
Journal of Informatics and Web Engineering

Vol. 5 No. 3 (October 2026)

eISSN: 2821-370X

A Framework for Pre-Pages Generator and Visualiser for Journal Publishing Systems: A Case Study

Karrthik Nadarajan¹, Jiaheng Lu^{2*}

¹Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia

²Department of Computer Science, University of Helsinki, Kumpula Campus, Pietari Kalmin katu 5, 00560 Helsinki, Finland

*corresponding author: (jiaheng.lu@helsinki.fi; ORCID: 0000-0003-2067-454X)

Abstract - In the domain of academic publishing, publishers must establish an efficient editorial workflow for managing their publication schedule while producing quality content, as these two processes will ultimately provide a good experience to readers. Many independent publishers utilizing Open Journal Systems (OJS) rely on a manual compilation of "pre-pages" (front matter), including the Issue Cover, the Editorial Board List, and the Table of Contents (TOC), through a manual workflow that is labour-intensive and prone to formatting and metadata errors. This study presents a web-based Pre-Pages Generator and Visualiser that automates the pre-publishing process, developed using Python and the Streamlit framework. The system utilizes a 5-step Wizard workflow to accomplish the entire pre-publishing process by integrating web scraping to extract existing editorial data from the target journal's website, cleansing the data, and leveraging template-driven rendering to populate Word templates with the appropriate OJS submission data. In addition, the Editorial Intelligence Dashboard provides an overview of issue composition and editorial insights into content trends across issues of the publication by transforming raw bibliometric data. Validation using Journal of Informatics and Web Engineering (JIWE) live production and five historical issues showed 100% formatting compliance across 154 papers, with no manual formatting corrections required. For a 24-paper issue, preparation time was reduced from approximately 90 minutes to 5 minutes (94%). These results demonstrate the practical value of the framework for reducing metadata risk and improving editorial efficiency for non-technical users.

Keywords— Open Journal Systems, Document Automation, Metadata Extraction, Web Scraping, Bibliometric Visualization, Academic Publishing

Received: 21 February 2026; Accepted: 25 August 2026; Published: 16 October 2026

This is an open access article under the [CC BY-NC-ND 4.0](#) license.



1. INTRODUCTION

In the publishing industry, the number of scholarly articles published has increased dramatically, leading the industry to almost completely move to digital workflows. However, many systems still repeat the print-based process, resulting in continued inefficiencies that preclude the full automation and machine-readability of scholarly communication [1]. Practically all journals publish and distribute their articles using a Journal Management System, with Open Journal Systems (OJS) being the international standard for all open-access journals and university-based journals that encourage the entire life cycle of publishing an article. OJS does maintain a very explicitly defined and organized collection of articles metadata but does not currently provide any tools for automatically generating formatted issue-level front matter (hereafter referred to as pre-pages), such as tables of contents and editorial board lists.

As a result, editors are required to extract metadata from OJS manually and re-enter it into a word processing platform. These processes are lengthy, costly, and increasingly subject to human error, introducing a high risk of data inconsistency between the system of record and the published output. The need for metadata accuracy and consistency is vital, and the addition of manually produced data increases the chances of typographical errors occurring in an author's name and Digital Object Identifier (DOI), negatively impacting indexation and lessening the visibility of the work [2]. Furthermore, existing document generation applications frequently have "hard-coded" formatting methods, which make it difficult to modify journal layout over time. Additionally, despite having access to substantial bibliography data via OJS, editors do not have access to integrated visual analytic tools that would allow them to track trends related to keyword use, author contributions, and how various issues are composed, all of which are becoming more critical in the decision-making process of editors [3].

This paper proposes an issue-level automation architecture, through the Pre-Pages Generator and Visualiser that bridges structured OJS metadata and formatted publication output through schema normalization and template-driven rendering. This web-based application comprises a total pre-publication process through the integration of three distinct functions: data extraction, data visualization, and document creation. The system obtains static pre-page data through two hybrid data acquisition methods – tabular export in Comma-Separated Values (CSV) from OJS via database export, and web-based data extraction (using Document Object Model (DOM) parsing) via direct collections. By utilizing a data cleaning layer and a template-based document generation engine, the system can allow non-technical editorial staff to compile compliant Word documents (.docx) without requiring programming skills. The primary contribution of this research is the development of an OJS-integrated tool that unifies issue-level pre-page generation with editorial analytics for non-technical users. Furthermore, the Visualiser component offers a Key Performance Indicators (KPIs) dashboard for editors to see on-demand visual insights on their publications based on uploaded datasets.

In this paper, we employ the Journal of Informatics and Web Engineering (JIWE) as a case study to validate the applicability of the proposed framework within a live OJS-based editorial environment. The case study enables evaluation using real submission metadata and actual issue-level workflows, demonstrating the practicality, scalability, and effectiveness of the system under operational publishing conditions.

2. LITERATURE REVIEW

2.1 Overview of Academic Publishing and Automation Technologies

The transformation of the academic publishing industry from traditional print media to digital-first and open-access models has greatly increased the need for efficient workflows to fulfil the demands for speed, quality, and cost [4]. The efficiency of workflows is also critical to supporting the sustainability of the academic economy [5], reflecting broader trends toward digital automation [6]. OJS has become the prevalent international standard that provides the publication infrastructure to support journal publishing [7]. However, OJS primarily serves as a submission engine and does not natively provide support for the automated generation of formatted pre-pages [8]. Consequently, independent journals must rely on editorial staff to manually input this data into a word processor [9]. This creates an increased risk of human error and could jeopardize the integrity of metadata, thus severely limiting the discoverability of scholarly material within international search indexes [10].

New-generation data pipelines use advanced extraction techniques with cutting-edge metadata extraction and an automated document publishing process to eliminate bottlenecks in the workflow, as referenced above. The conventional methods of extracting metadata employ rigid rule-based heuristics to find metadata [11] and require

multiple iterations before satisfactory results can be obtained, resulting in less than optimal efficiencies throughout the workflow. In contrast, modern techniques utilize "layout-aware" AI-based models and are therefore able to accurately extract document elements, including both text and Two-Dimensional (2D) spatial coordinates, regardless of the font or style used in the document template [12], [13]. In general, when data is extracted from documents as raw data, it has formatting noise and missing values and therefore must be sanitized before it can be used in a definitive manner [14], [15] and benchmarked against established data quality measures before finalizing the processed output [16], [17]. For final output generation, template-based architectures, which separate the variable content from presentation, provide greater flexibility when creating output than conventional techniques for generating hard-coded output [18]. By viewing a document as a structured hierarchy of data instead of a collection of pixels [19] and by using intermediate structured formats, such as eXtensible Markup Language (XML) [20], as seen in recent OJS integrations [21], automation tools can maintain the same output consistency across thousands of articles. Furthermore, once cleaned bibliographic data is represented in a format suitable for visual display, it enables the transformation of static databases into more intelligent strategic decision-support systems for editors [22].

2.2. Related Works in Metadata Extraction and Automation

There has been a significant amount of research on document automation in the academic publishing industry. Böschén [23] evaluated JATSdecoder as an automated text-extraction tool for retrieving statistical information from scientific reports. The study demonstrated that structured JATS XML can support automated extraction from scholarly documents, although extraction performance depends on the consistency and quality of the underlying document structure. This highlights the importance of standardized machine-readable formats in automated publishing workflows. As a result, the program is very limited by standardized XML schemas. The DOCX-to-JATS XML Converter Plugin was developed by [24] to assist with converting Word documents. This plugin utilizes a set of predefined conceptual flowcharts to convert Microsoft Word styles into JATS tags through a pre-defined process. However, because it heavily depends on the template, the plugin is unable to convert documents with minor formatting variations and may require manual intervention by the editor.

