International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 6, Nov-Dec-2026, ISSN (Online): 2395-566X

Nexel AI: A Multimodal Retrieval-Augmented Generation System for Intelligent Document Processing, Multilingual Learning, and Collaborative Study

Somil Salodiya[1] Dr. Pradeep[2]
Department Computer Science And Engineering (Cse
Hmr Institute Of Technology And Management, Delhi, India

Abstract- Students increasingly depend on digital documents such as lecture notes, textbooks, and past examination papers, yet turning those documents into active study material usually requires several disconnected tools. This paper presents Nexel AI, a deployed web application that converts uploaded PDF files into interactive study resources through a Retrieval-Augmented Generation (RAG) pipeline. The system extracts text with unpdf, divides it into chunks, embeds the chunks with the 768-dimensional jina-embeddings-v2-base-en model from Jina AI, and stores the vectors in Supabase PostgreSQL with pgvector. At query time the question is embedded, matched to stored chunks by cosine similarity, and passed as context to Groq-hosted language models (openai/gpt-oss-20b and openai/gpt-oss-120b), which produce answers with source citations. The same retrieval layer drives notes, flashcards, quizzes, a narrated lecture generator, a study roadmap generator, and an exam predictor. Prompts support English, Hindi, and Hinglish; an AI Tutor accepts text, voice, images, and PDFs, with Google Gemini providing image analysis; and Supabase Realtime supports collaborative study rooms. The application is built with Next.js 16, TypeScript, and Tailwind CSS, is installable as a Progressive Web App, and is deployed on Vercel. The paper describes the architecture, the engineering challenges met in serverless deployment, and a functional evaluation based on developer-observed behaviour. No benchmark figures or learning-outcome claims are made; a formal user study is identified as future work. The work shows that a single document-grounded pipeline can support several study modes across multiple languages within one integrated product.

Keywords: retrieval-augmented generation, educational technology, multilingual learning, multimodal language models, collaborative learning, Progressive Web App.

I. INTRODUCTION

1. Background

Digital documents have become the main medium through which university students meet course material. Lecture slides, scanned notes, textbooks, and past examination papers are most often shared as Portable Document Format (PDF) files, which preserve layout but offer little support for active study. At the same time, advances in neural language modelling have changed what software can do with text. The Transformer architecture [14] replaced recurrence with attention and made large-scale pre-training practical. Bidirectional pre-training, demonstrated by BERT [16], set a new standard for language understanding. Large autoregressive models then showed that a sufficiently scaled system can perform new tasks from a handful of examples placed in the prompt [12], and instruction-based fine-tuning was shown to improve zero-shot behaviour on tasks the model had not been trained on [11].

These capabilities make language models attractive as study assistants, but they also expose a familiar weakness: a model that answers only from its trained parameters can produce fluent statements that are not grounded in the learner's own course material, and fluency alone is not evidence of understanding or accuracy [15]. Retrieval-Augmented Generation (RAG) addresses this by combining a generative model with an external, searchable memory [1]. Rather than relying only on what the model has memorised, the system retrieves passages relevant to a question and supplies them as context for generation, so that answers can be tied to identifiable sources. In an educational setting, the natural external memory is the student's own set of documents.

2. Problem Statement

Four recurring problems motivate this work. They are described here from the perspective of a student preparing for university examinations.

- **Fragmented workflow.** A student who wants to study from a PDF typically moves between a PDF reader, a general-purpose chat assistant, a note-taking tool, a flashcard application, and a messaging group. Each hand-off requires copying text, re-explaining context, and reformatting output, and none of the tools shares an understanding of the underlying document.
- **English-centric tooling.** Many students in India think and discuss in Hindi or in Hinglish, the informal mixture of Hindi written in Roman script with English technical

© 2026 IJSRET

vocabulary. Most study tools assume English input and output, which creates friction exactly where a student would benefit from an explanation in a more comfortable language. Cross-lingual benchmarks such as XTREME also show that performance can differ markedly across languages [6].

- **Passive learning.** Reading and highlighting are comparatively passive activities. Research on test-enhanced learning and distributed practice suggests that retrieval and spacing strengthen long-term retention [3], [4], yet producing good practice questions from one's own material is laborious, so students often skip it.
- **Lack of collaboration.** Study is often a social activity, but document-centred tools rarely offer shared, real-time study spaces that stay connected to the material being studied [5].

3. Motivation And Proposed Approach

The motivation for Nexel AI is to place these activities behind one document-grounded pipeline. The student uploads a PDF once. The system extracts and indexes its content and then reuses the same indexed knowledge for question answering with citations, generated notes, flashcards, quizzes, narrated lectures, study roadmaps, and exam-topic prediction. Language preference is treated as a first-class input, so that English, Hindi, and Hinglish are supported by prompt design rather than by separate products. An AI Tutor extends interaction beyond typed text by accepting voice input, images, and additional PDFs, and collaborative study rooms allow several learners to work together in real time.

The proposed approach is deliberately built from managed services so that a single developer can deliver and operate it: Next.js on Vercel for the application, Supabase for authentication, storage, relational data, vector search, and realtime messaging, Groq for fast language-model inference, Jina AI for embeddings, and Google Gemini for image understanding. The contribution of the project lies in the integration of these parts into a coherent study environment and in the engineering decisions needed to make that integration work within serverless constraints.

