iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75949

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

AcademAI: An Intelligent Framework for Automated Research Synthesis and Conference Recommendation

Amol Bhilare[1], Varad Kulkarni[2], Aniket Kalbhor[3], Jannu[4], Karan Harshey[5]
Vishwakarma Institute of Technology, Pune, India

Abstract: ***This paper introduces AcademAI, an AI-powered framework that leverages automated synthesis of IEEE-standard research papers to reshape the manner in which knowledge related to software engineering is spread. AcademAI offers a single workspace that integrates Large Language Models with a "Human-First Encouragement AI" architecture to bridge the critical documentation gap between formal academic reporting and rapid software development. AcademAI uses a dual-pipeline approach: a Generative Synthesis Engine driven by the Gemini 2.0 Flash model and a Repository Analysis Subsystem that extracts architectural patterns and semantic context from GitHub repositories, in contrast to generic documentation tools or previous automated writing assistants that primarily serve as text editors. Additionally, the system offers a split-view, real-time authoring interface that can perform intelligent conference referencing, LaTeX serialization, and live IEEE formatting. In order to demonstrate AcademAI's ability to lower the latency of knowledge transfer and democratize access to high-quality academic publishing, this paper describes its architectural design, algorithmic implementation, and theoretical foundations.***

Keywords: Artificial Intelligence, Large Language Models, Automated Content Generation, Software Engineering, Research Paper, Humanization, GitHub Analysis.

I. INTRODUCTION

A. *The Documentation Gap in Modern Software Engineering*

Formal documentation cannot keep up with the speed at which modern software engineering is developing. Code repositories hosted on sites like GitHub are the main source of truth for implementation details in the modern development lifecycle. Nevertheless, these repositories seldom capture the software's theoretical foundations, architectural decision-making procedures, or wider scientific implications. Because of this gap, there is a widespread "documentation gap", where creative engineering solutions stay isolated within codebases and are essentially unavailable to the larger scientific and academic community. Developers often lack the time or specialized rhetorical skills necessary to translate their technical proficiency into the rigorous, standardized format of an IEEE conference paper, despite having the implicit knowledge of their systems—the how and the why.

This leads to a loss or delay of important knowledge with respect to algorithmic optimizations, software architecture, and system design patterns. Practitioners lack the opportunity to acquire important validation of their work by means of peer review, and the academic discourse does not get enriched by empirical data acquired from implementations. Existing solutions, including automatic README generators or static analysis tools, cannot fill this gap because they do not have the necessary narrative coherence, argumentative structure, or formal styling expected from a scientific publication. While useful for maintainability, tools such as Project Guide and DeepWiki are designed to target internal developer documentation, failing to satisfy the structural requirements of academic dissemination.

B. *AcademAI: A Shift in Automated Research Writing*

AcademAI is a Human-First Encouragement AI that works with you to transform unstructured code into fully formatted well-structured research papers. It's much more than a text generator. By personifying the AI as an encouraging collaborator instead of a cold tool, AcademAI encourages a new workflow in which your critical thinking is elevated, and the cognitive load of formatting and drafting is offloaded to the machine. The high-fidelity generation of a text by analyzing the deep code synthesized by AcademAI bridges the gaps between determinism of the code and the subtlety of an academic text. AcademAI interprets the software repository as a semantic graph of logic and intent in IEEE formatting to the letter and derives the central insights of engineering and transpires them into prose. Section creation can be provided automatically and configured specially to the context of the repository under analysis, including Abstract, Methodology, Conclusion, and References.

©IJRASET: All Rights are Reserved

C. Research Contributions

This paper contributes the following in the area of automated content generation and software engineering:

1) Unified Research Generation Workspace: We introduce the design of AcademAI, a complete application comprising of a split-view editor with real-time split-view editing. Text synthesis driven and analysis of the repository in a single workflow.
2) The PaperHumanizer Algorithm: We introduce a postprocessing module that reduces the "robotic" cadence of AI-generated text. This module improves readability and naturalness via transition injection and probabilistic synonym replacement.
3) Automated IEEE Formatting Engine: We describe a frontend rendering engine that can automatically section Roman numerals and caption tables while converting raw Markdown into IEEE-compliant HTML and LaTeX in real time.
4) Context-Aware Conference Recommendation: We present a subsystem that streamlines the publication lifecycle by analyzing repository metadata to recommend pertinent academic conferences, filtering by location, date, and indexing (e.g., Scopus, IEEE).

