

A Retrieval-Augmented Question Answering System Using BERT for Stream-Specific Educational Content

Submitted in partial fulfilment of the
requirements of the Degree: M.Tech in Data science and engineering

By

Jaykumar Chaudhary

2022DC04341

Under the supervision of

Mr. Lalkar Eknath Chhadawelkar

Technical Evangelist (Cybage Software, Pune)



BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

AUGUST, 2025

Acknowledgement

I would like to express my heartfelt gratitude to all those who have supported and guided me throughout the journey of this dissertation.

First and foremost, I am deeply thankful to my supervisor, **Mr. Lalkar Eknath Chhadawelkar**, for his invaluable guidance, continuous encouragement, and patient supervision at every stage of this project. His insightful feedback and expertise have been crucial in shaping my work and motivating me to achieve my best.

I would also like to sincerely thank **Mrs. Radhika Raghuvanshi** for her initial guidance and for helping me refine the core idea of my dissertation. Her suggestions on the overall roadmap and her clarity of thought gave direction to my efforts from the very beginning.

My special thanks to **Jayalakshmi Natarajan ma'am**, my BITS evaluator, for her thoughtful evaluation and constructive comments, which have helped me strengthen both my research and its presentation.

I sincerely thank **Mr. Rakesh Sharma**, my additional examiner, for his valuable time and insightful feedback. His observations and guidance have greatly enriched this dissertation.

Finally, I owe my deepest appreciation to my family for their unwavering support, constant encouragement, and understanding. Their faith in me has been my greatest source of motivation throughout this academic journey.

To all those who have helped me, directly or indirectly, in completing this work, I extend my sincere thanks.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

SECOND SEMESTER 2024-25

DSECLZG628T / AIMLCZG628T DISSERTATION

CERTIFICATE

This is to certify that the Dissertation entitled A Retrieval-Augmented Question Answering System Using BERT for Stream-Specific Educational Content

and submitted by Mr. Jaykumar Chaudhary ID No. 2022DC04341 in partial fulfilment of the requirements of DSECLZG628T Dissertation, embodies the work done by him/her under my supervision.

BITS ID No. 2022DC04341 Name of Student: Jaykumar Chaudhary

Name of Supervisor: Mr. Lalkar Eknath Chhadawelkar

Designation of Supervisor: Technical Evangelist

Qualification and Experience: M.Tech in Data Science and Engineering. 28 Years

Official E- mail ID of Supervisor: lalkar@cybage.com

Topic of Dissertation: A Retrieval-Augmented Question Answering System Using BERT for Stream-Specific Educational Content



Signature of Student

Date: 14-AUGUST-2025



Signature of Supervisor

Date: 14-AUGUST-2025

Abstract

In academic environments, students often struggle to quickly locate specific information across large, text-heavy educational documents such as course handouts. As elective subjects vary across semesters, navigating this material can be overwhelming, especially when trying to understand topics or explore subjects before making elective decisions. Traditional keyword-based search methods lack contextual understanding, leading to imprecise and inefficient information retrieval.

This dissertation proposes the design and development of a **Retrieval-Augmented Question Answering (QA) system** that leverages a **pretrained BERT-based model** to enable students to query academic content in natural language and receive accurate, contextually relevant answers. The system is primarily designed to process structured and semi-structured educational content — including course handouts — from the **Data Science** and **AI/ML streams**.

The proposed solution follows a **Retrieval-Augmented Generation (RAG) architecture** that separates the QA process into two components: **retrieval** and **reading**. First, a **retriever** uses **Sentence-BERT** to generate **embeddings** and identify relevant content, which is then indexed using **FAISS** for efficient **semantic search**. Next, a **reader model (BERT fine-tuned on SQuAD)** extracts the most likely answer span from the retrieved text.

Unlike conventional QA systems that require domain-specific fine-tuning or extensive labeled data, this system uses pretrained components and unsupervised document chunking. This makes it scalable, adaptable, and ideal for academic use. The system is designed to help students retrieve reliable answers to subject-related queries, assist in exam preparation, and support elective planning by giving clarity on subject depth and focus.

The expected result is a functional prototype capable of answering factual, definition-based, and conceptual questions using curriculum-aligned educational content. The system will be evaluated through QA metrics like **Exact Match** and **F1 Score**, along with qualitative feedback from users. This dissertation also discusses limitations such as the lack of support for **multi-hop reasoning** or generative answers. Nonetheless, it showcases a practical application of **NLP** in the academic domain, offering an intelligent way to support self-directed learning.