Moving beyond rule-based heuristics, Choi et al. [12] have proposed a new, layout-aware approach called Layout-MetaBERT that uses both text and spatial data. The researchers accomplish this by associating bounding box coordinates with the text tokens to better distinguish between headers, title blocks, and similar structural components in a document than using just text. The performance of their model substantially exceeds that of baseline systems, which only used plain text input. The authors noted that the use of spatial coordinates can frequently lead to issues within scanned documents, in which the spatial data are generally unreliable. In addition, Shen et al. [25] introduced the VILA framework for the extraction of metadata from complex multi-column scientific PDFs. VILA incorporates a visual block segmentation module that can perform pattern matching using a transformer-based classifier. VILA produced a relatively high F1-score compared to traditional text-processing extractors. However, it required a considerable amount of computational cost to process these images.

In practical implementations, Musap [26] described a modular automated conversion system to transform manuscripts stored as DOCX files into JATS XML format used by the HRČAK portal. Studies show that while there is a reduction in the amount of editorial workload, the inability to use machine learning techniques makes the system inflexible; therefore, the architecture is limited for complex structures, for instance, embedded images requiring manual intervention for refinement after conversion. Furthering this application, in a separate study, Musap et al. [27] introduced the JATS XML Converter Service to establish interoperability between publishers using the HRČAK and SRCE portal (a central portal that gathers Croatian scientific and professional journals). The service was evaluated in a live production environment, producing a high volume of well-formatted manuscripts and providing a low-volume conversion service for poorly formatted manuscripts. Consequently, the need for continued use of machine learning to improve the performance of the service was recommended.

Focusing on the analytical aspect of metadata, Pereira et al. [28] released an open-source Python library, pyBibX, for the analytical purpose of conducting bibliometric analysis through the use of metadata. pyBibX does an excellent job of constructing various complex visualizations, such as network graph representations of co-authorship and other metrics, but functions solely as an analytical tool. Therefore, the effectiveness of pyBibX is completely dependent on the quality of the data that is provided to it. In most cases, there are insufficient components to properly sanitize or cleanse noisy editorial metadata. In response to this issue, Lim et al. [22] provided a thorough framework for effectively cleaning bibliometric datasets. The authors found that by utilizing a systematic approach to cleaning

bibliometric data (e.g., disambiguation of authors and removal of duplicates) the quality and reliability of visualization tools based on that dataset improved. However, the authors were limited to utilizing expensive databases with subscription restrictions for their study.

Targeting the manuscript submission phase, Rizqi et al. [29] developed an OJS Metadata Extractor Plugin that automates the submission of manuscripts by allowing users to upload files instead of manually entering all of the necessary information. This plugin automatically populates the OJS interface fields as the user uploads files to OJS. The results of testing this plugin on Indonesian scholarly journals demonstrated that it has high accuracy in extracting standard information, such as titles, from these files, thereby greatly reducing the time required to manually enter such information. However, it was less successful in handling complex multi-affiliate structures. To establish standardized metrics for assessing the effectiveness of these technologies, Boukhers and Yang [17] introduced the "Metadata Extraction from Scholarly Documents" (MESD) shared task. The results from this task demonstrated definitively that layout-aware transformer models are superior to traditional rule-based systems for the extraction of metadata from scholarly documents. These results also demonstrated that transformer models trained on multiple template datasets exhibit significantly better generalizability to previously unseen layouts than those trained only on single-template datasets.

2.3 Comparative Analysis and Research Gap

Although researchers have previously developed numerous methods for extracting metadata, there is still a significant gap in the automated generation of issue-level pre-pages. Existing solutions are highly fragmented, with Native OJS plugins providing basic reporting capabilities but lacking document generation. Enterprise platforms like SciSpace have the ability to automate the typesetting of body text but are expensive and designed only for individual "article" submissions, and do not consider the aggregate of all "articles" as a single "issue." Tools like Bibliometrix require significant expertise in programming, making them unsuitable for most editors. Additionally, most existing systems generate hard-coded formats that are incapable of being modified to meet changing layouts of journals. As summarized in Table 1, which compares the potential functionality of the proposed system with these currently available solutions, there are no other solutions within the OJS framework to provide a comprehensive and easily accessible way for OJS editors to automate generating pre-page documents, as well as providing visual representations of pre-publication insights. This research proposes the creation of a Pre-Pages Generator and Visualiser, which will automate pre-page documents for use in the editorial workflow of non-technical editors by using templates. This represents a significant system innovation, as it combines hybrid data extraction with template-driven rendering for solving an issue-level aggregation problem, which cannot otherwise be processed by current XML-dependent automation tools without extensive programming skills.

Table 1. Comparison of Proposed Framework vs. Existing Solutions

Feature / Criteria	Proposed Framework	OJS Native Statistics	SciSpace (For Publishers)	Bibliometrix (R-Tool)
Primary Scope	Issue-Level (Whole Journal Issue)	Journal-Level (Readership)	Full Workflow (Enterprise Ecosystem)	Field-Level (Research Analysis)
Automation Target	Pre-Pages (TOC, Editorial Board)	None (Reporting only)	Article Body (XML/PDF)	None (Analysis only)
Input Data Source	OJS Export (XML/CSV)	Internal Database Logs	Word Document Upload	Scopus/WoS Export Files
Output Format	Editable Word (.docx)	CSV / On-screen Charts	PDF / JATS XML	Static Plots / Charts
Target User	Non-Technical Editors	Journal Managers	Commercial Publishers	Data Scientists / Researchers
Accessibility/Cost	Free / Open Source	Built-in	Enterprise License (Paid)	Open Source (Requires Coding)
Link	N/A	OJS Native Statistics	SciSpace (For Publishers)	Bibliometrix (R-Tool)

3. RESEARCH METHODOLOGY

3.1 Research Design and Data Acquisition

This section discusses the methodologies and technical approaches used to build both the Pre-Pages Generator and Visualiser. The research adopted the Systems Development Research Method (SDRM) design; thus, a functional prototype as the key research instrument for validating the proposed workflows is crucial. This study seeks to provide a system that addresses the operational deficiencies identified in the Literature Review by automating the manual formatting of journal pre-pages and providing a user-intuitive visual inspection of their data.

To support the reproducibility and validity of this work, datasets and software libraries were used to implement this prototype. In addition, we used real submission metadata from an OJS-based journal (JIWE) to illustrate that our approach can be easily applied to the actual editorial process in the journal systems. This system is created to take as its main input a standard OJS 3.0 native CSV export schema. It consists of information such as title, author's name and status of submission. The OJS export's inherent issues with "Schema Flattening" resulted in broken author names in many cases, null section headings, and inconsistent whitespace within the raw data.

To design the system, we selected Python due to its extensive libraries for data manipulation and text processing. The prototype used multiple libraries to address the separation of concerns within the application architecture. Table 2 will include a detailed explanation of the technical tooling and Python libraries used in this study.

Table 2. Summary of Python Libraries and Technical Tools

Category	Library / Tool	Purpose in Prototype
Front-End	Streamlit	Provides the web-based Graphical User Interface (GUI) and session state management.
Data Processing	Pandas	Handles CSV ingestion, cleaning (EDA), and dataframe manipulation.
Web Scraping	BeautifulSoup4	Parses HTML content to extract the Editorial Board list from the live journal website.
Networking	Requests	Sends HTTP requests to fetch raw HTML data from the target URL.
Document Generation	python-docx-template	Merges processed data into Microsoft Word (.docx) templates using Jinja2 tags.
Visualization	Matplotlib / WordCloud	Generates static images for the Co-Word Analysis (Keyword Cloud).

3.2 System Architecture

A Three-Tier Architecture is used as a framework for the system. The modular design allows for a clear separation of concerns and supports future growth and upgrades. The Three-Tier Architecture consists of three layers of logical design: the Presentation Layer, Logic Layer, and Data Layer.

