Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

SANKSHEP AI- READ LESS, KNOW MORE: AN AI-BASED TEXT SUMMARIZATION SYSTEM USING NLP AND TRANSFORMER ARCHITECTURES

[1]Sourabh Biswas, [2]Mr. Pawan Kumar Jaiswal
[1]Student, [2]Assistant Professor
[1,2]Amity University Chhattisgarh, Raipur

Abstract

Currently in our digital world there is a huge volume of text, such as articles, reports, and research documents being produced at a rapid pace creating many challenges around processing this information efficiently. This paper presents Sankshep AI-Read Less, Know More: an intelligent text summarization system that uses Artificial Intelligence (AI) and Natural Language Processing (NLP) techniques to help make content easier and faster to understand. The system uses state-of-the-art transformer architectures and an attention mechanism to help ensure that all summary sentences are semantically accurate and contextually relevant. Two primary approaches are employed within Sankshep — extractive summarization will identify and select the most significant sentences from the original source text, whereas abstractive summarization will generate new sentences that reflect the core idea(s) in a more natural, human-like way. Sankshep AI was built using Python, NLTK, Hugging Face Transformers, and on a user-friendly web interface built with Streamlit, powered by Google Gemini AI. In addition to plain-text input, many other formats (i.e., PDF documents, DOCX files, and web URLs) can also be submitted for processing. As a result of this wide variety of accepted input formats, Sankshep should show utility across many different types of real world applications. The study shows that when producing a summary of an original text, the Speed of Summarisation and Summary Size decrease while the content of the original text is preserved. Performance evaluations using various metrics are shown to confirm that the system is effective across many different types of datasets when evaluating performance. In general, Sankshep AI has demonstrated good potential for use within educational, research, and professional settings where it is important to manage information effectively.

Keywords: Text Summarization, Natural Language Processing, Transformer Models, Artificial Intelligence, Extractive Summarization, Abstractive Summarization

1. Introduction

In today's data-driven world, enormous volumes of textual information are generated daily from diverse sources including academic journals, news portals, blogs, research papers, and social media platforms. According to various studies, more than 2.5 quintillion bytes of data are created every day, with a significant proportion consisting of unstructured text. Reading and comprehending all available content is impractical and time-consuming, leading to a growing need for automated intelligent systems capable of distilling essential information from large textual corpora.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

Text summarization — a key application domain at the intersection of Artificial Intelligence (AI) and Natural Language Processing (NLP) — addresses this challenge by converting lengthy documents into shorter, contextually accurate summaries. This paper presents Sankshep AI: Read Less, Know More, a system designed to automatically generate meaningful summaries from large text using state-of-the-art NLP techniques. The name "Sankshep" derives from Sanskrit, meaning "concise" or "abbreviated," reflecting the system's core purpose.

Text summarization is broadly categorized into two primary paradigms: (1) Extractive Summarization, which identifies and selects the most significant sentences from the source text without altering their wording; and (2) Abstractive Summarization, which generates entirely new sentences that capture the essence of the original content, mimicking human-level comprehension and synthesis. Both approaches are incorporated into the Sankshep AI system, giving users flexibility in their summarization preferences.

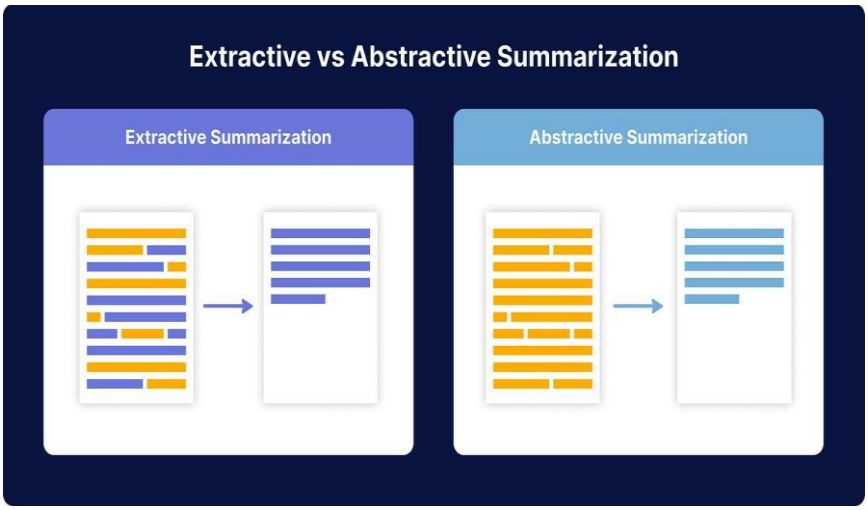

Fig. 1. Illustration of Extractive and Abstractive Approaches in Text Summarization.

