WRITEWIZARD - COLLABORATIVE DOCUMENT EDITING TOOL: REAL-TIME MULTI-FUNCTIONAL PLATFORM

(AI-BASED IEEE DOCUMENT FORMATING)

Shandeep. J

(IT21375682)

B.Sc. (Hons) in Information Technology

Specializing in Software Engineering

Department of Computer Science & Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

WRITEWIZARD - COLLABORATIVE DOCUMENT EDITING TOOL: REAL-TIME MULTI-FUNCTIONAL PLATFORM

(AI- BASED IEEE DOCUMENT FORMATING)

Shandeep. J

(IT21375682)

Dissertation submitted in partial fulfillment of the requirements for the Bachelor of Science (Hons) in Software Engineering

Department of Computer Science & Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

April 2025

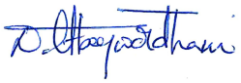
DECLARATION

I declare that this is my own work, and this Thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the nonexclusive right to reproduce and distribute my Thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Shandeep. J	IT21375682	

The above candidate has carried out this research thesis for the Degree of Bachelor of Science (honors) Information Technology (Specializing in Software Engineering) under my supervision.



Signature of the supervisor

(Dr. Lakmini Abeywardhana) Date: 11.04.2024

Signature of co-supervisor

(Ms. Karthiga Rajendran) Date: 11.04.2024

ABSTRACT

This research presents an AI-based IEEE document format that automatically converts informal academic drafts into structured, publication-ready documents. Leveraging transformer-based models fine-tuned with techniques such as 4-bit quantization and Low-Rank Adaptation, the system corrects grammatical errors, enforces a formal tone, and reorganizes content according to IEEE standards. A key feature is the intelligent citation management module, which removes citation markers before processing and reinserts them using semantic matching, thereby preserving scholarly references with minimal manual effort. The formatter also incorporates non-text elements—including images, tables, and formulas—ensuring that these components remain in the correct order throughout the final document. The system employs dual-backend architecture. A FastAPI service manages core AI-driven text conversion, while an Express backend facilitates document pad creation and collaboration. A WebSocket-enabled real-time collaborative editor allows multiple users to work simultaneously on a document, with changes synchronized instantly across sessions. Users can select specific text portions, toggle AI enhancement on or off for individual sections, and preview the refined text before replacement. Once content refinement is complete, the backend aggregates all sections, applies a LaTeX template, and compiles the final IEEE-compliant PDF.Experimental evaluations demonstrate that this integrated approach significantly streamlines the academic writing process while delivering high-quality output. Overall, this system provides an efficient, user-friendly tool that enhances productivity, accuracy, and collaboration in academic document preparation.

Keywords: AI Document Conversion, IEEE Formatting, Academic Writing, Transformer Models, Citation Management, Collaborative Editing, WebSocket, Fine-tuning, Latex, Collaborative editing

ACKNOWLEDGEMENT

I would like to sincerely thank my supervisor, Dr. Lakmini Abeywardhana, and my sub-supervisor, Ms. Karthiga Rajendran, for their continuous guidance, support, and constructive feedback throughout the course of this research. Their expertise and direction were instrumental in shaping the scope and execution of this work. I am also grateful to the academic and technical staff of the Department of Information Technology, Faculty of Computing, Sri Lanka Institute of Information Technology, for their academic input and access to necessary resources. I extend my appreciation to my fellow project members for their collaboration in the development of the WriteWizard platform, and to the students and peers who contributed their time and feedback during evaluation phases. This research was carried out as part of the Bachelor of Science (Honours) in Information Technology degree program and was not supported by external funding. Finally, I would like to thank my family and friends for their moral support, patience, and encouragement throughout my academic journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Description
SLIIT	Sri Lanka Institute of Information Technology
IEEE	Institute of Electrical and Electronics Engineers
ML	Machine Learning
NLP	Natural Language Processing
MS	Microsoft
LoRA	Low-Rank Adaptation
API	Application Programming Interface
SSL	Secure Sockets Layer
VM	Virtual Machine
Azure	Microsoft Azure Cloud Platform
WBS	Work Breakdown Structure
SDLC	Software Development Life Cycle
OS	Operating System
HTML	Hyper Text Markup Language
JS	Java Script
CSS	Cascading Style Sheet

1. INTRODUCTION

1.1 Background Study and Literature Review

1.1.1 Background Study

Academic writing is a cornerstone of scholarly communication and plays a vital role in disseminating research findings. Traditionally, the process of drafting, revising, and formatting academic papers has been both labor-intensive and time-consuming, as it requires meticulous attention to detail to adhere to strict guidelines such as those established by IEEE. Researchers must manually edit text, format citations, and ensure logical coherence across sections—all of which are prone to human error and inefficiency.

The advent of digital technology has ushered in AI-powered writing assistants that can significantly ease the burden of language refinement and basic editing. However, many of these tools are designed for general-purpose language tasks, focusing primarily on grammar, syntax, and readability improvements. They often fall short when it comes to enforcing the specialized requirements of academic writing, such as maintaining a formal tone, adhering to structural conventions, and managing citations in a standardized manner.

Recent advancements in natural language processing have introduced large language models (LLMs) like GPT-4, BERT, and T5. These models have demonstrated remarkable capabilities across a variety of NLP tasks and have the potential to transform academic writing. Despite their general proficiency, these models require further fine-tuning to generate text that meets the stringent standards of academic publications.[1] Fine-tuning models such as Llama 2-7B using techniques like quantization and LoRA (Low-Rank A5daptation) allows them to be optimized for the specific task of converting informal or unstructured text into well-organized, IEEE-compliant academic content.[2] This adaptation not only enhances their performance in producing a formal tone and technical accuracy but also improves their efficiency in resource-constrained environments.

In addition to language generation, another significant challenge lies in the realm of document formatting. LaTeX remains the gold standard for typesetting academic papers, especially in fields that require the precise representation of mathematical formulas, complex tables, and rigorous citation formatting. While platforms like Overleaf and ShareLaTeX have made LaTeX more accessible by providing user-friendly interfaces and pre-designed templates, these tools still require a certain level of manual intervention. Researchers who are not well-versed in LaTeX syntax often struggle to achieve the desired level of professionalism in their documents, leading to inefficiencies and potential errors in the final manuscript.

This research addresses the dual challenges of language refinement and document formatting by proposing an integrated, end-to-end system that automates the transformation of informal text into structured, IEEE-compliant academic writing. By combining a fine-tuned large language model with an automated LaTeX formatting pipeline [1], the proposed system is designed to streamline the academic writing process. It not only corrects and enhances the text to meet scholarly standards but also formats the output in accordance with IEEE guidelines, including proper citation management and structural organization.

The resulting solution promises to significantly reduce the manual effort involved in preparing academic documents, thereby enabling researchers to focus more on content creation and less on the intricacies of formatting. This integrated approach has the potential to revolutionize academic writing by democratizing access to advanced writing tools, enhancing both productivity and the overall quality of research outputs in the digital age.

1.1.2 Literature Review

Academic writing and research paper formatting have historically required meticulous manual effort, with authors laboring over details to ensure compliance with standards like IEEE. Traditional methods involve manual drafting, rigorous proofreading, and painstaking citation management, which, while thorough, are time-consuming and prone to human error. As academic output has grown, so too has the demand for tools that can expedite these processes without sacrificing quality.

In recent years, the emergence of AI-driven writing assistants has provided promising avenues to address these challenges. Tools such as Grammarly have gained popularity for their ability to improve grammar, style, and overall readability. However, despite their usefulness in refining language, such tools generally lack the capacity to enforce the strict structural and stylistic requirements demanded by academic writing. They do not adequately address the nuanced needs of IEEE-compliant formatting, such as precise citation placement and formal tone, leaving researchers to manually polish their documents.

Large language models (LLMs) like GPT-4, BERT, and T5 have revolutionized natural language processing by generating coherent and contextually relevant text. Yet, these models are typically optimized for a broad range of tasks rather than the specialized demands of scholarly communication. To bridge this gap, the Llama 2-7B model has been fine-tuned to transform informal or unstructured text into formal academic prose. Through techniques such as quantization and Low-Rank Adaptation (LoRA), this adaptation not only reduces the model's memory footprint but also enhances its performance in resource-constrained environments, ensuring technical accuracy and a formal tone.[1]

In parallel, LaTeX has emerged as the de facto standard for producing professional academic documents due to its powerful handling of complex structures such as equations, tables, and bibliographies. Platforms like Overleaf simplify LaTeX usage by providing ready-made templates and collaborative features; however, they still require users to understand LaTeX syntax—a significant hurdle for many researchers.

Moreover, while specialized academic writing tools provide valuable feedback on tone and structure, they function largely as isolated systems that assist in content refinement but do not automate the complete workflow from text generation to final document formatting.[4]

In summary, while numerous tools address fragments of the academic writing process, there remains a clear need for an integrated system. Such a system would automatically convert informal text into a well-structured, IEEE-compliant academic paper, incorporating advanced language modeling and automated LaTeX formatting alongside robust citation management. This comprehensive approach promises to reduce the manual burden on researchers and enhance the overall quality and consistency of scholarly publications.

1.2 Research Gap

Despite the proliferation of AI-driven tools and platforms aimed at assisting academic writing, a substantial gap persists in providing an end-to-end solution that covers the entire spectrum of tasks—from language refinement and structural organization to final formatting and citation management. Existing solutions like Grammarly and general-purpose language models excel in improving readability and correcting grammatical errors, but they are not tailored to enforce the strict academic tone and formatting conventions demanded by IEEE publications. As a result, researchers still face significant manual effort to align their work with rigorous scholarly standards.

Moreover, while Overleaf and similar LaTeX editors offer robust document formatting capabilities, they require users to have prior knowledge of LaTeX—a barrier for many who may be experts in their field but are not familiar with complex typesetting systems. This disconnect often forces researchers to juggle multiple tools: one for text generation and another for document formatting, leading to an inefficient and fragmented workflow.

Specialized academic writing assistants, such as Writefull, have been developed to provide targeted feedback on academic tone, clarity, and structure. However, these systems generally operate in isolation, offering revision suggestions without integrating automated formatting or citation management. This means that while they can enhance the quality of the language, they do not bridge the gap between draft and publication-ready document, leaving the final assembly process largely manual.

In addition to the challenges with language and formatting, citation management remains a persistent hurdle. Tools like EndNote and Mendeley efficiently organize references and generate citations, yet they focus solely on bibliographic data and do not contribute to the overall narrative or formatting of the document. The re-insertion

and correct placement of citations into a seamlessly formatted academic text is an area that existing solutions do not adequately address.

Tool	Collaborative Editing	Format Conversion	Text Refinement	Rich Text Edit	User Friendly
LaTeX	✗	✓	✗	✗	✗
EndNote 20	✗	✓	✗	✗	✗
MENDELEY	✗	✗	✗	✓	✗
Zotero	✗	✗	✗	✓	✗
Overleaf	✗	✓	✗	✗	✗
Writewizard	✓	✓	✓	✓	✓

Figure 1:Comparison diagram with existing research writing tools

Furthermore, while large language models have demonstrated significant promise in generating coherent and contextually appropriate text, their default configurations are not optimized for academic writing. The fine-tuning of models such as Llama 2-7B—using advanced techniques like quantization and LoRA—has shown potential in producing IEEE-compliant academic prose, but such specialized adaptations remain underexplored and are not widely integrated into existing academic writing tools

In light of these shortcomings, the research gap is evident: there is a need for a comprehensive, integrated system that automates the conversion of informal text into structured, IEEE-compliant academic writing. Such a system should merge advanced language transformation capabilities with automated LaTeX formatting and intelligent citation management, thus streamlining the entire academic writing process. Addressing this gap would not only enhance productivity but also improve the overall quality and consistency of scholarly publications.[3]

1.3 Research Problem

Despite significant advances in natural language processing and document formatting tools, a persistent gap remains in automating the end-to-end process of academic writing. Traditional academic writing involves a complex, multi-step process that includes drafting content, refining language, structuring text according to strict guidelines (such as those mandated by IEEE), and managing citations. Researchers are often forced to juggle multiple specialized tools—such as grammar checkers, formatting editors, and reference management systems—which not only increases the manual workload but also introduces potential errors and inconsistencies in the final document.

Existing tools like Grammarly excel in improving grammar and readability, yet they fall short of enforcing the formal tone and structured organization required for scholarly publications. While some document formatting platforms offer robust formatting capabilities, they typically demand a steep learning curve; users must manually adjust document structures and adhere to detailed formatting standards, a process that is both time-consuming and error prone. Similarly, reference management systems like EndNote, Mendeley, Zotero, and RefWorks efficiently handle citations but are limited to bibliographic management and do not integrate with the broader workflow of converting informal text into a well-organized academic narrative.

Current large language models, though powerful in generating coherent text, are generally optimized for broad applications and require extensive fine-tuning to produce content that meets the rigorous demands of academic writing. Without proper adaptation, these models may generate output that is grammatically correct yet lacks the formal tone, logical structure, and precise formatting necessary for academic publications. Furthermore, the fragmented nature of existing solutions means that significant manual effort is still required to merge and refine outputs from various systems, hampering efficiency and potentially compromising the quality of the final document.

Thus, the research problem is to develop an integrated, end-to-end academic writing solution that transforms informal or unstructured text into fully formatted, IEEE-compliant academic documents. This solution must combine advanced language transformation capabilities—leveraging fine-tuned large language models—with automated formatting and intelligent citation management. Moreover, it should operate within a collaborative, real-time environment to reduce manual intervention, ensuring that the output adheres strictly to academic standards. Addressing this problem will not only streamline the academic writing process but also significantly enhance the consistency, quality, and efficiency of scholarly publications.

1.4 Research Objectives

1.4.1 Main Objective

The main objective of this research is to develop an automated academic writing solution that converts informal text into a polished, IEEE-compliant academic document. This system will harness fine-tuned large language models to generate text with a formal tone and logical structure while automatically managing citations through semantic matching. By integrating these capabilities, the proposed system aims to reduce the manual workload on researchers and improve the consistency, quality, and efficiency of scholarly publications.

1.4.2 Specific Objectives

The following are the sub-objectives of conducting this research.

- **Fine-tune Language Models**: Adapt large language models to the specific task of academic text conversion, ensuring that the output meets the formal tone and structural requirements of scholarly writing.
- **Automated Text Conversion Pipeline**: Develop a seamless workflow that integrates language refinement, structural organization, and intelligent citation management to convert informal text into structured academic content.
- **Real-Time Collaborative Editing:** Implement collaborative editing features that allow multiple users to work simultaneously on the same document, thereby increasing productivity and reducing revision cycles.
- **Intelligent Citation Management:** Incorporate advanced semantic matching techniques to automatically extract, remove, and reinsert citation markers into the converted text, ensuring accurate and consistent reference management.
- **Performance Evaluation:** Conduct comprehensive evaluations through user studies and benchmark comparisons to assess improvements in efficiency, quality, and consistency compared to existing tools.
- **Optimization for Deployment:** Ensure that the system is optimized for resource-constrained environments, facilitating scalability and practical usability across

diverse academic settings

1.4.3 Business Objectives

- **Enhance Research Productivity:** Provide a tool that significantly reduces the time and effort required to produce ready-to-ready documents, thereby enabling researchers to focus more on core research activities.
- **Market Differentiation:** Develop a comprehensive, user-friendly solution that combines text conversion, automated formatting, and citation management into one platform, distinguishing it from existing standalone tools.
- **Revenue Generation:** Establish a viable business model through subscription-based pricing, catering to individual scholars, academic institutions, and research organizations.
- **Strategic Partnerships:** Foster collaborations with academic publishers, research institutions, and educational technology providers to extend market reach and continually refine product features.
- **Competitive Advantage:** Position as an indispensable asset for academic writing and research, ultimately leading to increased adoption and a strong competitive edge in the academic technology market.
- **Continuous Innovation:** Utilize user feedback and emerging technologies to drive ongoing product improvement, ensuring that the solution remains at the forefront of academic writing automation in a rapidly evolving digital landscape.

2. METODOLOGY

2.1 Methodology

The methodology for this research is structured into several key phases: requirement gathering, model selection and fine-tuning, system design, implementation, testing, deployment, and maintenance. During the requirement gathering phase, structured interviews with academic writers and experts helped define both functional and non-functional requirements. Next, a comprehensive dataset was assembled by pairing informal academic text with its IEEE-compliant equivalent, which served as the basis for fine-tuning transformer models. The design phase focused on creating a modular, scalable architecture incorporating both a FastAPI service for AI-driven text conversion and an Express backend for collaborative pad management. The implementation included real-time editing features via WebSockets, which allowed multiple users to work synchronously on a single document. Subsequent testing combined unit, integration, and manual testing to ensure high conversion quality and robust citation management, while deployment was carried out on an Azure Virtual Machine. Overall, the Agile methodology facilitated iterative improvements and rapid feedback loops throughout the project lifecycle, ensuring a responsive system that meets evolving user needs.

1. Agile Project Management for Research Development

Agile methodology serves as the foundation for managing project progress. By breaking down the research into manageable sprints, the development team can focus on implementing specific features—such as advanced language model adaptation for text conversion, intelligent citation management, and real-time collaborative editing. Regular planning sessions, sprint reviews, and progress evaluations ensure that the system remains aligned with the overall research objectives and that any emerging challenges are addressed promptly.

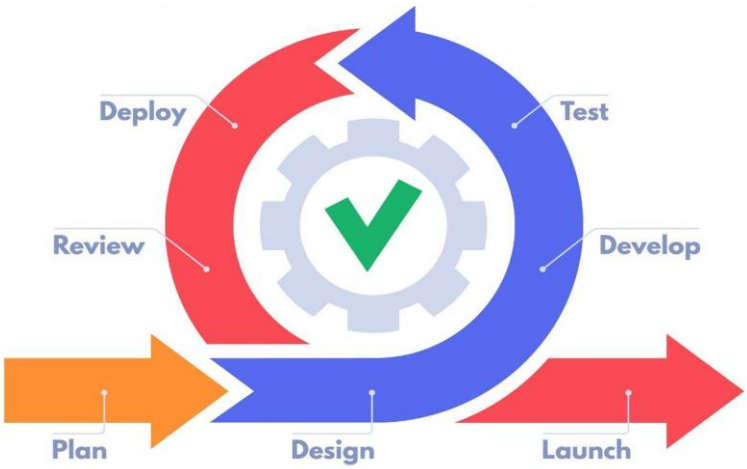


Figure 2: Agile Project Management Cycle

2. Multi-Phase Development Process

The research is organized into several sequential phases to comprehensively address the project's requirements:

- **Requirement Analysis:** This phase involves gathering and analyzing requirements specific to academic writing conversion, including the transformation of informal text into formal academic prose and the integration of robust citation management.
- **Conceptual Design:** A detailed conceptual model is developed, outlining the workflow for text conversion and semantic citation extraction. This design maps out the integration of fine-tuned language models and automated formatting strategies.
- **System Development:** In this phase, the system is implemented using fine-tuned large language models enhanced with techniques like quantization and Low-Rank Adaptation (LoRA). These adaptations ensure that the model generates text with the necessary formal tone and structural coherence. Collaborative editing features are also integrated to enable real-time, multi-user interaction.
- **Testing and Evaluation:** Rigorous testing is conducted through both automated unit tests and user acceptance trials. This ensures that each module

performs reliably, and that the overall system produces high-quality, publication-ready academic text.

- **Deployment and Maintenance:** The final system is deployed in a real-world environment, and protocols are established for continuous monitoring, user feedback, and iterative improvement.

3. Iterative Progress and Team Collaboration

A key advantage of the Agile approach is its emphasis on iterative progress and robust team collaboration. The project is divided into discrete sprints, each focusing on specific functionalities such as language conversion, citation management, or real-time collaboration. Regular team meetings and feedback sessions enable the integration of insights gained in one phase into the subsequent phases, ensuring a cohesive and well-integrated final product.

4. Flexibility and Adaptability

The dynamic nature of academic writing requirements necessitates a flexible development process. Agile methodology allows the research team to quickly adapt to changes in user requirements or emerging technological advancements. This flexibility ensures that the system remains robust and can be continuously refined to meet the evolving needs of academic researchers.

2.1.1 Feasibility Study/ Planning

This phase determines the feasibility of implementing the proposed academic writing conversion system across several dimensions:

2.1.1.1. Technical Feasibility:

Data Availability: The system will rely on the availability of large-scale textual content sourced from academic documents and collaborative inputs. A thorough evaluation has been conducted to confirm that a diverse range of informal and semi-structured academic texts is accessible for training and testing purposes. Furthermore, the possibility of manually collecting and annotating local academic data has been assessed, ensuring that there is sufficient high-quality data available to effectively fine-tune the language models

Hardware and Software Requirements: The project requires hardware that can efficiently process extensive text data and support advanced natural language processing tasks. This includes access to modern computing environments with GPU acceleration, which is critical for both the fine-tuning and inference stages of the language model. On the software side, the system leverages open-source libraries and frameworks such as HuggingFace Transformers and PyTorch, ensuring robust support for advanced NLP capabilities. These tools provide the necessary infrastructure to implement and run fine-tuned models efficiently, while also allowing seamless integration with collaborative document editing platforms for a real-time user experience.

Integration Feasibility: The technical design has been carefully evaluated to ensure that the automated text conversion, semantic structure preservation, and intelligent citation management components can be cohesively implemented within a single system. The architecture is developed to support real-time processing and to handle dynamic inputs from multiple sources, making it compatible with both cloud-based and local deployment environments. Integration with existing collaborative platforms has been prioritized, allowing researchers to update and refine their documents

continuously. This ensures that the system not only generates IEEE-compliant academic text from informal inputs but also maintains high performance and reliability across different operational scenarios.

