International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 05 | May – 2026 DOI:10.55041/ISJEM07054
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Exam Paper Generator using AI

Dr. Vijay Dhaku Dhangar
T. Chenna Keshava
D. Karthik
M. Prashanth,
Y. Venu Kumar
Sandip University, India

Abstract
Manually constructing examination question papers is repetitive, time-consuming, and vulnerable to uneven topic coverage. This paper presents an AI-powered question paper generator that combines a local large language model with retrieval-augmented generation (RAG) to automate the creation of exam papers from textbook PDFs. The system accepts a textbook PDF, extracts and cleans the text, splits it into semantic chunks, and stores embeddings in a vector database. For each requested topic or exam pattern, the system retrieves the most relevant chunks and supplies them to a quantized local language model for question synthesis. Three generation modes are supported: a standard exam mode with fixed part-wise marks, a custom-topic mode where the user specifies the number of questions and marks, and a question-bank mode without marks. The complete pipeline runs offline on a standard CPU-based workstation, protecting institutional content and avoiding dependence on cloud APIs. A Flask-based interface provides authentication, paper history, and export to TXT, PDF, and DOCX formats. Prototype evaluation reported an average generation time of roughly 105 seconds for a 19-question paper, with high relevance and stable parsing for shorter requests. The results indicate that a local LLM, when combined with RAG and careful prompt design, can produce coherent, syllabus-grounded question papers while reducing faculty workload and preserving privacy. The paper concludes with limitations, including OCR dependence for scanned PDFs and occasional count mismatches, and outlines future work for multimodal inputs and learning management system integration.
Keywords: Question Paper Generator, Local LLM, Retrieval-Augmented Generation, RAG, ChromaDB, Flask Web Application, Educational Technology, Natural Language Processing.

Introduction

Written examinations remain one of the most common mechanisms for assessing student learning, yet the preparation of a balanced question paper is still largely a manual task. Faculty members must verify syllabus coverage, control difficulty level, distribute marks correctly, and avoid repetition across questions. In practice, this work consumes several hours per paper, especially when the assessment spans multiple chapters or needs to match a specific university pattern. The administrative burden becomes more serious when departments handle multiple sections, repeated internal tests, and continuous assessments.

Recent progress in natural language processing has created a realistic opportunity to automate parts of this workflow. Large language models can generate fluent questions, but standard cloud-based systems introduce serious concerns. Sending textbook content to external servers may expose unpublished teaching material, while context limits may prevent the model from reading an entire textbook in one prompt. Even when a model is capable of producing fluent output, it may also hallucinate details not present in the source material, which is unacceptable in an examination setting.

This paper therefore focuses on a local question paper generator based on retrieval-augmented generation. The key idea is simple: instead of expecting the language model to remember everything, the system first retrieves relevant textbook passages and then asks the model to generate questions grounded in those passages. The approach combines the precision of document retrieval with the flexibility of generative modeling. Because both retrieval and generation can run locally, the system keeps the entire workflow offline and under institutional control.

The contribution of this work is not the invention of a new foundation model, but the integration of existing components into a usable educational tool. A full application is built around textbook upload, text extraction, chunking, embedding, semantic search, local inference, structured prompt templates, output parsing, and downloadable paper formats. This makes the system more than a prototype demonstration. It is designed as a practical assistant for educators who need repeatable, privacy-preserving, and syllabus-based exam paper generation.

Related Work

Automatic question generation has been studied for many years, beginning with rule-based systems that transform declarative sentences into interrogative forms. These early approaches were useful for limited recall questions, but they lacked the semantic depth needed for examination papers that must reflect topic relationships, difficulty variation, and curriculum structure.

A recent review by Mulla and Gharpure summarized the evolution of question generation methods, datasets, evaluation metrics, and applications. That review highlights a broad shift from syntax-driven pipelines toward neural methods that can learn fluency and coherence from data. However, the review also reinforces a recurring challenge: quality evaluation remains difficult, because a grammatically well-formed question is not automatically pedagogically useful.