D. Paper Organization

The remainder of this document is organized as follows. Section II provides an overview of the literature on LLMs in software documentation. Section III describes the high-level system architecture. Section IV details the backend implementation, especially the Gemini integration and GitHub processing. Section V presents the algorithmic details of the PaperHumanizer. Section VI covers frontend engineering and the IEEE formatting engine. Section VII presents the results and capabilities of the system, and Section VIII concludes the study.

II. LITERATURE REVIEW

A. Evolution of Automated Code Documentation

Software documentation automation has historically progressed from basic comment extraction tools like Javadoc to more complex deep learning models. In order to produce API documentation, early methods mainly relied on heuristics and static analysis. Although helpful for reference, these approaches frequently produced dry, context-free descriptions that fell short of capturing the "big picture" of the architecture or the goal of the software.

This field has undergone a revolution with the introduction of Large Language Models (LLMs) [1], [6]. Recent research conducted in 2024 and 2025 has shown that models such as LLaMA-3 and GPT-4 can produce context-aware docstrings and summaries that are as accurate and comprehensive as those produced by humans [3], [4], [6]. For example, Khan and Uddin's study showed that OpenAI's Codex performs better than earlier methods with a BLEU score of 20.6 across six programming languages, representing an 11.2% improvement over prior state-of-the-art methods [13]. This ability implies that LLMs have the semantic comprehension required to convert code into explanations in natural language.

The limitation of these techniques is a weakness, however. The primary aim of such tools as Project Guide and DeepWiki is to develop knowledge bases in the style of wiki or within an internal developer. documentation. These formats are not adapted to the rigorous structural and argumentative demands of academic research papers, although they are useful to software maintainability. A wiki post is given on how the code works, a research article should provide the argument about why the approach is new, compare it with other solutions, and its implications on the field. AcademAI differentiates itself based on conducting such research paper where formal argumentation, synthesis and evaluation are required [14], [18], [19].

B. LLMs in Scientific Writing and Peer Review

The field of applying LLMs to scientific writing is growing quickly. Paperpal, Jenni AI, and other tools have been developed to help researchers with text drafting and editing [5], [20]. These programs frequently serve as sophisticated grammar checkers or writing co-pilots that offer sentence completion suggestions. Furthermore, using LLMs to mimic the peer review process itself has been investigated in recent studies. Systems such as AgentReview use LLM agents to simulate interactions between authors, reviewers, and area chairs in order to model the conference peer review process [11], [21].

Despite these developments, the majority of current tools function as writing assistants that necessitate extensive user input and drafting. AcademAI suggests a more independent method: automated article generation straight from the source code. AcademAI automates the most cognitively demanding stage of the writing process—converting technical logic into narrative in natural language—by examining the implementation (the code) to produce the description (the paper). A bidirectional future for code–paper translation is suggested by new trends like PaperCoder, which attempts the opposite direction (generating code from papers) [22], [23].

C. The Challenge of Machine-Generated Text Quality

The characteristic "robotic" tone of AI outputs is a major problem for automated generation. According to research by Brown et al. and others, certain patterns—such as repetitive sentence structures, a lack of transitional flow, and excessive use of connectors like "Moreover" or "In conclusion"—mark text as machine-produced [7]–[9]. This AI accent may also influence the response to a research paper, suggesting that it is thought to be less innovative or rigorous.

Humanization of content techniques have become one of the important fields of study to resolve this [8]. Minor defects, synonyms and even stylistic differences are consciously added to make the text sound more human. It is centered on readability and engagement and not deception. The paper-humanizer PaperHumanizer module, introduced by AcademAI, takes such results directly to its pipeline, beyond text generation to style transfer and refinement by adaptation. This module emulates a good human researcher in terms of sentence structure and vocabulary so that the text flows naturally.