Key Words

BERT, Question Answering, Retrieval-Augmented Generation, Semantic Search, Educational NLP, Sentence-BERT, FAISS, Extractive QA

List of Symbols & Abbreviations

Symbol / Abbreviation	Description
NLP	Natural Language Processing
QA	Question Answering
RAG	Retrieval-Augmented Generation
LLM	Large Language Model
BERT	Bidirectional Encoder Representations from Transformers
SQuAD	Stanford Question Answering Dataset
FAISS	Facebook AI Similarity Search
ChromaDB	Chroma Vector Database
API	Application Programming Interface
PDF	Portable Document Format
DOCX	Microsoft Word Open XML Document Format
UI	User Interface
ID	Identification Number
RAGAS	Retrieval-Augmented Generation Assessment Suite
UUID	Universally Unique Identifier — a standardized 128-bit value
NV	Nvidia — refers to Nvidia-provided AI evaluation metrics.
BLEU	Bilingual Evaluation Understudy Metric
ROUGE	Recall-Oriented Understudy for Gisting Evaluation Metric
NLI	Natural Language Inference
JSON	JavaScript Object Notation
PyPDFLoader	Python-based loader for extracting text from PDF documents.
Docx2txtLoader	Loader for extracting text from DOCX files into plain text format.
Ollama	An LLM backend that enables local inference for RAG applications.
Gemini	Google's LLM backend used for QA and evaluation.
Groq	An LLM backend leveraging Llama models for QA and summarization.

NER	Named Entity Recognition — an NLP technique for detecting entities like names, dates, and places in text.
F1 Score	A harmonic mean of precision and recall, used to measure accuracy.
TP	True Positive
FP	False Positive
FN	False Negative
LlamaParse	LlamaIndex’s advanced parsing tool for complex documents, enabling structured output.
LlamaCloud	LlamaIndex’s cloud platform offering managed services for parsing and indexing.
Markdown	A lightweight markup language for formatting text, often used for structured document representation.

List of Tables

<i>Table 1: Chunks After Preprocessing and Hybrid Chunking</i>	15
<i>Table 2: Embedding Vectors for Text Chunks.....</i>	16
<i>Table 3: FAISS Retrieval Results for Sample Queries.....</i>	17
<i>Table 4: Example of Answer Sources and Types in Edge Cases.....</i>	21
<i>Table 5: comparison of the baseline and modern RAG pipeline architectures.....</i>	26
<i>Table 6: Uploaded document storage record</i>	29
<i>Table 7: Parsed documents with different Parsing Methods</i>	32
<i>Table 8: Metadata for sematically chunked document.....</i>	37
<i>Table 9: Example of Semantic Chunk Metadata.....</i>	37
<i>Table 10: Example Embeddings for Semantic Chunks</i>	38
<i>Table 11: Example Entries in ChromaDB Vector Store.....</i>	40
<i>Table 12: API parameters and response fields for question-answering endpoint.....</i>	44
<i>Table 13: Metadata in case of additional context flag enabled.....</i>	49
<i>Table 14: Aggregated Metric Table for Retrieval Category.....</i>	71
<i>Table 15: Aggregated Metric Table for Nvidia Category</i>	75
<i>Table 16: Aggregated Metric Table for Language Category</i>	82