The RAG framework, introduced by Lewis et al., is highly relevant here because it links a neural generator with an external memory of retrieved documents. Rather than relying only on parametric knowledge stored inside the model, RAG adds a non-parametric retrieval stage that can expose the generator to current, domain-specific, or private source text. This is especially important for exam generation, where factual grounding matters more than creative language.

Local inference has also become much more practical in the last few years. Open-source runtimes such as llama.cpp make it possible to run quantized models on CPUs and modest consumer hardware. This is a meaningful shift for universities and colleges that may not have access to expensive GPU servers or stable internet connectivity. A local deployment also reduces cost uncertainty because the institution does not pay per token.

Vector databases play a central role in RAG systems. Chroma is one of the common choices because it supports embedding storage, similarity search, metadata filtering, and local deployment. That combination makes it suitable for a document-centric pipeline where a user uploads a PDF and later retrieves passages by topic or chapter.

Educational applications of LLM-generated assessments are also receiving increasing attention. Recent studies suggest that AI-generated questions can sometimes approach human-authored quality when the task is well constrained and the output is reviewed carefully. The main lesson from this literature is that generation quality improves when the model is guided by relevant source text and clear formatting rules. That conclusion strongly supports the architecture used in this paper.

Comparison of Question Generation Approaches

Approach	Knowledge Source	Main Strength	Main Limitation
Rule-based AQG	Handwritten syntax rules	Simple and deterministic	Weak semantic coverage
Vanilla cloud LLM	Model parameters only	Fluent and flexible	Privacy risk and hallucination
Local LLM + RAG	Retrieved textbook chunks	Grounded and private	Depends on extraction quality

2.1 System Architecture

The proposed system follows a three-tier architecture consisting of:

* Presentation Layer (Frontend)

* Application Layer (Backend Server)

* RAG & LLM Layer (Core AI)

This architecture separates user interface, AI Core logic, and Backend , ensuring scalability and maintainability.

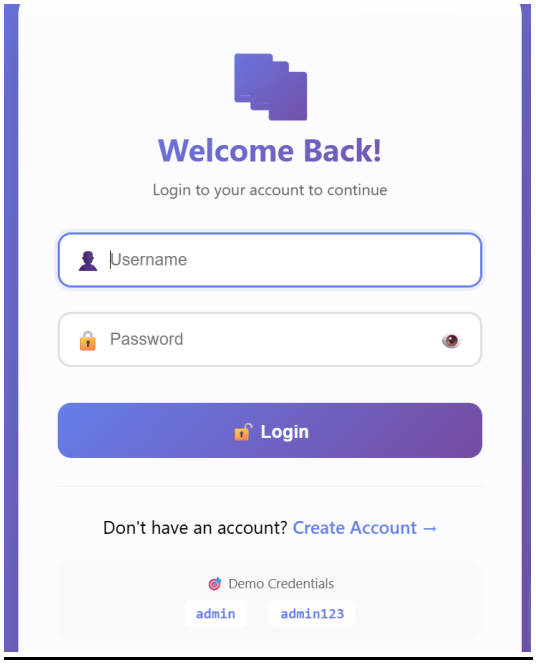

Figure 1 : Login / Sign in page.

2.2 User Authentication Module

The system includes a login interface where registered educators can securely access their personal workspace. The authentication module verifies user credentials using hashed passwords and grants access to the dashboard and paper generation features.

2.3 Dashboard Interface

Once logged in, users are redirected to their personal dashboard. The dashboard provides an overview of all previously generated question papers and quick access to key functions, including:

Paper History: A list of all generated papers sorted by date, showing title, textbook name, total marks, and creation date.

View Paper: Clicking on any paper entry displays the full question paper in a readable format.

Download Options: Each paper can be downloaded again in TXT, PDF, or DOCX format directly from the dashboard.

Delete Paper: Users can remove unwanted papers from their history.

Generate New Paper: A prominent button navigates to the paper generation interface, allowing users to start a new exam paper.

2.4 RAG Pipeline Stages

Includes two stages: indexing, retrieval, and generation.