The Sankshep AI system accepts multiple forms of input — plain text, PDF files, DOCX documents, and web URLs — processes the content through a multi-stage NLP pipeline, and produces a summary tailored to the user's preferred length (Short, Medium, Long) and style (Paragraph, Bullet Points, ELI5). The system leverages Google's Gemini AI model via API integration and is deployed through a Streamlit-based web interface, making it accessible without technical expertise to students, researchers, and professionals alike.

1.1 Objectives

The primary objectives of the Sankshep AI project are: (1) to develop a functional AI-based text summarization system supporting both extractive and abstractive paradigms; (2) to provide multi-modal input support for plain text, PDF, DOCX, and URL sources; (3) to implement a user-friendly web interface enabling non-technical users to interact with the system; (4) to integrate advanced language model APIs for high-quality abstractive summarization; and (5) to evaluate the system's performance using standard NLP evaluation metrics.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

2. Literature Review

The field of text summarization has evolved considerably over several decades. Early approaches relied exclusively on statistical methods. Term Frequency–Inverse Document Frequency (TF-IDF) became a foundational technique, ranking sentence importance based on the distribution of words across a document corpus. These methods, while computationally efficient, produced extractive summaries of limited coherence and linguistic fluency. The introduction of machine learning classifiers — including Naive Bayes and Support Vector Machines (SVMs) — improved summarization quality by incorporating richer contextual features such as sentence position, length, and cue words. Subsequent advances in deep learning led to adoption of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which better captured sequential dependencies in text, enabling rudimentary abstractive capabilities [1]. The landmark 2017 publication "Attention Is All You Need" by Vaswani et al. [2] introduced the Transformer architecture — now the foundation of modern NLP. The self-attention mechanism allows models to dynamically weight the relevance of any word with respect to any other word in a sequence, enabling rich contextual understanding at scale. BERT (Bidirectional Encoder Representations from Transformers) [3] extended this paradigm with masked language modeling and next-sentence prediction, achieving state-of-the-art results across NLP benchmarks. Pointer-Generator Networks [4] introduced copy mechanisms to address out-of-vocabulary word challenges in abstractive summarization.

Table 1. Comparative Analysis of Summarization Approaches.

Feature	Extractive Method	Abstractive Method
Working Principle	Selects key sentences verbatim	Generates new sentences from meaning
Complexity	Low	High
Output Naturalness	Less natural	More natural; human-like
Accuracy	Moderate	High (with well-trained models)
Computational Cost	Low	High
Use Case	Fact retrieval, legal documents	News, research, general articles

Research gaps identified in the existing literature include: difficulties in processing very long or domain-specific documents exceeding model context windows; high computational requirements limiting accessibility of abstractive models; limited multilingual support particularly for regional and low-resource languages; and persistent challenges in evaluating summary coherence, factual consistency, and faithfulness to source content. The Sankshep AI project directly addresses the accessibility gap by wrapping powerful language model APIs within an intuitive multi-modal interface.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

3. Methodology

The development of Sankshep AI followed a structured software engineering methodology encompassing requirements analysis, system design, implementation, integration testing, and performance evaluation. A modular architecture was adopted to enable independent development, testing, and future enhancement of each system component. The methodology integrated principles from both Agile iterative development — allowing continuous refinement based on testing feedback — and structured design to ensure system coherence.

3.1 System Architecture

The system is composed of five core functional modules: (1) Input Module — accepts text via direct entry, file upload (PDF/DOCX/TXT), or URL specification; (2) Preprocessing Module — performs tokenization, stopword removal, punctuation cleaning, and lowercase normalization; (3) Feature Extraction Module — computes word frequency distributions and sentence importance scores; (4) Summarization Engine — applies extractive frequency-based algorithms or routes requests to the Gemini abstractive API; and (5) Output Module — presents generated summaries through the web interface with configurable length and style options, alongside a download facility.

