Grenze International Journal of Engineering and Technology, June Issue

ReadAssist: An AI-Powered Intelligent Reading and Interactive Insight Generation System

Adhiraj Patil[1], Ganesh Bhutkar[2], Asmit Bhorade[3], Aniruddha Ghosh[4], Achintyaa Dinesh[5] and Nikhil Jathar[6]
[1-5]Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra, India
Email: rajesh.adhiraj24@vit.edu, ganesh.bhutkar@vit.edu, asmit.bhorade24@vit.edu, aniruddha.ghosh24@vit.edu, achintyaa.dinesh24@vit.edu
[6]AvanSaber Technologies Pvt. Ltd., Pune, Maharashtra, India
Email: nikhil.jathar@avansaber.com

***Abstract*— The rapid growth of digital content has increased the demand for intelligent tools that improve reading comprehension and knowledge retention. This paper presents ReadAssist, an AI-powered intelligent reading and insight generation system designed to enhance the digital reading experience through real-time contextual assistance and personalized learning support. The proposed system combines viewport-aware context extraction, hybrid AI inference, deterministic caching, and dynamic user interface generation to provide features such as contextual explanations, multilingual translation, knowledge blocks, quizzes, readability assessment, and smart note generation. A hybrid architecture utilizing local large language models through Ollama and cloud-based Gemini services ensures both privacy and scalability. Experimental evaluation demonstrates significant performance improvements through Redis-based caching and efficient context extraction strategies, enabling responsive and interactive user experiences. By integrating AI-driven educational assistance directly into the reading workflow, ReadAssist transforms passive document consumption into an active and engaging learning process.**

***Index Terms*— Artificial Intelligence, Interactive Reading, Text Summarization, Knowledge Enhancement, Educational Technology, Visualization.**

I. INTRODUCTION

Digital reading has become an essential part of modern education, research, and professional activities. Although modern PDF readers provide efficient document rendering, they offer limited support for comprehension and interactive learning. Readers often encounter complex terminology or language barriers that require additional explanation, forcing them to switch between search engines, dictionaries, and note-taking applications. This repeated context switching interrupts concentration and reduces overall learning efficiency.

Recent developments in Artificial Intelligence (AI) and Large Language Models (LLMs) have created new opportunities for improving reading comprehension. However, most existing AI-based systems operate independently of the reading environment, requiring users to manually copy and paste content. To address these limitations, this paper presents ReadAssist, an AI-powered intelligent reading system designed to transform passive reading into an active learning experience by integrating document viewing, contextual explanations, multilingual translation, quizzes, and smart note generation within a unified environment.

Grenze ID: 01.GIJET.12.2.62_26

ReadAssist employs a viewport-aware context extraction mechanism that processes only the visible content, reducing unnecessary computation. The system incorporates a hybrid AI inference architecture combining local language models (Ollama) with cloud-based Gemini services to achieve a balance between privacy, scalability, and performance. Additionally, deterministic caching using Redis minimizes latency. The primary objective is to enhance reading comprehension and long-term knowledge retention while providing a seamless, platform-independent experience.

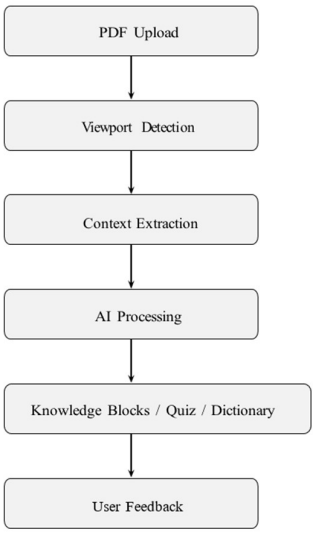

Fig. 1. Operational Workflow of the ReadAssist Pipeline

II. Literature Review

A. PDF-Based Document Rendering

Desai and Vasa [1] presented a detailed study of PDF.js, a framework for rendering PDF files directly within modern web browsers. ReadAssist builds upon PDF.js by extending its role beyond visualization, combining it with AI-driven processing to deliver interactive learning support.

For text extraction, Smith [2] introduced the Tesseract Optical Character Recognition (OCR) engine, widely used for extracting machine-readable text from scanned documents. Harley et al. [3] evaluated modern deep-learning-based OCR approaches, demonstrating that preprocessing methods such as denoising and segmentation significantly improve accuracy. ReadAssist incorporates these preprocessing techniques and OCR as a fallback mechanism for non-selectable text, ensuring reliable text extraction for downstream AI analysis.