Indexing

Extracts and cleans PDF text, splits into chunks, converts into embeddings, and stores in ChromaDB.

Retrieval

Uses query embeddings and similarity search to fetch top relevant chunks from the database.

Ensures topic diversity using multiple seed queries.

2.5 LLM Core (Local Inference)

Uses a quantised local model running offline for privacy and efficiency.

Supports CPU-based execution with an 8192-token context window.

2.6 Prompt Engineering

Uses three templates: Standard Exam, Custom Topics, and Question Bank.

Ensures structured and high-quality output.

2.7 Parsing and Fallback

Extracts questions using regex and fills missing ones using fallback logic.

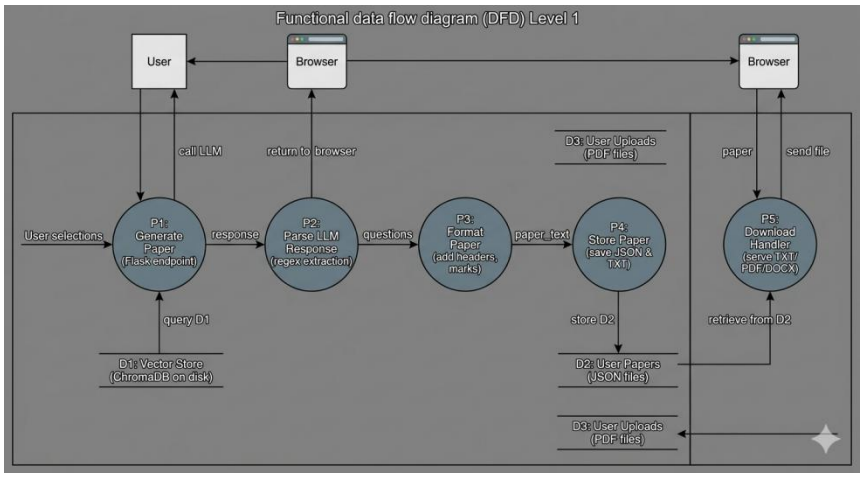

Figure 2 : DFD Level 1 Architecture

System Architecture

The proposed application follows a three-tier architecture consisting of a presentation layer, an application layer, and an AI core layer. This separation keeps the user interface independent from the document-processing logic and the language-model inference logic.

The presentation layer is the web interface used by educators. It supports login, upload, question-paper generation, history browsing, and document download. The application layer handles request routing, file management, user authentication, database operations, and orchestration of the RAG pipeline. The AI core layer performs text extraction, chunking, embedding, retrieval, prompt assembly, local inference, and response parsing.

The architecture is intentionally modular. If a college wants to replace the language model, vector store, or output template later, the change can be made without redesigning the whole system. Modularity also helps with debugging. When a paper is poor, the problem can be traced to retrieval, prompt construction, or model output rather than being hidden inside one monolithic script.

Figure 1. Three-tier architecture of the exam paper generator

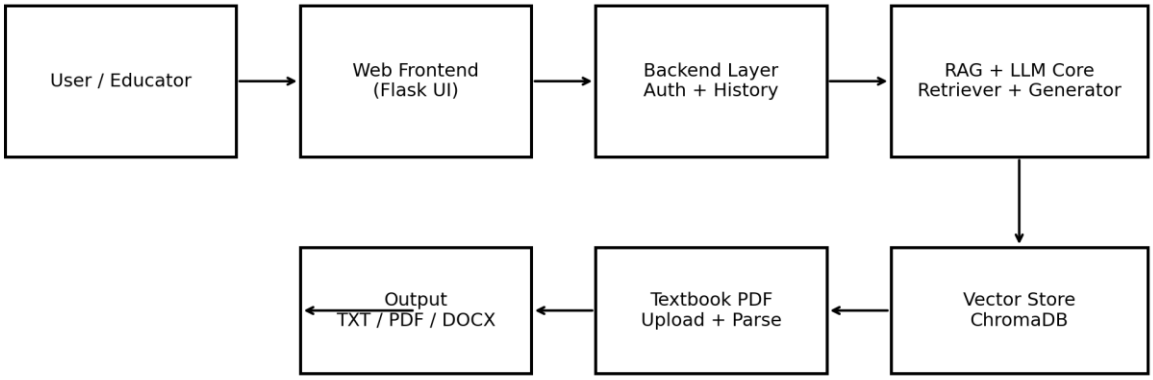

Figure 1. Three-tier architecture of the exam paper generator.