The Presentation Layer, also known as the Front-end of the system, serves as the interface between all users and the system. The Presentation Layer utilizes the Streamlit Framework to create a user-friendly GUI for the user and guides the user through a sequential workflow by facilitating all user-related interactions, including the file uploads. All user interactions in the Presentation Layer drive Logic Layer processing functions through their respective layers.

The Logic Layer, also known as the Back-end, supports the User Interface (UI) by supplying compute engines for the application's functions. The Data Cleaning Engine uses Pandas to standardize the raw CSV files, so that they follow a standard format. The Web Scraping module uses BeautifulSoup to scrape editorial lists, in static formats, directly from the journal's website. The Template Rendering Engine then takes these validated pieces of editorial content and

combines them with predefined .docx templates. Figure 1 depicts the high-level interaction between Presentation Layer, Logic Layer, and Data Layer.

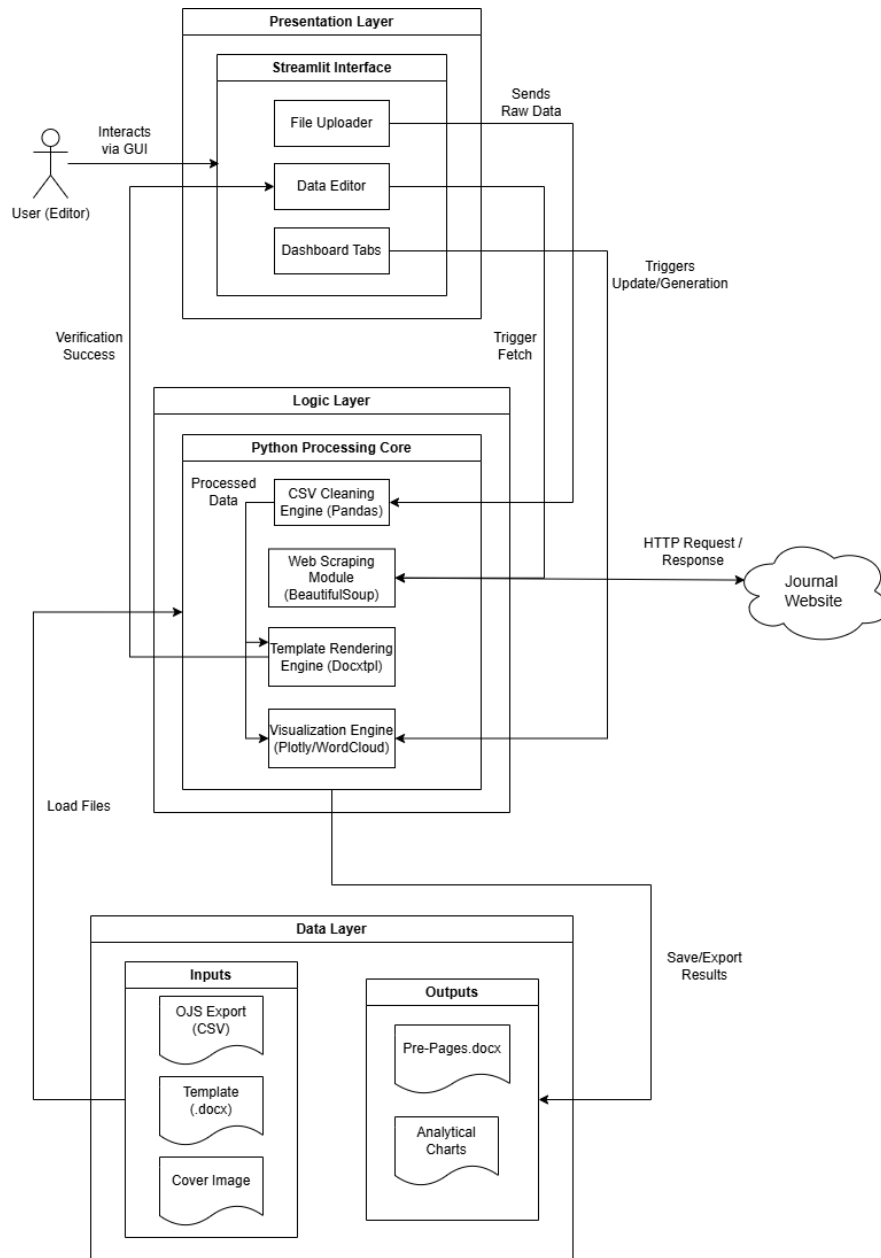


Figure 1. System Architecture Diagram

Finally, the Visualization Engine takes the final output from the Data Layer and uses Plotly and WordCloud libraries to generate graphs and charts that are used by Editors to analyse their results. The Data Layer is the component of the application where user input and output reside. It serves to ingest the OJS system-generated raw article reports and Word templates into a Session Store and provide a temporary storage location for processed dataframes until a final document export is created.

3.3 Process Flow

The overall flow of the system replicates what happens when an editor makes decisions about transforming unprocessed materials into content. The flow consists of several sequential steps that must be completed in order from beginning to end, as shown in Figure 2.

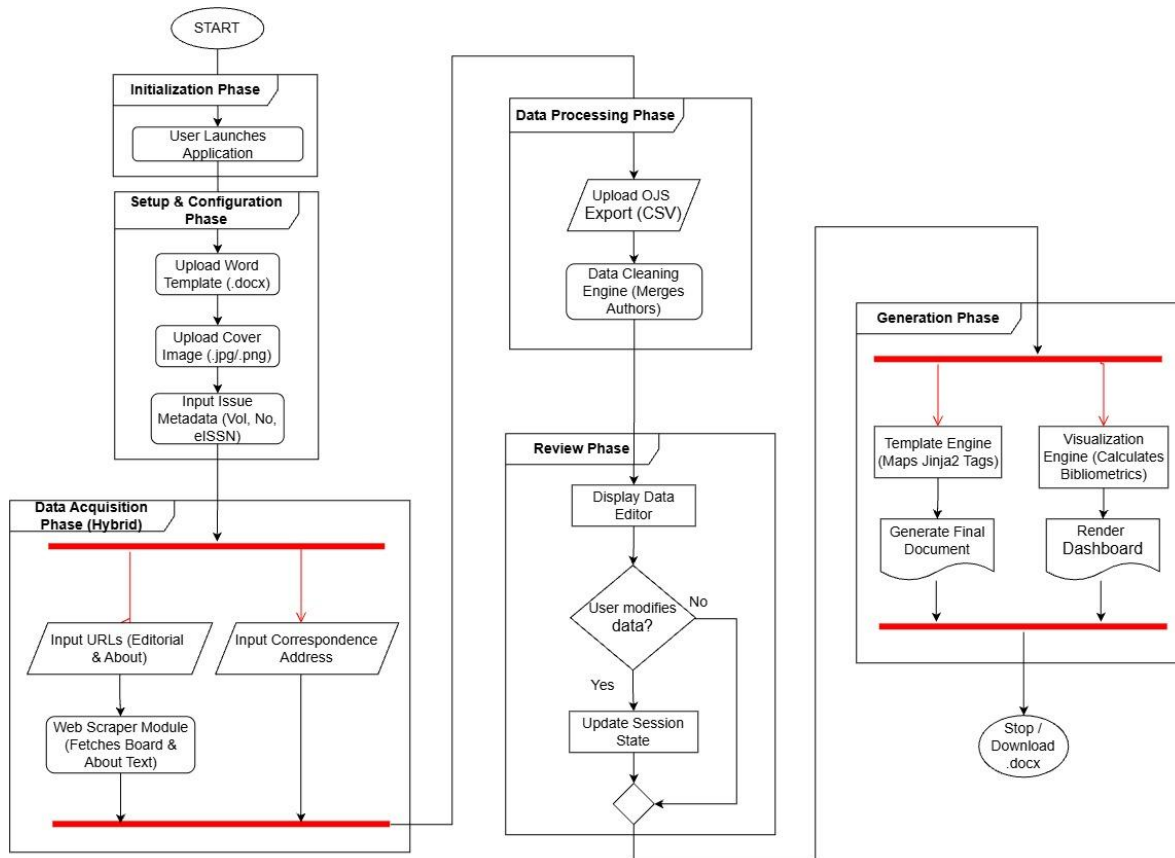


Figure 2. System Process Flowchart

The Initialization Phase begins when a new user first accesses the application and creates their own environment. During the Setup & Configuration Phase, the user uploads all necessary items, including a master template document and a cover image, which are required throughout the user's usage of the application. Next, the user moves into the Data Acquisition Phase, where static data is obtained by crawling the journal's website, and simultaneously, dynamic article metadata from the extracted OJS CSV file is retrieved and ingested into the system.