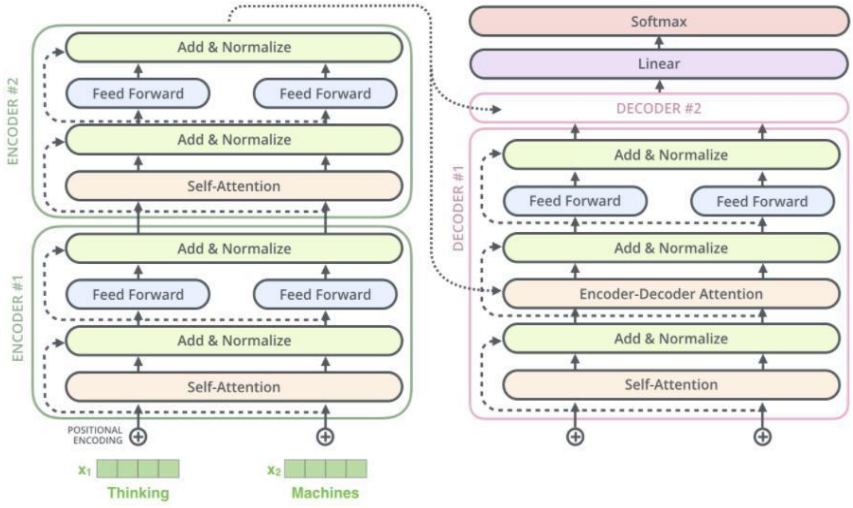

Fig. 2. System Architecture of the Sankshep AI Text Summarization Platform.

3.2 Extractive Summarization Algorithm

For extractive summarization, the system employs a word frequency–based sentence ranking algorithm. The algorithm proceeds through four stages: (1) text preprocessing to remove stopwords and normalize case; (2) word frequency computation using a dictionary accumulator; (3) sentence scoring, where each sentence receives a cumulative score based on the frequency of its constituent content words; and (4) top-N sentence selection using Python's heapq.nlargest function. The value of N is determined by the user's selected summary length preference. This approach is computationally lightweight and guarantees that selected

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

sentences are drawn verbatim from the source, preserving factual accuracy and linguistic integrity.

3.3 Abstractive Summarization via Gemini API

For abstractive summarization, structured prompts are dynamically constructed and transmitted to the Google Gemini AI API (model: gemini-3-flash-preview). Each prompt encapsulates two principal directives: a length constraint instruction mapping the user's Short/Medium/Long selection to approximate word-count targets; and a style formatting instruction specifying Paragraph (flowing prose), Bullet Points (itemized list), or ELI5 (Explain Like I'm 5 — simplified language) output format. The API response is post-processed to extract the summary text block, which is then rendered in the Streamlit interface. API keys are managed securely through environment variables using python-dotenv.

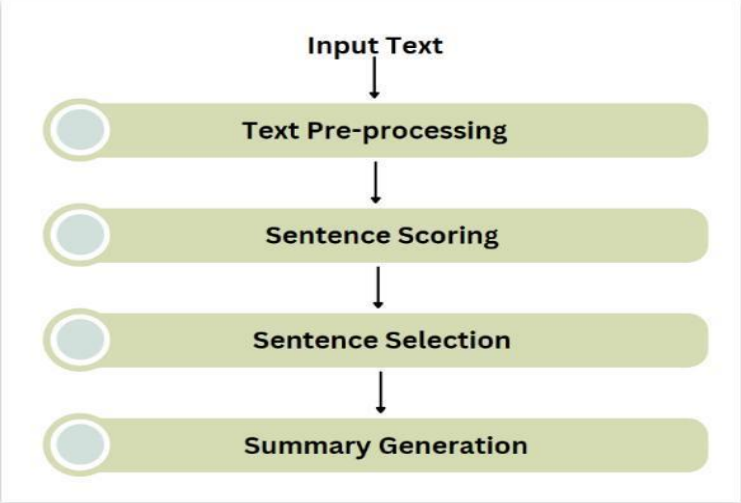

Fig. 3. Workflow of the Text Summarization Process.

3.4 Software and Hardware Requirements

Table 2. System Requirements Specification.

