. The URL includes the search query and pagination parameters to retrieve the desired results.

```
python
import requests
response = requests.get(url)
Parameters:
- url: The URL constructed to include the search query and other parameters.
```

2. Checking the Response Status

The status code of the HTTP response is checked to ensure that the request was successful (status code 200). If the request fails, an error message is printed.

```
python
if response.status_code == 200:
    Proceed with parsing
else:
   print(f"Failed to fetch search results. Status code: {response.status_code}")
```

3. Parsing the HTML Content

Upon a successful request, the HTML content of the response is parsed using BeautifulSoup. This involves creating a BeautifulSoup object to navigate and search through the HTML structure.

```
python
from bs4 import BeautifulSoup
soup = BeautifulSoup(response.content, 'html.parser')
```

4. Extracting Relevant Data

The parsed HTML content is searched for specific elements containing the citation information. In Google Scholar, search results are typically found within <div> elements with the class gs_ri. The script extracts the title, authors, and publication details from these elements.

```
python
results = soup.find_all('div', class_='gs_ri')
for result in results:
    title = result.find('h3', class_='gs_rt').text.strip() if result.find('h3', class_='gs_rt') else ""
    authors = result.find('div', class_='gs_a').text.strip() if result.find('div', class_='gs_a') else ""
    publication = result.find('div', class_='gs_pub').text.strip() if result.find('div', class_='gs_pub') else ""
    citations.append({'title': title, 'authors': authors, 'publication': publication})
```

5. Storing and Returning Data

The extracted data, including titles, authors, and publication details, are stored in a list of dictionaries. This list is then returned for further processing and formatting according to the selected citation style.

```
python
def parse_search_results(html_content):
    citations = []
    soup = BeautifulSoup(html_content, 'html.parser')
    results = soup.find_all('div', class_='gs_ri')
    for result in results:
        title = result.find('h3', class_='gs_rt').text.strip() if result.find('h3', class_='gs_rt') else
""
        authors = result.find('div', class_='gs_a').text.strip() if result.find('div', class_='gs_a')
else ""
        publication = result.find('div', class_='gs_pub').text.strip() if result.find('div',
class_='gs_pub') else ""
        citations.append({'title': title, 'authors': authors, 'publication': publication})
    return citations
```

5. Considerations and Challenges

1. Robustness:

   Web scraping scripts can be fragile and may break if the website's structure changes. To ensure robustness, the code should handle exceptions and be flexible enough to adapt to minor changes in the HTML structure.

2. Legality:

   It is crucial to ensure that scraping complies with the terms of service of the website being scraped. For Google Scholar, automated scraping might violate their terms, and using official APIs or obtaining proper permissions is recommended whenever possible.

6. Citation Formatting Explanation

The citation formatting functionality in this tool is designed to accommodate various academic styles, ensuring that users can easily generate citations in the format required by their specific discipline or publication. This section provides an explanation of how citation formatting is implemented in the tool, including the different styles supported and the process of applying these formats to the extracted data.

Supported Citation Styles

The tool supports several widely used citation styles, each with its own distinct rules for presenting authors, titles, publication details, and other relevant information [4]. The following citation styles are supported:

1. APA (American Psychological Association):
   - Format: Authors. (Publication Year). Title.
2. MLA (Modern Language Association):
   - Format: Authors. "Title" Publication.
3. Chicago:
   - Format: Authors. "Title" Publication.
4. Harvard:
   - Format: Authors (Publication Year). Title.
5. IEEE (Institute of Electrical and Electronics Engineers):
   - Format: Authors, "Title," Publication.

Implementation of Citation Formatting

The implementation of citation formatting involves defining functions for each citation style. These functions take a dictionary containing citation information as input and return a formatted string according to the rules of the specified style.

Applying Citation Formats

When the user inputs a search query and selects a citation style, the tool performs the following steps:

1. Search and Retrieve Data:

- The tool sends a request to Google Scholar, retrieves the HTML content of the search results, and parses this content to extract relevant citation information (titles, authors, and publication details).

2. Select Citation Style:

- The user selects a citation style from a dropdown menu. The selected style is used to choose the appropriate formatting function from the citation_styles dictionary.