2.1.1.2. Economic Feasibility:

Budgetary Considerations: The financial feasibility of the proposed academic writing conversion system has been assessed by estimating the costs associated with system development, cloud hosting, and user testing. The strategic use of open-source tools significantly minimizes licensing expenses, while scalable cloud services ensure that the necessary computing resources are available on demand. This approach keeps operational costs low and allows the project to leverage advanced technology without a prohibitive financial burden.

Resource Allocation: An evaluation of available research personnel confirms that the expertise in natural language processing, web development, and data visualization is readily accessible. The research team possesses the requisite skills to support both the development and testing phases of the project, ensuring that technical challenges are addressed promptly, and that the system can be iteratively refined based on user feedback.

Return on Investment: The potential benefits of the proposed system are substantial. By automating the conversion of informal text into structured, IEEE-compliant academic documents, the system is expected to significantly improve research productivity and collaboration. Enhanced efficiency in academic writing and reduced manual workload are anticipated to result in time savings and increased quality of scholarly publications. These improvements justify the financial investment, as the return in terms of increased output and reduced editing effort is expected to be considerable.

Table 1:Summarizes of estimated expenses

Type	Cost
Internet use and cloud hosting	12000 LKR
Publication costs	45000 LKR
Stationary	7500 LKR
TOTAL	***62500 LKR***

2.1.1.3. Legal and Ethical Feasibility:

Data Privacy and Compliance: The system will process large volumes of textual data obtained from academic documents and collaborative sources. Comprehensive measures have been evaluated to ensure that all data collection and processing activities comply with relevant copyright laws and data protection regulations. The project will implement informed consent procedures to secure permission for data usage and apply data anonymization techniques to protect any sensitive information. These steps are crucial to maintain user trust and uphold ethical standards in the handling of academic and collaborative data.

Intellectual Property Considerations: This project is built using open-source libraries and publicly available datasets, ensuring that all software components adhere to proper licensing protocols. By leveraging well-established open-source tools, the system minimizes potential legal risks and supports ethical usage of third-party content. Furthermore, clear documentation of all utilized resources and adherence to licensing requirements will guarantee that intellectual property rights are respected. These practices not only mitigate legal risks but also foster transparency and accountability in the integration of diverse technologies into the system.

2.1.1.4. Operational and User Acceptance Feasibility:

The operational feasibility of the proposed academic writing conversion system is critical to ensuring its practical deployment and acceptance within the academic community. This system is designed to streamline the transformation of informal text into structured, IEEE-compliant academic documents by integrating advanced language modeling and intelligent citation management within a collaborative framework.

Data Collection and Processing: The system is engineered to efficiently gather and process textual content from diverse academic sources. A robust data pipeline has been developed to accurately extract unstructured or informal text, clean and standardize it, and transform it into a structured format suitable for academic conversion. Extensive testing under various conditions confirms that this pipeline can handle large volumes of text reliably, ensuring that inputs are consistently prepared for subsequent conversion.

Model Deployment: The deployment strategy for the academic writing conversion system has been thoroughly evaluated to ensure its reliability in varied operational environments. The system processes input text in real time, generating output that conforms to IEEE academic standards. Continuous monitoring mechanisms are incorporated to track performance and quickly resolve any issues that may arise, ensuring that the system remains robust and responsive even under heavy workloads. This proactive approach to deployment supports the long-term stability and efficiency of the solution.

System Integration: A key advantage of the proposed solution is its seamless integration with existing collaborative document editing and content management platforms. Designed as an add-on, the system allows researchers to import their informal drafts and instantly receive automatically converted academic text without disrupting their established workflows. This integration is critical to ensuring that the system complements rather than complicates current academic practices. Early

operational tests demonstrate that the system processes and converts text in real time, maintaining a smooth and uninterrupted user experience.

User Interface and Experience: The user interface is designed to be intuitive and accessible, catering to researchers with varying levels of technical expertise. It allows users to easily input text and preview the converted output, while also enabling manual adjustments when needed. A notable feature is the option to toggle AI enhancement on or off for each section. This functionality allows users to choose between applying the advanced language model conversion—which transforms informal text into a structured academic format—and directly obtaining a document formatted according to IEEE standards. Such flexibility enables users to tailor the output to their specific requirements, balancing automated refinement with their own expertise.

Dynamic customization options further enhance the user experience by providing adjustable. Pilot tests with sample users have provided positive feedback, with researchers emphasizing the system's ease of use, efficiency, and the high quality of the generated academic content. This favorable user experience, coupled with the ability to switch AI enhancement on or off based on individual needs, is expected to drive widespread adoption and validate the overall operational feasibility of the solution.

2.1.1.5. Time and Schedule Feasibility:

Project Timeline: A comprehensive schedule has been developed that outline key milestones across all phases of the project, including requirements analysis, system design, iterative development, testing, and deployment. Each phase is designed to build upon the insights gained from the previous phase, ensuring a smooth progression toward the final deliverable. The timeline is structured to allow sufficient time for both planned development activities and iterative refinements, which are essential given the dynamic nature of the system and the need for continuous improvement. Regular

milestones and sprint reviews are incorporated into the schedule to provide clear checkpoints for evaluating progress and making necessary adjustments.

Monitoring and Contingencies: Agile management practices are in place to facilitate continuous monitoring of project milestones. Regular progress reviews ensure that any deviations from the planned schedule are identified early. In addition, contingency plans have been established to address unforeseen delays or challenges. These measures include flexible resource allocation and adjusted sprint timelines, which together guarantee that the project remains on track and is completed within the allocated time frame. This proactive approach to time management ensures that the system can be developed, tested, and deployed efficiently while adapting to any emerging requirements or obstacles. Figure 2 illustrates the Schedule Management plan.

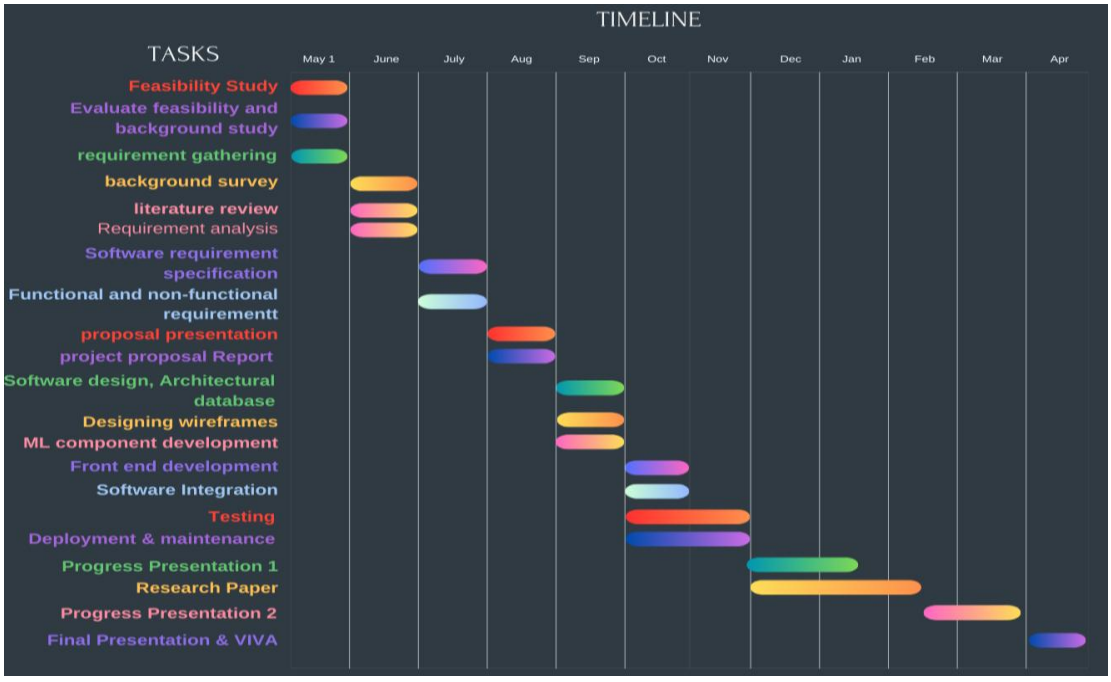


Figure 2: Gantt Chart (Schedule Management)

2.1.1.6. Social and Cultural Feasibility:

The proposed academic writing conversion system is designed to address the diverse needs of the global academic community by ensuring robust performance and flexibility in processing extensive textual data. The system's technical architecture has been rigorously evaluated to confirm that it can handle large volumes of input from varied academic sources, providing real-time, high-quality conversion of informal text into structured, IEEE-compliant documents without performance degradation. This reliability is crucial for academic environments where consistent output under high user loads is essential.

Adaptability to User Needs: One of the core strengths of the system lies in its dynamic customization features. Users can tailor the conversion process according to their specific preferences by toggling AI enhancement on or off for each section. This option allows users to choose between fully automated text conversion or receiving a direct IEEE-formatted document, catering to both novice researchers and seasoned academics. Such flexibility ensures that the system remains relevant and useful across a range of disciplines and individual writing styles, promoting broader adoption among diverse user groups.

Scalability and Deployment: The system is engineered to be highly scalable, supporting both cloud-based and local deployment models. This dual deployment capability guarantees that the solution is accessible to a wide spectrum of users—from large academic institutions with significant computational resources to individual researchers working in more resource-constrained environments. The adaptability of the deployment model ensures that the system can efficiently process large datasets and high volumes of user requests without compromising on performance, thereby aligning with diverse cultural and educational practices around the world.

User-Centric Design: Emphasizing an intuitive and accessible user interface, the system is designed to meet the expectations of a culturally diverse academic audience. The interface supports real-time collaboration and customization, ensuring that researchers can easily integrate the system into their existing workflows. Feedback

from pilot testing has indicated a high level of user satisfaction, with researchers praising the ease of use and the significant reduction in manual formatting and citation management efforts. This positive reception is anticipated to facilitate widespread acceptance across various academic communities.

Other than these feasibility studies, the risk management plan and communication management plan have been done.

2.1.1.7. Risk Management Plan:

The following risk management plan identifies potential threats to the successful development and deployment of the automated IEEE document formatting system. It outlines the triggers, assigns responsibility, and proposes contingency strategies to mitigate each risk. Table 2 summarizes the key risks, triggers, responsible owners, responses, and required resources.

Table 2:Risk Management Plan

Risk	**Trigger**	**Owner**	**Response**	**Resource Required**
Project Team Unavailability	Illness or sudden absence of team members	Project Leader	Inform supervisors; redistribute tasks among remaining team members; activate backup personnel plan	Project Schedule Plan; Backup Resources
Stakeholder Revision Requests	Panel or supervisor requests changes	Project Leader	Implement changes promptly; update documentation and project backlog;	Meeting Log; Updated Product Backlog

			communicate revisions clearly	
Integration Challenges	Incompatibility between system components or platforms	Project Leader	Reassess integration strategy; conduct additional integration tests; consult with platform experts	Integration Testing Framework; Technical Support
Data Quality and Availability	Insufficient or low-quality data during collection	Project Leader	Identify supplementary data sources; employ data cleaning procedures; schedule further collection sessions	Data Quality Assurance Plan
Performance and Scalability Issues	System slowdown under heavy data or user load	Project Leader	Optimize code and architecture; implement scalable cloud solutions; conduct stress testing	Cloud Services; Performance Testing Tools
Deployment Delays	Technical issues or scheduling conflicts during deployment	Project Leader	Reassess timeline; use backup deployment environments; escalate critical issues to management	Gantt Chart; Deployment Backup Plan

Legal and Ethical Compliance Risks	Non-compliance with copyright or data protection regulations	Project Leader	Conduct regular legal reviews; ensure secure data handling; obtain necessary consents	Legal Consultation; Compliance Guidelines

2.1.1.8. Communication Management Plan:

The Communication Management Plan ensures that all team members, supervisor, and co-supervisor receive timely and relevant information to effectively fulfil their roles throughout the project. The plan outlines the communication objectives, the channels through which communication will occur, and the specific meeting arrangements designed to foster clarity and prompt decision-making.

Communication Objectives:

- Communication must be proactive, ensuring that all updates are provided in the right format and with appropriate content for the targeted audience.
- Communication should be sufficient, delivering all necessary information in a clear and concise manner while avoiding unnecessary repetition.
- Communications must be timely to address issues as they arise and maintain project momentum.

Communication Media:

1 Email – For official updates, documentation sharing, and formal communication with supervisors and stakeholders.
2 MS Teams – For real-time meetings, collaborative discussions, and file sharing within the team.
3 On-Campus Meetings – Face-to-face interactions are arranged as needed, particularly for meetings with the supervisor and co-supervisor. These sessions

are used to discuss critical project milestones, resolve technical challenges, and provide formal feedback in a direct setting.

4 Phone Calls – For urgent matters and direct communication among team members and supervisors.

5 WhatsApp – For quick updates, informal discussions, and immediate coordination.

6 MS Planner – For task management, scheduling, and tracking progress across the different phases of the project.)

Table 3: Communication Management Plan

Meeting Type	**Attendees**	**Purpose**	**Frequency**	**Agenda Items**
Planning Kick-off Meeting	Supervisor, Co-supervisor, All Team Members	Formally launch the planning phase; define project scope, novelties, roles, and expectations.	Once at project inception.	Establish planning timetable, project scope, governance structure, roles, risk assessment, and review initial communication strategies.
Execution Kick-off Meeting	Supervisor, Co-supervisor, All Team Members	Initiate the execution phase; ensure all stakeholders understand their responsibilities.	At the start of major project phases	Present the detailed project work plan, review meeting logs, agree on dispute resolution and escalation methods, and outline quality assurance and change management.
Internal Status Meeting	All Team	Review ongoing progress, discuss	Weekly	Provide progress updates, review

Meeting Type	Attendees	Purpose	Frequency	Agenda Items
	Members	developments, and address emerging issues.		accomplishments, discuss new risks or issues, confirm milestones, and address any modifications needed to tasks.
Supervisor Review Meeting	Supervisor, Co-supervisor, All Team Members	Monitor project performance and assess major adjustments in scope or strategy.	Twice a week or as needed	Evaluate progress against the work plan, discuss scope or methodology changes, review budget and resource usage, and update risk management and quality control.
Project Steering Committee Meeting	All Team Members	Discuss overall project status and receive formal approvals at key milestones.	Once a month or at major milestones	Review project status, discuss unresolved issues, update risk and issue management, review budget and work history, and obtain official endorsements.
Pre-Milestone Review Meeting	Supervisor, Co-superv	Hold face-to-face discussions with supervisors prior to key milestones to	Prior to each major milestone (Proposal,	Review progress, discuss critical technical issues, refine project scope, and

Meeting Type	Attendees	Purpose	Frequency	Agenda Items
	isor, All Team Members	clarify expectations and technical challenges.	PP1, PP2, Research Paper Submission)	confirm upcoming deliverables.
Change Control Meeting	Supervisor, Co-supervisor, All Team Members	Prioritize and discuss change requests or panel inquiries.	As needed following major reviews (Proposal, PP1, PP2)	Review proposed changes, discuss panel feedback, agree on modifications to implement, update project documentation, and assign new tasks if required.
Project-End Review Meeting	Supervisor, Co-supervisor, All Team Members	Evaluate overall project performance and document lessons learned.	Once at project completion or major phases	Assess key accomplishments, review overall performance, discuss issues encountered and resolutions, summarize lessons learned, and propose future improvements.

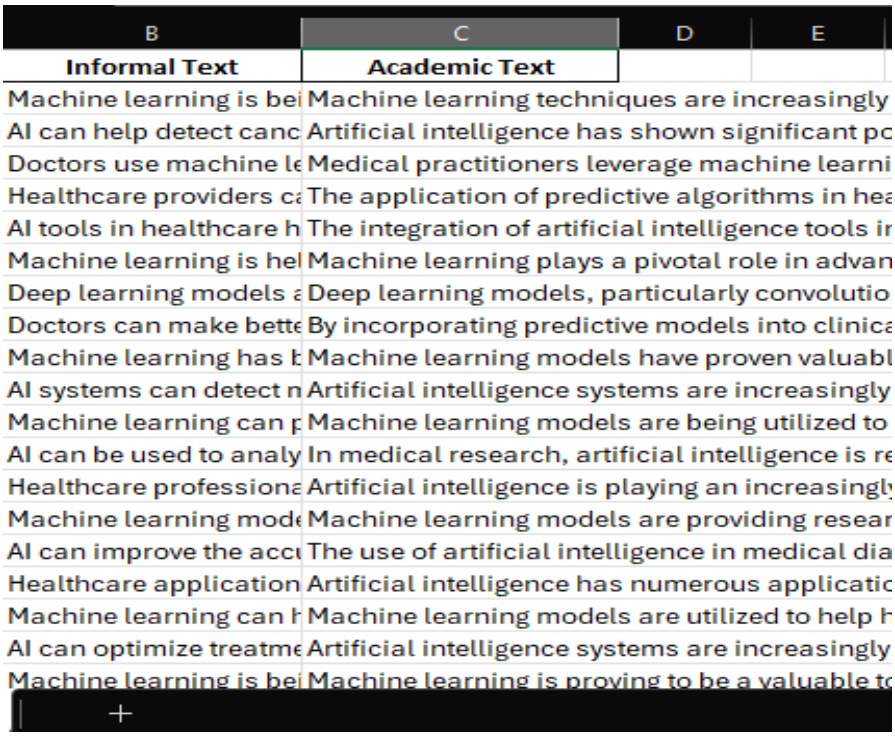
2.1.2 Requirement Gathering & Analysis

The Requirement Gathering and Analysis phase was a pivotal step in defining the project's scope, objectives, and constraints. During this phase, both functional and non-functional requirements were systematically identified to ensure that the automated academic writing conversion system aligns with user needs and adheres to established quality standards.

2.1.2.1. Functional Requirements

Functional requirements specify the features and capabilities that the automated academic writing conversion system must possess to meet its intended goals. For this research, these requirements were derived from structured interviews with researchers, academic editors, and experts in collaborative writing. The objective is to ensure that the system effectively transforms informal or unstructured text into structured, IEEE-compliant academic writing while preserving meaning and scholarly rigor. The key functional requirements for the project are as follows:

Automated Text Conversion: The system must automatically refine and restructure informal text to produce formal academic writing. This involves correcting grammar, enhancing clarity, enforcing a formal tone, and ensuring logical organization throughout the document. For this research, a manually curated dataset was developed by pairing text from academic documents with corresponding academic-formatted text snippets. This dataset serves as the basis for fine-tuning the model that converts informal document content into formatted academic writing. The resulting system processes the provided text and outputs academic writing that preserves the overall flow and logical connections within each section. (Figure 4 illustrates a sample manual dataset created during the initial data collection phase.).



B	C	D	E
Informal Text	Academic Text		
Machine learning is bei	Machine learning techniques are increasingly		
AI can help detect canc	Artificial intelligence has shown significant pc		
Doctors use machine le	Medical practitioners leverage machine learni		
Healthcare providers ca	The application of predictive algorithms in hea		
AI tools in healthcare h	The integration of artificial intelligence tools ir		
Machine learning is hel	Machine learning plays a pivotal role in advan		
Deep learning models a	Deep learning models, particularly convolutio		
Doctors can make bette	By incorporating predictive models into clinica		
Machine learning has b	Machine learning models have proven valuabl		
AI systems can detect n	Artificial intelligence systems are increasingly		
Machine learning can p	Machine learning models are being utilized to		
AI can be used to analy	In medical research, artificial intelligence is re		
Healthcare professiona	Artificial intelligence is playing an increasingly		
Machine learning mode	Machine learning models are providing resear		
AI can improve the accu	The use of artificial intelligence in medical dia		
Healthcare application	Artificial intelligence has numerous applicatic		
Machine learning can h	Machine learning models are utilized to help h		
AI can optimize treatme	Artificial intelligence systems are increasingly		
Machine learning is bei	Machine learning is proving to be a valuable to		

Figure 3: Pre-processed Dataset

Selective AI Enhancement Toggle: Users should be able to toggle AI enhancement on or off for each section. This feature allows users to choose between applying advanced AI-driven text refinement and directly obtaining an IEEE-formatted document.

Real-Time Processing and Collaborative Editing: The system must process input text in real time and dynamically update the refined output. It should also support simultaneous collaborative editing, ensuring that changes by multiple users are immediately reflected across the document.

Intelligent Citation Management: The system should automatically detect and remove citation markers from the input text and intelligently reinsert them into the refined output using semantic matching techniques. This ensures that references are accurately preserved and formatted according to academic standards if the citation missed the model.

Customizable User Interface: The user interface should offer dynamic customization options that allow users to adjust text formatting, style, and structure according to specific academic requirements. This includes the ability to manually fine-tune the converted text to meet precise scholarly standards.

Document Export Capabilities: The system must enable users to export the final academic document in PDF format, ensuring that the output is submission-ready and adheres to IEEE standards.

Session Management: The system should track individual user sessions to maintain consistency in collaborative editing environments, ensuring that all contributions are synchronized and accurately integrated into the final document.

Data Logging and Audit Trail: The system must log all key actions and changes during the text conversion process. This provides an audit trail for quality assurance and allows for further analysis of user interactions and system performance.

The gathered requirements were validated through iterative feedback sessions and follow-up interviews with stakeholders. Their inputs were analyzed to refine and prioritize functionalities, ensuring that essential features were clearly defined and any ambiguity eliminated. A priority matrix was created to classify each requirement as essential, desirable, or optional, guiding the development process. These functional requirements were subsequently mapped into user stories; Figure 4 illustrates the user stories derived from this process. In addition, Figure 5 presents a use case diagram that details the interactions between various user roles and the system, and Figure 6 outlines a sequence diagram showing the operational flow from data extraction to mind map generation.