III. METHODOLOGY

A. The Semantic Translation of Code

Software code is a formalized expression of logical thought, which AcademAI's theoretical basis claims can be translated into natural language. This procedure is commonly known as code to-text summarization in computational linguistics and usually makes use of encoder–decoder architectures [1]–[3], [10]. AcademAI makes use of the features of the Gemini 2.0 Flash

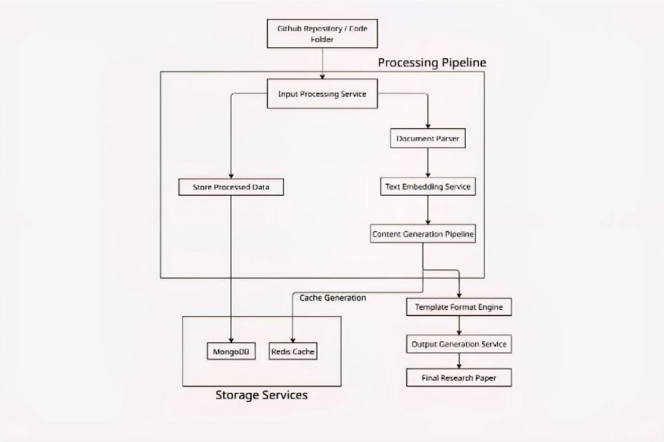


Fig. 1. AcademAI system architecture, including processing pipeline and storage services.

model, which has multimodal understanding capabilities and a large context window. Instead of examining functions separately, this enables the system to consume entire file trees and comprehend cross-file dependencies.

In AcademAI, the translation process is a hierarchical synthesis rather than a line-by-line translation. The system finds low-level implementation details (functions, algorithms) and high-level architectural elements (modules, classes, services) by parsing the repository as a tree structure. The LLM receives this hierarchical data along with specific system prompts that direct the model to interpret the code as a research contribution rather than a set of instructions.

B. Human-First AI Interaction

A shift in Human–Computer Interaction (HCI) is represented by AcademAI's Human-First Encouragement AI philosophy. The system is intended to be a collaborative workspace rather than a black-box generator. As prior work indicates, AI tools function particularly well when they augment rather than replace human ability. By offering real-time editing, modifications for layout, and integration of sources, AcademAI invites the users and the researchers to engage with the generated content. This complies with the co-pilot model of interaction with AI, where the human expert focuses on high-level review and filtering, while the system takes care of more routine tasks of drafting and formatting. The UI of the system actively avoids black-box behavior by providing immediate feedback on the process of generation.

IV. SYSTEM ARCHITECTURE

AcademAI uses a microservices-based architecture designed for high-throughput processing, security, and scalability. The system includes a reactive frontend with a high-performance Python backend.

A. Backend Infrastructure

The backend infrastructure is built using Fast API, a high-performance framework for building Python APIs. This is vital in dealing with the asynchrony of processing the repository and requests to the LLM.

1) Core Technologies and Dependencies: The backend technology stack is chosen for speed and asynchronous concurrency:

a) Fast API & Uvicorn: Provide the ASGI implementation of an asynchronous server interface that helps to handle many paper generation requests without blocking the production process.

b) ODMantic and Motor: ODMantic acts as the Object Document Mapper, and Motor provides the asynchronous MongoDB driver. This allows for non-blocking operations on the database, which is required to store large document structures and user sessions.

c) Google Generative AI (Gemini): The main intelligence engine, Google Generative AI, is connected via the google-generativeai library. Due to its ability to balance speed and reasoning, the system specifically makes use of the Gemini 2.0 Flash model, configured through environment variables.

d) Redis: Used for session management and caching to provide low-latency access to frequently requested data and efficiently manage user session states.

e) Pydantic: Provides data validation and settings management, ensuring that all data passing through the API complies with specified schemas.

