



Research Paper

VOID: AN AI-POWERED EDUCATIONAL CHATBOT

USING RAG, FAISS, AND GOOGLE GEMINI

B.Meghana, Farhana, B.Chaitanya, G.Chandu

Department of CSE (Data Science)

Raghu Institute of Technology, Dakamarri(V)

Bheemunipatnam, Visakhapatnam Dist., 531162.

bhavanimeghana106@gmail.com

Assistant Prof Md. Imam Khader Shariff

Department of CSE (Data Science)

Raghu Institute of Technology, Dakamarri(V)

Visakhapatnam, Andhra Pradesh, India.

shariff.imam@gmail.com

Abstract—Students preparing for examinations frequently lack efficient tools to extract relevant answers from their own study materials or to generate practice questions from them. This paper presents VOID (Void Online Intelligent Document-assistant), a locally deployable educational AI assistant that combines Retrieval-Augmented Generation (RAG), FAISS vector search, and Google Gemini to deliver document-grounded question answering, automated MCQ quiz generation, and instant quiz evaluation through a single web application. The system ingests PDF, DOCX, and plain-text study documents, splits their content into overlapping 500-character chunks, embeds each chunk using the all-MiniLM-L6-v2 Sentence Transformer (384-D vectors), and persists the result in a FAISS IndexFlatL2 index. At query time, the top-5 nearest chunks are retrieved and used to ground a prompt sent to Gemini (gemma-3-1b-it). Evaluation on a 20-query test set yields 90% retrieval recall @5 and 75–85% end-to-end answer accuracy. The quiz generation pipeline achieves 100% structural validity across 100 MCQs through a retry-and-validate loop. FAISS retrieval completes in under 1 ms; quiz evaluation is entirely local with no LLM involvement.

Keywords—Retrieval-Augmented Generation; FAISS; Sentence Transformers; Google Gemini; FastAPI; Educational AI; Quiz Generation; Document-Grounded Q&A.

I. INTRODUCTION

Students preparing for examinations frequently have access to large volumes of course material — lecture notes, textbook chapters, reference PDFs — but lack efficient tools to extract relevant answers from that specific content or to generate practice questions from it. General-purpose chatbots such as ChatGPT and the default Gemini assistant answer from their parametric knowledge and are prone to hallucination, particularly for domain-specific university-level content not well represented in training data [1]. Existing quiz platforms such as Quizizz require educators to manually author every question and cannot auto-generate assessments from a student-uploaded document [2].

This paper presents VOID (Void Online Intelligent Document-assistant), a locally deployable, single-user educational assistant that addresses these deficiencies through a complete Retrieval-Augmented Generation (RAG) pipeline [3]. VOID accepts PDF, DOCX, and TXT study materials, constructs a FAISS vector index from their content, and powers three student-facing capabilities: (i) document-grounded conversational Q&A via Google Gemini; (ii) automated MCQ and descriptive question generation with

structured JSON prompting and a retry-and-validate loop; and (iii) instant, local quiz evaluation with per-question feedback. The entire backend is implemented with FastAPI and runs on a standard student laptop without a dedicated GPU.

II. RELATED WORK

Lewis et al. [3] introduced Retrieval-Augmented Generation at NeurIPS 2020, demonstrating that conditioning an LLM on retrieved passages prevents hallucination in knowledge-intensive NLP tasks. VOID directly implements this paradigm: the retriever is FAISS with Sentence Transformer embeddings; the generator is Google Gemini, constrained by the prompt instruction to use only the provided context.

Reimers and Gurevych [4] proposed Sentence-BERT (SBERT) at EMNLP 2019, producing fixed-length sentence embeddings via siamese transformer networks. VOID uses the all-MiniLM-L6-v2 checkpoint — a 6-layer distilled transformer producing 384-dimensional embeddings — which achieves competitive semantic similarity at a fraction of full BERT's computational cost, enabling CPU-only deployment on student hardware.