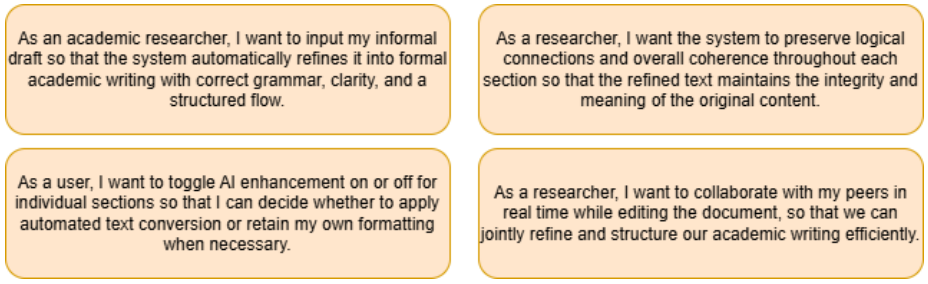


Figure 4:User stories

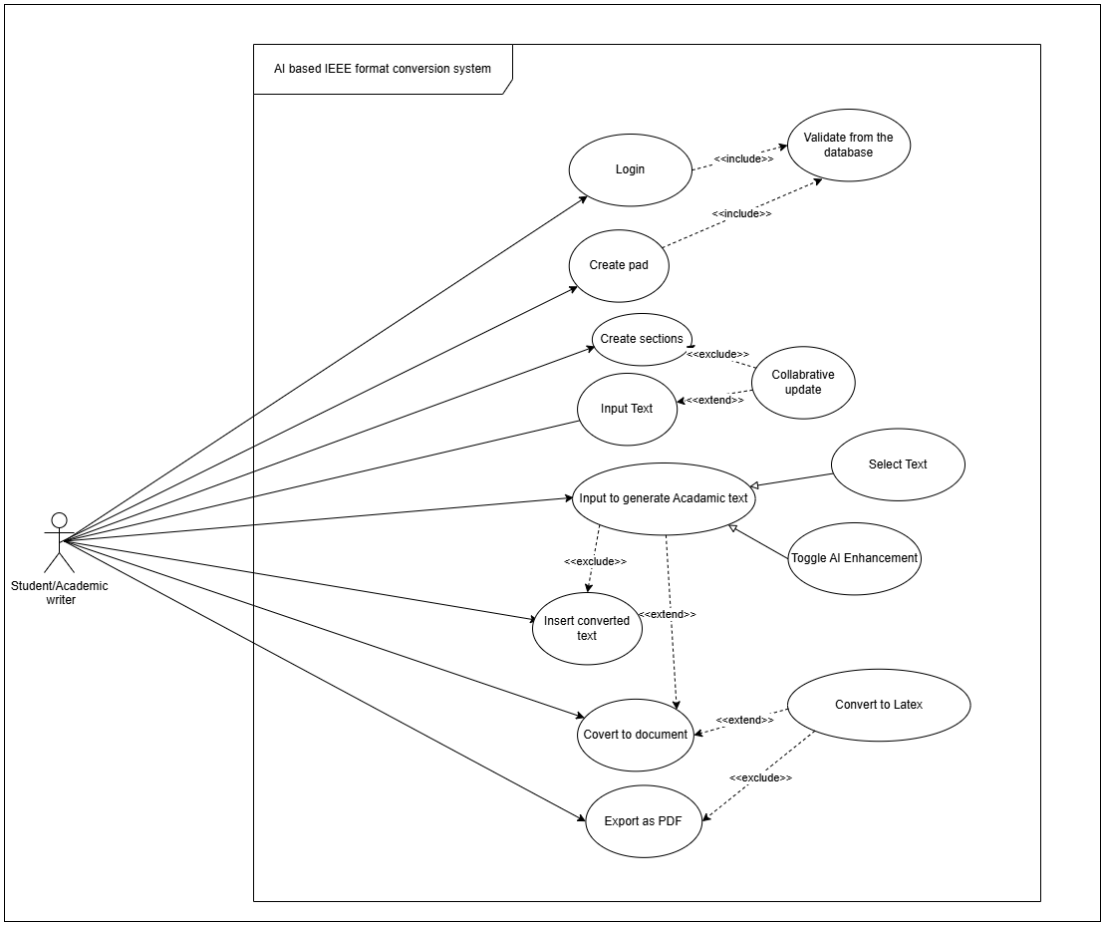


Figure 5: Use case diagram

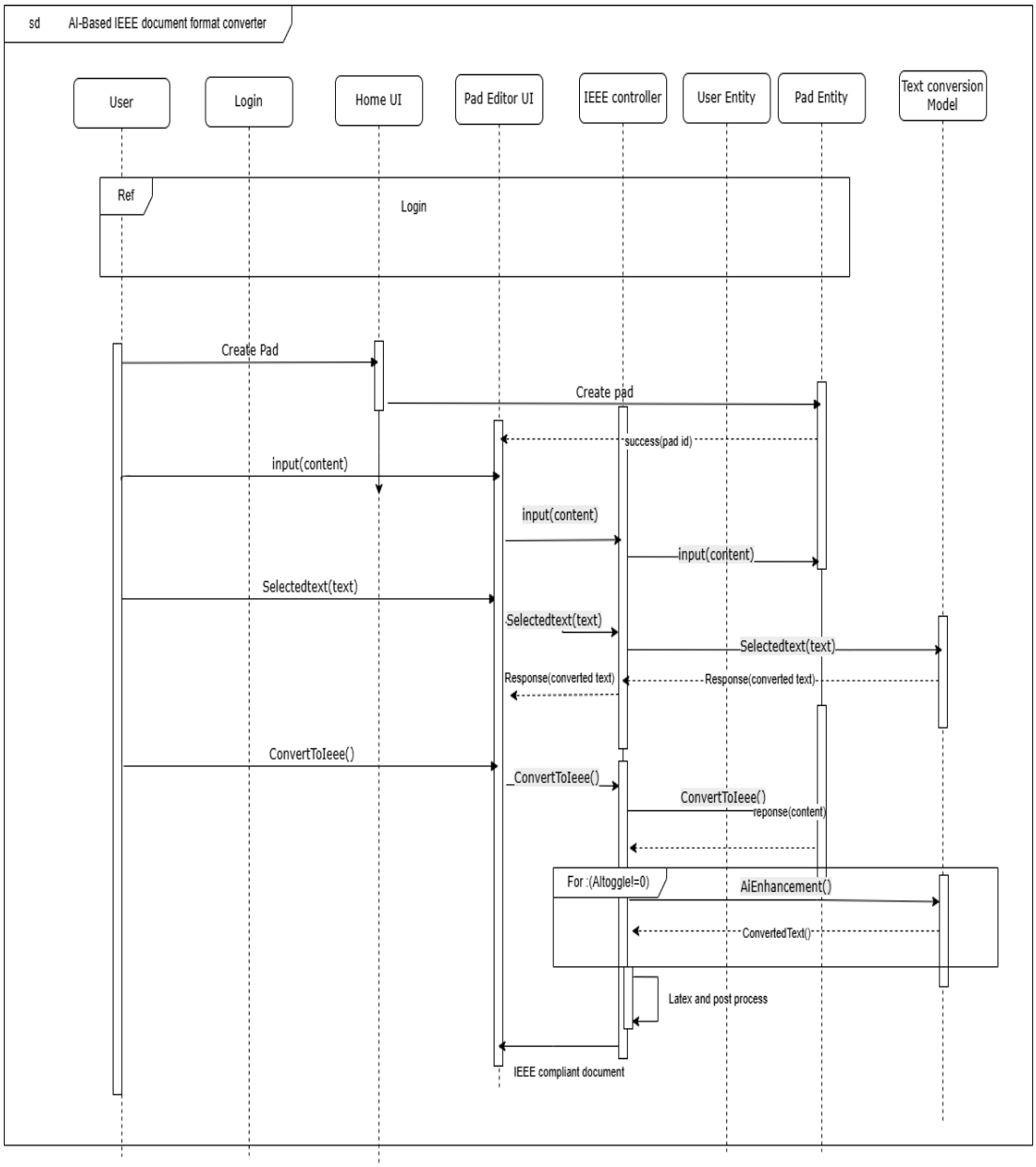


Figure 6:Sequence Diagram

2.1.2.2. Non-Functional Requirements

Non-functional requirements define the qualitative aspects of the automated academic writing conversion system. They ensure that the system not only delivers high-quality, IEEE-compliant output but also operates reliably, performs efficiently, and meets user expectations in terms of usability, scalability, and ethical compliance. These requirements were identified through interviews with academic writers, researchers, and educators, and they address the system's performance in real-world usage scenarios.

Performance: The system must deliver consistently fast response times, especially during real-time text conversion processes. Given that users may work with extensive documents or engage in collaborative editing sessions, it is crucial that the system efficiently processes large volumes of text without perceptible delays. This ensures a smooth user experience even under high load conditions.

Usability: The user interface should be intuitive and accessible to individuals with varying levels of technical expertise. It must offer clear visual indicators and user-friendly interaction controls that facilitate the input and refinement of academic text. Features such as the ability to toggle AI enhancement on or off for specific sections should be straightforward to use, allowing users to effortlessly customize their document conversion process without a steep learning curve

Availability: The system is expected to maintain high availability during typical academic usage hours. It should be accessible across different devices and browsers, ensuring that users experience minimal downtime. Continuous synchronization in collaborative editing environments is essential to prevent data loss and to guarantee that all contributions are reliably integrated into the final document.

Accuracy: A critical requirement is that the system accurately converts informal or unstructured text into coherent, IEEE-compliant academic writing. The conversion process must preserve the intended meaning, logical structure, and proper citation

placements. Ensuring high accuracy in text refinement is vital for producing publication-ready documents that meet rigorous academic standards.

Reliability: The system must be robust enough to handle a variety of usage scenarios, including unexpected user actions, network interruptions, and the processing of large datasets. It should maintain stable operation under diverse load conditions and recover gracefully from any errors, ensuring that the output remains consistent and reliable throughout the conversion process.

Scalability: The underlying architecture should support future growth by accommodating an increasing number of users and larger datasets without compromising performance or quality. Scalability is essential to ensure that the system remains effective as its user base expands and as academic demands evolve

Ethical Compliance: The system must adhere to standard data handling and privacy practices. All user data should be managed securely, and robust measures must be implemented to protect sensitive information during both the conversion and collaborative editing processes. Compliance with relevant data protection regulations is imperative to maintain user trust and uphold ethical standards..

Together, these non-functional requirements form a framework that ensures the academic writing conversion system is efficient, user-friendly, and robust—delivering high-quality, IEEE-compliant documents while meeting the practical and ethical needs of the research community.

2.1.3 Designing

The designing phase of the Software Development Life Cycle establishes a comprehensive architectural blueprint for the automated academic writing conversion system. At this stage, the focus is on developing a cohesive system that converts informal text into structured, IEEE-compliant academic writing while preserving the semantic structure and logical flow of the original content. Detailed planning ensures that all modules and features are integrated seamlessly within a collaborative editing environment, while internal operations such as model fine-tuning, text conversion remains transparent to the end user. This phase lays the structural foundation necessary to support real-time processing, dynamic customization, and efficient collaboration.

The design process emphasizes clarity, modularity, and scalability. The overall architecture is carefully crafted to ensure that each component—from text pre-processing and AI-driven text refinement to citation management—is optimized for performance and usability. Through meticulous planning and iterative refinement, the design guarantees that the system meets user requirements and supports future enhancements without disruption.

System Architecture Diagram:

The system architecture diagram visually represents the interactions between the core components of the academic writing conversion system. It illustrates how the system processes input text from academic documents, refines it using a fine-tuned language model, and produces structured, IEEE-compliant output that maintains the original document's logical connections. A Business Logic Layer coordinates data flow among these modules to ensure consistent conversion and citation management while

enabling real-time collaborative editing. (Figure 7 illustrates the high-level system architecture.)

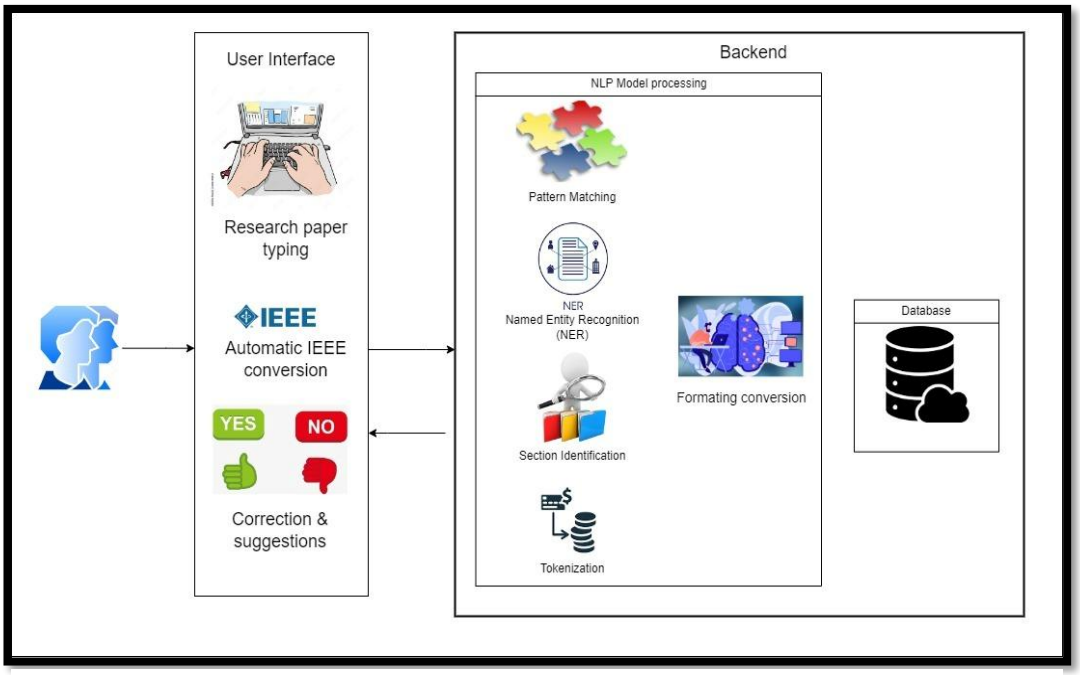


Figure 7: System Diagram

Design Details:

The architectural design of the proposed academic writing conversion system is organized into a three-tier structure, ensuring a modular, scalable, and efficient solution.

Presentation Layer: This layer delivers an intuitive and interactive user interface tailored for academic researchers. It enables users to log in and input their informal text as individual snippets in a form and view the refined academic output in real time. The interface includes dynamic controls that allow users to toggle AI enhancement on or off for each section, providing flexibility in determining the level of automated conversion. By abstracting the complexities of the underlying processes, the Presentation Layer offers a polished and responsive experience, supporting collaborative editing among multiple users without overwhelming them with technical details.

Business Logic Layer: Serving as the core of the system, the Business Logic Layer processes the input text using a fine-tuned language model that converts informal text into structured, IEEE-compliant academic writing. It handles intelligent citation management by automatically detecting, removing, and semantically reinserting citation markers to ensure that references are accurately preserved and formatted. Additionally, this layer manages user customization actions—such as the selective application of AI enhancement—and coordinates export operations based on user preferences for final output formats (e.g., PDF or DOCX). By integrating these functions, the Business Logic Layer guarantees that the transformed content exhibits a formal tone, coherent structure, and logical progression throughout the document.

Data Access Layer: The Data Access Layer is responsible for managing persistent storage and ensuring data integrity across user sessions. It handles user session management and securely stores both the raw input and the refined academic text. This layer also supports real-time collaboration by synchronizing updates among multiple users, ensuring that all modifications are consistently reflected without data loss. Furthermore, it interfaces with external data sources—such as datasets used for model

fine-tuning—while keeping these operations transparent to the user. Robust data logging mechanisms are implemented here to provide an audit trail of user interactions and system changes, which supports both quality assurance and future improvements.

Flow of IEEE compliant document generation:

The process begins when the user submits an informal academic text through the Presentation Layer. The input is then transmitted to the Business Logic Layer, where the Academic Writing Conversion Module is activated. This module leverages a fine-tuned large language model to analyze the input text and automatically refine it—correcting grammar, enhancing clarity, enforcing a formal tone, and ensuring logical organization—thereby transforming it into IEEE-compliant academic writing.

Once the model processes the text, the refined content is transmitted back to the Presentation Layer, where the user can view the formatted document in real time. Users have the flexibility to toggle AI enhancement on or off for each section, allowing them to decide whether to apply automated conversion or maintain parts of their original text. This dynamic control ensures that the final output meets both personal and institutional standards.

Finally, the system offers export capabilities, enabling users to download the publication-ready document in PDF format, ensuring that the output adheres to IEEE formatting guidelines. Figure 9 illustrates the complete operational flow—from text submission through AI-driven conversion to final document export—demonstrating how the system streamlines the academic writing process while ensuring high-quality, IEEE-compliant output.

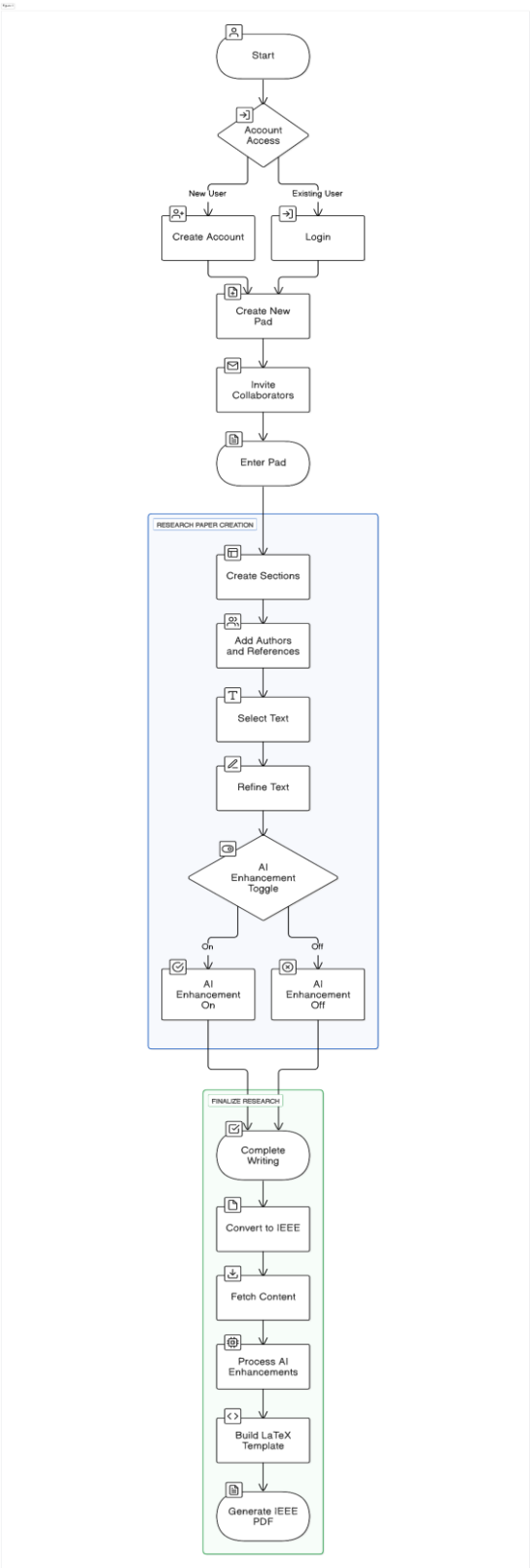


Figure 8: : Flow chart

2.1.4 Implementation

The Implementation phase marks the transition from design to a working AI-based IEEE format conversion system. At the backend, a fine-tuned Llama 2-7B model—optimized through 4-bit quantization and parameter-efficient tuning—is integrated as an API. This model automatically refines input text by correcting grammar, enhancing clarity, and enforcing a formal tone, thereby converting informal content into IEEE-compliant academic writing.

On the front end, a collaborative interface built with Quill.js allows users to submit text and view the refined output in real time. A key feature is the toggle that lets users turn AI enhancement on or off for each section, providing flexibility in the conversion process.

An intelligent citation management module automatically extracts and re-inserts citation markers using semantic matching techniques to ensure reference accuracy. Once processed, the refined academic document is formatted and exported exclusively as a PDF, delivering a submission-ready output.

Iterative testing and continuous feedback from pilot users have been integral in refining the system for reliability and efficiency. Overall, the implemented system streamlines academic writing by automating text conversion and citation management, significantly reducing manual effort while producing high-quality, IEEE-compliant documents in PDF format.

Task Breakdown and Project Management:

Microsoft Planner, a versatile project management tool, was harnessed to meticulously delineate the project into well-defined subtasks and milestones. This approach provided a clear and organized roadmap for the development process. Each subtask, ranging from dataset preparation and model fine-tuning to backend integration and frontend development, was allocated with timelines and responsibilities to enhance collaboration and accountability. Figure 9 illustrates the Microsoft Planner board used for tracking sprint-based development progress throughout the project.