4. Objectives

- Design and implement a RAG pipeline that ingests PDF files, indexes them semantically, and answers questions with source citations.
- Generate notes, flashcards, and quizzes from the same retrieved document context, so that passive reading can be converted into active practice.
- Produce long-form narrated lectures with chapter structure and voice output from an uploaded document.
- Support English, Hindi, and Hinglish in prompts, generated content, and speech interaction.
- Provide a multimodal AI Tutor that accepts text, voice, images, and PDFs, together with real-time collaborative study rooms.
- Deliver the system as a deployed, installable Progressive Web App and document the engineering challenges and limitations honestly.

5. Scope And Paper Organization

The scope of this paper is the description of the architecture and implementation of Nexel AI and a functional evaluation based on developer-observed behaviour of the deployed system. The paper does not report benchmark measurements, and it does not claim validated learning outcomes; both would require controlled studies that are outside the scope of a minor project and are listed as future work.

The remainder of the paper is organised as follows. Section 2 reviews related literature and identifies the research gap. Section 3 presents the methodology and system design. Section 4 describes the implementation and the challenges encountered. Section 5 presents the results and discussion, including a comparison with existing categories of tools and the threats to validity. Section 6 concludes the paper, lists limitations, and outlines future work. References and three appendices follow.

II. LITERATURE REVIEW

1. Retrieval-Augmented Generation

Lewis et al. introduced Retrieval-Augmented Generation as a way to combine a pre-trained sequence-to-sequence model, which acts as parametric memory, with a dense vector index of documents accessed through a neural retriever, which acts as non-parametric memory [1]. Their work showed that conditioning generation on retrieved passages is effective for knowledge-intensive tasks and that it offers a route to provenance, because the passages that influenced an answer can be inspected. Karpukhin et al. contributed the dense passage retrieval approach, in which questions and passages are encoded independently into a shared vector space and matched by inner product, and they showed that learned dense representations can outperform traditional sparse keyword retrieval for open-domain question answering [2]. Nexel AI adopts this retrieve-then-generate pattern, with a pre-trained embedding model for encoding and cosine similarity for matching, applied to a user's private documents rather than to a public corpus.

2. Language Models As Study Assistants

The Transformer [14] underlies nearly all current language models. BERT demonstrated the value of bidirectional pre-training for understanding tasks [16], while large autoregressive models such as those described by Brown et al. showed that in-context examples can steer behaviour

© 2026 IJSRET

without updating parameters [12]. Instruction tuning further improved the ability of models to follow natural-language task descriptions on unseen tasks [11]. These properties are what make a general model usable as a study assistant: the same model can be asked to explain, summarise, quiz, or outline simply by changing the instruction. The same literature also cautions that language models can generate plausible text without grounding and that scale brings social and environmental risks [15]. For a study tool this argues for grounding answers in the learner's own sources and for presenting model output as an aid rather than an authority.

3. Learning Science: Retrieval Practice And Spaced Repetition

The pedagogical design of Nexel AI draws on two well-established findings. The first is the testing effect. Roediger and Karpicke reported that taking memory tests improves long-term retention more than additional restudying, even though restudying can look more effective in the short term [4]. This supports generating quizzes and flashcards from study material. The second is the spacing effect, which originates in Ebbinghaus's early work on forgetting and the forgetting curve. Cepeda et al. synthesised a large body of experiments on distributed practice and found that spreading study sessions over time generally improves recall compared with massed study, with the best spacing depending on how long the material must be retained [3].

These findings concern learning in general and were not established with Nexel AI. They are used here as design motivation: the system lowers the cost of producing practice questions, which is the step that students

most often skip. The current implementation does not schedule repetitions adaptively, and adaptive spacing is therefore listed as future work.

4. Computer-Supported Collaborative Learning

Dillenbourg's edited volume on collaborative learning frames collaboration as a set of cognitive and computational processes in which learners explain, question, and negotiate meaning together, and it examines how computational systems can support these processes [5]. For a document-centred tool, the implication is that shared spaces are most useful when they remain connected to the learning material and allow interaction in real time. Nexel AI's study rooms follow this principle by providing a shared, synchronised space on top of the same platform that holds the user's documents.

5. Multilingual And Speech-Enabled Learning

XTREME is a multi-task benchmark covering 40 typologically diverse languages that evaluates how well multilingual models transfer knowledge across languages [6]. Its results highlight that cross-lingual performance is uneven and that gaps remain between English and many other languages. For Nexel AI this has two consequences. First, support for Hindi and Hinglish cannot be assumed to be equivalent to support for English and should be described cautiously. Second, language handling should be explicit in the design, through language-aware prompts and a visible language selector. For speech, the Web Speech API specification defines browser interfaces for speech recognition and speech synthesis [13], which makes voice input and spoken output available in a web application without a separate native client.