3. Format Citations:

- Each extracted citation is passed to the selected formatting function, which returns the citation formatted according to the specified style. The formatted citations are then displayed in the tool's interface.

```
python
def search_and_display_citations():
     Get the search query from the entry field
    query = entry_query.get()
    Get the selected citation style from the dropdown menu
    selected_style = style_var.get()
     Search Google Scholar with initial parameters
    start_index = 0
    num_results = 10
    citations = search_google_scholar(query, start_index, num_results)
     Clear the existing text in the text widget
    text_citations.delete(1.0, tk.END)
     Format and display the citations according to the selected style
    if citations:
        format_function = citation_styles[selected_style]
        for i, citation in enumerate(citations, start=1):
            formatted_citation = format_function(citation)
            text_citations.insert(tk.END, f"{i}. {formatted_citation}\n\n")
   Display pagination information
        total_results = len(citations)
        current_page = 1
        total_pages = (total_results + num_results - 1) // num_results
        text_citations.insert(tk.END, f"Page {current_page} of {total_pages}")
    else:
        text_citations.insert(tk.END, "No citations found.")
```

7. Search and Retrieval Process Explanation

The search and retrieval process is a critical component of the Google Scholar Citation Creation Tool. This section outlines the methodology used to extract relevant academic citations from Google Scholar based on user input. The process leverages web scraping techniques to collect and parse data, ensuring that the tool provides accurate and comprehensive citation information.

User Query Input

The search process begins with the user entering a search query into the tool's interface.

Constructing the Search URL

Once the user submits the query, the tool constructs a search URL tailored to Google Scholar's search parameters. This URL includes the query terms and any additional filters specified by the user. The base URL for Google Scholar search is:

https://scholar.google.com/scholar

Handling Rate Limiting

To prevent being blocked by Google Scholar due to excessive requests, the tool includes rate limiting. This is implemented by introducing delays between successive requests.

```
python
import time
 Function to enforce rate limiting
def rate_limited_request(url, delay=2):
    response = requests.get(url)
    time.sleep(delay)
    return response
 Usage
response = rate_limited_request(url)
```

Parsing the Search Results

Upon receiving a successful response, the tool uses BeautifulSoup to parse the HTML content and extract relevant citation information. The search results are typically structured with specific HTML elements that contain the title, authors, and publication details.

```
python
def parse_search_results(html_content):
    citations = []
    soup = BeautifulSoup(html_content, 'html.parser')
    results = soup.find_all('div', class_='gs_ri')
    for result in results:
        title = result.find('h3', class_='gs_rt').text.strip() if result.find('h3', class_='gs_rt') else ""
        authors = result.find('div', class_='gs_a').text.strip() if result.find('div', class_='gs_a') else
""
        publication = result.find('div', class_='gs_pub').text.strip() if result.find('div',
class_='gs_pub') else ""
        citations.append({'title': title, 'authors': authors, 'publication': publication})

    return citations
```

Displaying the Citations

The parsed citation data is then formatted according to the user-selected citation style and displayed in the tool's interface. The user can view the formatted citations and navigate through multiple pages of results using pagination controls.

```
python
def search_and_display_citations():
    query = entry_query.get()
    selected_style = style_var.get()
    start_index = 0
    num_results = 10
    citations = search_google_scholar(query, start_index, num_results)
    text_citations.delete(1.0, tk.END)
    if citations:
        format_function = citation_styles[selected_style]
        for i, citation in enumerate(citations, start=1):
            formatted_citation = format_function(citation)
            text_citations.insert(tk.END, f"{i}. {formatted_citation}\n\n")
        total_results = len(citations)
        current_page = 1
        total_pages = (total_results + num_results - 1) // num_results
        text_citations.insert(tk.END, f"Page {current_page} of {total_pages}")
    else:
        text_citations.insert(tk.END, "No citations found.")
```

8. Testing and Validation

Testing and validation are crucial components in the development of any software tool to ensure it functions correctly and meets user requirements. This section details the methods used to test and validate the Google Scholar Citation Creation Tool.

Unit Testing

Unit testing involves testing individual components of the application to verify their correctness. For this project, unit tests were written for the following functions:

1. Citation Formatting Functions:

- Each citation formatting function (APA, MLA, Chicago, Harvard, IEEE) was tested to ensure it correctly formats given citation data according to the respective style.
- Sample data was used to verify the output format.

```
python
def test_format_citation_apa():
    citation = {'authors': 'John Doe', 'title': 'Sample Title', 'publication': '2021'}
    expected = 'John Doe. (2021). Sample Title.'
    assert format_citation_apa(citation) == expected
```

2. Search and Parsing Functions:

- The parse_search_results function was tested with sample HTML content to ensure it correctly extracts citation data.
- Mock requests were used to test the search_google_scholar function and validate that it constructs the correct URL and handles the response appropriately.

```
python
def test_parse_search_results():
    html_content = '''
    <div class="gs_ri">
        <h3 class="gs_rt">Sample Title</h3>
        <div class="gs_a">John Doe</div>
        <div class="gs_pub">2021</div>
    </div>
    '''
    expected = [{'title': 'Sample Title', 'authors': 'John Doe', 'publication': '2021'}]
    assert parse_search_results(html_content) == expected
```

Integration Testing

Integration testing involves testing the interactions between different components of the application. For the citation tool, the integration of search, parsing, formatting, and displaying functions was tested to ensure they work together seamlessly.