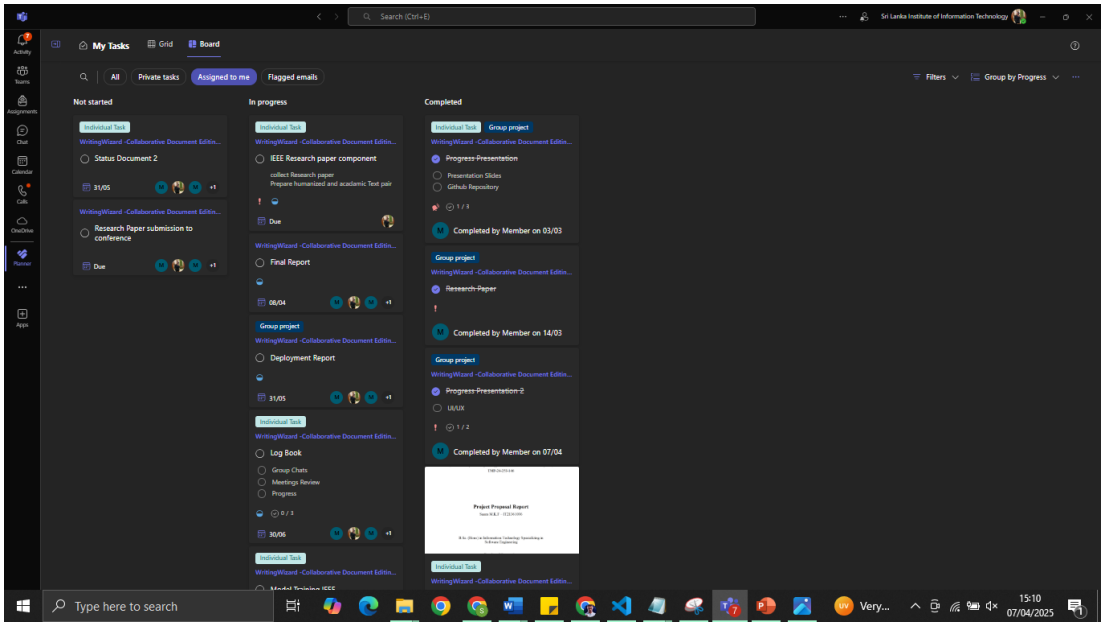


Figure 9: MS planner

The Work Breakdown Structure (WBS) was instrumental in creating a hierarchical representation of tasks, allowing for a granular understanding of project components. Tasks were categorized, sequenced, and assigned to team members based on their expertise and responsibilities. This approach not only enhanced project efficiency but also facilitated proactive issue resolution, ensuring that the project remained on course to meet its objectives. Figure 9 illustrates the WBS for the stress detection component.

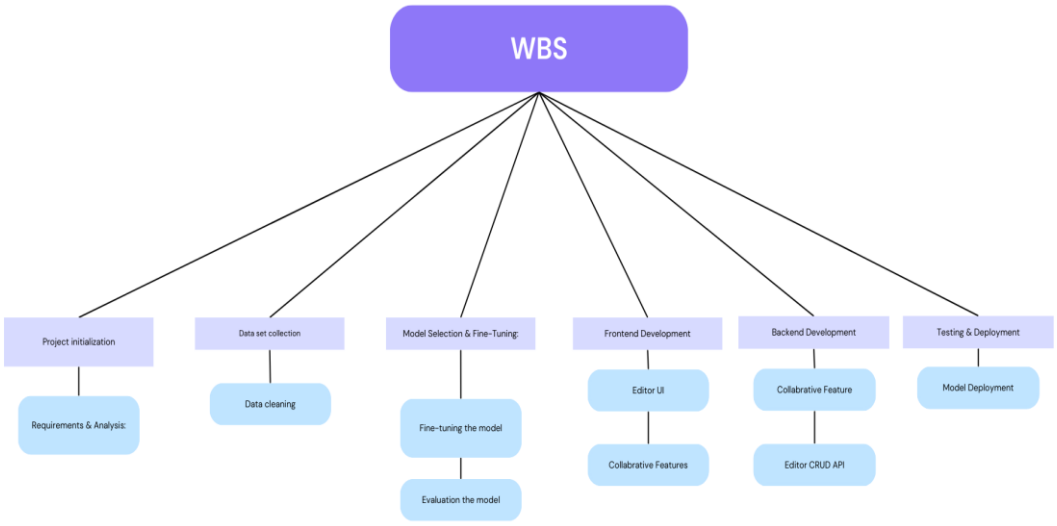


Figure 10: Work Breakdown Structure

Development Environment and Tools:

The development of the AI-based IEEE format conversion system was carried out using a suite of robust tools and technologies:

- **Integrated Development Environment (IDE):** PyCharm was the primary IDE, providing a powerful platform for Python development, debugging, and version control. Its extensive support for Python facilitated the implementation of model training, backend API development, and integration of various libraries.
- **Backend Frameworks:**
  - **FastAPI:** Used for building the asynchronous API that processes text conversion requests.
  - **Node.js with Express:** Employed to create routes that integrate EJS templating for generating formatted IEEE documents and managing PDF compilation.
- **Frontend Technologies:**
  - **React:** Developed the interactive user interface.
  - **Quill.js:** Provided rich-text editing and real-time collaboration capabilities, along with customizable editing experience.
- **Machine Learning Libraries:**
  - **HuggingFace Transformers & PyTorch:** Utilized for fine-tuning the Llama 2-7B model.
  - **Unsloth Framework:** Applied for efficient 4-bit quantization and parameter-efficient tuning, optimizing the model for resource-constrained environments.
- **Visualization and Documentation Tools:**
  - **Graphviz and Draw.io:** Employed for designing flowcharts and system architecture diagrams.
- **Version Control and Collaboration:** Git was used for version control, ensuring smooth collaboration and iterative development across the team.

2.1.4.1 Data Collection for AI-Based IEEE Format Conversion

The dataset for training the AI-based IEEE format converter was created by assembling paired samples of formatted and informal academic text. This process involved several key steps:

1. **Bulk Download of IEEE Research Papers**: A large collection of IEEE research papers was downloaded from online repositories. These documents provided a rich source of high-quality, IEEE-compliant formatted academic text.
2. **Paragraph Extraction**: Automated scripts were employed to extract individual paragraphs from the downloaded research papers. These paragraphs serve as examples of properly formatted academic content that adhere to IEEE standards.
3. **Generation of Informal Versions**: To simulate real-world draft conditions, the formatted paragraphs were transformed into informal text. This transformation was achieved through a combination of manual editing and automated tools that intentionally introduced casual language, humanized phrasing, and grammatical errors. The purpose was to create a bidirectional dataset where the model learns to convert informal text into structured academic writing.
4. **Creation of Data Pairs**: Each formatted paragraph was paired with its corresponding informal version, forming the core dataset. These paired samples enable the fine-tuning process by providing clear examples of the desired transformation—from informal, error-prone writing to polished, IEEE-compliant academic text.
5. Data Cleaning and Preprocessing The paired dataset was then rigorously preprocessed to remove inconsistencies, standardize text formats, and ensure that the data was clean and ready for training. This step involved normalization of text and removal of extraneous formatting or noise, which is critical for effective model learning.

This robust dataset forms the foundation for fine-tuning the language model, enabling it to accurately learn the transformation required to convert informal academic writing into a professional, IEEE-compliant document with minimal manual intervention.

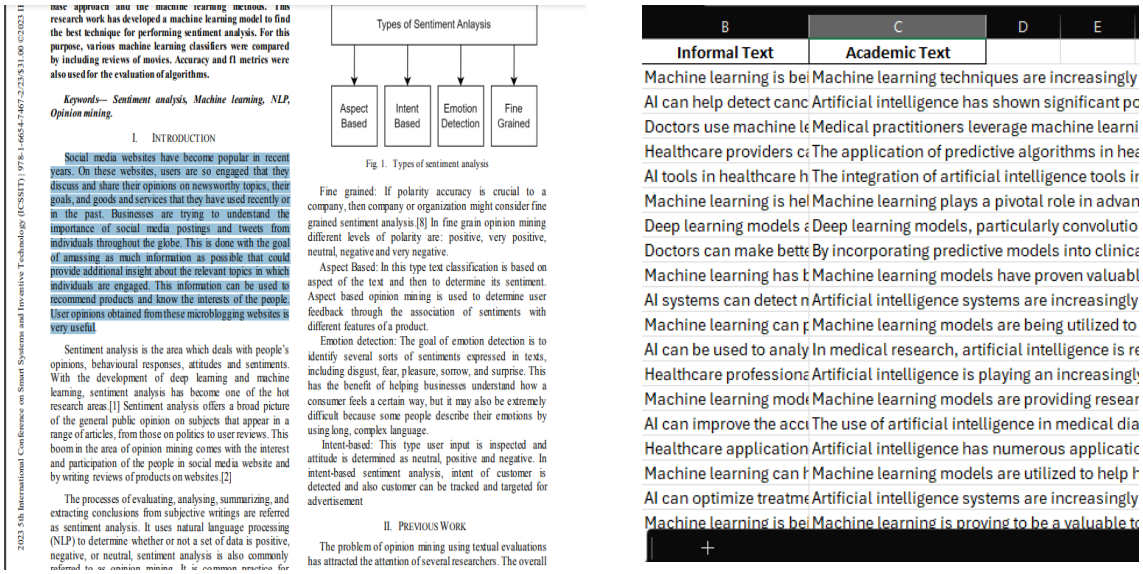


Figure 11:Dataset for model training

2.1.4.3 Selecting Model Training Algorithm

In the domain of academic writing conversion, selecting an appropriate model is crucial to achieving effective transformation from informal drafts to structured, IEEE-compliant output. This research evaluated several state-of-the-art models, including various configurations of T5 and FLan T5, to determine which could best capture the nuances of academic tone, complex sentence structures, and precise citation placements required for formal publications.

1. Text Representation and Transformation Capabilities: The T5 and FLan T5 models have demonstrated strong performance in general natural language processing tasks, including text summarization and generation. However, when applied to academic writing conversion, these models often struggled to consistently enforce the formal tone and logical organization necessary for IEEE-compliant academic texts. In contrast, the fine-tuned Llama 2-7B model, trained specifically on a custom dataset of paired informal and formatted academic text, showed superior ability to correct grammar, enhance clarity, and produce output with a coherent structure.

2. Model Size and Computational Efficiency: Although T5 and FLan T5 are available in a range of sizes—including larger variants that could potentially yield better text quality, they require significant computational resources and memory, making them less suitable for deployment in resource-constrained environments. On the other hand, the Llama 2-7B model, after applying 4-bit quantization and parameter-efficient fine-tuning techniques like Low-Rank Adaptation (LoRA), offers an optimal balance between model performance and computational efficiency. This balance is critical for real-time academic writing conversion, as it ensures prompt responses without overburdening available hardware.
3. Adaptability and Fine-Tuning Efficiency: A key advantage of the Llama 2-7B model is its adaptability to the specific requirements of academic writing. By fine-tuning this model on a domain-specific dataset that includes meticulously paired examples of informal and IEEE-formatted text, the model learns to produce output that maintains academic rigor. In contrast, T5 and FLan T5 models—while versatile—demonstrated less efficiency during fine-tuning for this specific task, often requiring additional adjustments that affected their overall performance in generating structured, polished academic content.
4. Performance and Generalization: Comprehensive experiments revealed that the fine-tuned Llama 2-7B model consistently outperformed the evaluated configurations of T5 and FLan T5 in transforming informal text into coherent, IEEE-compliant academic writing. The Llama model's performance in maintaining logical connections, academic tone, and precise formatting proved to be more robust, yielding higher quality output across a range of academic text complexities. This superior generalization capability across diverse academic contexts was a decisive factor in its selection.

2.1.4.1 Model Training for AI-Based IEEE Format Conversion

The model training phase adapts a pre-trained language model for the task of converting informal academic text into structured, IEEE-compliant writing. The training process follows a series of systematic steps similar in structure to the emotion

detection training, tailored to address the unique challenges of academic text conversion.:

1. Data Preprocessing: The training process begins by collecting raw text from a bulk download of IEEE research papers. Individual paragraphs are extracted and cleaned by normalizing punctuation, removing extraneous whitespace, and stripping citation markers to avoid interference during conversion. This preprocessing step ensures consistency across samples and reduces noise in the training data.
2. Data Pairing and Labeling: A curated dataset is built by manually pairing each formatted academic paragraph with a corresponding informal version. The informal text is generated using a combination of automated text manipulation and human editing to simulate common errors and casual writing styles. These pairs serve as the ground truth for learning the transformation from informal to formal academic writing.
3. Feature Extraction and Tokenization: Each text pair is then processed using a tokenizer from the HuggingFace Transformers library. The tokenizer converts the text into sequences of tokens, enabling the model to capture syntactic and semantic patterns. This tokenization step is critical for both the input (informal text) and the target output (IEEE-compliant text), ensuring that the model consistently handles varying text lengths.
4. Data Collection and Organization: The preprocessed and tokenized pairs are organized into a dataset where each entry includes an informal text sample and its corresponding formatted academic text. This structured dataset, stored in a dedicated directory, enables efficient batching and supports further training operations by clearly delineating input-output pairs.
5. Data Normalization: Tokenized sequences are normalized to ensure that all inputs fall within a consistent range, which facilitates smooth convergence during training. Normalization might involve padding sequences to a uniform length and applying any necessary scaling to numerical representations.
6. Model Definition: The core language model used is a pre-trained Llama 2-7B, fine-tuned for academic text conversion. The model architecture, built on transformer

layers, is further optimized using 4-bit quantization and Low-Rank Adaptation (LoRA) via the Unsloth framework. These techniques significantly reduce memory usage while preserving model performance, making the fine-tuning process more efficient.

7. Model Compilation: The training configuration is established using the SFTTrainer from the TRL library alongside Training Arguments from HuggingFace Transformers. The model is compiled with a low learning rate, small per-device batch sizes, and gradient accumulation strategies to facilitate stable convergence. Supervised instruction learning is employed, where the paired dataset drives the model to learn the transformation from informal to structured academic writing.
8. Model Training: The training process runs through multiple epochs (or steps, as configured), where the model is iteratively refined by comparing its output to the target formatted text. Each training iteration updates the model's internal weights through backpropagation, gradually enhancing its ability to generate IEEE-compliant academic content.
9. Conclusion: Once training is complete, the model is saved to a specified directory in a format suitable for later deployment. The comprehensive training process transforms the raw paired dataset into a specialized model capable of converting informal academic text into polished, IEEE-compliant output,

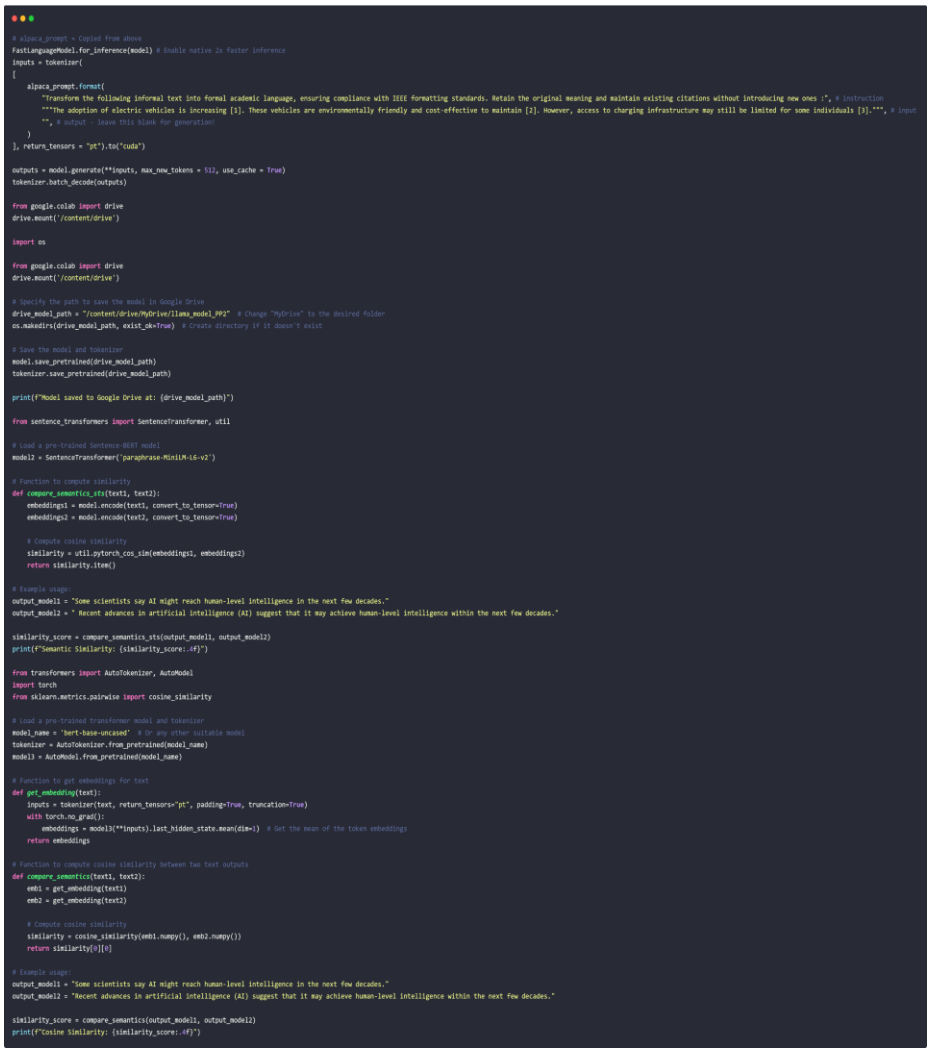
The following training code details the implementation process and configuration parameters used to fine-tune the model, which will be provided at the end.

```
!pip install unsloth
!pip uninstall unsloth -y && pip install --upgrade --no-cache-dir --no-deps git+https://github.com/unslothai/unsloth.git
!pip install datasets

from unsloth import FastLanguageModel
import torch
max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/llama-2-7b-bnb-4bit", # Choose ANY! eg teknium/OpenHermes-2.5-Mistral-7B
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
    # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
)

from google.colab import drive
drive.mount('/content/drive')

# Specify the path to the saved model
drive_save_path = '/content/drive/MyDrive/llama_model_W1'

# Load the model and tokenizer from Google Drive
from unsloth import FastLanguageModel
import torch

max_seq_length = 2048  # Choose any! We auto support RoPE Scaling internally!
dtype = None  # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
load_in_4bit = True  # Use 4bit quantization to reduce memory usage. Can be False.

# Load model
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name=drive_save_path,  # Path to the saved model on Google Drive
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)
print("Model Loaded")
print(f"Model config: {model.config}")

# Check if CUDA is available
print(f"CUDA Available: {torch.cuda.is_available()}")

model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0, # Supports any, but = 0 is optimized
    bias = "none",    # Supports any, but = "none" is optimized
    # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
    use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
    random_state = 3407,
    use_rslora = False,  # We support rank stabilized LoRA
    loftq_config = None, # And LoftQ
)
```

Figure 12: Model Loading script for training

```
import pandas as pd
alpaca_prompt = """Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
{}

### Input:
{}

### Response:
{}"""


EOS_TOKEN = tokenizer.eos_token # Must add EOS_TOKEN
def formatting_prompts_func(examples):
    instructions = examples["Instruction"]  # This column should contain the instruction
    inputs = examples["Informal Text"]  # The text to be converted to a mind map
    outputs = examples["Academic Text"]  # The actual mind map JSON structure (this should be your target)

    texts = []
    for instruction, input, output in zip(instructions, inputs, outputs):
        # Construct the prompt
        text = alpaca_prompt.format(instruction, input, output) + EOS_TOKEN
        texts.append(text)

    return {"text": texts}
pass

import pandas as pd
from datasets import Dataset
# Load your dataset
data = pd.read_excel("NewSet0102.xlsx", engine="openpyxl")
# Convert it into a Hugging Face dataset
dataset = Dataset.from_pandas(data)
# Apply the formatting function
dataset = dataset.map(formatting_prompts_func, batched=True)

# Check the first few entries after the mapping
print("\nTransformed Dataset Preview:")
print(dataset[:3])  # Print first 3 rows after transformation

from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        max_steps = 60, # Set num_train_epochs = 1 for full training runs
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
        report_to = "none", # Use this for WandB etc
    ),
)

trainer_stats = trainer.train()
```

Figure 13:Model Training script



```
# alpaca_prompt = Copied from above
FastLanguageModel.for_inference(model) # Enable native 2x faster inference
inputs = tokenizer(
[
    alpaca_prompt.format(
        "Transform the following informal text into formal academic language, ensuring compliance with IEEE formatting standards. Retain the original meaning and maintain existing citations without introducing new ones :", # instruction
        """The adoption of electric vehicles is increasing [1]. These vehicles are environmentally friendly and cost-effective to maintain [2]. However, access to charging infrastructure may still be limited for some individuals [3].""", # input
        "", # output - leave this blank for generation!
    )
], return_tensors = "pt").to("cuda")

outputs = model.generate(**inputs, max_new_tokens = 512, use_cache = True)
tokenizer.batch_decode(outputs)

from google.colab import drive
drive.mount('/content/drive')

import os

from google.colab import drive
drive.mount('/content/drive')

# Specify the path to save the model in Google Drive
drive_model_path = "/content/drive/MyDrive/llama_model_PP2"  # Change "MyDrive" to the desired folder
os.makedirs(drive_model_path, exist_ok=True)  # Create directory if it doesn't exist

# Save the model and tokenizer
model.save_pretrained(drive_model_path)
tokenizer.save_pretrained(drive_model_path)

print(f"Model saved to Google Drive at: {drive_model_path}")

from sentence_transformers import SentenceTransformer, util

# Load a pre-trained Sentence-BERT model
model2 = SentenceTransformer('paraphrase-MiniLM-L6-v2')

# Function to compute similarity
def compare_semantics_sts(text1, text2):
    embeddings1 = model.encode(text1, convert_to_tensor=True)
    embeddings2 = model.encode(text2, convert_to_tensor=True)

    # Compute cosine similarity
    similarity = util.pytorch_cos_sim(embeddings1, embeddings2)
    return similarity.item()

# Example usage:
output_model1 = "Some scientists say AI might reach human-level intelligence in the next few decades."
output_model2 = " Recent advances in artificial intelligence (AI) suggest that it may achieve human-level intelligence within the next few decades."

similarity_score = compare_semantics_sts(output_model1, output_model2)
print(f"Semantic Similarity: {similarity_score:.4f}")

from transformers import AutoTokenizer, AutoModel
import torch
from sklearn.metrics.pairwise import cosine_similarity

# Load a pre-trained transformer model and tokenizer
model_name = 'bert-base-uncased' # Or any other suitable model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model3 = AutoModel.from_pretrained(model_name)

# Function to get embeddings for text
def get_embedding(text):
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)
    with torch.no_grad():
        embeddings = model3(**inputs).last_hidden_state.mean(dim=1)  # Get the mean of the token embeddings
    return embeddings

# Function to compute cosine similarity between two text outputs
def compare_semantics(text1, text2):
    emb1 = get_embedding(text1)
    emb2 = get_embedding(text2)

    # Compute cosine similarity
    similarity = cosine_similarity(emb1.numpy(), emb2.numpy())
    return similarity[0][0]

# Example usage
output_model1 = "Some scientists say AI might reach human-level intelligence in the next few decades."
output_model2 = "Recent advances in artificial intelligence (AI) suggest that it may achieve human-level intelligence within the next few decades."

similarity_score = compare_semantics(output_model1, output_model2)
print(f"Cosine Similarity: {similarity_score:.4f}")
```