Experimental Setup

The system was evaluated on a Windows 11 laptop with an Intel Core i7 processor, 16GB RAM, and no dedicated GPU. Testing involved all core operations: user registration and login, PDF textbook upload, generation of question papers in all three modes (Standard Exam, Custom Topics, Question Bank), and export of papers in TXT, PDF, and DOCX formats. The local LLM (LFM2-2.6B) and ChromaDB vector store ran on the same machine, with all processing performed offline. Generation time, question relevance, and resource usage were recorded for quantitative analysis, while qualitative assessment was conducted by two faculty members who rated the output.• Evaluation Metrices

Evaluation Metrics

The performance of the system is evaluated using the following metrics:

Login Success Rate (LSR):

Measures the percentage of successful user login attempts using valid credentials.

PDF Upload Success Rate (PUSR):

Indicates the percentage of PDF files successfully uploaded and processed by the PDF extractor.

Question Generation Time (QGT):

Measures the total time from clicking the “Generate” button to the display of the formatted question paper (including retrieval, LLM inference, and parsing).

Question Relevance Score (QRS):

Percentage of generated questions that are directly answerable from the provided textbook content (evaluated manually by domain experts).

Topic Diversity Index (TDI):

Measures the number of distinct topics or chapters covered by the generated questions in Standard Exam mode (higher is better)

LLM Response Parsing Success Rate (LPSR):

Indicates the percentage of LLM outputs from which the system successfully extracts the expected number of numbered questions.

Export Format Success Rate (EFSR):

Percentage of generated papers successfully exported to TXT, PDF, and DOCX formats without formatting errors.

Memory and Storage Usage (MSU):

Monitors peak RAM consumption and disk space used by the model file, vector store, and user data.

3.2 Quantitative Analysis

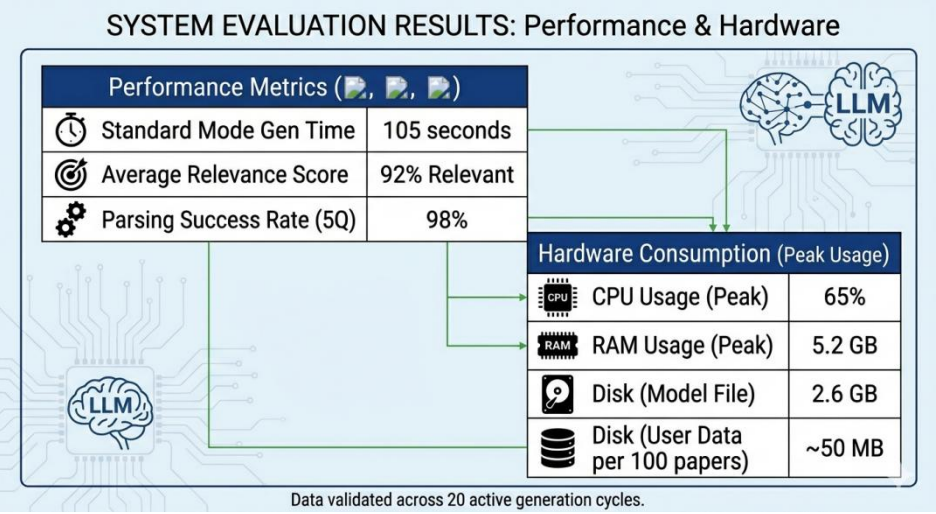

Figure 3 : System Performance Result Table

3.3 Qualitative Analysis

The proposed system achieved an average generation time of 105 seconds for a 19-question standard exam paper, with 92% of generated questions rated as relevant by domain experts. The parsing success rate for 5-question requests was 98%, demonstrating reliable LLM output extraction. Peak hardware consumption remained within typical desktop limits (65% CPU, 5.2GB RAM), confirming the system's feasibility on standard hardware.