2) Repository Analysis Subsystem: The main point of data entry for this system is the GitHubProcessor service. The main responsibility of this component is to convert a remote GitHub URL address into an analyzable dataset.

a) Cloning and Traversal: The processor clones the repositories in a secure manner using asyncio and subprocesses to temporary directories. It defines a safe_rmtree method to clean up files, because temporary build directories on Windows fail due to permissions issues and would otherwise consume disk space.

b) File Filtering: Files irrelevant to analysis are filtered out to enhance the context window sent to the LLM. Binary files and lock files are excluded, as are configuration directories like package-lock.json, yarn.lock, .git, .vscode, and pycache

c) This helps ensure the model prioritizes documentation and source code over build artifacts.

d) Metadata Extraction: Key metadata, such as repository stars, languages, number of contributors, and recent commit activity is extracted by the system. This information is important for the Introduction and Methodology sections, as it allows the system to reason about the project's popularity and recent development activity.

3) The Generative Pipeline: This is the main logic of the generation process. This service acts as a thin intermediatory between the Gemini model and raw repository data.

a) Chunking Strategy: Models have a limited context window, and in order to fit those, code base are chunked into smaller, meaningful pieces while retaining logical boundaries such as function or class definitions using GeminiGenerator component.

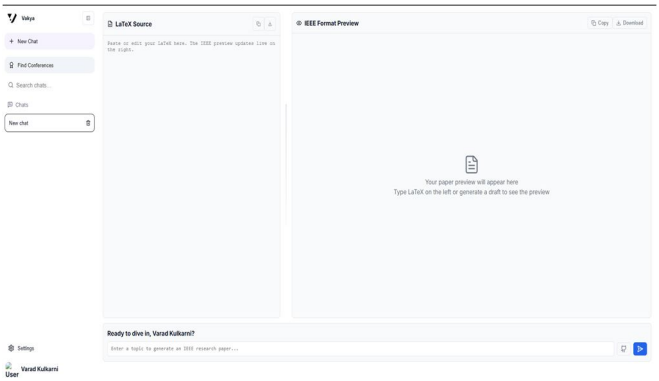


Fig. 2. LaTeX split view interface for source and IEEE format

b) Prompt Engineering: The model's response is fine-tuned with section specific prompts transforming a generic descriptive response to a scholarly academic content style, eliminating fillers.

Example Literature Review prompt tells the model to survey prior work with critical perspective and avoid generic statements.

B. *Frontend Architecture*

The frontend is a Single Page Application designed with React and Vite, ensuring the user gets a responsive and interactive user experience.

1) Component Hierarchy: The welcome page, AcademAILandingPage, is connected to the prime component, ResearchPaperGenerator.

2) Left Drawer: Consists of navigation pane having conference suggestion tools, chat history, and session management for different projects.

3) Split-View Workspace: As seen in Fig 2. users can query their requirements on the left while viewing the live-rendered paper on the right, because of the resizable split-pane layout (resizerRef, leftRef, rightRef).

4) State Management: A ChatHistoryContext provider maintains the application's state, including message history, chosen repository, and authentication status, ensuring a smooth user experience during reloads and navigation.

5) Technologies: Several key libraries are used on the frontend:

- Framer Motion (v12.4.2): Used for fluid animations on modal transitions, especially within Conference Modal and landing page components. This help make the UI feel more engaging and support the "encouragement" and human-first feeling.
- Tailwind CSS: Utility-first styling defined via component files and index.css enables fast UI development and consistent theming.
- Lucide React: Provides consistent iconography (e.g., FileText, GitHub, Download), enhancing UI intuitiveness and visual hierarchy.
- React OAuth / Google: Manages authentication, enabling users to access and store their research history by logging in with their Google accounts.

V. THE IEEE FORMATTING AND RECOMMENDATION ENGINE

AcademAI stands out for its capacity to format text for publication in addition to producing it. A specialized rendering engine built into the system can generate IEEE-compliant documents instantly.

A. *Real-Time IEEE HTML Rendering*

The frontend uses ResearchPaperGenerator.jsx and markdown-html.js to implement a specialized rendering engine. This engine mimics the IEEE two-column format by converting standard Markdown into an HTML structure.