Following the data collection phases, the application will go through a Data Processing phase where data cleaning algorithms will be executed on the data collected to standardize header formats, consolidate author names, or merge authors, and fill in data into any missing sections. Then, the last two steps will have to be completed: Review and Generation. An interactive data-editing tool will provide a user with a means to check the accuracy of the Table of Contents (TOC). When finished with this task, the application will map the data from the TOC to the corresponding placeholders in the Jinja2 template contained in the master Word template to produce the final document, while simultaneously creating an "Editorial Intelligence" dashboard.

3.4 Implementation Logic

Every aspect of the system's operation is implemented as separate Python functions to allow for code reusability. All of the code that performs Data Cleaning (EDA), Document Generation, and Visualization is logically separated from each other. The `clean_ojs_csv` function is the primary data cleansing function that uses a three-step algorithm to address the problem of "Schema Flattening" found in OJS exports. The first step is "Normalization" - stripping leading and trailing whitespace from all column headers so that `KeyError` exceptions will not occur when indexing due to invisible formatting characters. The next step is "Missing Value Imputation" within key fields. For example, the function checks if the "Section Name" is null, and if so, the function assigns the default value of "Articles" so that valid papers do not get discarded during grouping due to missing data. Finally, the system applies "Entity Resolution" to resolve fragmented author identities. To do this, the function iterates through 15 possible author columns (e.g.,

Author 1 First Name, Author 1 Last Name) with a loop that validates the presence of data and concatenates that author information together into a single unified "Authors" string. This allows the generation engine to receive data in the form of a structured comma-separated string, rather than from a sparse set of columns.

In this document creation module, the `generate_document` method is responsible for generating documents from pre-defined templates using a template-driven approach. This means that a document's presentation is separated from the business logic used to create the document, as opposed to traditional string replacement methods, which rely solely on replacing strings. Because the dynamic context mapping of specific data points is done during data structuring, the Jinja2 templating engine can map the cleaned dataframe, which is serialized into a Python dictionary, pairing each of the data points within the Word document to a corresponding Jinja2 tag. For example, each data point (article title) would be associated with its corresponding Jinja2 tag, `{{ article_title }}`. For dynamic elements like a TOC, the logic for how each item should be rendered is built into the Word template using control structures (for example, `{% for articles in toc_sections %}`). The Jinja2 templating engine uses dynamic rendering when rendering the document by iterating over the entire list of articles and applying the table style and formatting to each row of the table, allowing the document to continue growing and to have no limit to the number of articles it contains. Finally, the document will always match the same style, font, and margin as defined by the master template.

Using aggregation and frequency analysis algorithms, the Visualization Module takes raw, unstructured text data and produces analytical insights. To provide true analytics based on actual issue content, a critical Global Status Filtering logic is applied prior to analytics processing. The algorithm identifies records that have an active status (Published, Scheduled, Production, or Copyediting) and excludes any records that are rejected or archived from the submission. Co-word analysis is then performed using a Text Tokenization technique where all article titles are combined into one large text corpus. Then, all of the stop words that are common non-descriptive words, such as "the" and "and", are removed from the large text corpus. This enables the user to find research keywords that are most useful for developing an understanding of a particular area of research by counting how many times each keyword appears across the text corpus. The demographic distribution charts use a Group-By operation on (section name and author) the filtered dataset to generate counts or frequencies for the ways the combinations appear. The raw row data is then grouped into a frequency distribution (for each combination), so that they can be represented visually in interactive pie charts and bar charts, using the Plotly charting engine.

3.5 Pseudocode Description

The Pre-Pages Generator relies on a sequential pipeline that ingests data from the OJS and processes the data to ensure that accurate data is being used for creating documents and generating visual analytics. Examples of the processing activities performed in the sequential pipeline are provided in the form of pseudocode (Algorithm 1), which demonstrates how the data is processed for purposes of normalizing headers, author entity resolution, and producing visualizations of bibliometric data.

Algorithm 1: Pre-Pages Data Processing and Visualization Pipeline

```

Input: Raw CSV file exported from the OJS containing submission metadata.
Output: Cleaned dataframe for document generation and a set of visual analytics charts.
1  IMPORT pandas, matplotlib, wordcloud
2
3  FUNCTION process_and_visualize(raw_csv_file):
4    // Phase 1: Data Ingestion and Cleaning
5    raw_dataframe <- READ_CSV(raw_csv_file)
6
7    // Normalize column headers to remove whitespace
8    FOR each column_header IN raw_dataframe.columns:
9      column_header <- STRIP_WHITESPACE(column_header)
10   END FOR
11
12   // Impute missing Section Names
13   FOR each row IN raw_dataframe:
14     IF row["Section Name"] IS NULL OR EMPTY:
15       row["Section Name"] <- "Articles" // Default fallback value
16     END IF
17   END FOR

```

```

18
19 // Phase 2: Entity Resolution (Author Merging)
20 INITIALIZE new column "Authors"
21 FOR each row IN raw_dataframe:
22     authors_list <- [] // Empty list to store found authors
23
24     // Iterate through OJS dynamic author columns (Author 1 to 15)
25     FOR i FROM 1 TO 15:
26         first_name <- GET value of ("Author " + i + " First Name")
27         last_name <- GET value of ("Author " + i + " Last Name")
28
29         // Only merge if both name parts exist
30         IF first_name IS NOT NULL AND last_name IS NOT NULL:
31             full_name <- CONCATENATE(first_name, " ", last_name)
32             APPEND full_name TO authors_list
33         END IF
34     END FOR
35
36     // Join list into a single string for the final document
37     row["Authors"] <- JOIN(authors_list, delimiter=", ")
38 END FOR
39
40 // Phase 3: Advanced Sorting Logic (TOC Sequencing Fix)
41 // Extract numeric suffix from DOI for precise sorting
42 FOR each row IN raw_dataframe:
43     row["doi_suffix"] <- EXTRACT_LAST_NUMBER(row["DOI"])
44
45     // Exception Handling: Force "Editorial Preview" to top
46     IF row["Article Title"] CONTAINS "Editorial Preview":
47         row["doi_suffix"] <- -1
48     END IF
49
50     // Prioritize "Regular Issue" sections over others
51     IF row["Section Name"] CONTAINS "Regular" OR "Articles":
52         row["section_rank"] <- 0
53     ELSE:
54         row["section_rank"] <- 1
55     END IF
56 END FOR
57
58 // Sort by Rank (Primary), Section (Secondary), DOI (Tertiary)
59 sorted_dataframe <- SORT raw_dataframe BY ("section_rank", "Section
Name", "doi_suffix") ASCENDING
60
61 // Phase 4: Visualization Generation
62 INITIALIZE visual_outputs <- []
63 active_statuses <- ["Published", "Scheduled", "Production", "Copyediting"]
64
65 // Filter dataset to exclude rejected or archived submissions
66 filtered_dataframe <- FILTER sorted_dataframe WHERE "Status" IN
active_statuses
67
68 // Vis 1: Word Cloud (Topic Trends)
69 text_corpus <- ""
70 FOR each title IN filtered_dataframe["Article Title"]:
71     text_corpus <- CONCATENATE(text_corpus, " ", title)
72 END FOR
73 clean_tokens <- REMOVE_STOPWORDS(TOKENIZE(text_corpus))
74 word_cloud <- GENERATE_WORD_CLOUD(frequencies=clean_tokens)
75 APPEND word_cloud TO visual_outputs
76
77 // Vis 2: Section Breakdown (Pie Chart)