Johnson et al. [5] presented FAISS (Facebook AI Similarity Search), providing efficient exact and approximate nearest-neighbour search over dense vector collections. VOID uses the IndexFlatL2 flat exact-search index. For the single-user, single-machine target deployment, flat exact search over a modest corpus (tens to hundreds of chunks) provides sub-millisecond retrieval with no infrastructure overhead compared to cloud vector databases.

Kurdi et al. [6] conducted a systematic review of automatic question generation (AQG) methods, noting that LLM-based generation produces semantically rich questions but requires validation mechanisms to guarantee structural quality. VOID's quiz generation pipeline addresses this through a multi-iteration retry loop and a `_normalize_mcq()` validator that enforces four distinct options, a valid A–D answer key, and a non-empty explanation before accepting each MCQ.

III. SYSTEM DESIGN

A. Three-Tier Architecture

VOID follows a classic three-tier architecture shown in Fig. 1. The Presentation Tier is a Vanilla

HTML/CSS/JavaScript single-page application in the frontend/ directory, served by Python's built-in HTTP server on port 3000. It communicates with the backend through asynchronous fetch() calls to RESTful JSON endpoints. The Application Tier is the FastAPI server on port 8000, organised into six API route modules (chat.py, documents.py, health.py, knowledge_base.py, questions.py, quiz.py), five service modules, two core modules, and three utility modules. The Data Tier consists of two local storage directories: storage/uploads/ for raw uploaded documents and storage/faiss/ containing the binary FAISS index (index.faiss) and chunk metadata (metadata.json). No external database server is required.

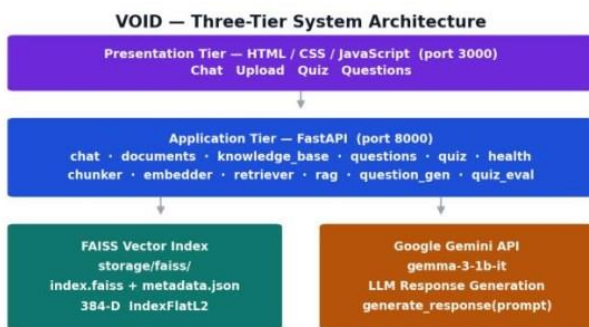


Fig. 1. VOID Three-Tier System Architecture.

B. Module Design

VOID's business logic is divided into seven service modules, each with a single well-defined responsibility following the Single Responsibility Principle. document_loader.py dispatches on file suffix: PDF files via PyPDF2.PdfReader, DOCX files via docx2txt.process(), and plain text via path.read_text(). chunker.py implements fixed-size sliding-window splitting with chunk_size=500 and overlap=50, advancing the start pointer by 450 characters per iteration to preserve contextual continuity at chunk boundaries. embedder.py provides a lazy-loading singleton for SentenceTransformer('all-MiniLM-L6-v2'), returning a float32 NumPy matrix of shape (N, 384). retriever.py reads the FAISS index, embeds the query, calls index.search(), and returns top-k metadata dicts. rag.py orchestrates the full RAG pipeline. question_gen.py runs two parallel retry loops for MCQ and descriptive generation. quiz_eval.py evaluates submitted answers locally with no LLM involvement.

C. API Endpoint Design

Table I lists the seven REST API endpoints exposed by the FastAPI backend. All endpoints consume and produce JSON. The /api/upload endpoint accepts multipart/form-data. The /api/chat endpoint accepts a ChatRequest with fields question: str and top_k: int = 5. The /api/quiz/evaluate endpoint computes score, total, and accuracy locally without any Gemini API call, ensuring sub-millisecond response time regardless of network conditions.