List of Figures

<i>Figure 1: RAG Question Answering Pipeline</i>	<i>10</i>
<i>Figure 2: Preprocessing and Hybrid Chunking Workflow</i>	<i>14</i>
<i>Figure 3: all-MiniLM-L6-v2 Model Architecture.....</i>	<i>15</i>
<i>Figure 4: Implementation Flow of Embedding and Retrieval Pipeline</i>	<i>18</i>
<i>Figure 5: BERT-Large SQuAD QA Pipeline Architecture</i>	<i>19</i>
<i>Figure 6: Decision Flow for Reader Module Edge Cases</i>	<i>21</i>
<i>Figure 7: Overview of the modern modular RAG pipeline architecture.....</i>	<i>26</i>
<i>Figure 8: Document upload and ingestion trigger workflow</i>	<i>28</i>
<i>Figure 9: Academic document upload panel with optional advanced parsing toggle</i>	<i>29</i>
<i>Figure 10: Flowchart for Standard Document Ingestion and Parsing.....</i>	<i>31</i>
<i>Figure 11: Confirmation message indicating successful document ingestion.....</i>	<i>31</i>
<i>Figure 12: Ingest Complex Documents with LlamaParse</i>	<i>33</i>
<i>Figure 13: Enhanced ingestion pipeline with LlamaParse</i>	<i>35</i>
<i>Figure 14: LlamaParsed Document available on cloud</i>	<i>36</i>
<i>Figure 15: Semantic Chunking Workflow</i>	<i>36</i>
<i>Figure 16: Embedding Workflow for Semantic Chunks.....</i>	<i>38</i>
<i>Figure 17: Document and query embeddings into ChromaDB</i>	<i>39</i>
<i>Figure 18: Construction of a retriever object from the ChromaDB vector index</i>	<i>40</i>
<i>Figure 19: LLM selection menu on ask question page.....</i>	<i>42</i>
<i>Figure 20: QA interface showing subject selection, LLM choice, and direct answer</i>	<i>45</i>
<i>Figure 21: LLM-Based Answer Generation and Polishing Workflow.....</i>	<i>47</i>
<i>Figure 22: Comprehensive overview of the answer generation and context preview</i>	<i>48</i>
<i>Figure 23: Generated answer with additional context</i>	<i>50</i>
<i>Figure 24: Feedback flow mechanism in QA system</i>	<i>52</i>
<i>Figure 25: Feedback providing UI option</i>	<i>53</i>
<i>Figure 26: Success notification for user feedback submission</i>	<i>53</i>
<i>Figure 27: End-to-End document summarization flow in QA generation</i>	<i>55</i>
<i>Figure 28: QA pair generation flow using summary and prompt.....</i>	<i>58</i>
<i>Figure 29: Input Configuration & Question Setup for Question paper generation.....</i>	<i>60</i>
<i>Figure 30: Additional Instructions & Summary Preview</i>	<i>61</i>

<i>Figure 31: Generated Questions Preview & Export</i>	<i>62</i>
<i>Figure 32: RAGAS evaluation pipeline</i>	<i>64</i>
<i>Figure 33: Bar chart for Context Precision scores per model</i>	<i>67</i>
<i>Figure 34: Bar chart for Context Recall scores per model</i>	<i>68</i>
<i>Figure 35: Bar chart for Faithfulness scores per model</i>	<i>69</i>
<i>Figure 36: Radar chart for Retrieval Metrics</i>	<i>70</i>
<i>Figure 37: Bar chart for NV Answer Accuracy</i>	<i>72</i>
<i>Figure 38: Bar Chart for NV Context Relevance</i>	<i>73</i>
<i>Figure 39: Bar Chart for NV Response Groundedness</i>	<i>74</i>
<i>Figure 40: Radar chart for Nvidia metrics Category</i>	<i>75</i>
<i>Figure 41: Bar Chart for Factual Correctness</i>	<i>76</i>
<i>Figure 42: Bar Chart for Semantic Similarity</i>	<i>77</i>
<i>Figure 43: Bar Chart for BLEU Score</i>	<i>78</i>
<i>Figure 44: Bar Chart for ROUGE score</i>	<i>79</i>
<i>Figure 45: Barc Chart for String Presence</i>	<i>80</i>
<i>Figure 46: Bar chart for Exact Match</i>	<i>81</i>
<i>Figure 47: Radar chart for Nvidia metrics Category</i>	<i>82</i>
<i>Figure 48: RAGAS per question drill down view</i>	<i>88</i>