```

```

78  section_counts <- GROUP_BY(filtered_dataframe, "Section
    Name").COUNT()
79  pie_chart <- PLOT_PIE(data=section_counts, labels="Section",
    values="Count")
80  APPEND pie_chart TO visual_outputs
81
82  // Vis 3: Top Contributors (Bar Chart)
83  all_authors <- []
84  FOR each author_string IN filtered_dataframe["Authors"]:
85      // Split merged string back into individual entities for counting
86      individual_authors <- SPLIT(author_string, delimiter=", ")
87      EXTEND all_authors WITH individual_authors
88  END FOR
89
90  author_freq <- COUNT_OCCURRENCES(all_authors)
91  top_10_authors <- SORT DESCENDING(author_freq).GET TOP(10)
92  bar_chart <- PLOT_BAR(data=top_10_authors, x="Author",
    y="Publications")
93  APPEND bar_chart TO visual_outputs
94
95  RETURN sorted_dataframe, visual_outputs
96 END FUNCTION

```

The algorithm begins by initializing the environment and ingesting the raw OJS CSV (L1-17). This process is referred to as "Data Hygiene" because it involves stripping invisible whitespace from the header columns. Additionally, if an article is missing a valid "Section Name", that value will be populated with a default fallback in order to maintain data integrity. In Step 2 (L19-38), the algorithm performs "Entity Resolution" on the author information by iterating through the 15 broken author columns and aggregating them into one single combined name (i.e., "Authors"). Finally, the last step of the algorithm is shown in (L40-59) and is titled "Advanced Sorting Logic". This step extracts the numeric DOI suffixes from each section (e.g., "Editorial Preview") and assigns a rank to each section in order to eliminate any sequencing anomalies represented in standard OJS reports.

The "Editorial Intelligence" Dashboard is created in the latter half of the algorithm. Filtering is accomplished in the first part of the "Editorial Intelligence" section (L61-66), where only documents currently in an active status (Published, Scheduled, Production) will pass through to the rest of the algorithm, while documents containing the status "Rejected" will be automatically discarded using Global Status Filtering logic. In the second part of the "Editorial Intelligence" section (L68-75), non-descriptive "stop words" are filtered out of the article title to produce a word cloud. The remaining lines (L77-93) aggregate report data by section for the Pie Chart and calculate unique author frequencies for the Top Contributors Bar Chart. The results will then be returned in a final clean dataframe containing the final graphical visualization assets (Lines 95-96).

3.6 Prototype UI

The UI of the prototype has been created using a linear "Wizard" style of interface that guides the editorial user through five sequential steps as they proceed through their workflow. The linear model method is used to create a lower cognitive load for the editor by breaking down the journal format task into easy-to-understand, manageable, and isolated stages.

3.6.1 Step 1: System Initialization

The workflow begins with the Setup interface as shown in Figure 3. In this screen, the user must upload necessary assets, namely, the master file (.docx) template created using Microsoft Word, and a Cover Image (.jpg or .png). The interface's design provides strict enforcement of the correct file type to prevent errors from occurring when generating an issue.

Figure 3. The Setup Interface Allowing Users to Upload the Word Template and Issue Cover Image

3.6.2 Step 2: Metadata Configuration

Upon uploading the requested assets, the user proceeds to the Issue Details screen, where all the necessary data captured within the application, as depicted above in Figure 4, is visible. Specifically, this screen displays all relevant metadata that will automatically populate all three areas of the document header, footer, and title page. Therefore, all Journal Name, Volume, Issue, Month and Year, and the Journal's eISSN will need to be entered manually by the user. The dropdown menus will help to maintain a standard for those two fields (Month and Year).

Figure 4. The Issue Details Form for Inputting Volume, Issue, and Date Metadata

3.6.3 Step 3: Editorial Board Acquisition

The third stage of the process, as shown in Figure 5, is an interface with the Web Scraping Module. Instead of requiring a user to type in the names of the entire editorial board into each document, the user simply pastes the URL of the journal's editorial team page into the input field on this screen. By clicking on the "Fetch" button on this screen, the web crawler will parse and extract the HTML from the journal's web page, retrieve the names of all editorial board members, and automatically categorize those names into the corresponding fields, such as Editor-in-Chief, Managing Editors, and Advisory Board, for verification by the editors using the system.

Figure 5. The Web Scraping Interface for Fetching Editorial Board Details

3.6.4 Step 4: Journal Scope and Address

Similar to the previous step, the About interface (Figure 6) will extract the journal's description and scope text from the URL provided by the user. It also includes a dedicated text area to enter or edit the publisher's address, which is used to populate the pre-pages with accurate and up-to-date contact details.

Figure 6. The About Interface for Fetching Journal Scope and Inputting the Address

3.6.5 Step 5: TOC Management

The final configuration stage is the TOC manager (Figure 7), which provides three main functions: first, this tool ingests the raw CSV export from OJS; second, it applies a default filter to show only "Scheduled" articles and exclude rejected and in-process submissions; and last, it enables users to assign page numbers to each article; users can either enter page ranges into the "Edit Content" table or upload the PDFs of the articles for the system to capture the page

count and populate this column. The final function of the TOC manager is the "Generate Document" button, which creates the pre-pages document.

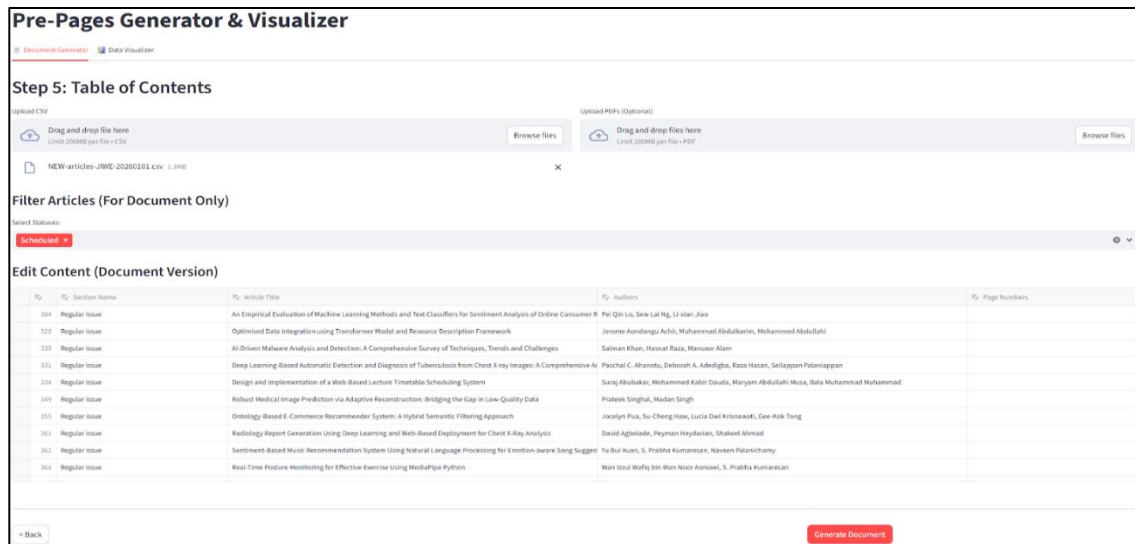


Figure 7. The Data Editor Interface Showing CSV Ingestion, Status Filtering, and the Table View