1. Search and Display Integration:
   - A full search query was performed, and the results were checked for correct formatting and display in the text widget.
   - Pagination functionality was tested to ensure it correctly handles navigation through multiple pages of results.

```
python
def test_search_and_display_citations():
    query = "machine learning"
    selected_style = "APA"
    citations = search_google_scholar(query, 0, 10)
    assert len(citations) > 0  Ensure that at least one citation is returned
    format_function = citation_styles[selected_style]
    formatted_citation = format_function(citations[0])
    assert "machine learning" in formatted_citation.lower()  Verify correct citation formatting
```

Validation

Validation ensures that the tool meets the needs and expectations of the users. For this tool, validation was performed through the following methods:

1. User Feedback:

- The tool was shared with a group of potential users, including students and researchers.
- Users provided feedback on the usability, functionality, and accuracy of the citation generation.

2. Comparison with Manual Citations:

- Citations generated by the tool were compared with manually created citations for the same references.
- This comparison ensured that the tool's output adhered to the standard formatting guidelines for each citation style.

3. Performance Testing:

- The tool's performance was tested to ensure it could handle multiple search queries efficiently without significant delays.
- The rate limiting mechanism was validated to ensure it prevented excessive requests to Google Scholar, thus avoiding potential blocking.

Test Results

The results of the testing and validation processes are summarized below:

- Unit Tests: All unit tests for citation formatting, search parsing, and rate limiting passed successfully.
- Integration Tests: Integration tests confirmed that the search and display functionalities work together correctly. Pagination was handled without issues.
- User Feedback: Users reported that the tool was easy to use and the citations generated were accurate and correctly formatted.
- Performance: The tool performed efficiently, with search results being retrieved and displayed within an acceptable time frame. The rate limiting mechanism effectively prevented excessive requests.

9. Future Enhancement

While the Google Scholar Citation Creation Tool in its current form provides a robust solution for generating citations from Google Scholar search results, several enhancements can be made to

ISSN: 2583-925X
Volume: 2
Issue: 1
Pages: 48-65

improve its functionality, usability, and scope. This section outlines potential future enhancements that could further augment the tool's capabilities.

1. Enhanced Search Filters

Description: Adding more granular search filters would allow users to refine their search results more precisely.

Implementation: Additional filters for language, exact phrase matching, inclusion/exclusion of patents, and sorting options (e.g., by relevance or date) could be incorporated.

2. Exporting Citations

Description: Providing options to export the generated citations in various formats such as .bib, .ris, or plain text would increase the tool's utility.

Implementation: Implement a feature that allows users to download their citations in their desired format. This could be achieved by integrating libraries that support these export formats.

3. Multi-language Support

Description: Adding support for multiple languages would make the tool accessible to a wider audience.

Implementation: Incorporate internationalization (i18n) and localization (l10n) frameworks within the application to support different languages and regional citation styles.

4. Browser Extension

Description: Developing a browser extension would allow users to generate citations directly from their web browser while browsing Google Scholar or other academic databases.

Implementation: Create a browser extension that interacts with the citation tool's backend to fetch and format citation data on-the-fly. The extension could provide a popup interface for entering search queries and displaying results.

5. Integration with Reference Management Software

Description: Integration with popular reference management tools like Zotero, EndNote, or Mendeley would streamline the research workflow for users.

Implementation: Utilize APIs provided by these reference management tools to enable seamless export and synchronization of citations.

6. Advanced Error Handling and User Notifications

Description: Improving error handling and providing detailed user notifications would enhance the overall user experience.

Implementation: Implement more robust error detection mechanisms to handle issues like network failures or changes in Google Scholar's HTML structure. Provide informative notifications to users about the status of their requests and any issues encountered.

7. Machine Learning for Citation Suggestions

Description: Leveraging machine learning algorithms to suggest relevant citations based on the user's research topic could add significant value.