6. Multimodal Large Language Models

Recent surveys describe the rapid development of multimodal large language models that connect a language model to encoders for images and other modalities. Caffagni et al. review the architectures, training strategies, and applications of these models [7]. Zhang et al. survey recent advances in multimodal language models, including design choices for aligning modalities [8]. Yin et al. provide a further survey of the field and its open challenges [9], and Bai et al. examine multimodal models from a data-centric perspective, emphasising the role of training data in capability and reliability [10]. Together these works justify a design in which a dedicated vision-capable model handles images while text generation and retrieval remain in the main pipeline. Nexel AI follows this division by delegating image analysis to Google Gemini and combining the result with document context.

7. Additional Foundation-Model Research

Beyond the surveys above, several foundation-model contributions shape the system's assumptions. Wei et al. show that instruction-tuned models generalise to new tasks in a zero-shot setting [11]. Brown et al. show strong few-shot behaviour from prompt examples alone [12]. Vaswani et al. provide the attention-based architecture on which these models rest [14], and Devlin et al. establish the pre-training paradigm for language representations [16]. Nexel AI does not train or fine-tune any model. It relies on these results to justify the use of instruction-following models that can be directed, through carefully written prompts, to produce notes, flashcards, quizzes, and lectures from supplied context.

III. RESEARCH GAP AND SYNTHESIS

The literature provides strong support for each individual ingredient of the system: retrieval-augmented generation [1], [2], retrieval practice and spacing [3], [4], collaborative learning [5], multilingual evaluation [6], speech interfaces [13], and multimodal models [7]–[10]. What is less commonly documented is a practical, deployed integration of these ingredients for students working from their own PDFs, with explicit attention to Hindi and Hinglish, to real-time collaboration, and to the constraints of serverless

© 2026 IJSRET

deployment. Nexel AI is positioned in this gap. It is a systems contribution, not a new algorithm, and its claims are limited accordingly. Table 1 summarises the literature themes and the design implications drawn from them.

Table 1. Literature themes and design implications

Theme	Representative sources	Design implication in Nexel AI
Retrieval-augmented generation	Lewis et al. [1]; Karpukhin et al. [2]	Embed document chunks, retrieve by similarity, and generate answers from retrieved context with citations.
Foundation language models	Vaswani et al. [14]; Devlin et al. [16]; Brown et al. [12]; Wei et al. [11]	Use instruction-following models through prompts instead of training new models.
Retrieval practice and spacing	Roediger and Karpicke [4]; Cepeda et al. [3]	Generate quizzes and flashcards from the learner's documents; plan adaptive spacing as future work.
Collaborative learning	Dillenbourg [5]	Provide real-time study rooms connected to the shared learning platform.
Multilingual evaluation	Hu et al. (XTREME) [6]	Make language an explicit input; describe Hindi and Hinglish support cautiously.
Speech interfaces	W3C Web Speech API [13]	Offer browser-based voice input and spoken output.
Multimodal models	Caffagni et al. [7]; Zhang et al. [8]; Yin et al. [9]; Bai et al. [10]	Delegate image understanding to a vision-capable model and combine it with document context.
Risks of language models	Bender et al. [15]	Ground answers in sources, show citations, and avoid presenting output as authoritative.

IV. METHODOLOGY

1. Research And System Design

The project follows an iterative engineering approach in which a working system was built incrementally, deployed early, and refined as practical constraints appeared. Design decisions were guided by the literature themes in Table 1 and by the requirement that the system run entirely on managed, internet-hosted services. The architecture has four layers. The presentation layer is a Next.js 16 application using the App Router, TypeScript, Tailwind CSS, and Framer Motion. The application layer consists of Next.js API Routes that run on Vercel and orchestrate every request. The data layer is Supabase, which provides authentication, PostgreSQL storage, file storage, and realtime channels. The AI services layer comprises Groq for language generation, Jina AI for embeddings, and Google Gemini for image analysis. API credentials for these services are held on the server side and are never exposed to the browser.

2. Component Responsibilities

Each component has a narrow responsibility so that individual parts can be replaced without redesigning the whole system. Table 2 lists the principal components.

Table 2. Component responsibilities

Component	Responsibility
Next.js front end	Renders the workspace, dashboard, tutor, quiz, lecture, roadmap, predictor, and study-room interfaces; handles voice input and speech output in the browser.
Next.js API Routes	Validate requests, call embedding, retrieval, and generation services, parse model output, and return structured responses.
Supabase Auth	Authenticates users so that each account sees only its own documents.
Supabase Storage and PostgreSQL	Store uploaded PDFs, document metadata, text chunks, generated study material, and analytics data.
pgvector	Stores 768-dimensional embeddings and performs cosine-similarity search inside the database.
Supabase Realtime	Synchronises state between participants in collaborative study rooms.
unpdf	Extracts text from uploaded PDF files in the serverless runtime.

© 2026 IJSRET

Component	Responsibility
Jina AI embeddings	Converts chunks and queries into 768-dimensional vectors using jina-embeddings-v2-base-en.
Groq models	Generate answers, notes, flashcards, quizzes, lectures, roadmaps, and predictions with openai/gpt-oss-20b and openai/gpt-oss-120b.
Google Gemini Vision	Analyses images submitted to the AI Tutor.

3. End-To-End Data Flow

The path from an uploaded file to a cited answer consists of the following ten steps.