3.6.6 Editorial Intelligence Dashboard

In addition to the linear generation workflow, the prototype comprises a Data Visualiser tab, as shown in Figure 8 and Figure 9. With the help of this dashboard, users have access to bibliometric insights regarding the dataset that has been uploaded, reflecting only the currently active submissions, processed by using the filtering logic described above (Section 3.4). The Data Visualiser Dashboard includes a Topic Trend Word Cloud which presents some of the most frequently used keywords found in the issue's article titles, allowing editors to view at a quick glance the overall themes of the research associated with the submissions. Furthermore, the Data Visualiser Dashboard also provides a Section Breakdown Pie Chart which shows how many articles were submitted under each category, such as Regular Issues and Thematic Issues. Finally, the Data Visualiser Dashboard also provides a Top Authors Bar Chart which ranks the top 10 most prolific and active authors in the current issue, making it easy for editors to see who the leading Key Researchers and Frequent Collaborators are.



Figure 8. The Topic Trends Visualization (Word Cloud) Generated from Article Titles

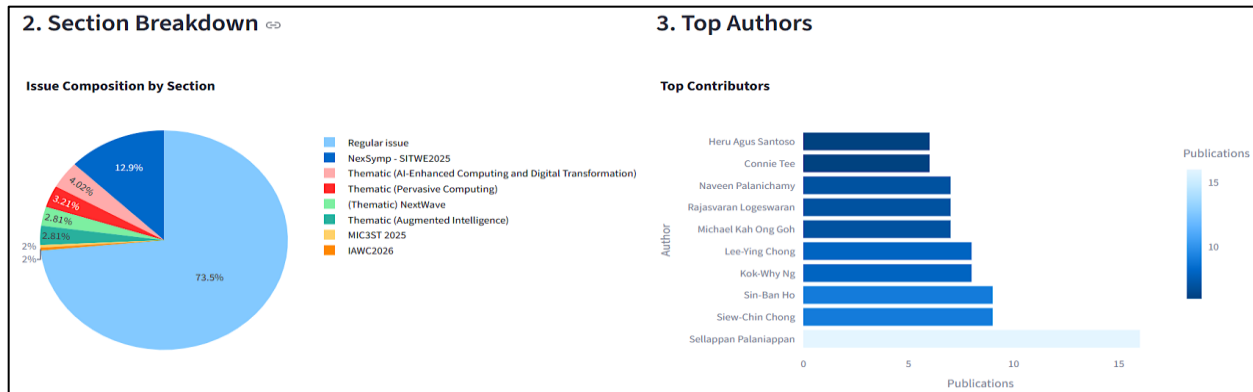


Figure 9. The Section Breakdown and Top Authors Analytics

4. RESULTS AND DISCUSSIONS

4.1 Document Generation Outputs

The implementation of the Pre-Pages Generator successfully produced a production ready .docx file with all the pre-pages required for a journal issue. The Pre-Pages Generator is based on template-driven rendering and dynamically inserts content based on the publishing house's formatting requirements, such as font type, margin size, and spacing between characters. Each of the generated outputs is further divided into major sections, which comprise: the Issue Cover, the Editorial Board, the Journal Scope, and the TOC.

The creation of the Issue Cover involved using a Dynamic Image Replacement method. The system identified a pre-defined placeholder in the Word template, and the photo uploaded by the user to be used as their cover was swapped in. As shown in Figure 10 (a), the configuration ensured that when the image uploaded by the user was displayed on the cover, it was displayed in a full-bleed manner around the edges of the issue, retaining its original aspect ratio (Dimensions), creating a new and different visual identity for the issue.

After the Cover page was successfully generated, the Editorial Information Page was also generated with the successful merging of data from dynamic metadata (Journal Name, Volume, Issue, Month, Year, eISSN) with the editor lists that were structured in the web scraping module. The Jinja2 looping logic allowed for the automatic production of the Editor-in-Chief, Managing Editors, and Advisory Board lists with the appropriate bulleted formatting, which eliminated the need to manually reformat the content as shown in Figure 10 (b).



Figure 10. (a) The Cover Page, and (b) The Formatted Editorial Board Generated Dynamically

The final and most complex output produced was the TOC. This was accomplished by using the control structures built into this template to iterate through the articles scheduled for publication that had been filtered in accordance with various control parameters. As illustrated in Figure 11, each submission received its own new row in the table, with Article Title, Author Name, and Page Number correctly aligned in the new row of the TOC. The end result of this output demonstrates that the application can produce very complex, tabular formatted documents with variable datasets, while still adhering to strict layout integrity.

Table of Contents		
Regular issue		
An Empirical Evaluation of Machine Learning Methods and Text Classifiers for Sentiment Analysis of Online Consumer Reviews <i>Pei Qin Lo, Sew Lai Ng, Li-xian Jiao</i>	1-10	
Optimised Data Integration using Transformer Model and Resource Description Framework <i>Jerome Aondongu Achir, Muhammad Abdulkarim, Mohammed Abdullahi</i>	11-15	
AI-Driven Malware Analysis and Detection: A Comprehensive Survey of Techniques, Trends and Challenges <i>Saiman Khan, Hasnat Raza, Mansoor Alam</i>	16-25	
Deep Learning-Based Automatic Detection and Diagnosis of Tuberculosis from Chest X-ray Images: A Comprehensive Analysis <i>Paschal C. Ahamotu, Deborah A. Adedigba, Raza Hasan, Sellappan Palaniappan</i>	26-40	
Design and Implementation of a Web-Based Lecture Timetable Scheduling System <i>Suraj Abubakar, Mohammed Kabir Dauda, Maryam Abdullahi Musa, Baia Muhammad Muhammad</i>	41-50	
Robust Medical Image Prediction via Adaptive Reconstruction: Bridging the Gap in Low-Quality Data <i>Prateek Singhal, Madan Singh</i>	51-60	
Ontology-Based E-Commerce Recommender System: A Hybrid Semantic Filtering Approach <i>Jocelyn Pua, Su-Cheng Haw, Lucia Dwi Krisnawati, Gee-Kok Tong</i>	61-65	

Figure 11. The Generated TOC Showing Correct Alignment of Titles and Page Numbers

4.2 Visual Analytics Outputs

An interactive Editorial Intelligence Dashboard Module has been developed that uses bibliometric data obtained from the generated articles to visually display the composition of the current journal issue via static documents and raw metadata.

The “Topic Trends” word cloud shown in Figure 12 contains the tokenized text of all article titles scheduled for publication, excluding the common “stop” words (i.e., “using” and “based”) that would not add meaning to an editor's ability to view research trends. By using n-gram text processing, limited to 2-word phrases (bigrams), the algorithm highlights the most significant research topics and themes found in the given submissions by displaying the specific phrases (i.e., “Machine Learning”, “Deep Learning”, etc.) that were used most often throughout all submissions to give editors a visual overview of the research domain for each submission subject in a single view. The dashboard provides editors with an overall summary of the latest research trends in their respective publications by visually grouping the specific phrases together based on their frequency.