Implementation: Train a machine learning model on a large corpus of academic papers to suggest citations that are highly relevant to the user's query. This feature could use natural language processing (NLP) techniques to understand the context of the user's research.

8. Enhanced User Interface

Description: Improving the graphical user interface (GUI) with a more modern design and better usability features.

Implementation: Use advanced GUI frameworks and libraries to create a more intuitive and visually appealing interface. Include features like drag-and-drop for citations, user preferences for saving search queries, and real-time updates.

10.Conclusion

The development of the Google Scholar Citation Creation Tool addresses a critical need within the academic community by simplifying the process of generating accurate citations. This project has successfully integrated web scraping techniques with a user-friendly graphical user interface to facilitate efficient citation generation from Google Scholar search results.

Key achievements of this project include:

1. User Interface: The tool provides an intuitive and interactive interface that allows users to input search queries, select preferred citation styles, and view formatted citations. This ease of use ensures that even those with minimal technical expertise can benefit from the tool.

2. Web Scraping: Utilizing BeautifulSoup, the tool effectively extracts and processes citation data from Google Scholar, ensuring that users receive accurate and detailed citation information.

ISSN: 2583-925X
Volume: 2
Issue: 1
Pages: 48-65

3. Citation Formatting: Supporting multiple citation styles, including APA, MLA, Chicago, Harvard, and IEEE, the tool meets diverse academic and publication requirements, enhancing its versatility and applicability.

The tool's current capabilities significantly improve the efficiency and accuracy of citation generation, making it a valuable resource for researchers, students, and academics. However, there are numerous opportunities for future enhancements, such as the addition of export options, multi-language support, browser extensions, integration with reference management software, and advanced machine learning features for citation suggestions.

In summary, the Google Scholar Citation Creation Tool not only meets the immediate needs of its users but also lays a solid foundation for future development and innovation. By continuing to evolve and incorporate user feedback, this tool has the potential to become an indispensable asset for the global research community, facilitating more streamlined and effective scholarly communication.

References

1. Furtado, D., & Pennington, M. (2018). Python Programming Blueprints: Build nine projects by leveraging powerful frameworks such as Flask, Nameko, and Django. Packt Publishing Ltd.
2. Padmanaban, K., & Apoorva, V. (2024). Message Encode-Decode Using Python.
3. Bad, U. I., & Good, U. I. (2010). User interface design.
4. Lanning, S. (2016). A modern, simplified citation style and student response. Reference Services Review, 44(1), 21-37.
5. Dhumane, A., Chiwhane, S., Mangore Anirudh, K., Ambala, S. (2023). Cluster-Based Energy-Efficient Routing in Internet of Things. In: Choudrie, J., Mahalle, P., Perumal, T., Joshi, A. (eds) ICT with Intelligent Applications. Smart Innovation, Systems and Technologies, vol 311. Springer, Singapore. https://doi.org/10.1007/978-981-19-3571-8_40
6. Dhumane, A.V., Kaldate, P., Sawant, A., Kadam, P., Chopade, V. (2023). Efficient Prediction of Cardiovascular Disease Using Machine Learning Algorithms with Relief and LASSO Feature Selection Techniques. In: Hassanien, A.E., Castillo, O., Anand, S., Jaiswal, A. (eds) International Conference on Innovative Computing and Communications. ICICC 2023. Lecture Notes in Networks and Systems, vol 703.

Springer, Singapore. https://doi.org/10.1007/978-981-99-3315-0_52

7. Dhumane, A., and D. Midhunchakkaravarthy. "Multi-objective whale optimization algorithm using fractional calculus for green routing in internet of things." Int. J. Adv. Sci. Technol 29 (2020): 1905-1922.

8. Dhumane, A., Chiwhane, S., Tamboli, M., Ambala, S., Bagane, P., Meshram, V. (2024). Detection of Cardiovascular Diseases Using Machine Learning Approach. In: Garg, D., Rodrigues, J.J.P.C., Gupta, S.K., Cheng, X., Sarao, P., Patel, G.S. (eds) Advanced Computing. IACC 2023. Communications in Computer and Information Science, vol 2054. Springer, Cham. https://doi.org/10.1007/978-3-031-56703-2_14

9. Dhumane, A., Pawar, S., Aswale, R., Sawant, T., Singh, S. (2023). Effective Detection of Liver Disease Using Machine Learning Algorithms. In: Fong, S., Dey, N., Joshi, A. (eds) ICT Analysis and Applications. ICT4SD 2023. Lecture Notes in Networks and Systems, vol 782. Springer, Singapore. https://doi.org/10.1007/978-981-99-6568-7_15

10. A. Dhumane, S. Guja, S. Deo and R. Prasad, "Context Awareness in IoT Routing," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697685.