- The user signs in through Supabase Auth and opens the workspace.
- The user uploads a PDF through the interface, which accepts PDF files up to 50 MB; the file is saved to Supabase Storage and a document record is created.
- The server extracts the text of the document with unpdf.
- The extracted text is cleaned and divided into overlapping chunks.
- Each chunk is sent to the Jina AI embedding service, which returns a 768-dimensional vector.
- The chunks and their vectors are written to PostgreSQL, where pgvector manages the vector column.
- When the user asks a question or requests a study artifact, the request text is embedded with the same model.
- pgvector ranks stored chunks by cosine similarity to the request embedding and returns the most similar chunks.
- The server assembles a prompt that contains the instructions, the language preference, and the retrieved chunks with source identifiers, and sends it to a Groq-hosted model.
- The response is parsed and validated, displayed with its source citations, and optionally read aloud through speech output.

4. Pdf Extraction And Chunking

Text extraction is the entry point of the pipeline, so its quality limits everything downstream. The system uses unpdf, a library suited to serverless JavaScript runtimes, to extract text page by page. Extracted text is then normalised by removing redundant whitespace and obvious artefacts. A language model has a finite context window and an embedding model represents a single vector per input, so the text cannot be embedded as one block. It is divided into chunks of limited length with some overlap between neighbouring chunks, so that a sentence or concept that straddles a boundary is not lost. Chunks keep a reference to their source document and position, which is what later allows a citation to point back to the original material. Chunk length and overlap are configuration parameters of the pipeline; the values used in the deployed build are recorded in Appendix B as reproducibility details.

Chunk size involves a trade-off. Very short chunks are precise but may lack the surrounding context needed to answer a question, whereas very long chunks carry more context but dilute the signal of any single topic in their embedding. Overlap reduces boundary losses at the cost of storing and embedding some text more than once.

5. Embeddings And Semantic Representation

Embeddings map text to points in a vector space in which semantically similar passages lie close together. Nexel AI uses the jina-embeddings-v2-base-en model from Jina AI, which returns 768-dimensional vectors. The approach is in the spirit of dense retrieval [2], in which questions and passages are encoded independently and compared by a similarity function. Calling a hosted embedding service, instead of running a model inside the application, keeps the serverless functions lightweight, as discussed in Section 4.2.

The model name indicates an English-oriented embedding model. Retrieval quality for queries or documents written in other languages is therefore expected to be weaker than for English, and the multilingual behaviour of the system is concentrated mainly in the generation stage. This is a limitation of the current design and is discussed further in Sections 5.4 and 6.2.

6. Vector Similarity

Given a query embedding q and a chunk embedding d, similarity is measured by the cosine of the angle between the two vectors:

$$\cos(q,d) = \frac{q \cdot d}{\|q\| \cdot \|d\|} \quad (1)$$

where the numerator $q \cdot d$ is the dot product of the two vectors and the denominator is the product of their Euclidean norms. The value lies between −1 and 1, and larger values indicate that the vectors point in more similar directions. Because cosine similarity depends on direction rather than magnitude, it is insensitive to

differences in vector length, which suits embeddings of passages with different lengths. In PostgreSQL with pgvector, cosine distance is available as an operator and equals one minus the cosine similarity, so ordering rows by

© 2026 IJSRET

ascending cosine distance is equivalent to ordering them by descending cosine similarity. The pipeline retrieves the highest-ranked chunks, and the number retrieved is a configuration parameter. Because embeddings are stored next to the relational data, retrieval can be restricted to the documents of the signed-in user within the same database query.

7. Context Construction And Generation

Retrieved chunks are assembled into a context block in which each chunk is labelled with its source identifier. The prompt instructs the model to answer from the supplied context, to cite the sources it used, and to state clearly when the context does not contain enough information instead of inventing an answer. Grounding of this kind follows the motivation of RAG [1] and responds to the concern that fluent model output may be unsupported by evidence [15].

Two Groq-hosted models are available to the generation layer. The smaller openai/gpt-oss-20b model is suited to short, latency-sensitive responses, and the larger openai/gpt-oss-120b model to longer or more demanding tasks. The same retrieval and prompt-construction steps are reused across features. For question answering the request is the user's question. For notes, flashcards, quizzes, lectures, roadmaps, and exam prediction, the prompt specifies the required artifact and its format, and the selected document content supplies the material.

8. Multilingual Prompt Design And Structured Output

Language handling is implemented at the prompt level. The interface exposes a language selector, whose default is automatic detection, and the chosen language is included in the instructions given to the model. English, Hindi, and Hinglish are the supported modes. Hinglish output is requested as Hindi written in Roman script with English technical terms retained, which matches how many students discuss technical subjects. Instruction-following models can adapt to such directions from the prompt alone [11], [12], although the quality of the result in each language is not guaranteed.

Features such as flashcards and quizzes must return data that the interface can render, so the model is instructed to answer in a structured JSON format with defined fields, for example a question, a set of options, the correct answer, and an explanation. Structured output is parsed on the server and checked before it reaches the interface. Multilingual text creates particular parsing hazards, which are described in Section 4.4.