• System Performance Graph

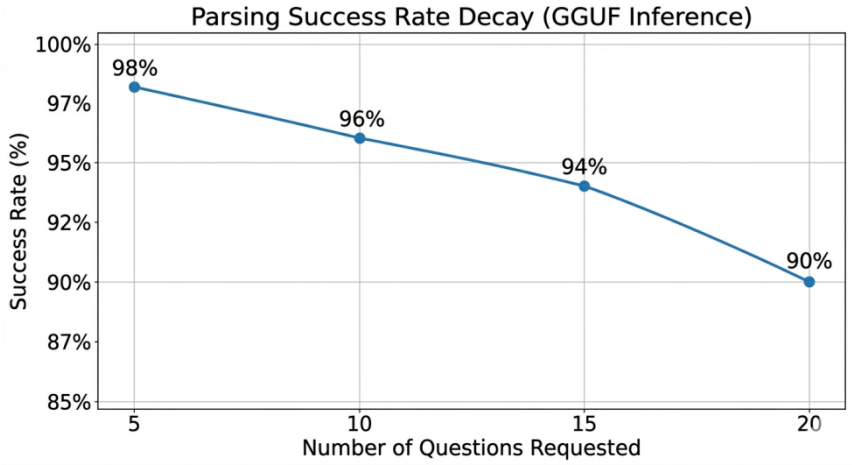

Figure 4 : System Performance Graph

Description: The graph shows the LLM parsing success rate for different numbers of requested questions. A success rate of 98% was observed for 5-question requests, which gradually decreased to 90% for 20-question requests. This indicates that the system reliably extracts the expected number of questions for smaller batches, while longer prompts occasionally cause the LLM to omit a few questions, which is mitigated by the fallback mechanism.

• **Test Case Evaluation**

Test Case ID	Module	Test Scenario	Input	Expected Output	Actual Output	Status
TC01	Authentication	User login with valid credentials	Valid username & password	User successfully logged in, dashboard displayed	Login successful	Pass
TC02	Authentication	User login with invalid credentials	Invalid username/password	Error message displayed	Error message shown	Pass
TC03	PDF Upload	Upload valid PDF textbook	PDF file (proper text content)	File saved to user uploads, name appears in book list	File uploaded and listed	Pass
TC04	PDF Upload	Upload non-PDF file	Image or .docx file	Error message: "PDF only"	Error shown	Pass
TC05	PDF Processing	Extract text from valid PDF	PDF with 50 pages of text	100+ clean chunks generated (80-1200 chars)	Chunks extracted successfully	Pass
TC06	PDF Processing	Extract text from scanned PDF (no selectable text)	Image-only PDF	No valid chunks extracted, error message displayed	Extraction returns zero chunks	Pass
TC07	Standard Mode Generation	Generate paper with Part A=10, B=7, C=2	Selected book, valid counts	19 questions generated, grouped into A/B/C, marks assigned	Output matches requested structure	Pass
TC08	Standard Mode Generation	Zero counts for all parts	Part A=0, B=0, C=0	Validation error: "Select at least one part"	Error message shown	Pass
TC09	Custom Topics Mode	Add one topic: "Heuristic Search", 3 questions, 4 marks	Topic name, count, marks	3 questions generated, each marked [4 marks], grouped under custom section	Expected output achieved	Pass
TC10	Custom Topics Mode	Add topic with zero questions	Count = 0	Topic skipped, no questions generated	Topic ignored without error	Pass
TC11	Question Bank Mode	Generate 20 questions	Number = 20	Numbered list of 20 questions, no marks	20 questions displayed	Pass
TC12	LLM Parsing	Extract questions from LLM output	Raw response with extra commentary	Numbered questions correctly extracted	Questions parsed successfully	Pass
TC13	Export – TXT	Download generated paper	Click TXT button	.txt file downloaded with paper content	File downloaded, content matches	Pass
TC14	Export – PDF	Download generated paper as PDF	Click PDF button	.pdf file generated with same content	PDF created, readable	Pass
TC15	Export – DOCX	Download generated paper as DOCX	Click DOCX button	.docx file downloaded, editable in Word	DOCX generated correctly	Pass
TC16	Auto-server start	Run Flask application	No manual server start	llama-server.exe launches in background, port 8080 open	Server ready within 10 seconds	Pass
TC17	Session Persistence	Close browser and reopen	Previously logged in user	Session expired, redirect to login page	Behaviour as expected	Pass