Category	Component	Specification
Software	Programming Language	Python 3.x
	NLP Libraries	NLTK, Hugging Face Transformers
	AI Model API	Google Gemini (gemini-3-flash-preview)
	Web Framework	Streamlit
	File Processing	PyPDF2, python-docx
	Web Scraping	Requests, BeautifulSoup4
Hardware	Processor	Intel i3 or equivalent
	RAM	Minimum 4 GB
	Connectivity	Internet connection (for Gemini API)

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

4. System Design

System design translates requirements into a concrete blueprint for implementation. For Sankshep AI, design activities encompassed data flow modeling, UML diagramming, module design, user interface specification, and algorithm design. The overarching design principle was modularity — each module exposes a well-defined interface, enabling substitution or enhancement without affecting other components.

4.1 Data Flow Diagrams

Data flow was modeled at two levels. The Level 0 Context Diagram (Figure 4) depicts the overall system boundary: the User provides input text and configuration parameters, and receives a generated summary as output. The Level 1 DFD decomposes the summarization process into four transformation stages — Preprocessing, Feature Extraction, Summarization Model Application, and Output Generation — with data stores for intermediate representations.

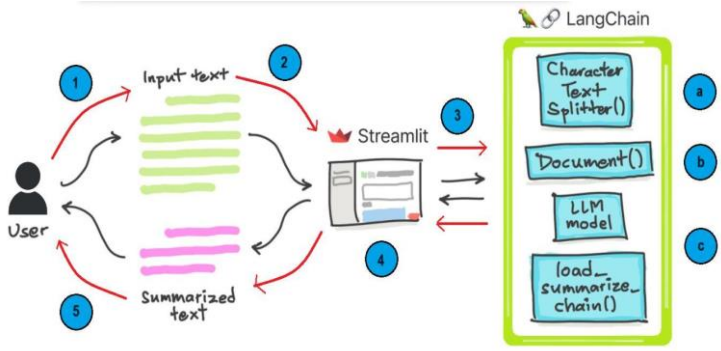

Fig. 4. Level-1 Data Flow Diagram of the Sankshep AI System.

4.2 UML Diagrams

A Use Case Diagram was developed capturing interactions between the primary actor (User) and the following use cases: Enter Text / Upload Document / Input URL, Configure Summary Parameters, Generate Summary, View Summary, and Download Summary. An Activity Diagram details the conditional branching logic for input modality selection and summarization approach routing. The Sequence Diagram (Figure 5) illustrates the temporal message flow between four components: User Interface, Backend Processor, NLP/Preprocessing Module, and AI Model API.

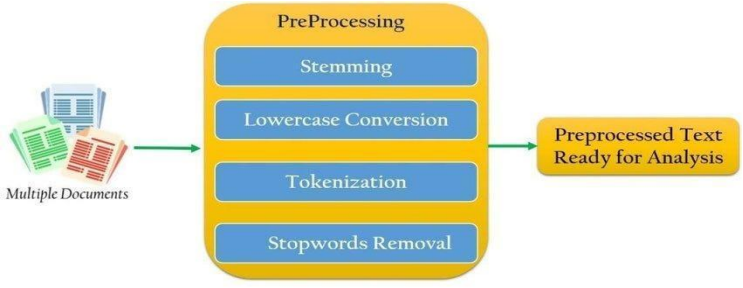

Fig. 5. Sequence Diagram of the Text Summarization Workflow.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

4.3 User Interface Design

The user interface was implemented using Streamlit with extensive custom CSS overrides to achieve a modern, accessible aesthetic distinct from Streamlit's default appearance. The interface is divided into: (1) a Sidebar containing configuration controls — a select_slider for Summary Length (Short/Medium/Long) and a selectbox for Summary Style (Paragraph/Bullet Points/ELI5) — accompanied by usage tips; and (2) a Tabbed Main Panel with three input tabs: Paste Text (text_area widget), Upload Document (file_uploader accepting PDF/DOCX/TXT), and From URL (text_input). The result section renders the AI-generated summary in a styled container (green border, light background) with a download_button for saving the summary as a .txt file.