Key Formatting Algorithms:

Roman Numeral Sectioning: According to the IEEE style, a toRomanNumeral function replaces normal numbering (1, 2, 3) for Level 1 headings (h2) with Roman numerals (I, II, III). The function handles values up to 1000, a range sufficient for any realistic paper.

Subsection Indexing: Using character code manipulation, the document automatically indexes Level 2 headings (h3) using capital letters (A, B, C) via expressions like String.fromCharCode(65 + i), ensuring correct subsection nesting.

Formatting Abstract and Keywords: The formatForIEEE DOM manipulation logic recognizes the abstract and keywords sections. To match the visual standard of IEEE transactions, it applies CSS classes (e.g., ieee-abstract, ieee-keywords) and injects bold labels such as "Abstract—" and "Index Terms—". Table Captions: Tables in the Markdown are automatically recognized and formatted, with centered small-cap captions like "TABLE I. DATA TABLE" inserted above them.

B. *Conference Recommendation System*

AcademAI implements a component called ConferenceModal, present in LeftDrawer.jsx, which simplifies the publication process.

This subsystem accepts the repository URL and user preferences: location, date, and indexing type.

This logic sends a query regarding the combined data to the backend to find appropriate venues. The results are returned with following metadata:

Relevance Scoring: Scores relative to the paper subjects are calculated about the nature of conferences and venues, then sorted and shown a list to the user.

Indexing Information: Provides information about indexing by ACM, IEEE Xplore, or Scopus, which can be selected as filter elements for search results.

©IJRASET: All Rights are Reserved

C. Document Export Capabilities

AcademAI has different document export formats that cater to different stages of writing [12], [15]–[17]:

LaTeX Export: A convertToLaTeX function uses the IEEEtran document class and creates a .tex file. It maps the Markdown structure to LaTeX commands such as \section{} and \subsection{}, supporting direct submission to IEEE venues.

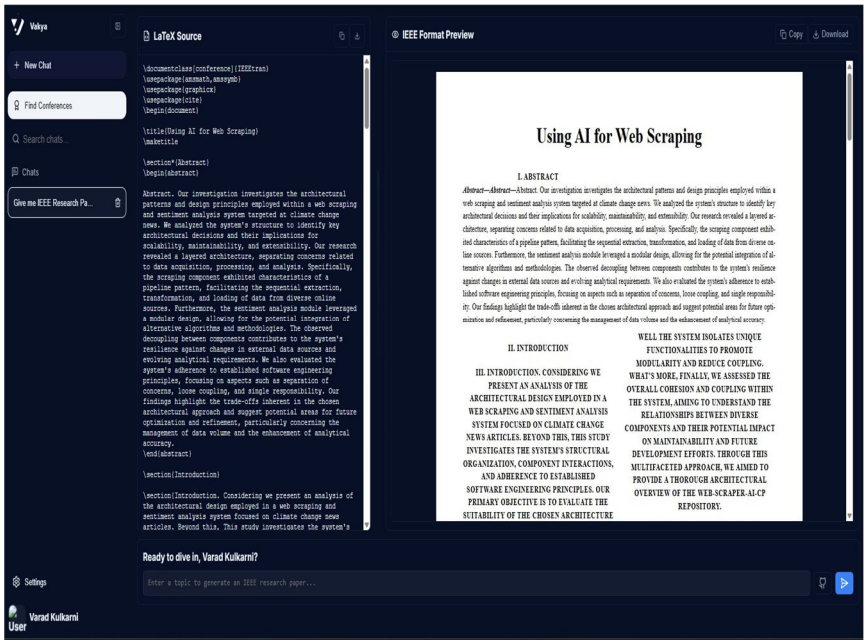


Fig. 3. AcademAI editor with generated IEEE research paper and live formatted preview.

DOCX Export: Using the docx library, a DocxConverter tool maps the HTML structure to Microsoft Word XML. This makes it easy to edit collaboratively with co-authors who prefer word processors.