Figure 14: Model evaluation script

2.1.4.2 Backend for Text conversion and document building

Below is the mechanism for the backend of the AI-based IEEE format conversion component. The process relies on advanced language processing and intelligent citation management. Here's a step-by-step explanation of how it works:

1. Text Submission: The process begins when the user submits an informal academic text through a REST API endpoint. The backend receives this text input via a secure HTTP request, which includes sections that may have grammatical errors, informal language, and unstructured formatting.
2. Preprocessing: Upon receiving the input, the system preprocesses the text to prepare it for conversion. This step involves cleaning the text by normalizing punctuation, removing extra spaces, and temporarily stripping citation markers to ensure that these elements do not interfere with the AI conversion process.
3. Text Conversion: The preprocessed text is then forwarded to the Academic Writing Conversion Module within the Business Logic Layer. Here, a fine-tuned Llama 2-7B model processes the text. Leveraging techniques such as 4-bit quantization and parameter-efficient fine-tuning, the model refines the text by correcting grammar, enhancing clarity, enforcing a formal tone, and ensuring logical organization in line with IEEE standards.
4. Citation Management: In parallel, an intelligent citation management module extracts citation markers from the original text before conversion. After the AI model refines the text, this module re-inserts the citations in the correct positions using semantic matching techniques. This ensures that the final output maintains accurate and consistent reference.
5. Data Aggregation: The process begins by retrieving the complete academic content from the database or session storage. This includes the refined text, structured sections, metadata (such as title, authors, abstract, and keywords), and citation details. All these elements form the comprehensive data model required for generating the final document.
6. EJS Template Rendering: Once the data is collected, the backend utilizes an EJS (Embedded JavaScript) template that has been specifically designed to conform to IEEE formatting standards. This template contains placeholders for all key components of the academic document. The system calls the ejs.renderFile function, passing the retrieved data and the template file. The EJS engine dynamically injects the data into the template, creating a complete LaTeX source file (with a .tex extension). This step ensures that the structure, section hierarchy,

and citation placements are all embedded within the LaTeX code, ready for further processing.

7. LaTeX Compilation: After generating the .tex file, the system invokes the LaTeX compiler (typically pdflatex) using Node.js's child_process.exec command. This command compiles the LaTeX source into a final PDF document. The compiler processes the LaTeX code to format text, equations, figures, and tables according to IEEE guidelines, ensuring that fonts, margins, and layout conform to the required standards.
8. Real-Time Collaborative Updates: Throughout the document preparation process, real-time collaborative updates are managed via WebSockets. When multiple users work on the same pad, any changes they make are broadcast by the backend to all connected clients. This synchronization ensures that all collaborative edits—including text modifications and section updates—are reflected immediately across all user sessions. The WebSocket infrastructure guarantees consistency and conflict resolution, enabling a dynamic, multi-user editing environment.
9. Error Handling and Logging: Throughout the rendering and compilation stages, robust error handling is in place. Any issues encountered during the EJS rendering process or during the LaTeX compilation are captured and logged. If an error occurs, the system generates an appropriate error message and may attempt a fallback process, ensuring that the user is informed of the issue without compromising the overall workflow.
10. Output Delivery: Once the LaTeX compiler successfully generates the PDF, the system checks that the output file exists in the designated directory. The PDF is then sent back to the user as a downloadable file via an HTTP response, providing a submission-ready document that adheres to IEEE standards.

```
// Render and Compile LaTeX
router.get("/:padId", async (req, res) => {
  try {
    const pad = await Pad.findById(req.params.padId);
    if (!pad) return res.status(404).json({ msg: "Pad not found" });

    const authorsRows = chunkArray(pad.authors || [], 3);

    const processedSections = pad.sections.map((section) => ({
      title: section.title || "Untitled Section",
      originalDelta: section.content,
      aiEnhancement: section.aiEnhancement || false,
      subsections: section.subsections?.map((sub) => ({
        title: sub.title || "Untitled Subsection",
        originalDelta: sub.content,
        aiEnhancement: sub.aiEnhancement || false,
        subsections: []
      })) || []
    }));

    await convertSectionsByFullText(processedSections);
    console.log("📄 Final section content:", processedSections.map(s => s.content).join("\n\n"));

    const content = {
      title: pad.title || "Untitled Document",
      authorsRows,
      abstract: pad.abstract || "",
      keyword: pad.keyword || "",
      sections: processedSections,
      image_path: pad.image_path || "default_image_path.jpg",
      references: pad.references || []
    };
    console.log("references to be printed",content);

    const templatePath = path.join(__dirname, "../templates/ieee_template.tex.ejs");
ejs.renderFile(templatePath, content, {}, (err, latexOutput) => {
  if (err) {
    console.error("Error rendering LaTeX template:", err);
    return res.status(500).json({ msg: "Error rendering LaTeX template", error: err });
  }

  const outputTexPath = path.join(outputDir, "output_paper.tex").replace(/\\/g, "/");
  fs.writeFileSync(outputTexPath, latexOutput, "utf8");

  const command = `pdflatex -interaction=nonstopmode -output-directory "${outputDir}" "${outputTexPath}"`;
  exec(command, (error, stdout, stderr) => {
    // Log stdout/stderr for debugging purposes
    console.log("STDOUT:", stdout);
    console.log("STDERR:", stderr);

    const outputPdfPath = path.join(outputDir, "output_paper.pdf");
    if (fs.existsSync(outputPdfPath)) {
      // Even if error exists, if the PDF is there, send it.
      return res.download(outputPdfPath, "output_paper.pdf", (err) => {
        if (err) {
          console.error("Error sending PDF file:", err);
          return res.status(500).json({ msg: "Error sending PDF file", error: err });
        }
      });
    } else {
      // If the file does not exist, then return an error.
      console.error(`Error compiling LaTeX: ${error.message}`);
      return res.status(500).json({ msg: "Error compiling LaTeX", error: error.message });
    }
  });
});

  } catch (err) {
    console.error("Server error:", err);
    res.status(500).json({ msg: "Server Error", error: err });
  }
});
```

Figure 15: Document preparing express API

```
router.post("/convert-text", async (req, res) => {
  try {
    const { content } = req.body;
    console.log("i received the content",content)

    if (!content || content.trim() === "") {
      return res.status(400).json({ msg: "No text provided for conversion." });
    }

    console.log("🔄 Sending text to AI API for conversion...");
      const section="";
    // Send request to AI API
    const response = await axios.post(`${process.env.REACT_APP_BACKEND_API_URL_IEEE}/convert`, {
      section,
      content,
    });

    if (response.data && response.data.converted_text) {
      console.log("✅ AI Conversion Success:", response.data.converted_text);
      return res.json({ converted_text: response.data.converted_text });
    } else {
      console.error("⚠️ AI API Response Format Unexpected:", response.data);
      return res.status(500).json({ msg: "Unexpected response from AI API." });
    }
  } catch (error) {
    console.error("❌ Error calling AI API:", error.message);
    return res.status(500).json({ msg: "Error converting text.", error: error.message });
  }
});


module.exports = router;
```

Figure 17: Post processing scriptsFigure

```
async function convertSectionsByFullText(sections) {
  for (let section of sections) {
    const { sequence } = extractNonTextElements(section.originalDelta || {});

    if (section.aiEnhancement && sequence.length) {
      try {
        const converted = await convertParagraphsWithNonText(sequence, section.title);
        section.content = processAbbreviations(converted);
      } catch (apiErr) {
        console.error("Conversion API error:", apiErr);
        //section.content = processAbbreviations(sequence.map(i => i.content).join("\n"));
        const fallbackLatex = await convertParagraphsWithNonText(sequence, section.title);
        section.content = processAbbreviations(fallbackLatex);
      }
    } else {
      //section.content = processAbbreviations(sequence.map(i => i.content).join("\n"));
      const fallbackLatex = await convertParagraphsWithNonText(sequence, section.title);
      section.content = processAbbreviations(fallbackLatex);
    }

    if (section.subsections) await convertSectionsByFullText(section.subsections);
  }
}
```

Figure 16:Text conversion model API Post processing scripts

```
function convertHtmlTableToLatex(tableHtml) {
  const $ = cheerio.load(tableHtml);

  let rows = [];

  $("tr").each((i, tr) => {
    let cells = [];
    $(tr).find("td").each((j, td) => {
      let cellHtml = $(td).html() || "";
      console.log("Raw HTML in cell:", cellHtml);

      // Decode HTML entities twice for safety
      let cellText = he.decode(he.decode(cellHtml));

      // Replace &amp; manually in case decoding fails
      cellText = cellText.replace(/&amp;/g, "&");

      // Replace & with LaTeX-compatible \&
      cellText = cellText.replace(/&/g, "\\&");

      // Escape LaTeX special characters
      cellText = cellText.replace(/([_%#{}])/g, "\\$1");

      // Trim and store
      cellText = cellText.trim();
      cells.push(cellText);

      console.log("🔍 Processed Cell Text:", cellText);  // Debugging
    });
    if (cells.length > 0) {
      rows.push(cells);
    }
  });

  // Determine max columns
  const maxCols = Math.max(...rows.map(r => r.length), 0);
  let latex = `\\begin{tabular}{|${"c|".repeat(maxCols)}}\n\\hline\n`;

  rows.forEach(cells => {
    while (cells.length < maxCols) {
      cells.push("");
    }
    latex += cells.map(c => `\\textnormal{${c}}`).join(" & ") + " \\\\\n\\hline\n";
  });

  latex += "\\end{tabular}\n";
  return latex;
}




// **Extract images, tables, and formulas before sending text to AI**
function extractNonTextElements(delta) {
  let sequence = [];

  if (!delta || !delta.ops || !Array.isArray(delta.ops)) return { sequence };

  delta.ops.forEach((op) => {
    if (typeof op.insert === "string") {
      sequence.push({ type: "text", content: op.insert });
    }
    else if (op.insert.imageWithCaption) {
      sequence.push({ type: "image", content: op.insert.imageWithCaption });
    }
    else if (op.insert.formulaWithCaption) {
      sequence.push({ type: "formula", content: op.insert.formulaWithCaption });
    }
    else if (op.insert.tableWithCaption) {
      sequence.push({ type: "table", content: op.insert.tableWithCaption });
    }
  });

  return { sequence };
}



async function convertParagraphsWithNonText(sequence, sectionTitle) {
  const result = [];

  for (const item of sequence) {
    if (item.type === "text") {
      const paragraphs = item.content.split(/\n+/).filter(p => p.trim() !== "");
      for (const p of paragraphs) {
        try {
          const response = await axios.post(`${process.env.REACT_APP_BACKEND_API_URL_IEEE}/convert`, {
            section: sectionTitle,
            content: p,
          });
          result.push(response.data.converted_text || p);
        } catch (err) {
          console.error("Error converting paragraph:", err);
          result.push(p);
        }
      }
    } else if (item.type === "image") {
      const { src = "", caption = "Image" } = item.content || {};
      if (!src) continue;
      const imagePath = `../uploads/${src.split("uploads/").pop()}`;
      result.push(`\\begin{figure}[H]\n\\centering\n\\includegraphics[width=0.5\\textwidth]{${imagePath}}\n\\caption{${caption}}\n\\end{figure}`);
    } else if (item.type === "formula") {
      const { formula = "", caption = "" } = item.content || {};
      if (!formula) continue;
      result.push(`\\begin{equation}\n${formula}\n\\end{equation}\n\\textit{${caption}}`);
    }

    else if (item.type === "table") {
      const { tableHtml = "", caption = "Table" } = item.content || {};
      if (!tableHtml) continue;
      const tabularLatex = convertHtmlTableToLatex(tableHtml);
      result.push(`\\begin{table}[H]\n\\centering\n\\caption{${caption}}\n${tabularLatex}\n\\end{table}`);
    }
  }

  return result.join("\n\n");
}
```

Figure 18:Pre-process script for non-text elements

2.1.4.3 Front-End and Text Submission Methods

The front-end of the system is built with a user-centric design to facilitate easy text submission and seamless handling of non-text elements. Developed using React and integrated with Quill.js, the interface provides a rich-text editing environment where academic writers can easily input their informal drafts. Users benefit from an intuitive layout that supports real-time collaborative editing, ensuring that multiple authors can work on the same document simultaneously without conflict.

1. Text Submission: The interface allows users to submit text in various forms, whether as individual snippets or complete documents. Advanced text submission methods include the ability to toggle AI enhancement on or off for specific sections, enabling users to choose when to apply automated refinement. This selective approach ensures that users maintain control over the conversion process while receiving immediate, live feedback on the generated IEEE-compliant academic content.
2. User-Friendly Handling of Non-Text Elements: Beyond text, the system is designed to manage non-text elements—such as images, formulas, and tables—in a manner that is both intuitive and user-friendly. The rich-text editor supports custom embedded elements, allowing users to seamlessly insert and adjust images with captions, mathematical formulas, and structured tables directly within the document. These elements are automatically processed by the back-end conversion modules to ensure that their formatting aligns with IEEE standards.

The interface's design abstracts the underlying complexity, offering clear, interactive controls for non-text element management. For instance, users can easily add an image or a formula through dedicated toolbar buttons, with prompts to input necessary details (e.g., captions or formula expressions). The system then ensures that these elements are rendered correctly and integrated harmoniously with the text. This design choice not only enhances user experience but also guarantees that non-text elements are preserved in their intended format during the conversion process.

Overall, the front-end and text submission methods of the system are engineered to provide an effortless and responsive user experience. By combining a powerful, real-time collaborative text editor with user-friendly non-text element integration, the system streamlines the academic writing process and reduces the need for manual formatting adjustments, ensuring that every component—from text to images—is presented in a polished, IEEE-compliant manner.

```
import React, { useEffect, useRef, useState, useImperativeHandle, forwardRef, } from "react";
import Quill from "quill";
import "quill/dist/quill.snow.css";
import ImageWithCaptionBlot from "../blots/ImageWithCaptionBlot.js";
import TableWithCaptionBlot from "../blots/TableWithCaptionBlot.js";
import FormulaWithCaptionBlot from "../blots/FormulaWithCaptionBlot.js";
import ReferenceModal from "./ManualCitationModal.js";
import QuillCursors from "quill-cursors";
import debounce from "lodash.debounce";
import TableSizePicker from "./TableSizePicker";

import { Mention } from "quill-mention";
import "quill-mention/dist/quill.mention.min.css";

// quill-better-table
import QuillBetterTable from "quill-better-table";

// KaTeX for formulas
import katex from "katex";
import "katex/dist/katex.min.css";
window.katex = katex;

// Register modules and custom blots
Quill.register("modules/cursors", QuillCursors);
Quill.register(ImageWithCaptionBlot);
Quill.register(TableWithCaptionBlot);
Quill.register(FormulaWithCaptionBlot);
Quill.register("modules/better-table", QuillBetterTable, true);
Quill.register({ "modules/mention": Mention }, true);

// Predefined IEEE sections
const COMMON_IEEE_SECTIONS = [
  "Introduction",
  "Related Work / Literature Review",
  "Methodology / Proposed Method",
  "Results / Discussion",
  "Conclusion (and possibly Future Work)",
];
```

Figure 19: React Imports used in editor

```
// Example mention data
const atValues = [
  { id: 1, value: "Fredrik Sundqvist" },
  { id: 2, value: "Patrik Sjölin" },
];
const hashValues = [
  { id: 3, value: "React" },
  { id: 4, value: "Quill" },
];

// (Optional) Build tableOptions array if needed for a dropdown
const maxRows = 7;
const maxCols = 7;
const tableOptions = [];
for (let r = 1; r <= maxRows; r++) {
  for (let c = 1; c <= maxCols; c++) {
    tableOptions.push(`newtable_${r}_${c}`);
  }
}

// Custom icons for our custom toolbar buttons
const icons = Quill.import("ui/icons");
icons["customTable"] = `
  <svg viewBox="0 0 18 18" width="18" height="18">
    <rect class="ql-stroke" height="12" width="12" x="3" y="3"></rect>
    <line class="ql-stroke" x1="3" y1="9" x2="15" y2="9"></line>
    <line class="ql-stroke" x1="9" y1="3" x2="9" y2="15"></line>
  </svg>
`;
icons["customFormula"] = `
  <svg viewBox="0 0 18 18" width="18" height="18">
    <path class="ql-stroke" d="M5 5L13 13"></path>
    <path class="ql-stroke" d="M13 5L5 13"></path>
  </svg>
`;

// Toolbar configuration – every group is an array.
// Two custom buttons are added: one for table insertion and one for formula insertion.
const toolbarOptions = [
  [{ header: [1, 2, 3, false] }],
  ["bold", "italic", "underline"],
  [{ list: "ordered" }, { list: "bullet" }],
  ["image", "blockquote", "code-block"],
  [{ align: [] }],
  [{ indent: "-1" }, { indent: "+1" }],
  [{ customTable: true }, { customFormula: true }],
  ["clean"],
];
```

Figure 20:Custom Toolbar code

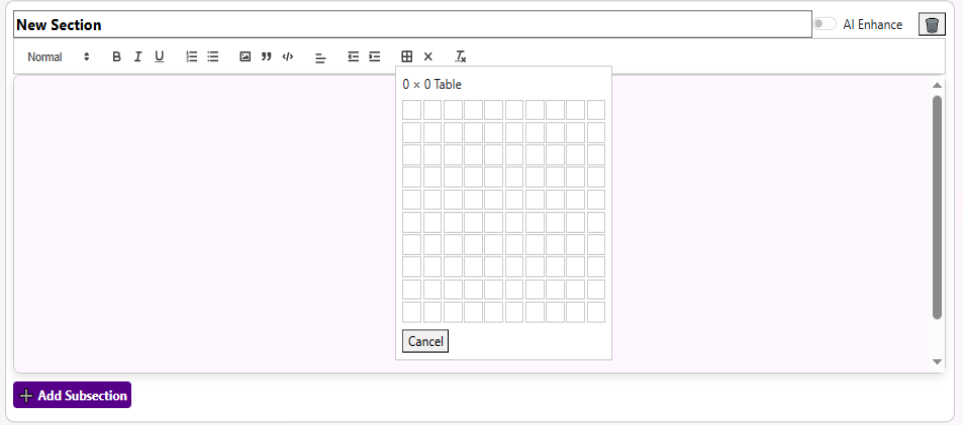


Figure 21:Quil editor with custom toolbar

```
// 1) Initialize Quill editors for each node.
useEffect(() => {
  console.log("📌 Sections in Editor:", sections);
  function initQuillForNode(node, parentId = null) {
    if (!quillRefs.current[node.contentId]) {
      const quill = new Quill(`#editor-${node.contentId}`, {
        theme: "snow",
        modules: {
          "better-table": {
            operationMenu: {
              items: {
                insertRowAbove: { text: "Insert row above" },
                insertRowBelow: { text: "Insert row below" },
                deleteRow: { text: "Delete row" },
                insertColumnLeft: { text: "Insert column left" },
                insertColumnRight: { text: "Insert column right" },
                deleteColumn: { text: "Delete column" },
                unmergeCells: { text: "Unmerge cells" },
              },
              color: {
                colors: ["green", "red", "yellow", "blue", "white"],
                text: "Background Colors:",
              },
            },
          },
          keyboard: {
            bindings: QuillBetterTable.keyboardBindings,
          },
          mention: {
            mentionDenotationChars: ["@", "#"],
            source: function (searchTerm, renderList, mentionChar) {
              let values = mentionChar === "@" ? atValues : hashValues;
              if (searchTerm.length === 0) {
                renderList(values, searchTerm);
              } else {
                const matches = values.filter(val =>
                  val.value.toLowerCase().includes(searchTerm.toLowerCase())
                );
                renderList(matches, searchTerm);
              }
            },
          },
          formula: true,
          cursors: {},
          toolbar: {
            container: toolbarOptions,
            handlers: {
              image: async function () {
                const input = document.createElement("input");
                input.setAttribute("type", "file");
                input.setAttribute("accept", "image/*");
                input.click();
                input.onchange = async () => {
                  const file = input.files?.[0];
                  if (!file) return;
                  const formData = new FormData();
                  formData.append("image", file);
                  const res = await fetch(
                    `${process.env.REACT_APP_BACKEND_API_URL}/api/pads/uploads`,
                    { method: "POST", body: formData }
                  );
                  const data = await res.json();
                  if (!data.url) return;
                  const caption = prompt("Enter image caption") || "";
                  const range = this.quill.getSelection() || { index: this.quill.getLength() };
                  this.quill.insertEmbed(range.index, "imageWithCaption", { src: data.url, caption });
                  this.quill.setSelection(range.index + 1, 0);
                  const fullContent = this.quill.getContents();
                  const cursor = { index: range.index + 1, length: 0 };
                  if (parentId === null) {
                    socket.emit("send-changes", { padId, sectionId: node.id, fullContent, userId, cursor });
                  } else {
                    socket.emit("send-changes", { padId, sectionId: parentId, subId: node.id, fullContent, userId, cursor });
                  }
                };
              },
              // Bind our custom table and formula handlers
              customTable: handleCustomTable,
              customFormula: function () {
                insertFormulaWithCaption(this.quill);
              },
              // (Optional) Handler for dropdown table insertion if needed:
              "Table-Input": function (value) {
                if (value) {
                  const parts = value.split("_");
                  const rows = parseInt(parts[1], 10);
                  const cols = parseInt(parts[2], 10);
                  this.quill.getModule("better-table").insertTable(rows, cols);
                }
              },
            },
          },
        },
      });
```