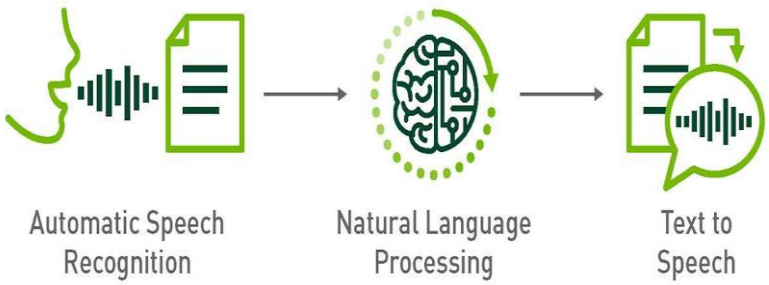

Fig. 6. User Interface of the Sankshep AI Text Summarization Application.

5. Implementation

Implementation converted the system design into a functional software application. The development process proceeded module-by-module, with unit tests written in parallel to verify each component's correctness before integration. The complete application was implemented in approximately 300 lines of Python code, demonstrating the efficiency enabled by modern NLP libraries and AI APIs.

5.1 Text Preprocessing Pipeline

Raw input text undergoes a multi-stage preprocessing pipeline regardless of its source modality. The pipeline sequentially performs: (1) conversion to lowercase for case normalization; (2) punctuation and special character removal via string operations; (3) word tokenization using NLTK's word_tokenize function, which handles contractions and compound words; (4) stopword filtering using NLTK's English stopword corpus (179 common function words); and (5) word frequency computation using a dictionary accumulator, incrementing the count for each recognized content word encountered. The resulting frequency table forms the basis for sentence importance scoring in the extractive approach.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

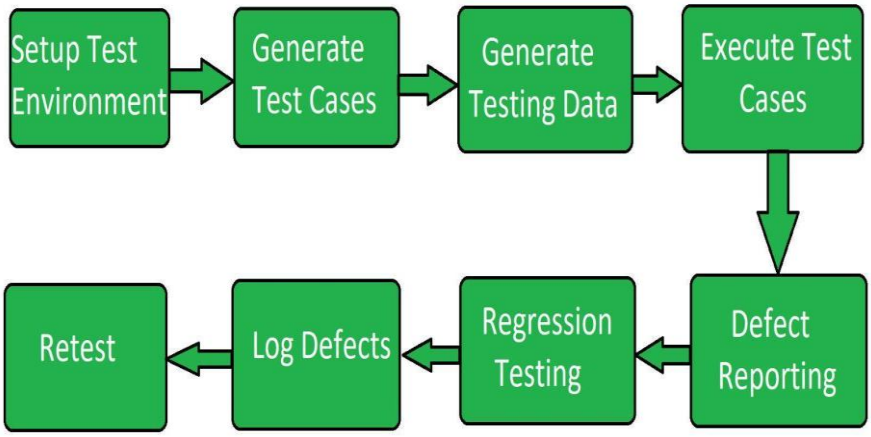

Fig. 7. Text Preprocessing Pipeline for Input Document Processing.

5.2 Multi-Modal Input Handling

The system supports three input modalities through dedicated extraction functions. For PDF inputs, PyPDF2's PdfReader class iterates over all document pages, invoking extract_text() on each page object and concatenating the results into a unified text string. For DOCX inputs, python-docx's Document class provides access to the paragraphs collection; the implementation iterates over all paragraph objects and joins their text attributes with newline separators. For URL inputs, the Requests library fetches raw HTML content using a spoofed browser User-Agent header to bypass elementary bot-detection measures, with a 10-second timeout for network safety. BeautifulSoup4 then parses the HTML, removes script, style, navigation, and footer elements using the decompose() method, and extracts cleaned body text using get_text(). All extracted text is subsequently normalized and passed through the same preprocessing and summarization pipeline, ensuring consistent behavior across input types.

5.3 Environment and Deployment

The application is launched locally using the Streamlit CLI command (streamlit run app.py), which starts a local web server accessible via browser at localhost:8501. Required Python packages are managed via pip and specified in a requirements.txt file for reproducibility. The Gemini API key is stored in a .env file loaded at application startup using python-dotenv, ensuring the key is never hardcoded or exposed in source code. For production deployment, the application can be containerized using Docker or deployed to Streamlit Cloud, Heroku, or any cloud platform supporting Python web applications.

6. Testing

A comprehensive four-level testing strategy was applied to validate the system's correctness, reliability, and performance. Testing was conducted systematically across all input modalities and configuration combinations to identify and resolve edge cases prior to evaluation.