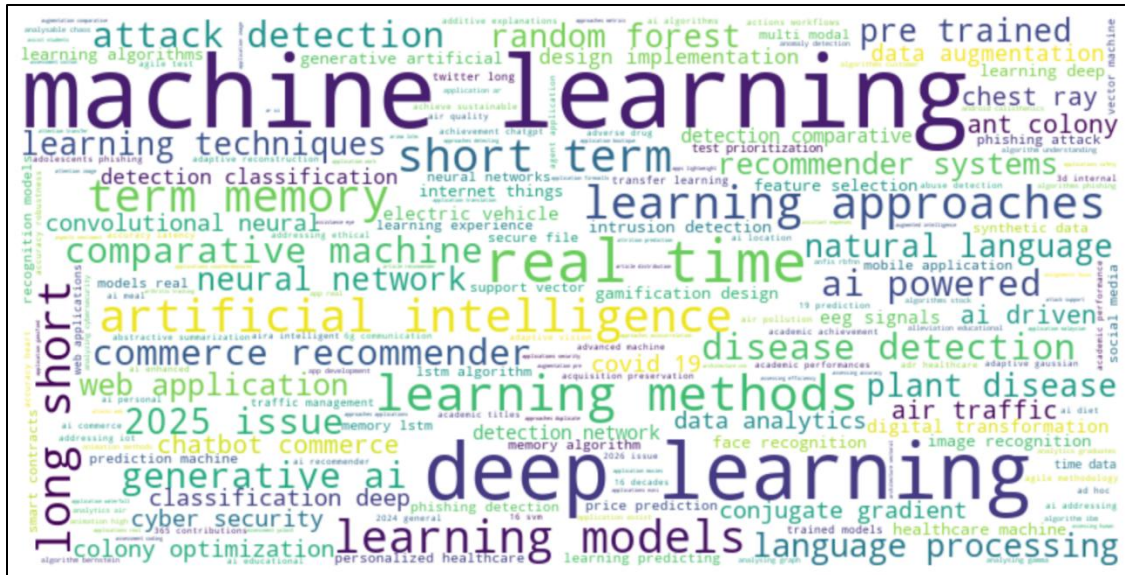


Figure 12. Topic Trends Visualization Highlighting Dominant Keywords from Article Titles

Additionally, a Section Breakdown Pie Chart was produced from the automated aggregation of structural content. Figure 13 illustrates that, after classification of the articles, Regular Issues represented 72.6% of the content, with the rest divided among thematic issues and conference proceedings. This provides the editorial staff with a means of analysing the distribution of content categories (i.e., structural content) before the content is published.

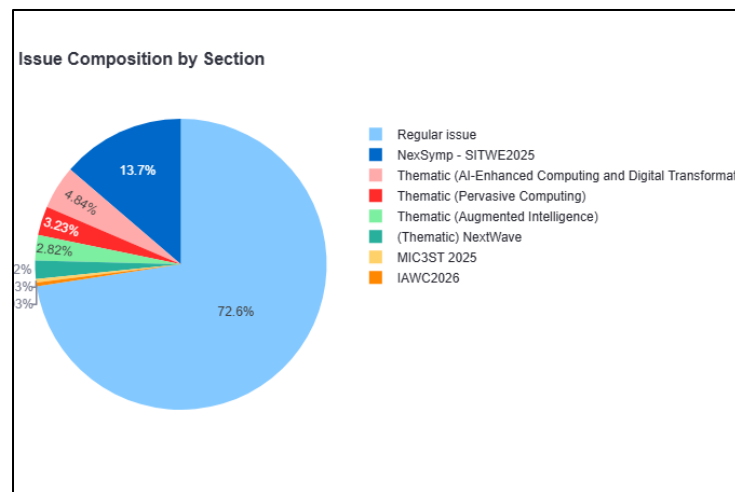


Figure 13. Section Breakdown of The Distribution of Articles

4.3 User Acceptance Testing (UAT) and Error Analysis

The final system was evaluated by performing a live UAT, historical regression testing, and comparing it with the existing manual workflow. For the former, the system was used for the first time for the pre-pages of *JIWE* Vol. 5, Issue 2. The issue consists of 24 manuscripts, the outputs of which were checked by the production editor for conformance with the journal's formatting and publication requirements. The ten test items were successfully passed, as seen in Table 3.

Table 3. Final UAT Item Breakdown

Test Item	Status
Issue Cover Page Generation	Passed
About the Journal (Aims & Scope) Formatting	Passed
Journal Address & Contact Information	Passed
Editorial Board List Formatting	Passed
TOC: Regular Issue Sequencing	Passed
TOC: Thematic Issue Sequencing	Passed
Page Number Calculation (Dual-Path Engine)	Passed
Author Metadata & Affiliation Mapping	Passed
Native OOXML Hyperlink Injection	Passed
Text Sanitization (No justification/HTML anomalies)	Passed

In addition, a historical regression test was performed using five previous issues of JIWE in their current form, amounting to 130 papers. The issues were re-generated with 100% formatting compliance and did not require any manual interventions. In addition, the system was able to process the live dataset five times with the same result, achieving identical outputs for all runs.

Finally, the system's impact on the preparation time was estimated and compared to the existing manual process. As seen in Table 4, a 24-paper issue required approximately 90 minutes of work to be prepared manually, while the automated system only took about 5 minutes. Thus, it can be seen that the preparation time was reduced by approximately 94%.

Table 4. Performance Comparison – Manual vs. Automated Workflow (24-Paper Issue)

Metric	Traditional Manual Workflow	Proposed Automated System
Total Formatting Time	~90 minutes	~5 minutes
Manual Corrections Required	5 to 8 corrections	0 corrections
Typographical Error Risk	High	Eliminated
Formatting Compliance	~85% (requires heavy revision)	100% (ready for publication)

Overall, the results show that the final system was able to produce consistent pre-pages while reducing the amount of manual work required. However, the findings are based mainly on the JIWE production environment, and further testing with other journals would be useful to determine how well the system performs across different publishing workflows.

5. CONCLUSION

This paper describes the creation of a web-based Pre-Pages Generator and Visualiser. The framework allows for the automatic generation of issue-level publishing workflows. It implements OJS data mining, web scraping and sanitisation, cloud storage of persistent data, page calculator, Word template-based generation, and visualisation of

the editing process. Live production and historical regression testing revealed that the framework consistently produced publisher-compliant pre-pages with significantly reduced manual effort.

The final framework achieved 100% formatting compliance in the live 24-paper JIWE issue and five historical issues containing 130 papers in total, with no manual formatting corrections needed. Comparison of the workflow also showed a reduction in manual preparation from approximately 90 minutes to only 5 minutes. The paper demonstrates that the framework is indeed viable for issue-level production in the test JIWE environment. However, the current system is not without its limitations. The web scraping component relies partially on pre-defined patterns for different editorial roles. Local PDF pagination requires careful input file management to avoid duplicates or outdated documents.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for the suggestions to improve the paper.

FUNDING STATEMENT

The authors received no funding from any party for the research and publication of this article.

AUTHOR CONTRIBUTIONS

Karrthik Nadarajan: Conceptualization, Data Curation, Methodology, Validation, Writing – Original Draft Preparation; Jiaheng Lu: Project Administration, Writing – Review & Editing.

CONFLICT OF INTERESTS

No conflict of interests were disclosed.

ETHICS STATEMENTS

Our publication ethics follow The Committee of Publication Ethics (COPE) guideline. <https://publicationethics.org/>

DATA AVAILABILITY

Derived data supporting the findings of this study are available from the corresponding author on request.

REFERENCES

- [1] P. M. Saugmann, “The precariousness of academic publishing in a digital world,” *European Review*, vol. 32, no. S1, pp. S75–S91, 2024, doi: 10.1017/s1062798723000613.
- [2] L. Cespedes *et al.*, “Evaluating the linguistic coverage of OpenAlex : An assessment of metadata accuracy and completeness,” *Journal of the Association for Information Science and Technology*, vol. 76, no. 6, pp. 884-895, 2025, doi: 10.1002/asi.24979.
- [3] D. Auclair, J. Carmichael, and J. Thibodeau, “Challenges and roadblocks to robust metadata in the scholarly communications industry,” *Insights: the UKSG Journal*, vol. 37, 2024, doi: 10.1629/uksg.642.