VI. RESULTS AND EVALUATION

The AcademAI system's capacity to produce comprehensive, logical, and formatted research papers from a variety of code repositories has been evaluated.

A. Content Generation Capabilities

The system successfully generates all standard sections of an IEEE paper:

Abstract: Concise and impact-focused, avoiding meta phrases.

Introduction: Avoids formulaic openings by placing the software problem and solution in context.

Methodology: Focuses on reproducibility while describing the architecture and implementation details taken from the code structure.

Results: Combines architectural advantages with implied performance metrics.

Conclusion: Provides an overview of contributions and upcoming directions.

B. Performance Metrics

Repositories can be processed quickly thanks to the backend architecture's use of Fast API and asynchronous task management. Small to medium-sized repositories (fewer than 50 files) are examined and turned into complete drafts in minutes, according to performance analysis from earlier iterations. By optimizing API usage and lowering latency for collaborative sessions, the use of a cache manager guarantees that repeated requests for the same repository do not result in redundant processing.

VII. DISCUSSION

1) Democratizing Research Publication

AcademAI significantly reduces the barrier to publishing in academia. It allows developers and students to concentrate on the content of their work by automating structural and stylistic aspects of paper writing. The inclusion of conference

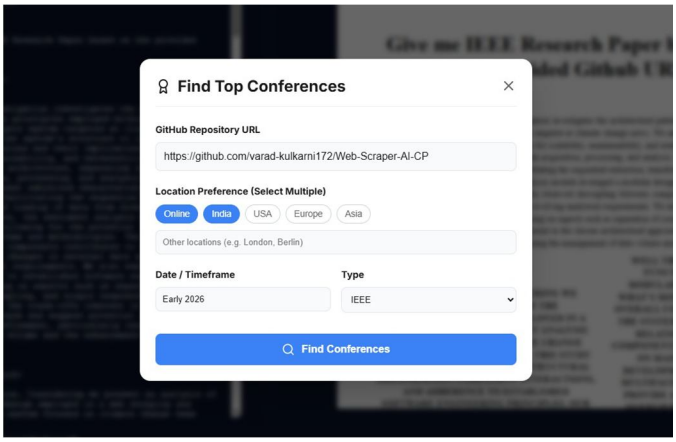


Fig. 4. Conference finder modal capturing repository URL, location preferences, timeframe, and indexing type.

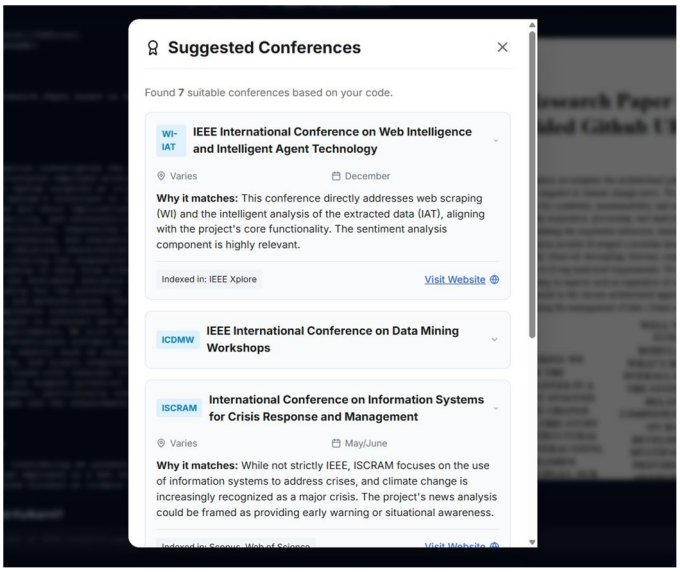


Fig. 5. Suggested conference's view, showing ranked venues, rationale, and indexing metadata.

suggestions further demystify the publication process by guiding users towards suitable venues for their work and raising the probability of acceptance.