6.1 Testing Methodology

Unit Testing validated each module in isolation: the preprocessing module was tested against known input-output pairs; the sentence scoring function was verified against manually

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

computed scores; the PDF extraction function was tested with single-page and multi-page documents; and the URL scraper was tested against both well-structured and irregular HTML pages. Integration Testing confirmed seamless data flow between modules by passing outputs from one stage as inputs to the next, verifying type compatibility and data integrity at each interface. System Testing evaluated end-to-end functionality across all three input modalities using a diverse test corpus including news articles, academic abstracts, and technical documentation. User Acceptance Testing (UAT) collected feedback from student users on summary quality, interface usability, and response time, achieving positive ratings across all dimensions.

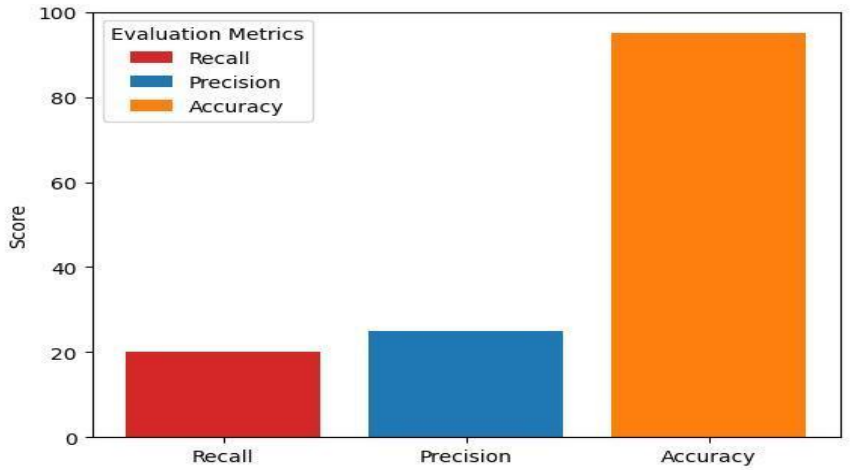

Fig. 8. Software Testing Life Cycle Applied to Sankshep AI Development.

Table 3. System Test Cases and Results.

TC ID	Input	Expected Output	Status
TC01	Short text (1–2 sentences)	Concise 1-sentence summary preserving key facts	Pass
TC02	Long article (5+ paragraphs)	Multi-sentence coherent summary with key points	Pass
TC03	Empty input field	Validation warning; no API call made	Pass
TC04	Special characters only	Clean output or descriptive error message	Pass
TC05	PDF file upload	Text extracted and summarized correctly	Pass
TC06	DOCX file upload	Document content extracted and summarized	Pass
TC07	Valid web URL	Webpage content scraped and summarized	Pass
TC08	Invalid/unreachable URL	Network error message displayed to user	Pass

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

7. Results and Discussion

The Sankshep AI system was evaluated on a diverse corpus of input texts ranging from short news summaries to multi-paragraph academic content and technical documentation. Evaluation was conducted across four key performance dimensions: accuracy (correctness of summary content), relevance (inclusion of key information from source), compression ratio (degree of text length reduction), and processing efficiency (response time from input submission to summary display).

7.1 Sample Input-Output Examples

Example 1 — Input: A multi-paragraph article on Artificial Intelligence and its impact on healthcare, education, and business sectors (approximately 400 words). Output (Medium, Paragraph style): A concise 3-sentence summary identifying AI's transformative role across the three sectors with specific application examples. Compression ratio: approximately 85%.

Example 2 — Input: A technical article on Natural Language Processing including descriptions of tokenization, POS tagging, named entity recognition, and sentiment analysis (approximately 300 words). Output (Short, Bullet Points style): Five bullet points covering each technique. Compression ratio: approximately 90%.