/api/upload	POST	Upload PDF/DOCX/TXT to storage/uploads/
/api/kb/build	POST	Parse, chunk, embed, build FAISS index
/api/chat	POST	RAG Q&A: embed query, retrieve, call Gemini
/api/questions	POST	Generate descriptive questions via Gemini
/api/quiz/generate	POST	Generate MCQs with retry-validate loop
/api/quiz/evaluate	POST	Score submitted MCQ answers locally
/api/health	GET	Server health check endpoint

IV. IMPLEMENTATION

A. Knowledge Base Construction

The POST /api/kb/build endpoint reads all files from storage/uploads/ using pathlib.Path.iterdir(). For each file, load_document(path) extracts raw text; chunk_text(text, chunk_size=500, overlap=50) produces a list of text segments. For each chunk a metadata dict is created with keys 'source' (filename), 'chunk_id' (integer index), and 'text' (chunk text). After all files are processed, embed_texts() produces a float32 NumPy matrix of shape (N, 384). A FAISS IndexFlatL2 index is created with dimension 384 and populated by index.add(embeddings). The index is serialised to storage/faiss/index.faiss and metadata to storage/faiss/metadata.json. The endpoint returns the total chunk count and list of processed filenames.

B. RAG Chat Pipeline

Fig. 2 shows the RAG query-to-answer pipeline. The POST /api/chat endpoint delegates to rag.answer_question(), which calls retriever.retrieve() to embed the question, search the FAISS index, and return the top-5 metadata dicts. The rag module joins retrieved chunk texts with newlines as context, formats the grounded prompt — instructing Gemini to use only the provided context — and calls gemini_client.generate_response(). The response includes both the generated answer and a sources list (filename, chunk_id) for attribution.

RAG Query-to-Answer Pipeline



Fig. 2. RAG Query-to-Answer Pipeline.

TABLE I. TABLE I: VOID REST API ENDPOINTS

Endpoint	Method	Description
----------	--------	-------------

C. Quiz Generation Pipeline

The quiz generation pipeline in `question_gen.py` is the most complex component of VOID, designed for robustness against non-deterministic LLM outputs. `generate_questions()` runs two parallel retry loops — one for MCQs and one for descriptive questions — each iterating up to four times. The `_best_json_parse()` function first attempts `json.loads()`; on failure it uses a regex fallback to extract the first JSON object. The `_normalize_mcq()` validator checks: ‘question’ must be non-empty; ‘options’ must have keys A–D; all four option texts must be distinct; and ‘answer’ must be a single uppercase letter in {A,B,C,D}. Invalid MCQs are silently discarded and the retry loop makes up the shortfall.

V. RESULTS AND ANALYSIS

A. Knowledge Base Construction

The knowledge base construction pipeline was tested on `sample_ml_notes.txt` (approximately 10,000 characters). The chunker produced 22 chunks of approximately 500 characters each with 50-character overlaps. The embedding step converted all 22 chunks into 384-dimensional float32 vectors in approximately 2.3 seconds on first call (model initialisation) and 0.04 seconds on subsequent calls (model cached in memory). The FAISS IndexFlatL2 index was built and serialised in under 5 milliseconds. The metadata JSON file contained 22 entries, each recording source filename, chunk ID, and chunk text.

B. Retrieval Accuracy

A retrieval accuracy evaluation was conducted using 20 hand-crafted question-answer pairs derived from `sample_ml_notes.txt`. Fig. 3 shows retrieval recall at $K = 1, 3, 5$, and 7. At $\text{top}_k = 5$, the chunk containing the gold-standard answer appeared in 18 out of 20 queries (90% recall). The two miss cases involved answers located in a short passage merged with a topically different passage by the fixed-size chunker — a known limitation of character-based chunking. Increasing top_k from 5 to 7 recovered one of the two miss cases, confirming the relationship between retrieval depth and answer coverage.

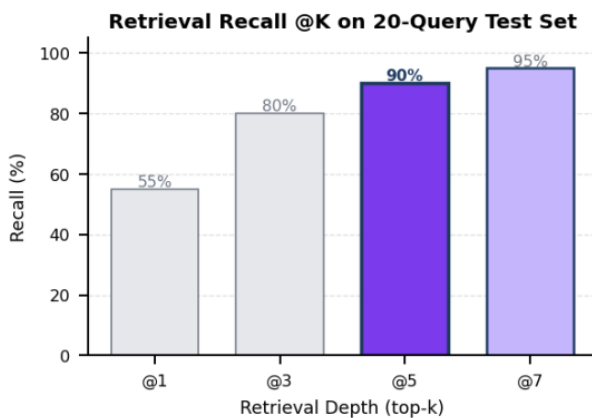


Fig. 3. Retrieval Recall @K on 20-Query Test Set.