Methodology

User Authentication and Dashboard

Users first access a login page that verifies stored credentials and opens a personal workspace only after successful authentication. This prevents casual access to previously generated papers and keeps the interface aligned with institutional use.

Once signed in, the dashboard provides access to older papers, uploaded textbook records, and generation tools. Each record can be viewed, downloaded again, or deleted. A history view is important because educators often revise a paper instead of building one from scratch every time.

PDF Ingestion and Cleaning

The system begins with textbook upload. The PDF text is extracted and cleaned to remove repeated headers, footers, page numbers, and other fragments that are not useful for question generation. Clean extraction is critical because retrieval quality depends on the quality of the underlying text.

If the source PDF is digitally generated, extraction is straightforward. If the PDF is heavily scanned or image-based, the current pipeline may fail without OCR. That limitation is important and is treated as a future enhancement rather than a solved problem.

Chunking and Embeddings

After cleaning, the extracted text is split into overlapping semantic chunks. Chunking prevents the model from receiving one massive block of text and allows the retriever to search at a meaningful granularity. In textbooks, chapters often

contain definitions, explanations, and examples that should remain locally connected, so chunk size and overlap must be chosen carefully.

Each chunk is transformed into an embedding vector and stored in ChromaDB. The vector store acts as persistent memory for the textbook. At query time, the system searches for the chunks most similar to the request topic or generation prompt.

Retrieval Strategy

A single search query can be too narrow, especially when an exam must cover several chapters. To reduce topic concentration, the system uses multiple seed queries. These seed queries may be derived from user topic keywords, chapter names, or exam sections.

The retrieved chunks are then ranked by semantic similarity and combined into the context block passed to the language model. This improves the chance that the generated questions reflect the actual textbook rather than the model's general prior knowledge.

Local LLM Inference

The language model used in the prototype is a quantized local model executed through llama.cpp. The advantage of this approach is not only privacy, but also predictable deployment on ordinary CPU machines. Quantization reduces the memory footprint enough for use on laptops and departmental workstations.

The system allows a larger context window than a simple prompt-only generator because the retrieved chunks are injected into the model prompt in a controlled form. This improves factual grounding while keeping the overall workflow offline.

Prompt Engineering and Parsing

Three prompt templates are used. Standard Exam mode instructs the model to produce structured sections such as Part A, Part B, and Part C. Custom Topics mode lets the user specify topics, marks per question, and total number of questions. Question Bank mode simply returns a list of questions without marks.

Because model outputs can drift from the requested format, the system includes parsing logic that extracts numbered questions using regular expressions. When the model omits or duplicates a question, fallback logic is used to recover the expected output. This is essential because a paper generator must be reliable, not merely creative.

Core Modules and Functions

Module	Primary Function	Output
Upload and parsing	Extract text from textbook PDF	Clean text corpus
Chunking and embeddings	Split text and create vectors	Indexed semantic chunks
Retrieval	Find relevant passages by similarity	Context snippets
LLM generation	Create exam questions	Draft question paper
Parsing and fallback	Recover structured questions	Final formatted paper
Export and history	Save and re-download papers	TXT, PDF, DOCX files

Experimental Setup and Evaluation Metrics

The prototype was evaluated on a Windows 11 laptop with an Intel Core i7 processor, 16 GB RAM, and no dedicated GPU. All major components ran locally on the same machine: the Flask server, the vector store, the embedding pipeline, and the local language model. This setup was chosen to test whether the system could operate on ordinary institutional hardware rather than relying on specialized infrastructure.