B. Optical Character Recognition

Smith [2] introduced the Tesseract Optical Character Recognition (OCR) engine, one of the most widely used open-source OCR systems. Tesseract performs document layout analysis, character recognition, and language-based correction to extract machine-readable text from scanned documents and images.

The OCR engine has been successfully applied in numerous digitization projects involving books, reports, and archival records. However, recognition accuracy can degrade when processing noisy scans, skewed pages, low-resolution images, or complex document layouts. Consequently, additional image preprocessing techniques are often required to improve OCR performance.

ReadAssist incorporates OCR as a fallback mechanism for documents that do not contain selectable text. To improve recognition accuracy, preprocessing techniques such as image enhancement, denoising, contrast adjustment, and deskewing are applied before OCR execution. This approach enables reliable text extraction while minimizing errors that could affect downstream AI-based analysis.

C. Deep Learning-Based OCR Enhancement

- To address this, ReadAssist introduces the following novel contributions:
- Viewport-aware context extraction that processes only visible content to reduce computational overhead.

- Hybrid AI inference combining local models (Ollama) and cloud services (Gemini) for optimal privacy and speed.
- Dynamic generation of UI components like contextual explanations, knowledge blocks, and quizzes.
- Deterministic Redis-based caching that minimizes response latency and eliminates redundant computations.

D. Semantic Retrieval Using Sentence-BERT

Reimers and Gurevych [4] proposed Sentence-BERT (SBERT), an extension of the BERT architecture designed for efficient semantic similarity computation. SBERT has been widely adopted in applications such as semantic search, question answering, recommendation systems, and information retrieval.

ReadAssist utilizes SBERT-based embeddings to represent document content in a semantic vector space. When a user requests an explanation or asks a question, the system retrieves the most relevant content segments using similarity matching. This mechanism enables efficient contextual retrieval and improves the relevance of AI-generated responses.

E. Retrieval-Augmented Generation

Lewis et al. [5] introduced Retrieval-Augmented Generation (RAG), a framework that combines information retrieval with language generation. Instead of relying solely on knowledge stored within model parameters, RAG retrieves relevant information from an external knowledge source and uses it during response generation. This approach improves factual accuracy and reduces hallucinations in generated content.

The RAG framework demonstrated strong performance across knowledge-intensive tasks such as open-domain question answering, fact verification, and information retrieval. However, most implementations focus on large-scale external knowledge bases rather than document-centric reading environments.

ReadAssist adopts a document-centric RAG architecture in which relevant passages are retrieved directly from the uploaded document. These passages are then supplied to the language model to generate grounded explanations, summaries, and answers. This approach ensures that responses remain contextually relevant and traceable to the original document.

F. Research Gap and Novel Contributions

Existing studies primarily focus on individual technologies such as PDF rendering, OCR, semantic retrieval, and retrievalaugmented generation. While these technologies are effective within their respective domains, they are typically implemented as independent solutions and do not provide a unified environment for reading comprehension and knowledge enhancement.

Traditional document viewers focus on visualization and navigation, OCR systems focus on text extraction, and AI systems focus on retrieval or content generation. Few existing solutions integrate these capabilities into a single platform that actively assists users while reading.

To address this gap, ReadAssist introduces the following novel contributions:

- Viewport-aware context extraction that processes only the content currently visible to the reader, reducing computational overhead and improving contextual relevance.
- Hybrid AI inference combining local language models through Ollama with cloud-based Gemini services to balance privacy, scalability, and response quality.
- Dynamic generation of educational user interface components including contextual explanations, knowledge blocks, quizzes, and interactive learning aids.
- Deterministic Redis-based caching that minimizes response latency and eliminates redundant AI computations.
- Integration of document rendering, semantic retrieval, AI assistance, and educational content generation within a single unified reading environment.

These contributions distinguish ReadAssist from traditional PDF readers, standalone OCR systems, and generic AI assistants by providing an interactive, personalized, and context-aware reading experience.

III. METHODOLOGY

A. Abbreviations and Acronyms

ReadAssist follows a modular, multi-layered architecture designed for scalability. The frontend is developed using React.js and React Native [6], providing a consistent user experience. The backend utilizes FastAPI in Python [7] for high-performance asynchronous API handling. PostgreSQL [8] serves as the primary database,

while Redis [9] is used for low-latency caching. Document processing is performed via PDF.js [10], and AI inference integrates local models through Ollama [11] and cloud models via Google Gemini [12].