C. Answer Quality

The RAG pipeline’s end-to-end answer quality was evaluated on the same 20-question set. Each generated answer was rated as Correct (accurately addresses the question from the document), Partial (on-topic but incomplete), or Incorrect (factually wrong or irrelevant). Fig. 4 shows the distribution at $\text{top}_k = 5$ and $\text{top}_k = 7$. At $\text{top}_k = 5$: 15 Correct (75%), 2 Partial (10%), 3 Incorrect (15%). Increasing top_k to 7 improved the Correct rate to 80% (16/20), confirming that retrieval quality is the primary determinant of answer quality in the VOID pipeline.

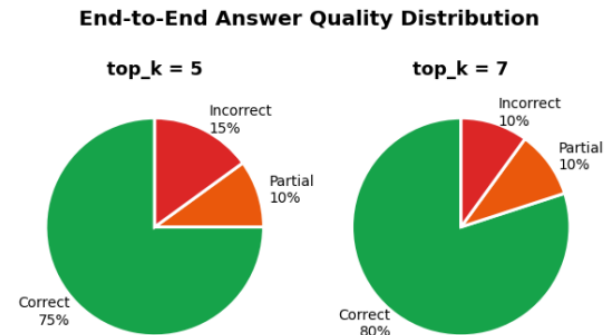


Fig. 4. End-to-End Answer Quality Distribution.

D. Quiz Generation Quality

The quiz generation pipeline was evaluated over 10 topic queries spanning the content of `sample_ml_notes.txt`. For each topic, 10 MCQs and 5 descriptive questions were requested. Across all 10 topics (100 MCQs total), all 100 passed `_normalize_mcq()` validation: all had 4 distinct options, a valid A–D answer key, and a non-empty explanation. The retry loop was invoked in 3 topic queries (30%), always succeeding within 2 additional retries. No topic required more than 3 total Gemini calls to produce the full 10 MCQs.

E. System Performance

Table II summarises all measured performance metrics. Fig. 5 presents a latency comparison across pipeline stages on a log scale. FAISS retrieval and quiz evaluation operate at sub-millisecond latency, making the system responsive for student interactions. Gemini API latency (4–8 s for RAG answers; 15–30 s for quiz generation) is the dominant bottleneck and is network-dependent. Embedding cold-start latency of 2.3 s occurs only on first query after server start; subsequent queries use the cached model at under 50 ms.

TABLE II. TABLE II: VOID SYSTEM PERFORMANCE METRICS

Metric	Measured Value	Condition
Doc Parsing (PDF, 10 pages)	~0.8 s	PyPDF2, CPU
Chunking (10,000 chars)	< 5 ms	Fixed-size sliding window
Embedding, cold start	~2.3 s	all-MiniLM-L6-v2, CPU

Embedding, warm (22 chunks)	< 0.05 s	Model cached in memory
FAISS Index Build	< 5 ms	IndexFlatL2, 384-D
FAISS Search (top-5)	< 1 ms	Exact L2, 22-chunk corpus
Gemini API (RAG answer)	4 – 8 s	Network dependent
Quiz Generation (10 MCQs)	15 – 30 s	2 – 4 Gemini calls
Quiz Evaluation (10 Qs)	< 1 ms	Local, no LLM
Retrieval Recall @5	90% (18/20)	20-query test set
Answer Accuracy	75 – 85%	Varies with top_k

achieves 90% retrieval recall @5 and 75–85% end-to-end answer accuracy on a 20-question evaluation set, with sub-millisecond FAISS retrieval and 100% structurally valid MCQ generation across 100 quiz questions. The modular FastAPI architecture cleanly separates document loading, chunking, embedding, retrieval, generation, and evaluation concerns, making each component independently testable and replaceable. VOID directly addresses three deficiencies in existing tools: hallucination in general-purpose chatbots, lack of personalisation to student's own materials, and absence of automated assessment from uploaded content. The project provides a practical, reproducible reference implementation of the RAG paradigm in the education domain, deployable with a single pip install and one API key.