9. Voice And Image Interaction

Voice input and speech output rely on the speech recognition and speech synthesis capabilities of the browser, as standardised in the Web Speech API specification [13]. Spoken questions are converted to text before entering the same pipeline as typed questions, and generated answers and lecture narration can be read aloud. Image interaction is handled by Google Gemini Vision. When a user attaches an image in the AI Tutor, the image is sent to the vision model, and the resulting description or analysis is combined with the user's question and, where relevant, retrieved document context. This reflects the division of labour suggested by the multimodal literature [7]–[10]. Figure 1 shows the AI Tutor interface, which exposes file attachment, a language selector, a microphone control, and a text input in a single composer.



Figure 1. The AI Tutor interface with file attachment, language selector (Auto), voice input, and text entry, together with suggested prompts for explaining, summarising, and studying.

10. Real-Time Collaboration

Collaborative study rooms are built on Supabase Realtime. Participants who join a room subscribe to a shared channel, and updates made by one participant are broadcast to the others without a page refresh. The design follows the view of collaborative learning as an interactive, shared activity [5] by keeping the room inside the same platform as the learner's documents. Access to rooms is mediated by authenticated accounts. The present implementation focuses on shared real-time interaction; richer facilitation, moderation, and learning analytics for groups are left for future work.

V. IMPLEMENTATION

1. Technology Stack

Table 3 summarises the technologies used and their roles. All services are accessed over the network, so the application itself remains a thin orchestration layer.

© 2026 IJSRET

Table 3. Technology stack

Layer	Technology	Role
Front end	Next.js 16 (App Router), TypeScript, Tailwind CSS, Framer Motion	User interface, routing, styling, and animation
Back end	Next.js API Routes	Request handling and orchestration of AI services
Platform services	Supabase (Auth, PostgreSQL, Storage, Realtime)	Accounts, data, files, and live synchronisation
Language models	Groq: openai/gpt-oss-20b, openai/gpt-oss-120b	Answer, notes, quiz, lecture, roadmap, and prediction generation
Embeddings	Jina AI: jina-embeddings-v2-base-en (768 dimensions)	Semantic representation of chunks and queries
Vector search	Supabase pgvector, cosine similarity	Nearest-neighbour retrieval of chunks
Vision	Google Gemini 3.6 Flash	Image analysis in the AI Tutor
PDF processing	unpdf	Text extraction from PDFs
Deployment	Vercel	Hosting and serverless execution

2. Challenge 1: Embedding Execution On Vercel

A first design choice was whether to generate embeddings inside the application, with a locally executed model, or to call an external service. Serverless functions on Vercel have limits on memory, package size, execution time, and cold-start behaviour, which make it impractical to bundle and run a transformer-based embedding model inside each function. The system therefore calls the Jina AI embedding service over HTTP. This keeps functions small and start-up times short, and it moves the computational cost of embedding to a specialised provider. The trade-offs are dependence on an external service, network latency on every embedding request, and the need to batch chunk embeddings so that a long document does not exceed request limits.

3. Challenge 2: Generation Rate Limits

Hosted language-model inference is subject to rate limits on requests and tokens, and heavy features such as the lecture generator make many calls for a single document. The system responds in several ways. Tasks are assigned to either the smaller or the larger Groq model according to their demands. Long outputs are divided into stages, for example chapter by chapter in a lecture, so that no single request is excessively large. Requests that are rejected for rate-limit reasons are retried after a delay, and the interface reports failures clearly instead of failing silently. These mitigations reduce but do not remove the dependence of the system on provider quotas, which is recorded as a limitation in Section 6.2.

4. Challenge 3: Multilingual Json Parsing

Structured output is convenient for rendering but fragile, and the fragility is greater for Hindi and Hinglish. Devanagari characters, mixed-script text, quotation marks, and line breaks inside generated strings can produce output that is not valid JSON, for example through unescaped quotes or truncated structures. The server therefore treats model output as untrusted input. It removes surrounding formatting such as codefences, extracts the JSON portion, parses it defensively, validates the expected fields, and where necessary repairs or regenerates the response. This defensive parsing is an essential part of the multilingual pipeline, not an optional refinement.

5. Challenge 4: Pdf Library Compatibility

Several popular Node.js PDF libraries depend on native modules, on file-system assumptions, or on browser-specific objects, and these assumptions can fail in a serverless runtime. The choice of unpdf resolved this problem because it is designed to work in serverless and edge-style JavaScript environments. Even so, text extraction from PDFs is imperfect: scanned documents without a text layer, complex multi-column layouts, tables, and mathematical notation may not be extracted faithfully, and these cases are listed among the limitations.

6. Pwa And Responsive Delivery

Nexel AI is delivered as a Progressive Web App. A web application manifest and service worker allow the application to be installed on supported devices and to open in its own window, as in the screenshots in this paper, which show the installed application without browser controls. The interface is built with responsive layouts in Tailwind CSS so that it can be used on desktop and mobile screens. Delivery as a PWA avoids the cost of maintaining separate native applications, although it provides fewer device integrations than a native client would, as noted in Section 6.3.