TABLE I. COMPARISON WITH EXISTING APPROACHES

Feature	PDF.js	OCR Tools	RAG Systems	ReadAssist
Document Rendering	Yes	No	No	Yes
Contextual Explanations	No	No	Partial	Yes
Interactive Quizzes	No	No	No	Yes
Viewport Awareness	No	No	No	Yes
Hybrid AI Inference	No	No	Partial	Yes
Knowledge Blocks	No	No	No	Yes
Real-Time Assistance	No	No	Partial	Yes

B. System Architecture and Cross-Platform Orchestration

ReadAssist follows a modular multi-layered architecture designed for scalability, maintainability, and cross-platform compatibility.

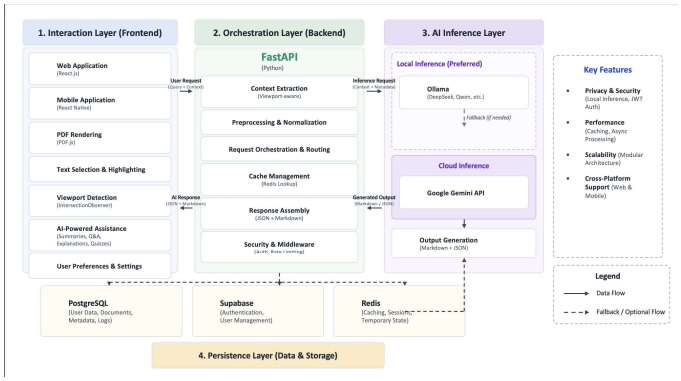

Fig. 2. Overall Architecture of Readassist

TABLE II: TECHNOLOGY STACK SUMMARY

Component	Technology	Rationale
Frontend	React / React Native	Cross-platform UI with code reuse
Backend	FastAPI (Python)	High-performance asynchronous API framework
Database	PostgreSQL / Supabase	Relational storage with authentication support
Cache	Redis	Low-latency caching and state management
AI Inference	Ollama / Google Gemini	Hybrid privacy-aware inference architecture

The architecture consists of three primary layers:

Interaction Layer

The interaction layer is responsible for document rendering, user interaction, and educational content visualization. It is implemented using React.js for web platforms and React Native for mobile devices. Key responsibilities include:

- Document rendering and navigation
- Text selection and highlighting
- Interactive educational content
- User preference management
- Cross-platform user interface support

A WebView-based rendering engine enables dynamic integration of AI-generated educational content directly within the reading interface.

Orchestration Layer

The orchestration layer is implemented using FastAPI and acts as the central processing unit of the system.
Its responsibilities include:

- Context extraction and preprocessing
- Request routing and orchestration

- AI inference management
- Security enforcement
- Response generation

FastAPI was selected due to its asynchronous execution capabilities, high throughput, and seamless integration with Python-based AI frameworks.

Persistence Layer

The persistence layer manages long-term and temporary data storage.

- PostgreSQL stores user profiles, metadata, and application records.
- Supabase manages authentication and session control.
- Redis provides high-speed caching and temporary state storage.

This layered architecture improves maintainability, reliability, and scalability.

C. Viewport-Aware Context Extraction Mechanism

A key innovation of ReadAssist is its viewport-aware context extraction mechanism. Instead of processing an entire document, the system focuses only on the content currently visible to the reader.

The workflow is as follows:

- The document is rendered using PDF.js inside a WebView.
- The IntersectionObserver API monitors visible text elements.
- Visibility thresholds are computed using bounding box analysis.
- Visible content is extracted and serialized.
- The extracted context is forwarded to the AI inference pipeline. This approach offers several advantages:
  - Reduced computational overhead
  - Improved contextual relevance
  - Lower memory consumption
  - Faster response generation
  - Enhanced privacy through limited content processing

D. Intelligent AI Inference Pipeline

The ReadAssist AI pipeline employs a hybrid inference architecture that combines local and cloud-based language models.

Local Inference

Local inference is performed using Ollama [11]-hosted language models such as DeepSeek and Qwen.

Benefits include:

- Reduced latency
- Improved privacy
- Offline processing capability
- Lower operational costs

Cloud Fallback

Cloud-based inference using Google Gemini [12] is activated when:

- Local inference exceeds predefined latency thresholds
- Hardware resources become constrained
- Complex reasoning tasks are detected

This fallback mechanism ensures uninterrupted service and consistent response quality.