11. Ambala, S., Mangore, A. K., Tamboli, M., Rajput, S. D., Chiwhane, S., & Dhumane, A. "Design and Implementation of Machine Learning-Based Network Intrusion Detection." International Journal of Intelligent Systems and Applications in Engineering, (2023), 12(2s), 120–131. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/3564

12. Kurle, A. S., & Patil, K. R. (2015). Survey on privacy preserving mobile health monitoring system using cloud computing. International Journal of Electrical, Electronics and Computer Science Engineering, 3(4), 31-36.

13. Meshram, V., Meshram, V., & Patil, K. (2016). A survey on ubiquitous computing. ICTACT Journal on Soft Computing, 6(2), 1130-1135.

14. Omanwar, S. S., Patil, K., & Pathak, N. P. (2015). Flexible and fine-grained optimal network bandwidth utilization using client side policy. International Journal of Scientific and Engineering Research, 6(7), 692-698.

15. Dong, X., Patil, K., Mao, J., & Liang, Z. (2013). A comprehensive client-side behavior model for diagnosing attacks in ajax applications. In 2013 18th International Conference

on Engineering of Complex Computer Systems (pp. 177-187). IEEE.

16. Patil, K. (2016). Preventing click event hijacking by user intention inference. ICTACT Journal on Communication Technology, 7(4), 1408-1416.
17. Patil, K., Dong, X., Li, X., Liang, Z., & Jiang, X. (2011). Towards fine-grained access control in javascript contexts. In 2011 31st International Conference on Distributed Computing Systems (pp. 720-729). IEEE.
18. Patil, K., Laad, M., Kamble, A., & Laad, S. (2019). A Consumer-Based Smart Home with Indoor Air Quality Monitoring System. IETE Journal of Research, 65(6), 758-770.
19. Shah, R., & Patil, K. (2018). A measurement study of the subresource integrity mechanism on real-world applications. International Journal of Security and Networks, 13(2), 129-138.
20. Patil, K., & Braun, F. (2016). A Measurement Study of the Content Security Policy on Real-World Applications. International Journal of Network Security, 18(2), 383-392.
21. Patil, K. (2017). Isolating malicious content scripts of browser extensions. International Journal of Information Privacy, Security and Integrity, 3(1), 18-37.
22. Shah, R., & Patil, K. (2016). Evaluating effectiveness of mobile browser security warnings. ICTACT Journal on Communication Technology, 7(3), 1373-1378.
23. Patil, K. (2016). Request dependency integrity: validating web requests using dependencies in the browser environment. International Journal of Information Privacy, Security and Integrity, 2(4), 281-306.
24. Patil, D. K., & Patil, K. (2016). Automated Client-side Sanitizer for Code Injection Attacks. International Journal of Information Technology and Computer Science, 8(4), 86-95.
25. Patil, D. K., & Patil, K. (2015). Client-side automated sanitizer for cross-site scripting vulnerabilities. International Journal of Computer Applications, 121(20), 1-7.
26. Kawate, S., & Patil, K. (2017). An approach for reviewing and ranking the customers' reviews through quality of review (QoR). ICTACT Journal on Soft Computing, 7(2).
27. Jawadwala, Q., & Patil, K. (2016). Design of a novel lightweight key establishment mechanism for smart home systems. In 2016 11th International Conference on Industrial

ISSN: 2583-925X
Volume: 2
Issue: 1
Pages: 48-65

and Information Systems (ICIIS) (pp. 469-473). IEEE.

28. Patil, K., Vyas, T., Braun, F., Goodwin, M., & Liang, Z. (2013). Poster: UserCSP-user specified content security policies. In Proceedings of Symposium on Usable Privacy and Security (pp. 1-2).

29. Patil, K., Jawadwala, Q., & Shu, F. C. (2018). Design and construction of electronic aid for visually impaired people. IEEE Transactions on Human-Machine Systems, 48(2), 172-182.

30. Kawate, S., & Patil, K. (2017). Analysis of foul language usage in social media text conversation. International Journal of Social Media and Interactive Learning Environments, 5(3), 227-251.

31. Patil, K., Laad, M., Kamble, A., & Laad, S. (2018). A consumer-based smart home and health monitoring system. International Journal of Computer Applications in Technology, 58(1), 45-54.