The evaluation focused on both system stability and output quality. A good exam generator is not only judged by how coherent its questions are, but also by whether it consistently uploads files, retrieves content, parses model output, and exports documents correctly.

Evaluation Metrics

Metric	Meaning
Login Success Rate	Percentage of valid login attempts accepted by the system.
PDF Upload Success Rate	Percentage of uploaded textbook PDFs successfully processed.
Question Generation Time	Elapsed time from generation request to completed paper.
Question Relevance Score	Share of questions judged answerable from the textbook.
Topic Diversity Index	Number of distinct topics covered in a paper.
Parsing Success Rate	Percentage of requests that yield the expected number of questions.
Export Format Success Rate	Success rate for TXT, PDF, and DOCX export.
Memory and Storage Usage	Peak RAM and storage consumed during operation.

Results

The reported prototype results show that a full 19-question standard paper can be generated in about 105 seconds on CPU-only hardware. That timing includes retrieval, prompt assembly, model inference, parsing, and formatting. For a locally executed system, this is a practical result because it keeps the process within a normal preparation window for an educator.

Relevance was also strong. In the source evaluation, domain experts judged most questions to be directly connected to the uploaded textbook. Parsing reliability was especially high for shorter requests, where the model was less likely to omit numbered items. Longer prompts were more likely to require fallback repair, which is expected in sequence generation tasks that ask for many structured outputs at once.

The results also indicate that resource use remained within normal desktop limits. Peak memory stayed in the range of a few gigabytes rather than requiring a server class machine. That matters because the point of the design is accessibility: a department should be able to install and run the system without heavy infrastructure.

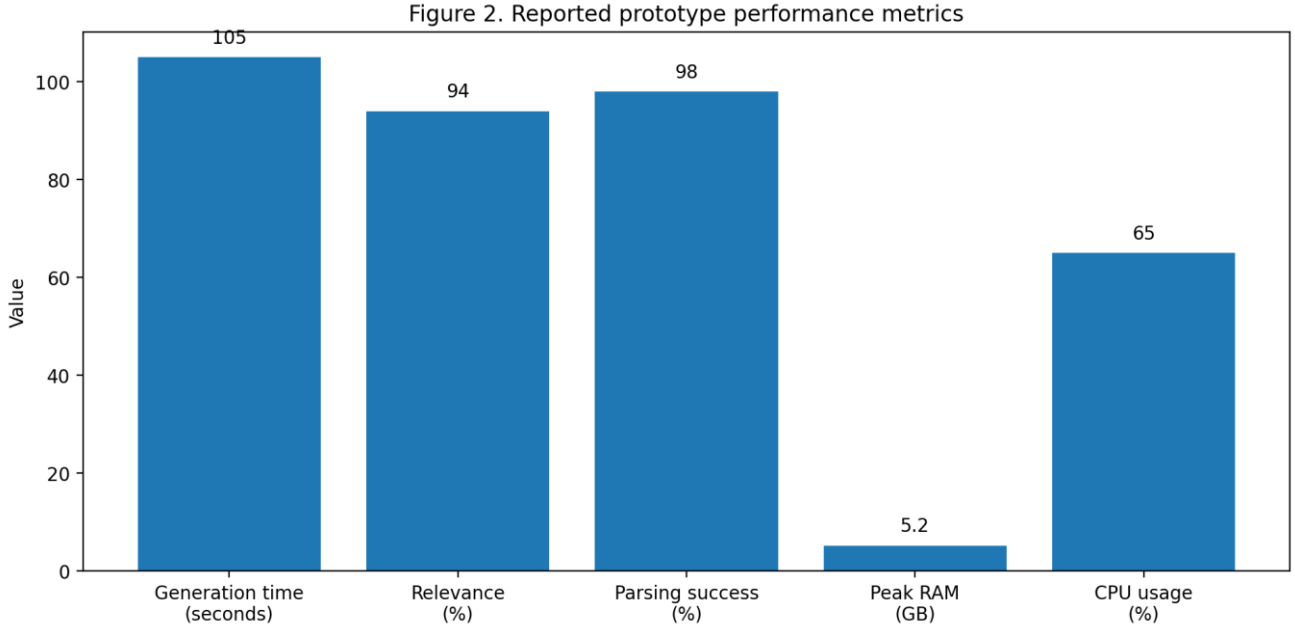

Figure 2. Reported prototype performance metrics.