Output Synthesis

The AI pipeline generates two output formats:

1) Markdown content for educational explanations and summaries. 2) JSON metadata for frontend rendering.

The JSON structure contains quiz configurations, contextual definitions, styling information, and layout instructions, allowing dynamic generation of educational components.

E. Dynamic UI Synthesis

ReadAssist employs a JSON-driven rendering architecture to transform AI-generated metadata into interactive educational components.

Supported UI elements include:

- Knowledge blocks

- Interactive quizzes
- Contextual definitions
- Expandable explanations
- Educational tooltips The rendering workflow consists of:
  - Parsing AI-generated JSON metadata.
  - Mapping metadata to frontend components.
  - Dynamic rendering within the reading interface. 4) Event-driven interaction management.

This architecture enables adaptive and context-aware user interfaces that enhance reading comprehension and engagement.

F. Performance Optimization and Deterministic Caching

To achieve real-time responsiveness, ReadAssist employs a deterministic caching strategy using Redis. A unique SHA-256 hash is generated from:

- User query
- Extracted context
- Model parameters
- Processing configuration

The generated hash serves as a cache key.
The caching workflow is:

- Generate request hash.
- Query Redis for existing results.
- Return cached results if available.
- Execute AI inference on cache miss.
- Store newly generated results.

Benefits include:

- Reduced latency
- Lower API costs
- Improved scalability
- Consistent responses

G. Readability Metrics

ReadAssist evaluates document complexity using standard readability measures.

$$FKGL = 0.39(ASL)+11.8(ASW)-15.59 \quad (1)$$

$$FRE = 206.835-1.015(ASL)-84.6(ASW) \quad (2)$$

where ASL represents Average Sentence Length and ASW represents Average Syllables per Word.
These metrics help the system adapt explanations and educational content according to document complexity.

H. Security Framework

Security is maintained through multiple protection layers.

- JWT-based authentication using Supabase.
- Input validation and sanitization.
- API rate limiting.
- Secure session management.
- Local inference support for privacy-sensitive content.

IV. Results and Discussion

A. Performance Metrics, Latency Profiling, and System Throughput

A comprehensive evaluation of ReadAssist was conducted to analyze system latency, throughput, and responsiveness under different operating conditions. The evaluation considered multiple factors including cache state, inference mode, document complexity, and user interaction patterns.
During cold-cache execution using local inference through Ollama (DeepSeek-7B) [11], the end-to-end response latency ranged between 4.2 and 7.8 seconds. This latency includes context extraction, preprocessing, prompt generation, model inference, and response formatting. Experimental observations indicate that the majority of execution time is consumed by language model inference, while the remaining time is spent on preprocessing

and system-level operations. The implementation of deterministic caching using Redis significantly improved system responsiveness. For repeated requests resulting in cache hits, average response time was reduced to less than 45 milliseconds. This represents a latency improvement of more than 98% compared to cold-cache execution.

The hybrid inference architecture further improves system stability by dynamically routing requests between local and cloud-based models. When local resource utilization exceeds predefined thresholds, requests are redirected to Google Gemini, ensuring consistent performance under varying workloads.

Experimental testing demonstrated that the system can sustain an average throughput of approximately 12 requests per minute in a single-instance deployment while maintaining stable response times. The use of asynchronous processing and non-blocking API operations contributes to improved resource utilization and overall scalability. The observed performance improvements demonstrate the effectiveness of combining viewport-aware processing with deterministic caching. By restricting AI processing to visible content, the system reduces unnecessary token consumption and computational overhead. This optimization contributes directly to lower response times and improved resource utilization. Furthermore, the hybrid inference architecture enables the platform to maintain service availability even under varying workload conditions, providing a practical balance between privacy, scalability, and computational efficiency.

The quantitative performance metrics obtained during experimental evaluation are summarized in Table III.

The overall functionality of the major system modules was validated during experimental testing. Table IV summarizes the implementation status and observed outcomes of the core features integrated within the ReadAssist platform. The experimental observations further indicate that the combination of viewport-aware processing and hybrid AI inference significantly improves overall system efficiency. By restricting analysis to visible content, unnecessary processing is avoided, resulting in lower token consumption and reduced computational overhead. Furthermore, the hybrid inference strategy enables the system to maintain consistent service quality while balancing privacy requirements and resource availability. These findings demonstrate the practical feasibility of deploying ReadAssist in educational and professional environments where responsiveness and scalability are critical requirements.