Table of Contents

Abstract.....	1
Chapter 1 Introduction.....	8
1.1 The Challenge of Academic Information Access.....	8
1.2 Problem Statement	8
1.3 Motivation and Project Evolution.....	9
1.4 System Overview	10
1.5 Objectives of the Dissertation	10
Chapter 2 Literature Survey.....	12
2.1 Introduction.....	12
2.2 Classic RAG and BERT-Based QA	12
2.3 Advances in Chunking and Embedding.....	12
2.4 Modern Modular RAG with LLMs	13
2.5 Summary	13
Chapter 3 Baseline RAG System (Traditional Approach)	14
3.1 Introduction.....	14
3.2 Document Chunking & Pre-processing	14
3.2 Embedding & Retrieval	15
3.2.1 Semantic Embedding with all-MiniLM-L6-v2	15
3.2.2 Efficient Retrieval with FAISS.....	16
3.2.3 Implementation Workflow: Embedding and Retrieval.....	17
3.3 Reader Module.....	18
3.3.1 Answer Extraction with BERT QA Model.....	19
3.3.2 Handling Edge Cases: Keyword and Fall-back Heuristics	20
3.4 Observations & Limitations	22
Chapter 4 Modern RAG System with LLMs & Ecosystem Tools	23
4.1 Motivation for Advancing the Pipeline	23
4.2 New Architecture Overview.....	24
4.2.1 System Design and Data Flow.....	25
4.3 Modular Retrieval Layer: Upload to Semantic Retrieval.....	27
4.3.1 Document Upload & Subject Mapping.....	27
4.3.2 Document Ingestion & Parsing.....	30
4.3.3 Semantic Chunking of Academic Documents	36
4.3.4 Advanced Embedding with FastEmbed (BAAI/bge-base-en-v1.5).....	37
4.3.5 Persistent Vector Storage with ChromaDB	38
4.3.6 Retriever Object Creation from the Vector Store	40

Chapter 5 Reader Module and User Interaction Layer	41
5.1 Multi-LLM Answer Generation and Orchestration	41
5.1.1 Overview of Each LLM	42
5.1.2 Unified Backend Workflow for Answer Generation	43
5.1.3 In-Memory Caching Mechanism for Optimized RAG Execution	45
5.1.4 Prompt Construction with Retrieved Context	46
5.1.5 LLM-Based Answer Generation Workflow	46
5.2 Answer Presentation with Context Preview and Metadata	47
5.3 Feedback and Retrieval Adaptation	50
5.3.1 Feedback-Aware Re-ranking Logic	52
Chapter 6 Document Summarization & Prompt-Driven Q/A Generation	54
6.1 Summarization Pipeline: Reducing Context to Its Core	54
6.2 End-to-End Summarization Pipeline	55
6.3 Prompt Construction and Response Generation	57
6.4 Educator View: Summary and Generated Questions	60
Chapter 7 Evaluation & RAGAS-Based Benchmarking	63
7.1 RAGAS Evaluation Pipeline	64
7.2: Retrieval Metrics	66
7.3 Nvidia Metrics	71
7.4 Natural Language Comparison Metrics	75
Chapter 8 Conclusions and Recommendations for Future Work	83
8.1 Conclusions	83
8.2 Recommendations and Future Work	83
Chapter 9 Bibliography / References	85
Chapter 10 Appendix	87
Appendix A: Llama Ecosystem	87
A.1 LlamaParse – Advanced Document Parsing Capabilities	87
A.2 LlamaCloud Parsing Modes Overview	87
Appendix B: Extended RAGAS Metrics and Per-Question Drill-Down View	88
B.1 RAGAS Per-Question Drilldown Analysis View	88
B.2 Additional RAGAS Evaluation Metrics	88

Chapter 1

Introduction

1.1 The Challenge of Academic Information Access

In the digital age, students and educators have unprecedented access to academic resources—PDF handouts, lecture notes, assignments, and supplementary readings—often spanning hundreds or even thousands of pages. Paradoxically, this abundance can make it more difficult to pinpoint precise, contextually accurate answers to academic questions. Traditional keyword-based search tools fall short in this environment: they fail to capture semantic relationships, often misinterpret technical terminology, and cannot reliably handle mathematical or tabular content.

Recent advances in Natural Language Processing (NLP) offer transformative possibilities. **Retrieval-Augmented Generation (RAG)** has emerged as a powerful paradigm that combines semantic search with large language models (LLMs) to produce direct, context-rich responses. By retrieving relevant passages from large collections and then generating an answer grounded in those passages, RAG systems move beyond keyword matching to deliver explanations in natural, domain-appropriate language.

However, applying RAG effectively to the educational domain introduces new challenges. Academic documents are diverse in format, often containing complex structures such as multi-column layouts, equations, and embedded tables. Privacy and offline access are also critical for many institutions, creating demand for local inference options. To meet these needs, this dissertation advances a robust, modular RAG system capable of:

- Parsing and indexing multi-format academic documents (**PDF, DOCX**) with structure preservation.
- Handling complex layouts through tools like **LlamaParse**.
- Supporting both **cloud LLMs (Gemini, Groq)** and **local LLMs** via **Ollama** for privacy-preserving deployment.
- Integrating user feedback to iteratively improve retrieval accuracy and answer quality.