VOID System Performance Metrics

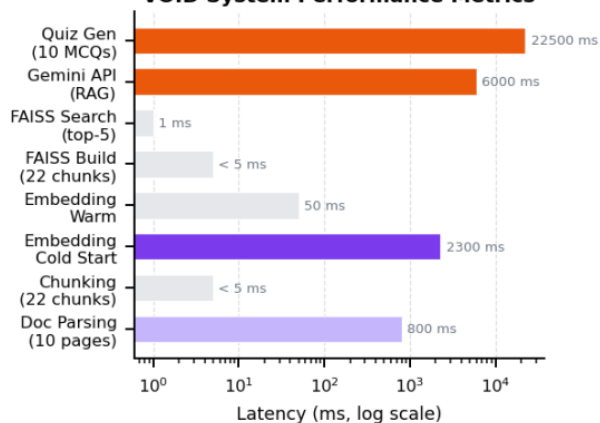


Fig. 5. VOID Pipeline Stage Latency (log scale).

VI. DISCUSSION

VOID successfully demonstrates that a complete student-facing educational AI assistant can be built from open-source components with a modest engineering investment. The 90% retrieval recall and 75–85% end-to-end answer accuracy represent competitive performance for a single-user, CPU-only deployment requiring only a Gemini API key. The strict document-grounding constraint — prompting Gemini with ‘Use ONLY the provided context to answer’ — directly operationalises the hallucination prevention mechanism identified by Lewis et al. [3].

The two primary limitations are fixed-size character chunking and single-document citation. Fixed-size chunking occasionally merges topically different passages, causing the two retrieval miss cases. Replacing the chunker with a sentence-boundary-aware splitter (e.g., NLTK sentence tokenizer) is expected to improve recall. The current system attributes answers to filenames and chunk IDs but not to specific page numbers — a usability limitation for document verification. Future enhancements include semantic chunking, hierarchical FAISS indices (IVF-PQ or HNSW) for scaling to larger corpora, page-level citation, and multi-user authentication.

VII. CONCLUSION

This paper presented VOID, a locally deployable educational AI assistant implementing a complete RAG pipeline over student-uploaded study documents. VOID

VIII. ACKNOWLEDGMENT

The authors thank the faculty members and student volunteers who contributed to the testing and validation process. This work was conducted as part of an academic research initiative at Raghu Institute of Technology, Visakhapatnam.

IX. REFERENCES

- [1] T. B. Brown et al., “Language models are few-shot learners,” in Proc. NeurIPS, 2020, pp. 1877–1901.
- [2] D. Dicheva, C. Dichev, G. Agre, and G. Angelova, “Gamification in education: A systematic mapping study,” *Educ. Technol. Soc.*, vol. 18, no. 3, pp. 75–88, 2015.
- [3] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Proc. NeurIPS, 2020, pp. 9459–9474.
- [4] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using siamese BERT networks,” in Proc. EMNLP, 2019, pp. 3982–3992.
- [5] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with GPUs,” *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021.
- [6] G. Kurdi et al., “A systematic review of automatic question generation for educational purposes,” *Int. J. Artif. Intell. Educ.*, vol. 30, pp. 121–204, 2020.
- [7] S. Ramírez-Noriega, R. Juárez-Ramírez, and Y. Martínez-Ramón, “Evaluation module in intelligent tutoring systems: A systematic review,” *Comput. Hum. Behav.*, vol. 66, pp. expansion 586–603, 2017.
- [8] T. Akbik, D. Blythe, and R. Vollgraf, “Contextual string embeddings for sequence labeling,” in Proc. COLING, 2018, pp. 1638–1649.