B. Discussion

The experimental results demonstrate the effectiveness of the proposed ReadAssist architecture in improving both system performance and user experience. The viewport-aware context extraction mechanism reduces the amount of content processed during inference by focusing only on the visible portion of the document. This approach minimizes unnecessary computational overhead while maintaining contextual relevance.

The Redis-based caching layer plays a significant role in improving responsiveness by eliminating redundant AI computations. The observed reduction in latency confirms the effectiveness of deterministic caching for repeated user queries and similar reading scenarios.

Unlike conventional PDF readers that primarily focus on document visualization, ReadAssist actively supports learning through contextual explanations, readability assessment, intelligent summaries, knowledge blocks, and interactive quizzes. The integration of local and cloud-based language models further ensures a balance between privacy, performance, and scalability.

Overall, the results indicate that ReadAssist successfully transforms passive reading into an interactive learning experience while maintaining practical performance for real-world deployment.

TABLE III. PERFORMANCE EVALUATION RESULTS

Metric	Observed Value
Cold Cache Latency	4.2 – 7.8 s
Warm Cache Latency	< 45 ms
Latency Improvement	> 98%
Average Throughput	12 requests/min
Cache Lookup Complexity	O(1)
Inference Modes	Local + Cloud Hybrid

TABLE IV. FEATURE VALIDATION RESULTS

Feature	Status
Viewport Context Extraction	Successful
AI Explanations	Successful
Smart Dictionary	Successful
Knowledge Blocks	Successful

Quiz Generation	Successful
Translation Support	Successful
Deterministic Caching	Successful
Hybrid AI Routing	Successful

V. CONCLUSION

This paper presented ReadAssist, an AI-powered intelligent reading system designed to enhance digital learning by transforming passive document consumption into an active, engaging experience. The system integrates viewport-aware context extraction, hybrid AI inference (Ollama and Gemini), and deterministic caching to effectively balance privacy, scalability, and computational efficiency.

A key contribution of this framework is the seamless integration of real-time educational assistance—such as contextual explanations, interactive quizzes, and knowledge blocks—directly within the reading workflow. Experimental evaluations demonstrated that this architecture, particularly the Redis-based caching mechanism, significantly reduces latency and ensures highly responsive, context-aware assistance suitable for real-world deployment.

Despite its advantages, the system's performance remains dependent on the capabilities of the underlying language models and the user's available hardware resources. Resource-constrained devices may require more frequent routing to cloud-based inference services, which can potentially increase dependency on network connectivity and external API availability.

Future work will focus on integrating multimodal understanding capabilities to analyze visual elements like charts and diagrams, alongside adaptive learning mechanisms to personalize educational content based on user behavior. Ultimately, ReadAssist provides a practical and scalable solution, representing a significant step toward next-generation intelligent reading systems.

REFERENCES

[1] A. Desai and S. Vasa, “PDF.js: A Case Study of JavaScript-Based PDF Rendering and Performance,” in Proc. IEEE Int. Conf. Web Technologies, 2021, pp. 45–52.

[2] R. Smith, “An Overview of the Tesseract OCR Engine,” in Proc. Int. Conf. Document Analysis and Recognition (ICDAR), 2007, pp. 629–633.

[3] A. W. Harley, A. Ufkes, and K. G. Derpanis, “Evaluation of Deep Learning OCR Techniques for Document Digitization,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 44, no. 12, pp. 8921–8932, 2022.

[4] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings Using Siamese BERT Networks,” in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2019, pp. 3982–3992.

[5] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[6] Facebook Inc., “React: A JavaScript Library for Building User Interfaces,” 2024. [Online]. Available: https://react.dev

[7] FastAPI, “FastAPI Framework Documentation,” 2024. [Online]. Available: https://fastapi.tiangolo.com

[8] PostgreSQL Global Development Group, “PostgreSQL Documentation,” 2024. [Online]. Available: https://www.postgresql.org

[9] Redis Labs, “Redis: In-Memory Data Structure Store,” 2024. [Online]. Available: https://redis.io

[10] Mozilla Foundation, “PDF.js Documentation,” 2024. [Online]. Available: https://mozilla.github.io/pdf.js/

[11] Ollama, “Running Large Language Models Locally,” 2024. [Online]. Available: https://ollama.ai

[12] Google, “Gemini: A Family of Highly Capable Multimodal Models,” 2024. [Online]. Available: https://deepmind.google