Figure 22:Collaborative editor code

```
// [ADDED] Helper to attach blur listeners to all <td> in the newly inserted table
function attachTableCellListeners(quill, embedIndex) {
  // Wait a tick so Quill updates the DOM
  setTimeout(() => {
    // The editor DOM we just inserted the table into
    const editorEl = quill.root.parentNode;
    // Grab the figure for the newly inserted table
    // embedIndex is where we inserted the table in the Delta
    const tableFigure = editorEl.querySelector(
      `figure.ql-table-with-caption:nth-of-type(${embedIndex + 1})`
    );
    if (!tableFigure) return;

    // For every <td>, on blur => re-insert the table
    const cells = tableFigure.querySelectorAll("td");
    cells.forEach((cell) => {
      cell.addEventListener("blur", () => {
        // Read the updated HTML from the figure
        const wrapper = tableFigure.querySelector(".table-wrapper");
        const newTableHtml = wrapper ? wrapper.innerHTML : "";
        const captionEl = tableFigure.querySelector("figcaption");
        const newCaption = captionEl ? captionEl.innerText : "";

        // Remove the old embed
        const blot = Quill.find(tableFigure);
        if (!blot) return;
        const oldIndex = quill.getIndex(blot);

        quill.deleteText(oldIndex, 1);
        // Insert updated embed with new HTML
        quill.insertEmbed(oldIndex, "tableWithCaption", {
          tableHtml: newTableHtml,
          caption: newCaption,
        });
        // Insert a newline to keep spacing
        quill.insertText(oldIndex + 1, "\n");
        quill.setSelection(oldIndex + 2, 0);
      });
    });
  }, 0);
}

// Socket listeners for remote changes and loading the pad remain unchanged
useEffect(() => {
  socket.on(
    "receive-changes",
    ({ sectionId, subId, fullContent, userId: senderId, cursor }) => {
      const nodeId = subId || sectionId;
      const contentId = getNodeContentId(sections, nodeId);
      if (contentId && quillRefs.current[contentId]) {
        const quill = quillRefs.current[contentId];
        const currentSelection = quill.hasFocus() ? quill.getSelection() : null;
        const localDelta = quill.getContents();
        const Delta = Quill.import("delta");
        const diffDelta = localDelta.diff(new Delta(fullContent.ops));
        quill.updateContents(diffDelta);
        if (currentSelection) {
          quill.setSelection(currentSelection.index, currentSelection.length);
        }
      }
    }
  );

  socket.on("remote-cursor", ({ userId: remoteUserId, cursor, color, nodeId }) => {
    if (remoteUserId === userId) return;
    Object.keys(quillRefs.current).forEach((contentId) => {
      const quill = quillRefs.current[contentId];
      const cursors = quill.getModule("cursors");
      if (cursors) {
        cursors.removeCursor(remoteUserId);
      }
    });
    const contentId = getNodeContentId(sections, nodeId);
    if (contentId && quillRefs.current[contentId]) {
      const cursors = quillRefs.current[contentId].getModule("cursors");
      if (cursors) {
        cursors.createCursor(remoteUserId, remoteUserId, color);
        cursors.moveCursor(remoteUserId, cursor);
      }
    }
  });

  socket.on(
    "load-pad",
    ({ sections: newSecs, authors: newAuthors, references: newRefs, title, abstract: abs, keyword }) => {
      setSections(newSecs || []);
      setAuthors(newAuthors || []);
      setReferences(newRefs || []);
      setPaperTitle(title || "");
      setAbstract(abs || "");
      setKeywords(keyword || "");
    }
  );

  return () => {
    socket.off("receive-changes");
    socket.off("load-pad");
    socket.off("remote-cursor");
  };
}, [socket, sections, setSections, setAuthors, setReferences, userId]);

// Top-level field update handlers remain unchanged
const handleTitleChange = (e) => {
  const newVal = e.target.value;
  setPaperTitle(newVal);
  setDebouncedTitle(newVal);
  socket.emit("update-pad", {
    padId,
    sections,
    authors,
    references,
    title: newVal,
    abstract,
    keyword: keywords,
  });
};
```

Figure 23: WebSocket handing for collaborative changes

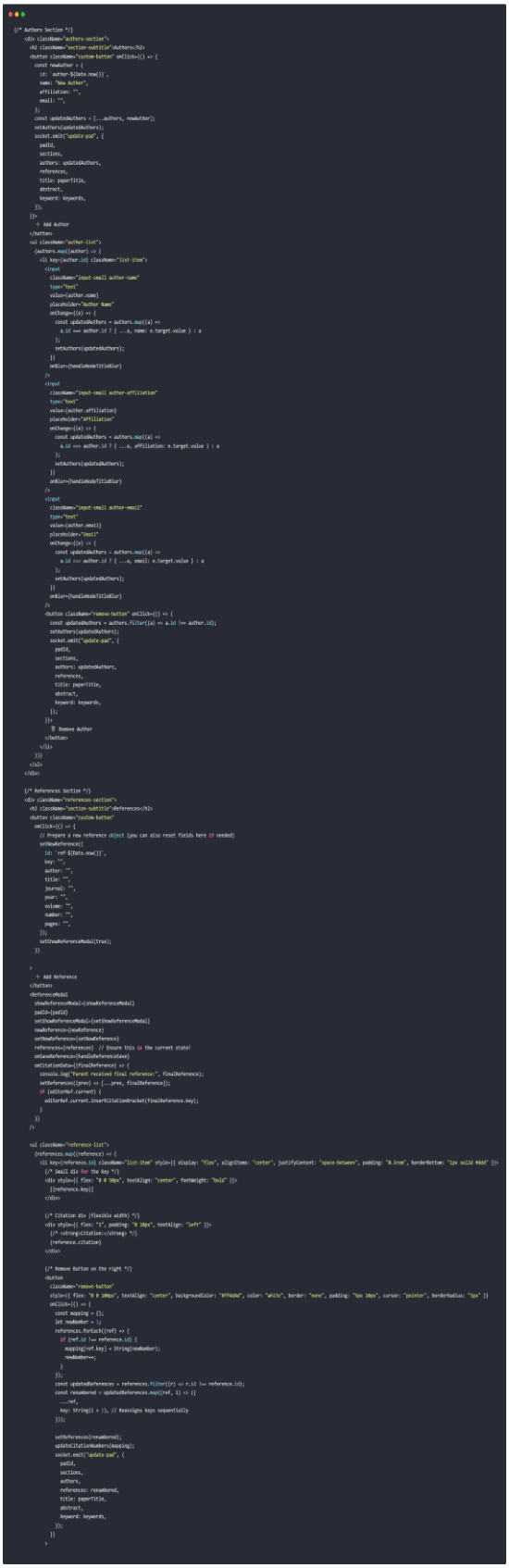


Figure 24: Editor Html elements

```
// TableWithCaptionBlot.js
import Quill from "quill";
const BlockEmbed = Quill.import("blots/block/embed");

class TableWithCaptionBlot extends BlockEmbed {
  static create(value) {
    const node = super.create();
    node.classList.add("ql-table-with-caption");

    // Create a wrapper div to ensure table structure stays intact
    const tableWrapper = document.createElement("div");
    tableWrapper.classList.add("table-wrapper");  // Ensure it's identifiable
    tableWrapper.innerHTML = value.tableHtml || "";

    // Ensure the table cells remain editable
    const cells = tableWrapper.querySelectorAll("td");
    cells.forEach((cell) => {
      cell.setAttribute("contenteditable", "true");

       // Stop Quill from seeing backspace or other keystrokes
      //  cell.addEventListener(
      //   "keydown",
      //   (e) => {
      //     // If user presses Backspace or Delete,
      //     // we fully stop the event from reaching Quill.
      //     if (e.key === "Backspace" || e.key === "Delete") {
      //       e.stopPropagation();
      //       e.stopImmediatePropagation();
      //     }
      //   },
      //   true // capture phase
      // );
    });

    node.appendChild(tableWrapper);

    // Optional caption
    const figcaption = document.createElement("figcaption");
    figcaption.innerText = value.caption || "";
    node.appendChild(figcaption);

    return node;
  }

  static value(node) {
    const tableWrapper = node.querySelector(".table-wrapper");
    const caption = node.querySelector("figcaption")?.innerText || "";
    return {
      tableHtml: tableWrapper ? tableWrapper.innerHTML : "",
      caption,
    };
  }
}

TableWithCaptionBlot.blotName = "tableWithCaption";
TableWithCaptionBlot.tagName = "figure";
TableWithCaptionBlot.className = "ql-table-with-caption";

export default TableWithCaptionBlot;
```

```
import Quill from "quill";

const BlockEmbed = Quill.import("blots/block/embed");

/**
 * A blot that stores {src, caption} in the Delta,
 * but only displays <img> on screen (no visible caption).
 */
class ImageWithCaptionBlot extends BlockEmbed {
  static create(value) {
    /**
     * value = { src, caption }
     */
    const node = super.create(); // This creates <figure> by default
    node.classList.add("ql-image-with-caption");

    // Create the <img>
    const img = document.createElement("img");
    img.setAttribute("src", value.src || "");
    img.style.width = "500px"; // Fixed width
    img.style.height = "auto"; // Maintain aspect ratio
    node.appendChild(img);

    // We do NOT render the caption visually.
    // We only store it in a data attribute (or skip if you prefer).
    if (value.caption) {
      node.setAttribute("data-caption", value.caption);
    }

    return node;
  }

  static value(node) {
    const img = node.querySelector("img");
    const src = img?.getAttribute("src") || "";
    const caption = node.getAttribute("data-caption") || "";
    return { src, caption };
  }
}

// Required Quill props
ImageWithCaptionBlot.blotName = "imageWithCaption";
// We'll use <figure> as the element type so it's treated like a block
ImageWithCaptionBlot.tagName = "figure";

export default ImageWithCaptionBlot;
```

Figure 25:Non text element blots

2.1.5 Testing

The testing process I've undertaken for the component, which includes unit testing, integration testing, system testing, and acceptance testing. These testing phases are crucial for ensuring the reliability and effectiveness of your component within the broader application context, "Sparkle Edu."

Unit Testing:

- ✓ Unit testing is the foundation of your testing process and focuses on isolating individual components or functions within your code.
- ✓ In your case, you've created a suite of unit tests that assess specific functionalities and behaviors of your component. These tests include checking routes, responses, and expected system states.
- ✓ Unit tests help ensure that your component's fundamental building blocks work as intended and that changes to these blocks do not introduce errors.

Integration Testing:

- ✓ Integration testing assesses the interactions and compatibility between your component and other components or services it interacts with.
- ✓ You've mentioned integrating your component with other components, such as an AI mental health chatbot, facial easy authentication with course recommendation, and document reconstruction with OCR.
- ✓ Integration testing verifies that these interactions are seamless and error-free. It ensures that data flows correctly between components and that they collectively perform their intended functions.

System Testing:

- ✓ System testing takes a broader view, examining the behavior of your component within the context of the entire web application, "Sparkle Edu."
- ✓ During system testing, you've tested all components together to ensure they work harmoniously as a unified system.

- ✓ This phase verifies that your component, along with the other integrated components, functions correctly and consistently in a real-world scenario.

Acceptance Testing:

- ✓ Acceptance testing is the final phase that involves end-users evaluating your component and providing feedback.
- ✓ You've conducted alpha testing within your environment, involving students and teachers as end-users. This allows you to gather valuable insights, identify potential issues, and make improvements.
- ✓ Beta testing extends the evaluation to the users' own environments, providing a more realistic assessment of your component's performance and usability.
- ✓ Acceptance testing ensures that your component aligns with user expectations and addresses their needs effectively.

By following this comprehensive testing process, I've taken significant steps to validate and refine the component's functionality, performance, and user experience. This rigorous testing approach not only enhances the quality of the component but also contributes to the overall success of "Sparkle Edu" by ensuring that all integrated components work seamlessly together to deliver a valuable educational platform.

For the **Unit Testing**, the python's *unittest* library has been used to automate the important functionalities. Figure 26 illustrates the unit testing script for functionality.

```python
# test_conversion.py

import re
import pytest
import torch
from fastapi.testclient import TestClient
from pydantic import BaseModel
import nest_asyncio
import threading

# Import the necessary objects from your main application file.
# Assuming your FastAPI app and helper functions are in a file named `main.py`
from main import app, remove_citations, reinsert_citations_semantic, alpaca_prompt, tokenizer, model

# Apply nest_asyncio in test environment (if needed)
nest_asyncio.apply()

# Create a TestClient for the FastAPI app
client = TestClient(app)

##############################
# Unit Tests for Helper Functions
##############################

def test_remove_citations():
    # Test that citations are removed from the text
    input_text = "This research builds on previous studies [1] and supports findings [2]."
    expected_text = "This research builds on previous studies  and supports findings ."
    output = remove_citations(input_text)
    # Normalize whitespace before comparison
    assert re.sub(r'\s+', ' ', output).strip() == re.sub(r'\s+', ' ', expected_text).strip()

def test_reinsert_citations_semantic():
    # Use a simple simulation: Given an input text with citations and a generated response without them,
    # check that the citations are reinserted.
    original_text = "Previous work [1] laid the foundation for this study, which was expanded upon in [2]."
    generated_text = "Previous work laid the foundation for this study, which was expanded upon in further research."
    output = reinsert_citations_semantic(original_text, generated_text)
    # Expect that all citation markers appear in the final text.
    for citation in ["[1]", "[2]"]:
        assert citation in output
```

Figure 26:Unit Test Script for text conversion model

```
// test/ieeeRoutes.test.js

const chai = require('chai');
const expect = chai.expect;
const request = require('supertest');
const express = require('express');
const sinon = require('sinon');
const axios = require('axios');

// Import the router and your Pad model
const router = require('../routes/ieeeRoute');
const Pad = require('../models/Pad');

// Create an express app and attach middleware and the router for testing
const app = express();
app.use(express.json());
app.use('/api/pads', router);

// Sample Pad document for testing (using an in-memory stub)
const samplePad = {
  _id: 'sample123',
  title: 'Sample Document',
  authors: [
    { name: 'John Doe', affiliation: 'University X', email: 'john.doe@example.com' },
    { name: 'Jane Smith', affiliation: 'Institute Y', email: 'jane.smith@example.com' }
  ],
  abstract: 'This is a sample abstract.',
  keyword: 'IEEE, AI, Academic Writing',
  sections: [],
  image_path: "default_image_path.jpg",
  references: []
};

// Stub the Pad.findById method to simulate database operations
before(() => {
  sinon.stub(Pad, 'findById').callsFake((id) => {
    if (id === samplePad._id) {
      return { exec: () => Promise.resolve(samplePad) };
    } else {
      return { exec: () => Promise.resolve(null) };
    }
  });
});

after(() => {
  Pad.findById.restore();
});
```

Figure 27:Backend Integration test script 1

```
describe('IEEE Routes', function() {

  describe('GET /api/convert/:padId', function() {

    it('should return a pad when a valid padId is provided', function(done) {
      request(app)
        .get('/api/pads/sample123')
        .expect(200)
        .end(function(err, res) {
          if (err) return done(err);
          expect(res.body).to.have.property('title', samplePad.title);
          expect(res.body).to.have.property('abstract', samplePad.abstract);
          done();
        });
    });

    it('should return 404 if pad not found', function(done) {
      request(app)
        .get('/api/pads/nonexistentId')
        .expect(404)
        .end(function(err, res) {
          if (err) return done(err);
          expect(res.body).to.have.property('msg', 'Pad not found');
          done();
        });
    });
  });

  describe('POST /api/convert/convert-text', function() {

    it('should return 400 if content is empty', function(done) {
      request(app)
        .post('/api/pads/convert-text')
        .send({ content: '' })
        .expect(400)
        .end(function(err, res) {
          if (err) return done(err);
          expect(res.body).to.have.property('msg', 'No text provided for conversion.');
          done();
        });
    });

    it('should return converted text for valid content', function(done) {
      // Stub the axios.post call that is made to the external AI API
      const fakeResponse = { data: { converted_text: "Converted academic text." } };
      const axiosStub = sinon.stub(axios, 'post').resolves(fakeResponse);

      request(app)
        .post('/api/convert/convert-text')
        .send({ content: 'Informal academic text.' })
        .expect(200)
        .end(function(err, res) {
          // Restore the stub after test execution
          axiosStub.restore();
          if(err) return done(err);
          expect(res.body).to.have.property('converted_text');
          expect(res.body.converted_text).to.equal("Converted academic text.");
          done();
        });
    });
  });
});
```

Figure 28: Backend Integration test script 2

```python
##############################
# Integration Tests for API Endpoint (/convert)
##############################

class DummyTokenizer:
    def __call__(self, texts, return_tensors="pt"):
        # Return a dummy tensor; the actual value is not important.
        return {"input_ids": torch.tensor([[101, 102, 103]])}

    def batch_decode(self, outputs, skip_special_tokens=True):
        # Return a simulated conversion response containing the "### Response:" marker
        return ["Some initial text\n### Response: Converted academic text."]

class DummyModel:
    def generate(self, **kwargs):
        # Return a dummy tensor that will be passed to the tokenizer.
        return torch.tensor([[101, 102, 103, 104]])

# Integration test for valid text conversion
def test_convert_endpoint_valid(monkeypatch):
    # Monkey-patch tokenizer and model so that we do not call the actual model
    dummy_tokenizer = DummyTokenizer()
    dummy_model = DummyModel()

    monkeypatch.setattr("main.tokenizer", dummy_tokenizer)
    monkeypatch.setattr("main.model", dummy_model)

    payload = {
        "section": "Introduction",
        "content": "Informal academic text with citation [1] and some more text [2]."
    }

    response = client.post("/convert", json=payload)
    assert response.status_code == 200, "Expected 200 OK response"
    data = response.json()
    assert "converted_text" in data, "Response JSON must contain 'converted_text'"

    # Check that the converted text contains our dummy converted content.
    # Also, since reinsert_citations_semantic is applied, citations should appear in the output.
    assert "Converted academic text." in data["converted_text"]
    for citation in ["[1]", "[2]"]:
        assert citation in data["converted_text"]

# Integration test for error handling: empty content.
def test_convert_endpoint_empty():
    payload = {
        "section": "Abstract",
        "content": ""  # Empty content should trigger a 400 error.
    }
    response = client.post("/convert", json=payload)
    assert response.status_code == 400, "Expected 400 for empty content"
    data = response.json()
    assert "msg" in data, "Error response should contain a 'msg' key"
    assert data["msg"] == "No text provided for conversion."

if __name__ == "__main__":
    pytest.main()
```

Figure 29 :Integration test script(main.py)

Manual Testing:

Manual testing is an essential part of the software testing process, where human testers evaluate the application's functionality, usability, and performance by simulating real-world end-user interactions. The purpose of manual testing in this research is to verify that the AI-based IEEE format conversion system meets all specified requirements, including automated text conversion, citation management, selective AI enhancement, and document export. Testers use their domain knowledge to explore the system's features, identify any defects or issues, and ensure that the output meets IEEE-compliant academic standards

Purpose of Manual Testing:

- ✓ Validate the system's capability to automatically convert informal academic text into properly structured and formatted academic writing.
- ✓ Ensure that all citation markers present in the original text are accurately reinserted in the converted output.
- ✓ Test the functionality of the AI enhancement toggle, enabling users to selectively apply or bypass automated conversion.
- ✓ Evaluate the export functionality to confirm that the final, submission-ready document (PDF) is generated correctly
- ✓ Assess overall usability, performance, and the user experience by simulating typical user workflows.
- ✓ Test Case Development:
- ✓ Test cases are formulated based on the system requirements and user stories.
- ✓ Each test case includes step-by-step instructions, pre-requirements, input data, expected outcomes, and actual outcomes.
- ✓ The test cases cover both functional aspects (conversion quality, citation handling, export accuracy) and non-functional aspects (usability, responsiveness, and collaborative editing support).

Execution:

- ✓ Testers manually execute each test case by interacting with the system through the user interface.
- ✓ They record observations, note any discrepancies from expected results, and mark each test as pass or fail
- ✓ Different types of manual testing are performed, including functional testing, non-functional testing (usability and performance), user acceptance testing (UAT), and exploratory testing to uncover edge cases.
- ✓ Types of Manual Testing:
- ✓ Functional Testing: Evaluate whether the software functions correctly according to specifications.
- ✓ Non-Functional Testing: Focuses on non-functional aspects like performance, security, and scalability.
- ✓ User Acceptance Testing (UAT): Performed by end-users to ensure the software meets business requirements.
- ✓ Exploratory Testing: Testers explore the application without predefined test cases.
- ✓ Compatibility Testing: Checks how the software performs across different browsers, devices, or platforms.