7. Deployment And External-Service Integration

The application is deployed on Vercel at https://nexel-ai-green.vercel.app, with API Routes executing as serverless functions. Credentials for Supabase, Groq, Jina AI, and Google Gemini are supplied through environment variables that are available only on the server. Integration with several external services means that the availability and behaviour of the whole system depend on each of them. Errors from any

© 2026 IJSRET

provider are caught at the API layer and translated into messages the user can act on.

8. Feature Integration

The principal engineering effort lies in integrating many features around a shared core. The workspace shown in Figure 2 is the entry point. It presents the document library, the upload control, and navigation to the Tutor, Roadmap, Rooms, Lecture, Predictor, Search, Stats, Notes, Study, Quiz, and Video areas, together with a guided tour. Behind these entries, features share the same document records, chunks, embeddings, and generation utilities, so that adding a feature mostly requires a new prompt, an output schema, and an interface, not a new data pipeline.

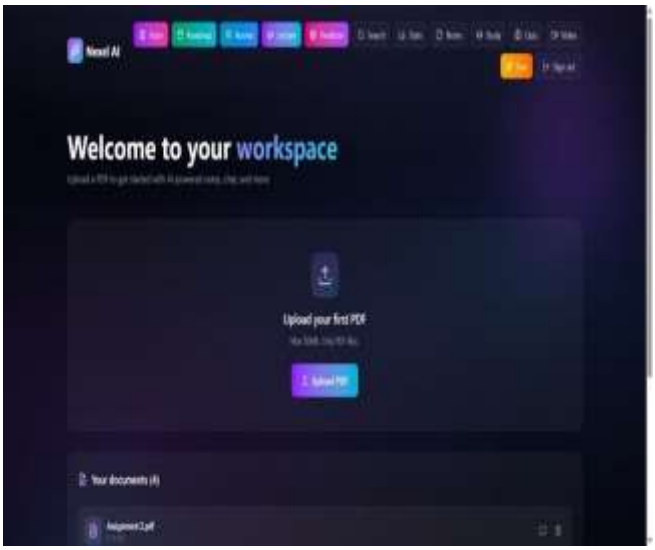


Figure 2. The Nexel AI workspace, showing the PDF upload area (maximum 50 MB, PDF files only), navigation to the main features, and the user's document library.

VI. RESULTS AND DISCUSSION

1. Evaluation Approach

The evaluation reported here is a functional evaluation of the deployed system, based on developer observation. The system was exercised through its web interface using the developer's own course-related PDFs, which included a programming assignment, a web-technology examination guide, and a training report, as visible in the screenshots. For each feature the question asked was whether the feature runs end to end in the deployed build and produces output of the expected type. No controlled experiment was run, no benchmark dataset was used, and no learning outcomes were measured. The results should therefore be read as evidence that the system works as designed, not as evidence of educational effectiveness or of comparative accuracy.

2. Functional Coverage

Table 4 lists the capabilities of the system, the interface through which each is reached, and the status observed. Figures 3 and 4 show two of the study-oriented interfaces. In the quiz generator, the user selects one of the uploaded PDFs and the system generates a set of five questions to test understanding. In the exam predictor, the user selects between two and eight uploaded documents, such as past papers, and the system analyses them to suggest the topics and questions that are most likely to matter; the interface guidance recommends three to five past papers.

Table 4. Evaluation matrix (developer-observed functional status)

Capability	Interface entry	Observed status	Remarks
PDF upload and processing	Workspace upload control	Functional	PDF files only; 50 MB limit shown in the interface.
Document Q&A with citations	Tutor, Search	Functional	Answers drawn from retrieved chunks with source references.
Notes generation	Notes	Functional	Generated from document context.
Flashcards	Study	Functional	Structured output rendered as cards.
Quiz generation	Quiz	Functional	Per-document quiz of five questions.
Full lecture generator	Lecture	Functional	Chaptered lecture with voice output; long jobs are staged.
Study roadmap	Roadmap	Functional	Plan generated from document content.
Exam predictor	Predictor	Functional	Analyses two to eight selected documents.

© 2026 IJSRET

Capability	Interface entry	Observed status	Remarks
Multilingual output	Language selector	Functional	English, Hindi, Hinglish; quality not measured.
Multimodal AI Tutor	Tutor	Functional	Text, voice, images, and PDFs accepted.
Voice input and speech output	Tutor, Lecture	Functional	Depends on browser speech support.
Image analysis	Tutor	Functional	Delegated to Google Gemini Vision.
Collaborative study rooms	Rooms	Functional	Real-time synchronisation through Supabase Realtime.
Analytics dashboard	Stats	Functional	Usage and study statistics.
Installable PWA	Browser install prompt	Functional	Runs in a standalone window once installed.



Figure 3. The quiz generator, in which the user selects an uploaded PDF and the system generates five questions on its content.



Figure 4. The exam predictor, in which the user selects uploaded exam papers for pattern analysis before generating a prediction.

3. Processing Time And Retrieval Latency

No timing measurements are reported in this paper, and no numeric performance claims are made. A qualitative account of where time is spent is nevertheless useful. Document processing consists of text extraction, chunking, embedding, and storage. Extraction and chunking run inside the serverless function, whereas embedding depends on calls to the Jina AI service, so processing time is expected to grow with document length and with network conditions. At query time, the latency has three components: embedding the query, running the similarity search in PostgreSQL, and generating the answer with a Groq-hosted model. The generation step is expected to dominate for long outputs, and the similarity search is expected to be comparatively inexpensive for per-user document collections. These expectations are hypotheses; Section

proposes a protocol to measure them properly.