32. Meshram, V. V., Patil, K., Meshram, V. A., & Shu, F. C. (2019). An Astute Assistive Device for Mobility and Object Recognition for Visually Impaired People. IEEE Transactions on Human-Machine Systems, 49(5), 449-460.

33. Meshram, V., Patil, K., & Hanchate, D. (2020). Applications of machine learning in agriculture domain: A state-of-art survey. International Journal of Advanced Science and Technology, 29(5319), 5343.

34. Sonawane, S., Patil, K., & Chumchu, P. (2021). NO2 pollutant concentration forecasting for air quality monitoring by using an optimised deep learning bidirectional GRU model. International Journal of Computational Science and Engineering, 24(1), 64-73.

35. Meshram, V. A., Patil, K., & Ramteke, S. D. (2021). MNet: A Framework to Reduce Fruit Image Misclassification. Ingénierie des Systèmes d'Information, 26(2), 159-170.

36. Meshram, V., Patil, K., Meshram, V., Hanchate, D., & Ramteke, S. (2021). Machine learning in agriculture domain: A state-of-art survey. Artificial Intelligence in the Life Sciences, 1, 100010.

37. Meshram, V., & Patil, K. (2022). FruitNet: Indian fruits image dataset with quality for machine learning applications. Data in Brief, 40, 107686.

38. Meshram, V., Thanomliang, K., Ruangkan, S., Chumchu, P., & Patil, K. (2020). Fruitsgb: top Indian fruits with quality. IEEE Dataport.

39. Bhutad, S., & Patil, K. (2022). Dataset of Stagnant Water and Wet Surface Label Images for Detection. Data in Brief, 40, 107752.

40. Laad, M., Kotecha, K., Patil, K., & Pise, R. (2022). Cardiac Diagnosis with Machine Learning: A Paradigm Shift in Cardiac Care. Applied Artificial Intelligence, 36(1), 2031816.

41. Meshram, V., Patil, K., & Chumchu, P. (2022). Dataset of Indian and Thai banknotes with Annotations. Data in Brief, 108007.

42. Bhutad, S., & Patil, K. (2022). Dataset of Road Surface Images with Seasons for Machine Learning Applications. Data in Brief, 108023.

43. Pise, R., & Patil, K. (2022). Automatic Classification of Mosquito Genera Using Transfer Learning. Journal of Theoretical and Applied Information Technology, 100(6), 1929-1940.

44. Sonawani, S., Patil, K., & Natarajan, P. (2023). Biomedical Signal Processing For Health Monitoring Applications: A Review. International Journal of Applied Systemic Studies, 44-69.

45. Meshram, V., & Patil, K. (2022). Border-Square net: a robust multi-grade fruit classification in IoT smart agriculture using feature extraction based Deep Maxout network. Multimedia Tools and Applications, 81(28), 40709-40735.

46. Suryawanshi, Y., Patil, K., & Chumchu, P. (2022). VegNet: Dataset of vegetable quality images for machine learning applications. Data in Brief, 45, 108657.

47. Sonawani, S., & Patil, K. (2023). Air quality measurement, prediction and warning using transfer learning based IOT system for ambient assisted living. International Journal of Pervasive Computing and Communication, Emerald.

48. Meshram, V., Patil, K., Meshram, V., & Bhatlawande, S. (2022). SmartMedBox: A Smart Medicine Box for Visually Impaired People Using IoT and Computer Vision Techniques. Revue d'Intelligence Artificielle, 36(5), 681-688.

49. Meshram, V., Patil, K., Meshram, V., Dhumane, A., Thepade, S., & Hanchate, D.

(2022). Smart low cost fruit picker for Indian farmers. In 2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA) (pp. 1-7). IEEE.

50. Chumchu, P., & Patil, K. (2023). Dataset of cannabis seeds for machine learning applications. Data in Brief, Elsevier, 108954.

51. Meshram, V., Patil, K., & Bhatlawande, S. (2022). IndianFoodNet: Dataset of Indian Food images for machine learning applications. Data in Brief, 107927.

52. Meshram, V., Patil, K., & Ruangkan, S. (2022). Border-net: fruit classification model based on combined hierarchical features from convolutional deep network for Indian fruits. Multimedia Tools and Applications, 81, 4627-4656.

53. Meshram, V., & Patil, K. (2023). Border-Net: fruit classification model based on combined hierarchical features from convolutional deep network for Indian fruits. Multimedia Tools and Applications, 82, 22801-22830.

54. Patil, K., & Pise, R. (2023). Automation of coconut plantation system using sensors and wireless technology for smart agriculture. IETE Journal of Research.