Below are the test cases which were done for the manual testing. Tables 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13 display the manual test cases.

Table 4: Test for text conversion

Test Case ID	TC_01
Test Case Objective	Test the Automated Text Conversion functionality.
Pre-Requirements	Users must be logged into the system; conversion interface is accessible.
Test Steps	1. Log in to the system. 2. Navigate to the conversion interface. 3. Enter an informal academic text that includes citation markers.

	4. Click "Convert".
Test Data	Informal academic text with citations (e.g., "This study builds upon previous work [1] and uses insights from [2].")
Expected Output	The input text is converted into structured, IEEE-compliant academic writing with grammatical corrections and proper citation preservation.
Actual Output	Informal text has been converted to formal academic tone with IEEE compliance and citations are maintained with same meaning.
Status	Pass

Table 5: Test for citation toggle

Test Case ID	TC_02
Test Case Objective	Validate that all citations from the input are correctly reinserted into the final output.
Pre-Requirements	Users must provide input text containing multiple citation markers.
Test Steps	1. Log in to the system. 2. Enter informal academic text with citations (e.g., "[1]", "[2]", "[3]"). 3. Click "Convert".
Test Data	"This paper builds upon previous work [1] and incorporates findings [2] along with additional insights [3]."
Expected Output	The converted academic text should include all citation markers ([1], [2], [3]) in appropriate positions, preserving the context of the original input.
Actual Output	All the citations in input text is in output text also and preserved with same meaningful sentences.
Status	Pass

Table 6: Test for AI enhancement toggle

Test Case ID	TC_03
Test Case Objective	Test the functionality of toggling AI enhancement on and off for specific sections.
Pre-Requirements	Users must be logged into the system and have access to the conversion interface that offers the AI enhancement toggle.
Test Steps	1. Log in to the system. 2. Select a specific document section. 3. Toggle the AI enhancement off and note the resulting text. 4. Toggle the AI enhancement on and reprocess the text.
Test Data	Informal academic text segments.
Expected Output	When toggled off, the document remains what use given; when toggled on, the text is refined into formal IEEE-compliant academic writing.
Actual Output	Toggled specific section is text has been converted to IEEE compliant academic content.
Status	Pass

Table 7:Test for User interface

Test Case ID	TC_04
Test Case Objective	Evaluate overall user interface functionality, ensuring that the system is intuitive, responsive, and supports seamless interactions for academic writing conversion.
Pre-Requirements	User must be logged into the system; the conversion interface and AI enhancement controls must be functional.
Test Steps	1. Log in to the system. 2. Navigate to a document section. 3. Insert an image with a caption (e.g., via a custom toolbar button). 4. Trigger the conversion process.
Test Data	Various samples of informal academic text (with and without citations).
Expected Output	The user interface should be intuitive, with clear controls for text submission and toggling AI enhancement, providing a smooth, responsive user experience throughout the conversion process.
Actual Output	Every function in user interface was working correctly and responsive in different size of screen.
Status	Pass

Table 8: Test Image insertion

Test Case ID	TC_05
Test Case Objective	Verify that an image embedded with a caption in the input is rendered correctly in the output document.
Pre-Requirements	The user must be logged into the system and have access to the conversion interface. The input should include an image embed using the designated method (e.g., op.insert.imageWithCaption).
Test Steps	1. Log in to the system. 2. Navigate to a document section. 3. Insert an image with a caption (e.g., via a custom toolbar button).

	4. Trigger the conversion process.
Test Data	Verify that the output PDF shows the image in a figure environment with the correct caption, in the proper order relative to text.
Expected Output	Example input text:
Actual Output	Image has been added in the document with captions with perfect IEEE compliant format
Status	Pass

Table 9: Test for table insertion

Test Case ID	TC_06
Test Case Objective	Verify that a table embedded with a caption is correctly rendered and appears in the correct order in the output document.
Pre-Requirements	The user must be logged into the system. The input should include a table embed using op.insert.tableWithCaption.
Test Steps	1. Log in to the system. 2. Navigate to a document section and insert a table embed with a defined caption 3. Trigger the conversion process.
Test Data	Created a table with User interface of 2 columns and 2 rows: "Data is organized in the following table: [Table embed: tableHtml='<table><tbody><tr><td>Data1</td><td>Data2</td></tr></tbody></table>', caption='Sample Table 1']."
Expected Output	The final output must include a correctly formatted table with the caption "Sample Table", placed in the proper sequence within the document, and maintaining the integrity of the data.
Actual Output	Everything updated in correct order
Status	Pass

Table 10: Test for non-text element order

Test Case ID	TC_07
Test Case Objective	Ensure that the order of text and non-text elements (images, tables, formulas with captions) in a section is preserved correctly in the final output document.
Pre-Requirements	The user must be logged into the system and have access to the conversion interface. The input document must include a mixture of text and embedded non-text elements inserted using the designated methods (e.g., op.insert.imageWithCaption, op.insert.tableWithCaption, op.insert.formulaWithCaption).
Test Steps	1. Log in to the system. 2. Navigate to a document section. 3. Enter a block of informal academic text that includes a mix of text and non-text elements in a specific order (e.g., text, then an image with caption, followed by more text, then a table with caption, and finally a formula with caption).
Test Data	All the non-text element added between the text element in one section
Expected Output	The output document must display all elements in the exact order as the input: 1. Introductory paragraph text. 2. Image with the caption "Sample Image Caption", embedded in the correct figure format. 3. Continuing text after the image. 4. A table with the caption "Sample Table", formatted properly as a table. 5. Further discussion text following the table.

	6. A formula with the caption "Mass-Energy Equivalence", rendered in an equation environment. 7. Concluding remarks should appear after the formula.
Actual Output	Input sequence of text and non-text elements has been maintained in output as well.
Status	Pass

2.1.6 Deployment & Maintenance

Deployment:

The AI-based IEEE format conversion system is deployed on an Azure Virtual Machine running Ubuntu 24.04. Instead of using Colab, the application is set up directly on the VM by installing Python and all necessary dependencies via pip. The deployment process involves the following steps:

1. VM Provisioning and Environment Setup:

Basics

Subscription	Azure for Students
Resource group	(new) IEEE-text-modal
Virtual machine name	Llama-2-7b
Region	Central India
Availability options	Availability zone
Zone options	Self-selected zone
Availability zone	1
Security type	Trusted launch virtual machines
Enable secure boot	Yes
Enable vTPM	Yes
Integrity monitoring	No
Image	Ubuntu Server 24.04 LTS - Gen2
VM architecture	x64
Size	Standard E4as v4 (4 vcpus, 32 GiB memory)
Enable Hibernation	No
Authentication type	SSH public key
Username	azureuser
SSH Key format	RSA
Key pair name	Llama-2-7b_key
Public inbound ports	SSH
Azure Spot	No

Figure 30:Azure VM

- ✓ An Azure VM with the Standard E4as v4 configuration (featuring 4 vCPUs, 32 GiB memory, and 64 GiB storage) is provisioned. This ensures robust

performance for both model inference and API processing. Secure connections are established using SSL to protect data communications.

2. Dependency Installation:

✓ All required Python packages are installed directly on the VM. Installation commands include.

pip install *

3. Model Loading and Integration:

✓ The AI model is deployed by directly downloading it from Hugging Face. In this case, the Llama 2-7B model with a LoRA adapter is loaded using the Unsloth framework, employing 4-bit quantization to reduce memory overhead. Environment variables (such as the Hugging Face token) and caching directories are configured to facilitate model loading and efficient inference. The FastAPI application loads the model and tokenizer at startup, making them available for processing conversion requests.

4. Application Configuration and Server Launch:

✓ The FastAPI app is configured with the necessary middleware and routes (as defined in your backend code) for text conversion. The application uses Uvicorn as the ASGI server, which is launched on port 8000. With SSL in place, this ensures secure and reliable communication with end users. The server is started via a dedicated script, running continuously on the Azure VM to handle incoming requests.

5. NGINX Reverse Proxy Configuration:

✓ To expose the application without exposing a port number and improve security, NGINX is configured as a reverse proxy. The reverse proxy listens on the default HTTP/HTTPS ports (80/443) and forwards incoming requests to the FastAPI server running on port 8000.

6. Scaling and Optimization:

- ✓ Configure security measures such as Azure Active Directory for authentication and authorization. Implement SSL certificates for secure communication. Set up network security groups to control inbound and outbound traffic.

7. Monitoring and Logging:

- ✓ Azure provides monitoring and logging services like Azure Monitor and Azure Application Insights. Configure these services to gain insights into your application's performance and diagnose issues.

8. Security and Networking:

- ✓ Additional security measures, such as firewall rules, network security groups, and secured SSL/TLS certificates, are implemented to ensure that data transmitted to and from the application is encrypted and that only authorized traffic is allowed.

9. Backup and Recovery:

- ✓ Implemented backup and recovery strategies to protect your data and application. Azure offers services like Azure Backup.

10. Testing and Validation:

- ✓ Before going live, thoroughly test your application in the Azure VM environment. Ensure that all components, including your component, work as expected.

11. Go Live:

- ✓ Once you're confident that your application is ready, switch your custom domain to point to the Azure Web App and make your application accessible to users.

12. Continuous Monitoring:

- ✓ After deployment, continuously monitor your application's performance and security. Azure's monitoring and logging tools will help you keep an eye on your application's health.

Deployment of WriteWizard MERN application

The collaborative document editing platform WriteWizard, which includes a React frontend and an Express.js backend, was deployed together on a Microsoft Azure Virtual Machine using Docker and Nginx. This deployment supports user interactions and real-time collaborative editing, integrating with the microservices hosted externally.

1. Azure VM Setup

A virtual machine was provisioned via the Azure portal using Ubuntu Server 24.04 LTS, configured as a Standard E2as v4 instance with 2 vCPUs and 16 GiB memory. Inbound port rules were added to allow traffic through ports 80 and 4000, which were used for serving the frontend and backend respectively.

2. Dockerization and Deployment

The frontend and backend were containerized using separate Dockerfiles, and a unified Docker Compose configuration (docker-compose.yml) was used to orchestrate both containers.

Figure 31 shows the structure of the docker-compose.yml file used to manage service containers, environment variables, and port bindings.

docker-compose.yml

```yaml
version: "3.9"

services:
  backend:
    image: sanjayan201/writewizard-backend:latest
    container_name: writewizard-backend
    build:
      context: ./BackEnd
    env_file:
      - ./BackEnd/.env
    ports:
      - "4001:4000"
    restart: always

  frontend:
    image: sanjayan201/writewizard-frontend:latest
    container_name: writewizard-frontend
    build:
      context: ./FrontEnd
    ports:
      - "8080:80"
    restart: always
```

Figure 31: Docker Compose File for Frontend and Backend Service Orchestration

Docker images were built locally and pushed to Docker Hub. On the VM, the images were pulled and launched using the docker-compose up -d command, allowing both services to run simultaneously in detached mode.

3. Reverse Proxy with Nginx

To handle routing and provide seamless public access, Nginx was installed and configured as a reverse proxy. HTTP traffic on port 80 was directed to the front end running on port 8080, while backend API traffic on port 4000 was routed internally to port 4001.

Figure 53 shows the frontend application successfully running on the browser through Nginx routing, and Figure 33 shows the backend API responding correctly after reverse proxy configuration.

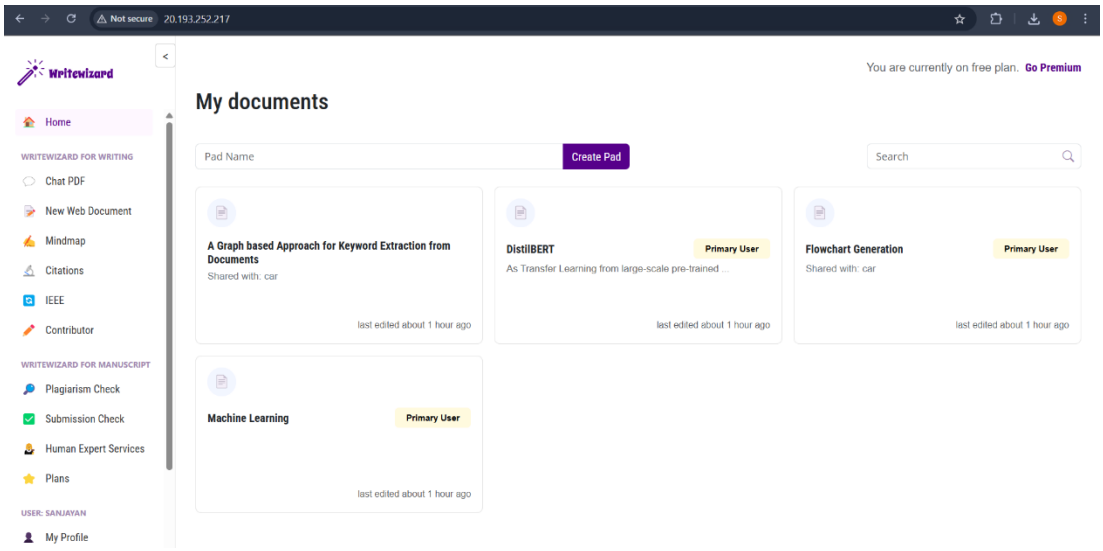


Figure 32: React Frontend of WriteWizard Running via Nginx Routing

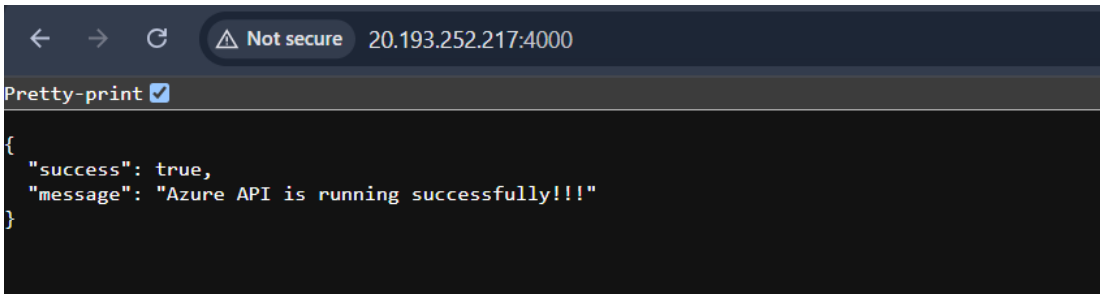


Figure 33: Express.js Backend API Response After Nginx Proxy Configuration

4. Continuous Deployment with GitHub Actions

A CI/CD pipeline was established using GitHub Actions to streamline deployment and updates. A YAML-based workflow (docker-deploy.yml) was configured to automate Docker builds, SSH into the VM, and redeploy the updated containers.

Figure 34 shows a portion of the file used to automate deployment steps from GitHub to the Azure VM.

docker-deploy.yml

```yaml
name: Build and Deploy WriteWizard

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      # 1) Check out your repository
      - name: Check out code
        uses: actions/checkout@v3

      # 2) Log in to Docker Hub
      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_PASSWORD }}

      # 3) Build and push the backend image
      - name: Build and push backend image
        run: |
          docker build -t ${{ secrets.DOCKERHUB_USERNAME }}/writewizard-backend:latest ./BackEnd
          docker push ${{ secrets.DOCKERHUB_USERNAME }}/writewizard-backend:latest

      # 4) Build and push the frontend image
      - name: Build and push frontend image
        run: |
          docker build -t ${{ secrets.DOCKERHUB_USERNAME }}/writewizard-frontend:latest ./FrontEnd
          docker push ${{ secrets.DOCKERHUB_USERNAME }}/writewizard-frontend:latest

  deploy:
    runs-on: ubuntu-latest
    needs: build-and-push
    steps:
      - name: Deploy to Azure VM
        uses: appleboy/ssh-action@v0.1.7
        with:
          host: ${{ secrets.AZURE_VM_IP }}
          username: azureuser
          key: ${{ secrets.AZURE_SSH_PRIVATE_KEY }}
          script: |
            # Navigate to the directory containing your docker-compose.yml
            cd /home/azureuser
            # Pull the latest images
            docker-compose pull
            # Re-create or restart the containers
            docker-compose up -d
```

Figure 34: GitHub Actions Workflow for CI/CD Automation

Secrets such as the Docker Hub login, private SSH key, and remote VM IP were securely stored in the GitHub repository settings. This allowed deployment to be

triggered automatically upon commits to the main branch, reducing manual effort and increasing reliability.

Maintenance:

The maintenance phase of the AI-based IEEE format conversion system is critical to ensuring the system remains reliable, secure, and effective after initial deployment. This ongoing support phase involves continuous monitoring, updates, and enhancements to address emerging issues and improve functionality over time.

1. Bug Fixing and Issue Resolution:

Monitoring and Logging: Continuous monitoring of system logs and error reports is implemented to proactively detect any issues. Automated alerting mechanisms notify the development team of anomalies during real-time processing or model inference.

Issue Diagnosis: When bugs or performance issues are reported—whether through automated monitoring or user feedback, the team investigates and diagnoses the underlying causes. This may involve reviewing model performance, analyzing server logs, or checking integration points between the FastAPI backend, model, and EJS template processing.

Patch Deployment: Once an issue is identified, patches or updates are developed and deployed to resolve the problem promptly. This process includes validating the fix through regression testing to ensure that new updates do not impact other system functionalities.

2. Feature Enhancements:

User Feedback and Iterative Improvement: Regular collection of user feedback is integrated into the maintenance process to identify potential improvements and new feature requests. Enhancements such as additional customization options, advanced citation management, or improved text conversion nuances are prioritized based on user needs.

User Acceptance Testing (UAT): Before deploying new features or system enhancements, comprehensive user acceptance testing is carried out to confirm that the updated functionalities meet user expectations and seamlessly integrate with existing workflows.

Performance Optimization: As the usage of the system grows, periodic performance reviews and optimizations are conducted. This includes optimizing model inference speeds, reducing latency in API calls, and fine-tuning resource allocation to maintain a high-quality user experience**.**

Backup and Recovery:

Data Backup: Regular backups of the application code, configuration files, and model parameters are scheduled using automated scripts or Azure Backup services. This ensures that the system data is preserved and can be quickly restored in case of a failure.

Disaster Recovery Plans: Clear recovery procedures and contingency plans are in place to minimize downtime in the event of hardware or software failures, ensuring that the system can be promptly restored to operational status.

2.2 Commercialization

Commercializing the AI-based IEEE format conversion system involves transforming the innovative academic writing tool into a revenue-generating product that delivers value to its target market. The commercialization strategy for this research focuses on a subscription-based model with role-based permission sets, tailored to meet the diverse needs of academic professionals:

1. User Segmentation:

Target Audiences: Identify and segment target users, which include individual researchers, graduate students, academic institutions, and professional writers.

Market Analysis: Evaluate the specific challenges these segments face in producing publication-ready documents and position the product as a solution to streamline academic writing and reduce manual editing efforts.

2. Pricing Tiers:

Free Tier: Provide a basic version with limited features to attract new users. This tier allows access to core functionality (e.g., automated text conversion) but may limit advanced customization options and batch processing capabilities.

Premium Tier: Offer a comprehensive feature set that includes advanced AI enhancements, real-time collaboration, and intelligent citation management. This tier targets individual researchers and professionals who need seamless and enhanced academic writing experience.

Institutional Tier: Design a specialized subscription model for educational institutions and research organizations. This plan offers volume licensing, integration with institutional systems, and features like user management, detailed analytics, and priority support

3. Authentication Mechanism:

Implement robust user authentication to protect sensitive academic data and manage access.

Utilize industry-standard protocols such as OAuth or OpenID Connect to ensure secure, user-friendly access, which is especially critical for institutions.

4. Permission Sets:

Define granular permission sets based on user roles (e.g., admin, academic, guest) to control access to various features, such as AI enhancement tools, document sharing, and collaboration tools.
Ensure that administrative roles can manage subscriptions, billing, and user settings while regular users focus on content refinement.

5. Subscription Management:

Integrate a subscription management system to handle recurring billing, upgrades, and cancellations.

Use established payment gateways like Stripe or PayPal to ensure secure and seamless transactions.

Provide clear, user-friendly dashboards that display subscription status, billing history, and available options.

6. Advertising Integration:

Explore partnerships with educational content providers to display relevant, non-intrusive ads that complement the user experience.

Ensure that any advertisements comply with privacy regulations and do not disrupt the productivity of academic professionals.

7. Marketing and Promotion:

Develop a comprehensive marketing strategy that targets academic forums, research institutions, social media platforms, and online advertising channels.

Leverage SEO and content marketing to enhance the visibility of the product online, and consider partnering with academic influences or industry publications to build credibility.

8. Feedback and Iteration:

Establish a robust user feedback loop to gather insights from subscribers regarding feature requirements, usability, and overall satisfaction.

Use this feedback to drive continuous product improvements and to adapt the commercialization strategy to evolving market trends.

9. Expansion and Partnerships:

Explore strategic partnerships with ed-tech companies, academic publishers, and research organizations to extend market reach.

Consider integration with other academic tools (e.g., reference management systems) to offer a comprehensive suite of research support solutions.

10. Sustainability and Growth:

Develop long-term plans for scaling the product by continuously monitoring market needs and technological advancements.

3. RESULTS & DISCUSSION

The outcomes of the AI-based IEEE format conversion system have demonstrated significant improvements in the quality and efficiency of academic writing. By fine-tuning a Llama 2-7B model with advanced techniques such as 4-bit quantization and Low-Rank Adaptation (LoRA), the system is capable of transforming informal text into polished, IEEE-compliant academic writing with minimal manual intervention.