4. Multilingual Behaviour

Developer observation of the deployed system indicates that the language selector changes the language of generated responses in ordinary use, and the AI Tutor's default setting detects the language automatically. These observations are informal. No systematic comparison of answer quality across English, Hindi, and Hinglish has been carried out. Given the uneven cross-lingual performance reported in the literature [6], quality should not be assumed to be equal across languages. The English-oriented embedding model also means that retrieval for non-English queries may be less reliable than for English queries, which would affect answers before generation begins. A possible mitigation, not implemented in the present system, is to translate or rewrite non-English queries into English before embedding, or to adopt a multilingual embedding model.

© 2026 IJSRET

5. Comparison With Existing Tools

Table 5 compares Nexel AI with general categories of existing tools. The comparison is qualitative and describes typical characteristics of each category, which vary between products and change over time. It is not a benchmark of any specific product, and no claim is made that Nexel AI outperforms them. The purpose is to show where the integrated design differs in emphasis.

Table 5. Qualitative comparison with categories of existing tools

Capability	General chat assistants	Document-chat tools	Flashcard and quiz platforms	Nexel AI
Grounding in the user's PDFs	Possible when text is pasted or attached	Core feature	Usually manual entry or import	Core feature through RAG
Source citations	Varies	Often provided	Not applicable	Provided with answers
Notes, flashcards, and quizzes from one document	On request, without a persistent study workflow	Varies	Core feature, often user-authored	Integrated in one workspace
Long narrated lectures	Not typical	Not typical	Not typical	Included
Hindi and Hinglish prompts	Often possible	Varies	Varies	Explicit language selector
Real-time study rooms	Not typical	Not typical	Sometimes	Included
Exam-topic prediction from past papers	On request	Not typical	Not typical	Dedicated feature

1. User Experience And Learning Implications

The user-experience aim of the system is to reduce the friction between having a document and studying from it. A single upload leads to many study modes, and the same navigation exposes all of them, as the workspace in Figure 2 illustrates. From a learning perspective, the design is consistent with research on retrieval practice [4] and distributed practice [3], because it makes quiz and flashcard creation cheap. It also supports collaboration [5] and spoken interaction [13]. These are design implications drawn from the literature. They are not findings about the effect of Nexel AI on learning, which has not been tested.

2. Reliability, Safety, And Data Governance

Generated content can be wrong, incomplete, or biased [15]. Grounding answers in retrieved context and showing citations lets the learner check claims against the source, but it does not guarantee correctness, and the model may still misinterpret a passage. The exam predictor in particular produces suggestions based on patterns in the supplied documents and should be treated as a planning aid, not as a forecast. Students should verify important material against their textbooks and instructors.

Uploaded documents may contain personal or institutional information. Documents are tied to authenticated accounts, credentials for external services are kept on the server, and file and database access is handled through Supabase. Text from documents and user prompts is transmitted to third-party AI providers for embedding and generation, so users should avoid uploading sensitive material, and a production deployment would need a clear privacy policy, retention rules, and an account-level deletion mechanism.

3. Evaluation Framework Proposed

A rigorous evaluation of the system should combine technical and educational measures. A proposed framework has four parts.

- Retrieval quality. Build a set of questions with known source passages from several documents, and measure

© 2026 IJSRET

recall at k and mean reciprocal rank for English, Hindi, and Hinglish queries, which would also quantify the effect of the English-oriented embedding model.

- Answer faithfulness. Have independent raters judge whether answers are supported by the cited chunks, and measure citation precision and the rate of unsupported statements, using documents the raters know well.
- Latency and reliability. Record processing time per page, retrieval time, and generation time across document sizes, and report medians, high percentiles, and error rates, including rate-limit failures.
- Learning and usability. Run a user study with a control group, comparing pre-test and post-test scores and delayed retention, and collect usability ratings with a standard instrument and qualitative feedback.

4 Threats To Validity

- **Internal validity.** The evaluation was performed by the developer, who knows how the system is meant to be used and may unintentionally avoid failure cases. Language-model output is non-deterministic, so a single successful run does not show that a feature is reliable.
- **Construct validity.** Functional status shows that a feature runs, not that its output is accurate or pedagogically useful. The comparison in Table 5 uses broad categories, and individual products may differ from the characterisations given.
- **External validity.** The test documents come from one student's coursework in a small number of subjects, and the results may not generalise to other disciplines, to scanned or image-heavy PDFs, or to other languages and learners.
- **Conclusion validity.** Because no quantitative measurements are reported, the paper draws no statistical conclusions. External services and models may also change their behaviour, pricing, or limits, which affects repeatability.