-
- [4] S. Serpa *et al.*, “Future of scholarly publishing: A perspective,” *Science Insights*, vol. 39, no. 3, 2021, doi: 10.15354/si.21.re239.
- [5] S. Baker, “Assessing open source journal management software,” *Journal of Electronic Publishing*, vol. 23, no. 1, 2020, doi: 10.3998/3336451.0023.101.
- [6] R. Kannan and J.-J. Chin, “Backup automation using power automate for Malaysian vaccination centres,” *Journal of Informatics and Web Engineering*, vol. 1, no. 1, 2022, doi: 10.33093/jiwe.2022.1.1.3.
- [7] B. D. Edgar and J. Willinsky, “A survey of the scholarly journals using Open Journal Systems,” *OJS pa dansk*, vol. 1, no. 1, 2010, doi: 10.7146/ojsb.v1i1.2707.
- [8] B. Tabatadze, “Technological aspects of Open Journal Systems (OJS),” *Journal of Technical Science and Technologies*, vol. 8, no. 1, 2024, doi: 10.31578/jtst.v8i1.151.
- [9] M. Farmanbar and P. Kolstrup, “Publishing with Open Journal Systems (OJS): The engaged university and library support at the University of Stavanger,” *Septentrio Conference Series*, no. 4, 2020, doi: 10.7557/5.5604.
- [10] W. A. M. Gregg, C. Erdmann, L. Paglione, J. Schneider, and C. Dean, “A literature review of scholarly communications metadata,” *Research Ideas and Outcomes*, vol. 5, 2019, doi: 10.3897/rio.5.e38698.
- [11] M. Waqas, N. Anjum, and M. T. Afzal, “A hybrid strategy to extract metadata from scholarly articles by utilizing support vector machine and heuristics,” *Scientometrics*, vol. 128, no. 8, 2023, doi: 10.1007/s11192-023-04774-7.
- [12] J. Choi, H. Kong, H. Yoon, H. Oh, and Y. Jung, “LAME: Layout-aware metadata extraction approach for research articles,” *Comput. Mater. Contin.*, vol. 72, no. 2, pp. 4019–4037, 2022, doi: 10.32604/cmc.2022.025711.
- [13] H. Kim, J. Choi, S. Park, and Y. Jung, “Layout aware semantic element extraction for sustainable science & technology decision support,” *Sustainability (Switzerland)*, vol. 14, no. 5, 2022, doi: 10.3390/su14052802.
- [14] F. Ridzuan and W. M. N. Wan Zainon, “A review on data cleansing methods for big data,” *Procedia Comput. Sci.*, vol. 161, pp. 731–738, 2019, doi: 10.1016/j.procs.2019.11.177.
- [15] J. Zhu, X. Zhao, Y. Sun, S. Song, and X. Yuan, “Relational data cleaning meets Artificial Intelligence: A survey,” *Data Science and Engineering*, vol. 10, no. 2, pp. 147–174, 2024, doi: 10.1007/s41019-024-00266-7.
- [16] N. Meuschke, A. Jagdale, T. Spinde, J. Mitrovic, and B. Gipp, “A benchmark of PDF information extraction tools using a multi-task and multi-domain evaluation framework for academic documents,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2023, doi: 10.1007/978-3-031-28032-0_31.
- [17] Z. Boukhers and C. Yang, “MESD: Metadata Extraction from Scholarly Documents- A shared task overview,” 2025. [Online]. Available: <https://zeyd.boukhers.com>
- [18] M. Ahmadi Achachlouei, O. Patil, T. Joshi, and V. N. Nair, “Document automation architectures and technologies: A survey,” *arXiv preprint arXiv:2109.11603*, 2021, doi: 10.48550/arXiv.2109.11603.
- [19] A. Krewinkel and R. Winkler, “Formatting open science: Agilely creating multiple document formats for academic manuscripts with pandoc scholar,” *PeerJ Computer Science*, vol. 2017, no. 5, 2017, doi: 10.7717/peerj-cs.112.
- [20] C. Borchert, R. Cozatl, F. Eichler, A. Hoffmann, and M. Putnings, “Automatic XML extraction from word and formatting of E-Book formats: Insight into the Open Source Academic Publishing Suite (OS-APS),” *Publications*, vol. 11, no. 1, 2023, doi: 10.3390/publications11010001.
- [21] C.-X. Ling, K.-W. Ng, and H. A. Santoso, “Mapping relational database to Full-text XML for Open Journal System cross-platform article distribution,” *Journal of Informatics and Web Engineering*, vol. 4, no. 3, pp. 259–277, Oct. 2025, doi: 10.33093/jiwe.2025.4.3.16.

- [22] W. M. Lim, S. Kumar, and N. Donthu, "How to combine and clean bibliometric data and use bibliometric tools synergistically: Guidelines using metaverse research," *Journal of Business Research*, vol. 182, p. 114760, 2024, doi: 10.1016/j.jbusres.2024.114760.
- [23] I. Boschen, "Evaluation of JATSdecoder as an automated text extraction tool for statistical results in scientific reports," *Scientific Reports*, vol. 11, no. 1, 2021, doi: 10.1038/s41598-021-98782-3.
- [24] OJS-Services, "DOCX to JATS XML Converter Plugin," OJS Services. Accessed: Dec. 05, 2025. [Online]. Available: <https://ojs-services.com/ojs-plugins/docx-to-jats-xml-converter-plugin/>
- [25] Z. Shen, K. Lo, L. L. Wang, B. Kuehl, D. S. Weld, and D. Downey, "VILA: Improving structured content extraction from scientific PDFs using visual layout groups," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 376–392, 2022, doi: 10.1162/tacl_a_00466.
- [26] L. J. Musap, "Enhancing scientific publishing: Automatic conversion to JATS XML," *Eur. Sci. Ed.*, vol. 49, Art. no. e114977, pp. 1–7, 2023, doi: 10.3897/ese.2023.e114977.
- [27] L. Jertec Musap, D. Celjak, P. Čačić, and I. Jelenić, "Enhancing machine-readability in scientific publishing: Introducing the JATS XML Converter Service," *OPERAS Innovation Lab*, Nov. 21, 2024. Accessed: Dec. 5, 2025. [Online]. Available: <https://lab.operas-eu.org/2024/11/21/enhancing-machine-readability-in-scientific-publishing-introducing-the-jats-xml-converter-service/>
- [28] V. Pereira, M. P. Basilio, C. Henrique, and T. Santos, "pyBibX: A Python library for bibliometric and scientometric analysis powered with artificial intelligence tools," *arXiv preprint arXiv:2304.14516*, 2023, doi: 10.48550/arXiv.2304.14516.
- [29] M. I. Rizqi, M. Z. Abdullah, and M. A. Hendrawan, "The Development of Meta data Extractor Plugin for Open Journal System," in *Proceedings of the International Conference on Applied Science and Technology on Engineering Science 2023 (iCAST-ES 2023)*, pp. 985–991, 2024, doi: 10.2991/978-94-6463-364-1_90.

BIOGRAPHIES OF AUTHORS

	Karrthik Nadarajan is a final-year student pursuing his Bachelor of Computer Science (Hons) in Data Science at Multimedia University. His research interests focus on data mining, machine learning, and automated document generation workflows. He can be contacted at email: 1211111427@student.mmu.edu.my or karrthiknadarajan123@gmail.com.
	Jiaheng Lu is a computer scientist and a teacher, with a research interest in databases and data management. His recent topics include multi-model database management systems, quantum computing for databases and job optimization for big data platform. He can be reached at jiaheng.lu@helsinki.fi