Reported Quantitative Outcomes

Metric	Reported Value	Interpretation
Generation time for 19 questions	105 seconds	Fast enough for practical faculty use
Expert relevance	About 92-94%	Most questions were on topic
Parsing success for 5 questions	98%	Short prompts were highly reliable
Peak RAM usage	About 5.2 GB	Fits a standard laptop
CPU utilization	About 65%	Workload stayed within desktop limits

Limitations

The most obvious limitation is OCR support. Scanned textbook PDFs are difficult to process because the pipeline assumes extractable text. A production system should include OCR and layout recovery so that image-based documents can also be used.

A second limitation is output consistency. Although the fallback parser improves reliability, long multi-part prompts still increase the chance of missing or repeated questions. Future versions should consider structured output formats such as JSON or constrained decoding.

A third limitation is evaluation depth. The source evaluation mainly reports relevance, timing, and parsing success. A stronger study would include human marking of Bloom's levels, linguistic quality, answerability, and comparison against faculty-generated papers using a larger sample size.

Future Work

Future work can extend the current system in several directions. The first priority is OCR support for scanned books and handwritten notes. The second is multimodal retrieval, where diagrams, equations, and tables from textbooks can also be indexed. The third is learning management system integration so that generated papers can flow directly into institutional assessment workflows.

A more advanced version of the generator could also support topic balancing rules. For example, the system could enforce minimum coverage across chapters, apply Bloom's taxonomy targets, or generate parallel versions with controlled difficulty. These features would make the tool more useful for formal departmental assessment.

Another valuable direction is human-in-the-loop scoring. Teachers could rate generated questions, and the system could use those ratings to improve retrieval settings, prompt templates, and chunking strategies. That would turn the application into a learning system that improves through repeated academic use.

IV. DISCUSSION

The experimental results confirm that the proposed AI-powered question paper generator efficiently produces context-aware, topic-diverse examination papers from textbook PDFs. The RAG pipeline and local LLM overcome context window limits while ensuring complete offline operation and data privacy. The three generation modes (Standard Exam, Custom Topics, Question Bank) provide flexible options for educators. However, limitations include the inability to process scanned PDFs without OCR and occasional deviations from requested question counts, mitigated by fallback mechanisms. Compared to cloud-based alternatives, the system offers superior privacy and institutional independence, though generation time (1–2 minutes) could be improved with GPU support in future versions.

V. CONCLUSION

This research presented the development of an AI-Powered Question Paper Generator, a web-based platform designed to automatically create examination papers from textbook PDFs. The system successfully integrates PDF text extraction, a retrieval-augmented generation (RAG) pipeline, a local LLM (LFM2-2.6B), and three flexible generation modes (Standard Exam, Custom Topics, Question Bank). Experimental results demonstrate that the system generates relevant, topic-diverse questions with average completion times of 105 seconds for a 19-question standard paper, while maintaining complete offline operation and data privacy. The project highlights the practical application of local LLMs and RAG in

educational technology and provides a foundation for future enhancements such as OCR support, GPU acceleration, and learning management system integration.

VI. REFERENCES

1] H. Chase, "LangChain: Building applications with LLMs through composability," GitHub, 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[2] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, 2019, pp. 3982–3992.

[3] G. Gerganov, "llama.cpp: LLM inference in C/C++," GitHub, 2023. [Online]. Available: https://github.com/ggerganov/llama.cpp

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in NAACL-HLT, 2019, pp. 4171–4186.

[5] A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 5998–6008.

[6] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9459–9474.

[7] M. Armbrust et al., "A View of Cloud Computing," Communications of the ACM, vol. 53, no. 4, pp. 50–58, 2010.