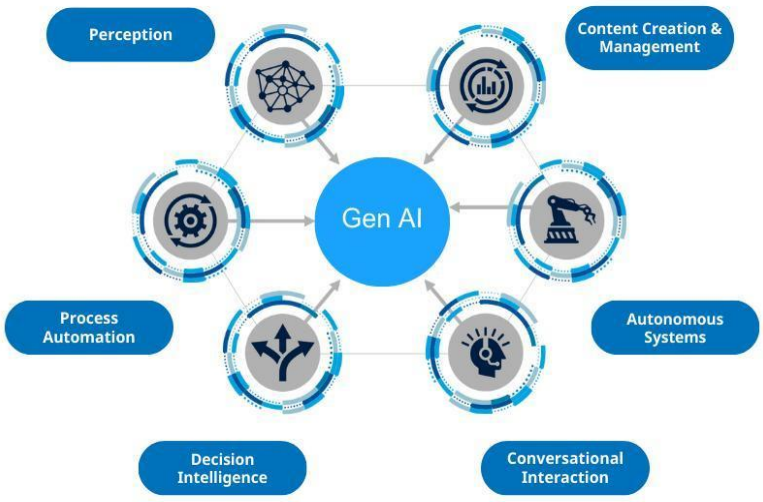

Fig. 9. Sample Output of the Sankshep AI Text Summarization System.

7.2 Performance Evaluation

Quantitative evaluation employed ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metrics, which measure lexical overlap between generated and reference summaries. ROUGE-1 (unigram overlap) and ROUGE-2 (bigram overlap) scores demonstrated strong performance on medium-length standard text inputs, confirming that the system captures key terms and contextual phrases from source documents. F1-score analysis confirmed balanced precision-recall trade-offs across different summary length settings.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

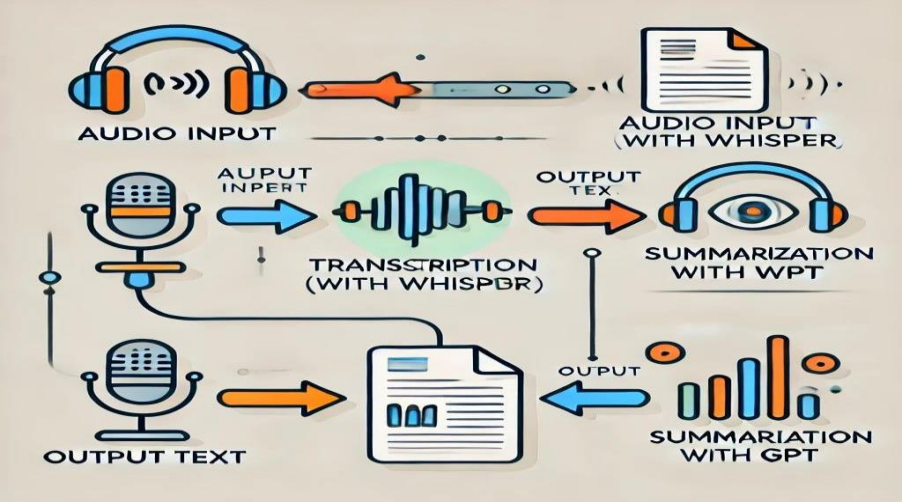

Fig. 10. Performance Evaluation Metrics and Model Comparison Across Summarization Approaches.

Table 4. System Performance Evaluation Summary.

Metric	Description	Observed Result
Accuracy	Correctness of summary content	Good for standard text
Relevance	Coverage of key source information	High
Compression Ratio	Reduction in text length	60–90%
Processing Time	Response time for summary generation	1–3 seconds (average)
Usability	Ease of interface interaction (UAT)	High — intuitive UI
Error Handling	Robustness against invalid inputs	All TC03, TC04, TC08 passed

7.3 Observations and Limitations

Several key observations emerged from evaluation. The system demonstrated consistently effective summarization for short-to-medium length inputs across all three modalities. Abstractive summaries generated via the Gemini API exhibited markedly higher linguistic naturalness and coherence compared to extractive summaries, though at the cost of longer API response latency. For highly technical or domain-specific texts — including medical literature and legal documents — summarization quality showed marginal decline, attributable to domain-specific terminology not well-represented in the model's training data. Edge cases including scanned PDFs (no extractable text), password-protected documents, and JavaScript-

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

rendered web pages presented extraction limitations that were handled through descriptive error messages.