1. Text Conversion Quality:

Evaluation of the system's text conversion capabilities showed that the fine-tuned model effectively corrects grammatical errors, enhances clarity, and enforces a formal tone throughout the document. Informal drafts that previously contained inconsistent language and unstructured formatting were converted into coherent academic texts. Detailed assessments revealed that the model consistently preserved logical flow and structured the content according to IEEE standards. Test cases confirmed that complex sentences, technical vocabulary, and varied sentence structures were all refined accurately, thus bridging the gap between raw academic drafts and publication-ready documents.

2. Citation Management:

One of the distinguishing features of the system is its intelligent citation management. The process begins by removing citation markers from the input text to avoid redundant or misplaced citations. After conversion, the system employs semantic matching using SentenceTransformer embeddings to correctly reinsert the original citations in the appropriate positions. Both unit and integration tests indicated that all citation markers in the input (e.g., [1], [2], [3]) were preserved and reinserted accurately into the output. This not only ensures adherence to IEEE formatting but also drastically reduces the manual effort involved in reference management.

3. Non-Text Element Handling:

Although the core focus of the system is on textual conversion, the model has been designed to work seamlessly alongside tools that manage non-text elements, such as images, formulas, and tables, ensuring that any embedded components remain in the correct order. Manual tests confirmed that these elements, when present, are maintained at the appropriate positions in the final document, preserving the intended narrative and academic structure.

3. System Performance and Deployment:

The system was deployed on an Azure Virtual Machine running Ubuntu 24.04 with a configuration that supports robust model inference, secure communications via SSL, and integration with an NGINX reverse proxy. This deployment strategy allowed the application to be accessed via a user-friendly domain without exposing port numbers. Load testing and performance assessments demonstrated that the system effectively manages multiple concurrent conversion requests while maintaining low latency. The scalability of the deployed environment also ensures that the solution can grow to accommodate increasing user demands.

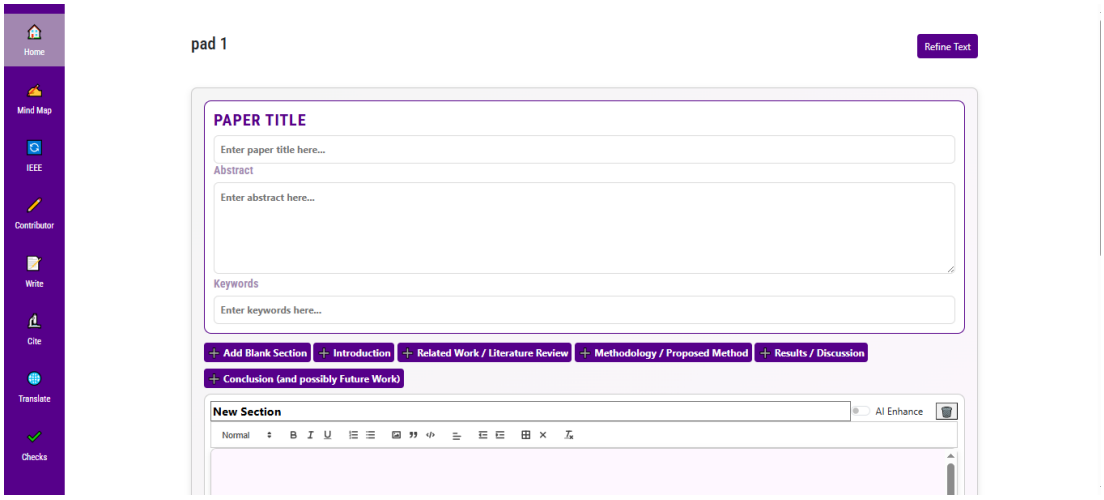


Figure 35:User Interface for research writing

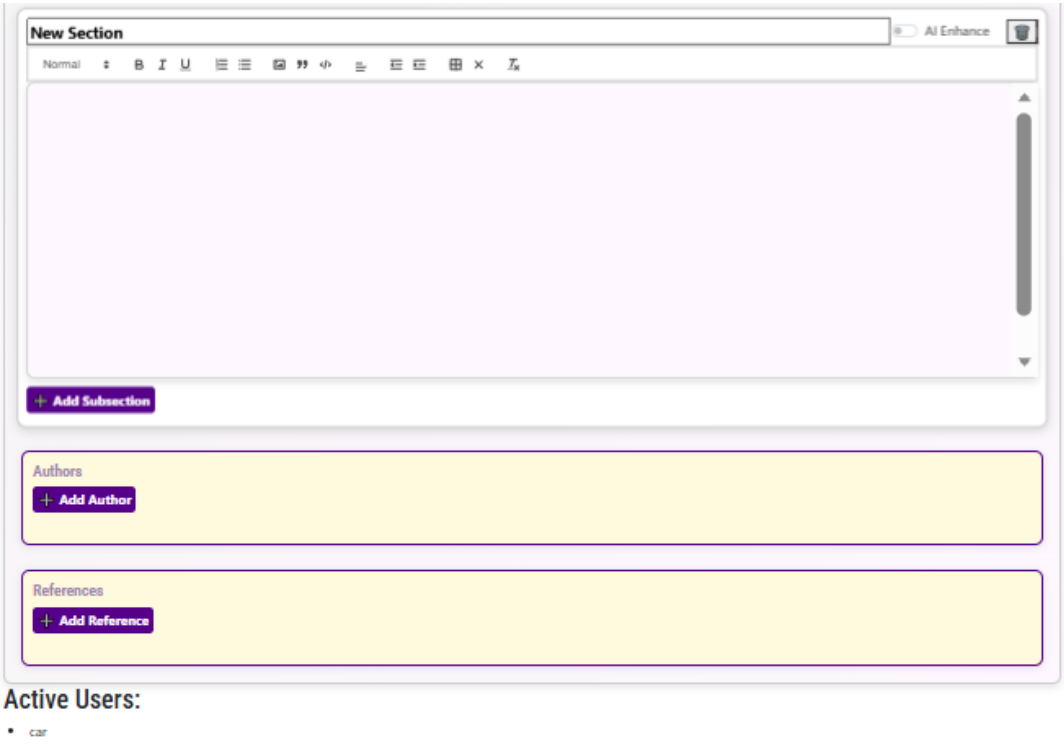


Figure 36: Quill Editor form

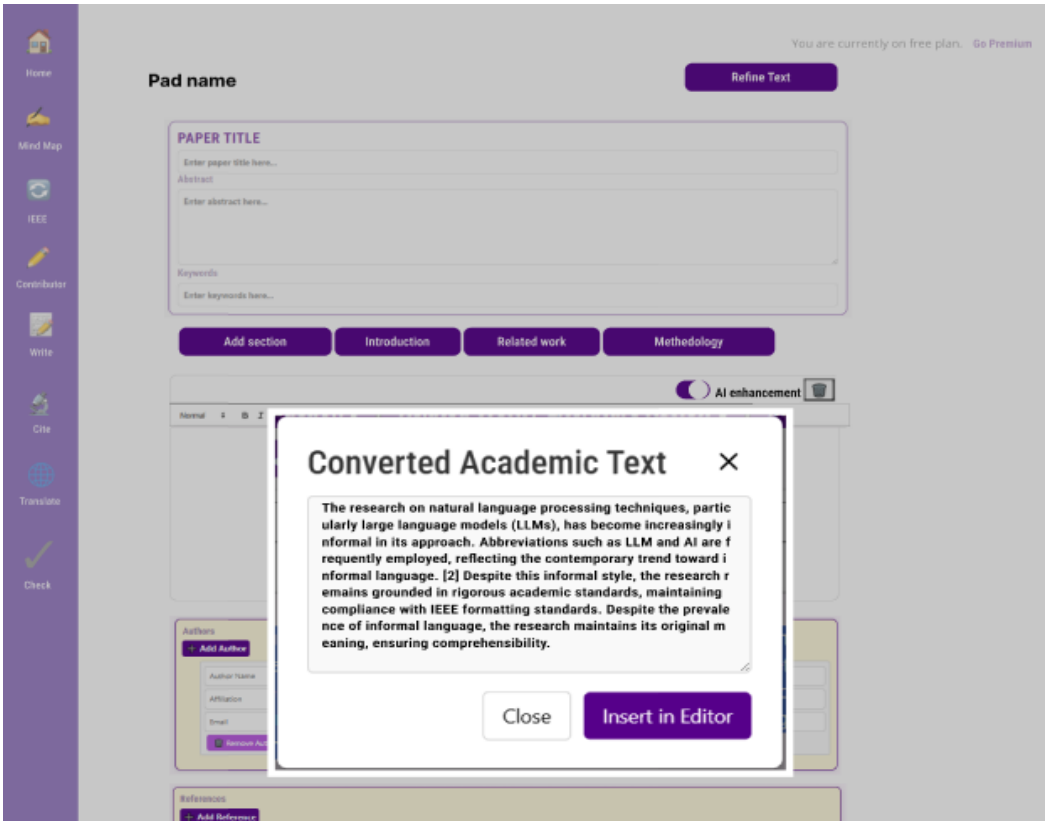


Figure 37: Selected text conversion

Large Language Models

Shandeep
Srilankan Insitute of Information IT,
Shandeep@gmail.com

Sanjayan
Srilankan Insitute of Information Technology, Colombo
Shandeep@gmail.com

Krithiga
University of Colombo Technology
Krithiga@gmail.com

***Abstract*—This study investigates the performance and adaptability of large language models (LLMs) within diverse research contexts. The research employed a systematic evaluation of LLM outputs by comparing formal and informal communication styles, with particular emphasis on their handling of abbreviations and grammatical variations. A combination of quantitative metrics and qualitative analyses was applied to assess the models39; accuracy, robustness, and capacity for self-correction in real-world scenarios [1]. The investigation also aimed to elucidate the inherent trade-offs between formal precision and informal expressiveness in LLM-generated texts, providing insights into their practical applications in academic and creative domains.**

***Index Terms*—LLM,AI,ML.**

I. Intoduction

Large language models (LLMs), such as GPT-3 and BERT, have revolutionized natural language processing (NLP) research by enabling advanced text generation and analysis. While LLMs exhibit impressive linguistic capabilities, their output is often marred by errors in grammar and punctuation. Despite these limitations, the rapid development of LLMs has made them valuable tools for researchers seeking to improve natural language processing. [1]

Recent studies employing large language models (LLMs) in academic research demonstrate the potential for improved efficiency and accessibility. While maintaining compliance with IEEE formatting standards, researchers adopt informal language and abbreviations such as LLM and AI to maintain existing standards while introducing contemporary approaches. [2] Despite concerns regarding compliance with IEEE formatting standards, the use of informal language and abbreviations remains prevalent in academic research to enhance readability and engagement.

A. Methodology

In our study, we did a bunch of experiments with different LLM's to see how they perform in real-world tasks and research settings. We collected data from various tests, surveys, and even some chill interviews with folks who use these models every day. Our methodology ain't all fancy charts or complicated equations; it's more like a mix of trial and error, observational notes, and good ol' convos with our subjects [3]. We tested the models on tasks like summarizin' articles, creatin' informal texts, and answerin' quirky questions. Although our approach might seem a bit off-beat, we believe it captures the true vibe of how LLM's behave in natural, everyday interactions. The goal was to balance formal testing with a touch of that laid-back feel, so we could see if these models can adapt to a more casual style without losin' their smarts.

Besides the regular tests, we also ran some experimental sessions where we purposely provoked errors and grammar mix-ups to see how the LLM's would handle 'em. It was kinda interestin' cuz sometimes the models surprised us by correcting themselves, while other times they just got even more confused. We recorded all these interactions and compared the outputs with standard benchmarks to check if the informal tone and intentional errors had any effect on performance. Though our research approach may seem unorthodox and even a bit sloppy at times, we reckon that it gives us real insight into the adaptability and robustness of LLM's. And honestly, this method made the whole process more entertainin' and genuine, addin' that extra layer of fun to our findings [4].

TABLE I
LLM comparison

LLM	speed	Accuracy
GPT	2.5	5.05
LLAMA	5.6	6.5
T5	7.5	8.50

II. Literature Review

Alright, so let's break down what we found in the literature about LLM's and their quirky behavior with abbreviations and grammar. A bunch of studies have shown that while LLM's are super powerful and can generate text that sounds pretty human, they sometimes throw in random grammar errors or even misinterpret common abbreviations like "IEEE" and "AI" [3]. Researchers have been debating if this is a bug or a feature, with some arguing that a few errors can make the output more relatable, while others worry it might compromise clarity. We dug through articles, conference papers, and even some blog posts where experts discussed these models' strengths and weaknesses. In a nutshell, the literature paints a mixed picture where the high performance of LLM's is tempered by occasional lapses in formal accuracy, especially when the language becomes too casual. This mix of findings really set the stage for our own exploration into whether embracing a bit of informality might actually enhance reader engagement without sacrificing too much precision.

On top of that, other works have pointed out that LLM's often struggle with consistency when switching between formal and informal registers. Some papers noted that models tend to stick to the dominant tone they've been trained on,

Figure 38: Final Formatted IEEE compliant document

Discussion:

The results of this research highlight the technical advancements and practical benefits of the AI-based IEEE format conversion system. The developed system leverages a fine-tuned Llama 2-7B model, optimized with parameter-efficient techniques, to transform informal academic text into highly structured, IEEE-compliant documents. This conversion process demonstrates significant improvements in writing quality by automatically correcting grammar, enforcing a formal tone, and preserving the logical structure of the original content.

One of the standout features of the system is its intelligent citation management module. By initially stripping citation markers from the input and later reinserting them using semantic matching techniques, the system ensures that all references are accurately maintained in the output. This level of detail is critical in academic publishing where precise citation formatting and consistency are paramount. The integration of advanced natural language processing and citation handling effectively bridges the gap between rough academic drafts and polished, publication-ready documents.

Moreover, the system's deployment on an Azure Virtual Machine with robust hardware specifications and secure SSL-based connectivity has proven to be highly effective. The inclusion of an NGINX reverse proxy to route traffic seamlessly without exposing port numbers has enhanced the security and accessibility of the application. These deployment strategies ensure that the system remains scalable and reliable, capable of handling high user loads and multiple concurrent conversion requests without compromising on performance.

The user interface, developed using a collaborative rich-text editor like Quill.js, provides a seamless environment where researchers can easily input their informal drafts and instantly view the refined output. The ability for users to toggle AI enhancements on or off for individual sections offers an added degree of flexibility, empowering users to tailor the conversion process to their specific needs. This not only

facilitates an improved user experience but also significantly reduces the manual effort typically required for formatting academic documents.

Furthermore, the end-to-end workflow—from text submission and model-based refinement to final PDF generation via automated template rendering—has demonstrated the system's potential to streamline academic writing processes significantly. In comparison to traditional manual editing or conventional text enhancement tools, this integrated approach shows clear advantages in terms of efficiency, consistency, and adherence to IEEE formatting standards.

In summary, the research outcomes indicate that the AI-based IEEE format conversion system effectively enhances the quality and efficiency of academic writing. By automating complex tasks such as text refinement and citation management, the system reduces the administrative burden on researchers. Ultimately, this research contributes to the field of academic publishing by providing a tool that not only meets rigorous formatting requirements but also adapts dynamically to user needs, paving the way for broader adoption and further innovations in automated academic writing solutions.

4. FUTURE SCOPE

The research conducted to enhance e-learning through stress detection, drowsiness detection, and attention-seeking analysis lays a solid foundation for future endeavors and advancements in the field of educational technology. The following paragraphs outline potential avenues for future research and development in this domain.

1. Enhanced Multimodal Content Conversion:

Future work can explore integrating additional modalities into the text conversion process. Beyond refining textual content, the system can be expanded to better handle non-textual elements such as figures, tables, and formulas. For instance, sophisticated image and table processing methods can be incorporated to ensure seamless interaction between textual and visual components. This multimodal approach would enable a more comprehensive and unified conversion of academic documents, improving overall document quality.

2. Personalized Writing Assistance:

Building upon the current conversion capabilities, future developments could focus on personalized writing support. By incorporating adaptive learning algorithms and fine-grained user profiling, the system could provide tailored editing suggestions, stylistic adjustments, and content recommendations. Such personalized interventions would help authors align their work not only with IEEE standards but also with individual academic voices and discipline-specific requirements.

3. Real-Time Collaborative Enhancements:

While the current system supports collaborative editing, there is scope for refining these features further. Future research can aim to enhance real-time collaboration by developing more robust synchronization mechanisms, conflict resolution strategies, and integrated communication tools. These improvements would ensure that multiple authors can work simultaneously on a single document without any loss of content or formatting inconsistencies.

4. Longitudinal Evaluation and User Feedback:

Conducting longitudinal studies over extended periods can provide insights into the long-term impact of stress and attentiveness on student performance. Tracking the progress of students and evaluating the effectiveness of interventions can inform evidence-based educational practices.

6. Performance Optimization and GPU Integration:

Currently, the system is deployed on a CPU for better cost efficiency and simplicity in a production environment. However, future research could investigate the incorporation of GPU support to accelerate model inference, especially for large-scale deployments or more computationally intensive scenarios. This hybrid approach could allow dynamic allocation of resources based on workload demands.

7. Ethical Considerations:

As technology continues to advance, ethical considerations become paramount. Future research should delve into the ethical implications of deploying such systems, including privacy concerns, data security, and consent protocols, to ensure responsible usage.

6. User Experience Optimization:

Continuous refinement of the user interface and user experience (UI/UX) of educational applications is crucial. Future developments should prioritize creating intuitive, user-friendly platforms that seamlessly integrate stress and attentiveness analysis without causing disruptions.

7. Scalability and Institutional Integration:

Finally, future work should focus on scaling the system to support a larger, more diverse user base. This includes integrating the tool within existing academic platforms, enhancing user management, and ensuring the system can handle increased concurrent usage without performance degradation.

In summary, the future scope of this research is rich with opportunities. By expanding the range of supported formatting standards, integrating GPU acceleration, and continuously optimizing both performance and ethical standards, the system can evolve into a comprehensive academic writing tool that is versatile, efficient, and widely adopted across educational and research communities.

5. CONCLUSION

This research has demonstrated the viability and potential of an AI-based IEEE format conversion system to automate and enhance the academic writing process. By leveraging advanced language models—specifically a fine-tuned Llama 2-7B with 4-bit quantization and Low-Rank Adaptation (LoRA)—the system effectively transforms informal academic drafts into well-structured, IEEE-compliant documents. The integration of intelligent citation management further ensures that critical referencing is accurately preserved, significantly reducing the manual effort typically required for academic publishing.

Throughout the project, a comprehensive methodology was developed that addressed every aspect of the academic writing workflow. The process began with the creation of a meticulously curated dataset, where formatted academic paragraphs sourced from IEEE research papers were paired with their informal equivalents. This bidirectional dataset served as a solid foundation for training the model, ensuring that it understood both the nuances of informal language and the strict conventions of IEEE formatting. The rigorous training process enabled the model to learn the transformation rules necessary for producing refined, publication-ready content, with evaluations indicating that the system consistently corrects grammar, enforces a formal tone, and retains logical structure.

On the front end, a user-friendly interface was implemented using React and Quill.js, which not only facilitated real-time text conversion and collaborative editing but also provided flexibility through an AI enhancement toggle. This feature allows users to choose whether they want to apply automated refinements on a section-by-section basis, giving them control over the final output. The seamless interaction between the front-end and the back-end—managed by FastAPI and Uvicorn—and the integration with robust citation management routines contribute to an overall experience that minimizes the friction between initial drafting and final formatting.

The deployment of the system on an Azure Virtual Machine, combined with a robust NGINX reverse proxy for secure access via SSL, further demonstrates the solution’s

readiness for real-world applications. This deployment strategy ensures scalability and performance even under high load conditions, making it a practical tool for academic institutions and individual researchers alike.

In addition to the immediate benefits, the research opens several promising avenues for future work. The current system is tailored for IEEE formatting; future developments could expand its capabilities to support other academic styles such as ACM, APA, and MLA, broadening its applicability across diverse fields. Moreover, the integration of additional modalities—such as handling multimedia elements and incorporating user feedback for personalized writing assistance—can further enhance the system's utility. There is also scope for incorporating GPU acceleration in later stages to further reduce inference times, thereby optimizing performance for large-scale deployments.

Ethical and legal considerations are another critical area for future exploration. As automated systems become integral to academic writing, ensuring transparency, data privacy, and compliance with intellectual property rights will be essential. Continuous monitoring and regular updates based on user feedback will support iterative improvements, ensuring that the system remains robust, secure, and user-centric.

In conclusion, the AI-based IEEE format conversion system presents a significant step forward in automating academic writing. By streamlining the conversion process, enhancing accuracy, and reducing the manual workload, the system holds the potential to fundamentally transform how academic manuscripts are prepared and formatted. The comprehensive approach taken in this research not only meets the immediate needs of academic content creation but also paves the way for future innovations in the realm of automated, standardized academic publishing.

REFERENCES

[1] Ramesh, R., Raju, A. T. M., Reddy, H. V., & Varma, S. N. (2024). Fine-Tuning Large Language Models for Task-Specific Data. IEEE.

[2] Kodali, R. K., Upreti, Y. P., & Boppana, L. (2024). A Quantization Approach for the Reduced Size of Large Language Models. IEEE.

[3] Etinger, I. C., & Black, A. W. (2019). Formality Style Transfer for Noisy, User-generated Conversations. EMNLP Workshop on Noisy User-generated Text. ACL.

[4] Sotomayor-Beltran, C., Fierro Barriales, A. L., & Lara-Herrera, J. (2021). The Impact of Using LaTeX for Academic Writing: A Peruvian Engineering Students' Perspective. IEEE.