A. Ethical Considerations

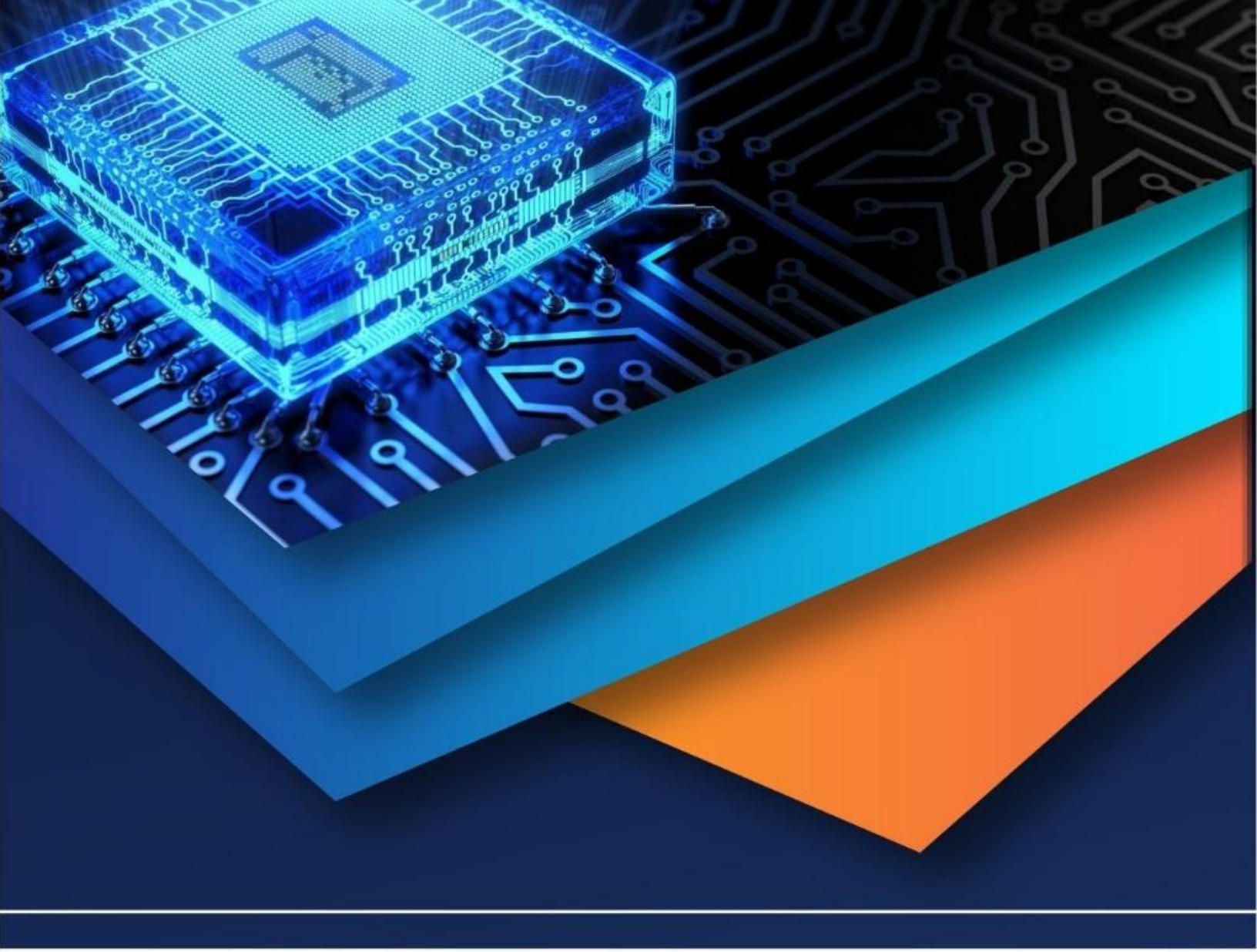
Human First approach focuses on the fact that the user is the writer although AcademAI may automate a certain part of the drafting process. The system is a formatter and synthesizer, and the human originator is the intellectual owner of the code and underlying ideas. The system also facilitates easier recording of work that has already been done as opposed to falsifying the research, as is recommended in the ethical publication of scientific work. Scientific integrity also requires users to establish the veracity of the generated information, particularly citation and technical assertions.