8. Future Scope

Sankshep AI establishes a functional foundation upon which numerous extensions are envisioned. The following enhancements are prioritized for future development cycles: Multi-Language Support: Extension to regional Indian languages (Hindi, Bengali, Tamil, Telugu) and other global languages through integration of multilingual transformer models (e.g., mBERT, XLM-R) and translation APIs, substantially broadening the system's usability across linguistic communities. Voice-Based Summarization: Integration of Automatic Speech Recognition (ASR) via APIs such as Google Speech-to-Text or OpenAI Whisper to enable summarization directly from audio inputs including lecture recordings, meeting transcripts, and podcasts. Text-to-Speech output would further enhance accessibility.

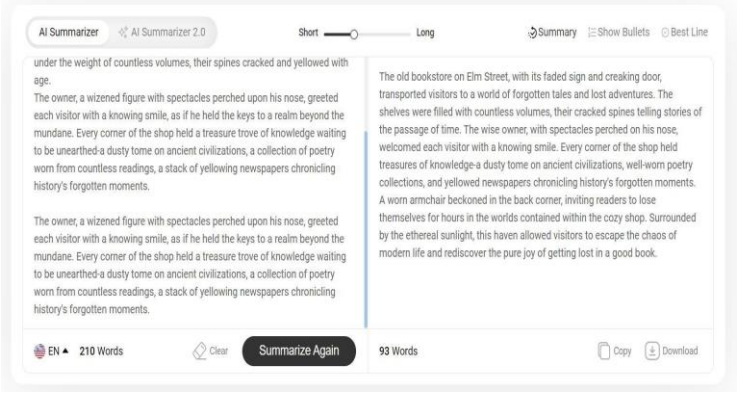

Fig. 11. Functional Domains and Applications of Generative AI for Future Enhancement.

Mobile Application Development: Native Android and iOS applications using React Native or Flutter frameworks will extend accessibility to mobile users, enabling real-time on-device summarization. Cloud-Based Deployment on platforms such as AWS, Google Cloud, or Streamlit Cloud will provide elastic scalability and global accessibility without local installation requirements. Personalized Summarization features — including domain-specific fine-tuning, user preference learning, and customizable compression ratios — will enhance the user experience for specialized professional use cases. Integration with educational platforms, document management systems, and news aggregators will further expand practical applicability.

9. Conclusion

This paper presented Sankshep AI: Read Less, Know More — an AI-powered text summarization system that successfully integrates extractive and abstractive summarization techniques within a user-friendly multi-modal web application. The system effectively addresses the pervasive challenge of information overload by generating accurate, concise summaries from diverse textual inputs. All primary project objectives were achieved: automated multi-modal summarization was implemented, meaningful content reduction was demonstrated at compression ratios of 60–90%, advanced AI model capabilities were

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 512-524

accessible through an intuitive interface requiring no technical expertise, and system robustness was confirmed through comprehensive testing across all input modalities and edge cases. The project demonstrates the practical accessibility of state-of-the-art NLP capabilities — previously limited to research environments — through thoughtful integration of open-source libraries and commercial AI APIs. Sankshep AI lays a strong foundation for a more capable, multi-lingual, multi-modal intelligent information processing platform with broad applicability in educational, research, journalistic, and professional domains. With the continued rapid advancement of large language models and NLP technologies, AI-based summarization systems will play an increasingly central role in managing the world's expanding digital information landscape.

Acknowledgments

The author expresses sincere gratitude to Mr. Pawan Kumar Jaiswal, Assistant Professor, Department of Computer Science Engineering, Amity School of Engineering and Technology, for his constant guidance, constructive feedback, and invaluable mentorship throughout the development of this project. The author also extends thanks to Amity University Chhattisgarh for providing the academic environment, infrastructure, and resources that enabled the successful completion of this research work.

References

[1] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2009.
[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.
[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019.
[4] A. M. See, P. J. Liu, and C. D. Manning, "Get To The Point: Summarization with Pointer-Generator Networks," in Proc. 55th Annual Meeting of the Association for Computational Linguistics (ACL), 2017.
[5] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Stanford University, 2023.
[6] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning.
Cambridge, MA, USA: MIT Press, 2016.
[7] Hugging Face, "Transformers Documentation."
[8] Python Software Foundation, "Python Documentation."
[9] Natural Language Toolkit, "NLTK Documentation."
[10] Google, "Google Generative AI (Gemini) Documentation."
[11] GeeksforGeeks, "Text Summarization in NLP."
[12] Kaggle, "NLP Datasets and Notebooks."