VII. CONCLUSION AND FUTURE WORK

1. Conclusion

This paper presented Nexel AI, a deployed multimodal RAG system that turns PDFs into interactive study materials. By indexing a document once and reusing the resulting embeddings, the system supports cited question answering, notes, flashcards, quizzes, narrated lectures, study roadmaps, and exam-topic prediction. It adds a multimodal AI Tutor, English, Hindi, and Hinglish prompting, voice interaction, image analysis, and real-time collaborative study rooms, and it is delivered as an installable Progressive Web App on Vercel. The paper documented the architecture, the main engineering challenges, and a functional evaluation based on developer observation. The work demonstrates that a managed-service stack can support a broad, integrated study environment, while leaving the questions of accuracy, multilingual quality, and learning impact for systematic study.

2. Limitations

- No quantitative evaluation. The paper reports functional status only; retrieval accuracy, answer faithfulness, latency, and learning outcomes have not been measured.
- English-oriented retrieval. The embedding model is English-oriented, so retrieval over Hindi or Hinglish queries and documents may be weaker than for English.
- Extraction quality. Scanned documents, complex layouts, tables, and mathematical notation may not be extracted faithfully, which affects every downstream feature.
- Model reliability. Generated content can be inaccurate or incomplete, and citations show where text came from without proving that the answer is correct [15].
- Dependence on external services. The system relies on Groq, Jina AI, Google Gemini, Supabase, and Vercel, and is exposed to their rate limits, outages, pricing changes, and policies.
- Single-document-centred retrieval. Cross-referencing several documents in one answer is limited in the current design.
- Browser and device variability. Speech recognition and synthesis depend on browser support, and the PWA offers fewer device integrations than native applications.

3. Future Work

- **Multi-Agent Architecture**
  Separate cooperating agents could handle retrieval, question generation, answer verification, and lesson planning. A verification agent that checks each answer against the cited chunks could reduce unsupported statements.
- **Native Mobile Applications**
  Native Android and iOS clients could provide better offline support, notifications, background audio for lectures, and more reliable speech services than a browser-based PWA.
- **Adaptive Spaced Repetition**
  The retrieval-practice [4] and spacing [3] literature suggests scheduling flashcards and quizzes according to each learner's past performance. A scheduler that tracks per-item recall and adjusts intervals would turn the current generation features into a long-term study system.
- **Multi-Pdf Cross-Referencing**
  Retrieval and prompts could be extended to combine evidence from several documents in one answer, reconcile differences between sources, and cite each,

© 2026 IJSRET

which would be useful for subjects with several textbooks and lecture series.

- **Formal User Study**
  A controlled study with students, using the framework in Section 5.8, would measure learning gains, retention, usability, and trust, and would compare languages and features. Ethical approval and informed consent would be required.
- **Accessibility And Responsible Ai**
  Future work includes screen-reader support, keyboard navigation, captions for narrated lectures, adjustable text and contrast, clear disclosure of AI-generated content, privacy controls, data deletion, and bias and error reporting, in line with the concerns raised in [15].
- **Final Remarks**
  Nexel AI shows that combining retrieval, generation, speech, vision, and real-time collaboration around a student's own documents is feasible with current managed services and a modest engineering effort. It is presented as a working prototype whose usefulness for learning still has to be established through careful evaluation. The author hopes that the architecture and the engineering lessons described here will be useful to other students building document-grounded learning tools.

REFERENCES

1. P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," NeurIPS, vol. 33, pp. 9459-9474, 2020.
2. V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," EMNLP, pp. 6769-6781, 2020.
3. N. J. Cepeda, H. Pashler, E. Vul, J. T. Wixted, and D. Rohrer, "Distributed Practice in Verbal Recall Tasks: A Review and Quantitative Synthesis," Psychological Bulletin, vol. 132, no. 3, pp. 354-380, 2006.
4. H. L. Roediger III and J. D. Karpicke, "Test-Enhanced Learning: Taking Memory Tests Improves Long-Term Retention," Psychological Science, vol. 17, no. 3, pp. 249-255, 2006.
5. P. Dillenbourg, Ed., Collaborative Learning: Cognitive and Computational Approaches. Oxford: Elsevier, 1999.
6. J. Hu et al., "XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalization," ICML, PMLR, vol. 119, pp. 4411-4421, 2020.
7. D. Caffagni et al., "The Revolution of Multimodal Large Language Models: A Survey," Findings of ACL 2024, pp. 13590-13618, 2024.
8. D. Zhang et al., "MM-LLMs: Recent Advances in Multi-Modal Large Language Models," Findings of ACL 2024, pp. 7735-7756, 2024.
9. S. Yin et al., "A Survey on Multimodal Large Language Models," National Science Review, vol. 11, no. 12, nwae403, 2024.
10. T. Bai et al., "A Survey of Multimodal Large Language Model from a Data-Centric Perspective," arXiv:2405.16640, 2024.
11. J. Wei et al., "Finetuned Language Models Are Zero-Shot Learners," ICLR, 2022.
12. T. Brown et al., "Language Models are Few-Shot Learners," NeurIPS, vol. 33, pp. 1877-1901, 2020.
13. W3C, "Web Speech API," W3C Community Group Draft, 2023.
14. A. Vaswani et al., "Attention Is All You Need," NeurIPS, vol. 30, 2017.
15. E. M. Bender et al., "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?," FAccT, pp. 610-623, 2021.
16. J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL-HLT, pp. 4171-4186, 2019.

© 2026 IJSRET