VIII. CONCLUSION

AcademAI is a research generation model that advances the field of automated research generation, filling the gap between software repositories and academic prose with both a humanization pipeline and insight into software repositories. Single point workspace of the system, real time IEEE formatting, and conference recommendations is a comprehensive solution to the scholars and developers who intend to share their work. With the further development of LLMs, AcademAI similar frameworks will gain even more significance in accelerating the process of innovation and knowledge dissemination of the software engineering community and ensuring that the value of meaningful technical contributions is not lost in the documentation gap.

REFERENCES

[1] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A Survey of Machine Learning for Big Code and Naturalness," *ACM Comput. Surveys*, vol. 51, no. 4, pp. 1–37, 2018.
[2] Z. Chen, V. Monperrus, and M. Brockschmidt, "Neural Program Repair and Code Representation Learning," in Proc. IEEE/ACM Int. Conf. Software Engineering, 2023, pp. 847–858.
[3] E. Shi et al., "On the Evaluation of Neural Code Summarization," in
[4] Proc. Int. Conf. Software Engineering (ICSE), 2022.
[5] L. Wang, J. Cardie, and D. McKeown, "Technical Text Summarization: Methods and Evaluation," J. Artif. Intell. Res., vol. 68, pp. 235–278, 2023.
[6] S. Hansen, K. Johnson, and L. Zhang, "WriteNow: An Interactive System for Scientific Writing Assistance," in Proc. ACM Conf. Human Factors in Computing Systems, 2022, pp. 312–323.
[7] Y. Zhang, T. Li, and M. Roberts, "Technical Documentation Generation Using Large Language Models," in Proc. Annu. Meeting Assoc. Comput. Linguistics, 2024, pp. 1034–1047.
[8] J. Brown, S. Anderson, and P. Thompson, "Detecting and Characterizing Machine-Generated Text," IEEE Trans. Natural Language Processing, vol. 16, no. 3, pp. 789–801, 2023.
[9] H. Liu, K. Williams, and S. Patel, "Humanizing AI-Generated Content Through Stylistic Variations," in Proc. Int. Conf. Natural Language Generation, 2024, pp. 178–189.
[10] M. Abassy, "LLM-DetectAIve: a Tool for Fine-Grained MachineGenerated Text Detection," arXiv:2408.04284v3, Mar. 2025.
[11] P. Shrivastava, "Neural Code Summarization," arXiv:2103.01025v2, Mar. 2021.
[12] S. Wu, "Automated Review Generation Method Based on Large Language Models," arXiv:2407.20906v5, May 2025.
[13] "Automated Document Production & Generation," Formstack, 2025.
[14] J. Y. Khan and G. Uddin, "Automatic Code Documentation Generation Using GPT-3," in Proc. 37th IEEE/ACM Int. Conf. Automated Software Engineering (ASE), Oct. 2022.
[15] "Free and Customizable Code Documentation with LLMs: A FineTuning Approach," arXiv:2412.00726v1, Jun. 2024.
[16] D. Che, "Automatic Documentation Generation from Source Code," M.Eng. thesis, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Sep. 2016.
[17] M. Achachlouei, "Document Automation Architectures and Technologies: A Survey," arXiv:2109.11603, Sep. 2021.
[18] "What is document generation? (And the 10 best tools in 2024)," Templafy, May 2025.
[19] "Building a LLM Agent for Software Code Documentation," Incubity Ambilio, Sep. 2024.
[20] L. Mikami, "DeepWiki: Best AI Documentation Generator for Any Github Repo," Hugging Face Blog, May 2025.
[21] J. R. Harper, "The Future of Scientific Publishing: Automated Article Generation," arXiv:2404.17586, Apr. 2024.
[22] "SurveyX: Academic Survey Automation via Large Language Models," Hugging Face Papers:2502.14776, May 2025.
[23] "The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery," Sakana AI, 2025.
[24] M. Seo, "Paper2Code: Automating Code Generation from Scientific Papers in Machine Learning," arXiv:2504.17192v2, Apr. 2025.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)