1.2 Problem Statement

While academic resources are more accessible than ever, extracting precise, well-contextualized answers from them remains a challenge. Existing search and Q/A tools often exhibit three key limitations:

1. **Semantic Gap** – Lack of deep understanding of academic language and structure, resulting in irrelevant or incomplete answers.
2. **Format and Structure Blindness** – Difficulty processing non-linear layouts, formulas, and tables in PDFs or Word documents.
3. **Limited Trust and Adaptability** – Minimal transparency in how answers are derived, and little capacity to adapt to user feedback.

The goal of this work is to design and implement an intelligent, context-aware question answering system that addresses these issues by combining advanced retrieval methods, modular LLM orchestration, and robust document parsing—delivering answers that are both accurate and traceable to their sources.

1.3 Motivation and Project Evolution

This project began as a classic, BERT-based QA pipeline employing semantic chunking, Sentence-BERT embeddings, and extractive answer selection. This early system proved that context-aware retrieval in the educational domain is feasible but also revealed opportunities for substantial improvement.

The second phase, which forms the focus of this dissertation, builds on that foundation by integrating:

- **LangChain** for modular orchestration of retrieval and generation steps.
- **ChromaDB** for scalable, persistent vector storage.
- **Multiple LLM backends:**
 1. **Gemini** for high-quality, context-rich responses.
 2. **Groq** for ultra-low latency inference.
 3. **Ollama** for running open-weight LLMs locally, ensuring privacy and offline capability.
- **LlamaParse** for accurate parsing of complex PDFs, including multi-column layouts, formulas, and tables.
- **Multi-format ingestion** to support both PDF and DOCX academic documents.
- **User feedback integration** for adaptive retrieval improvement.
- **Document summarization** and **prompt-driven Q/A generation** to enable topic-specific learning materials.
- **RAGAS evaluation** to systematically measure answer faithfulness, context recall, and relevance.

This evolution represents a deliberate and strategic enhancement—not a pivot due to limitations—ensuring that the final system is flexible, scalable, and tailored to real academic needs.

1.4 System Overview

The following diagrams illustrate both the baseline and advanced architectures implemented in this project. This project implements a modern, modular Retrieval-Augmented Generation (RAG) application for educational question answering. The RAG methodology blends efficient document retrieval with advanced language generation, enabling the system to deliver precise, context-aware answers tailored to user queries.

The **high-level workflow** of the system is outlined in Figure 1 below:

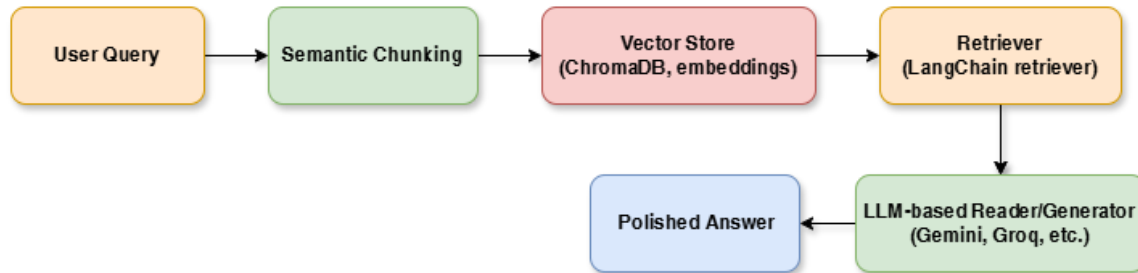


Figure 1: RAG Question Answering Pipeline

This modular architecture enables each pipeline component to be independently developed, tuned, or upgraded. Academic documents in various formats are ingested, chunked, and embedded using advanced models, with vectors stored in ChromaDB for fast semantic retrieval. LangChain orchestrates retrieval workflows, supporting context-aware, multi-hop queries. Retrieved passages and user questions are processed by state-of-the-art LLM readers (Gemini, Groq), which generate explanatory or synthesized answers. The design also anticipates integration of local LLMs via Ollama, ensuring privacy and control for institution-specific deployments. This foundation positions the system for rapid adaptation as new models and retrieval tools emerge.

This streamlined RAG workflow underpins the flexible and robust academic QA capabilities discussed in greater depth throughout Chapters 3 and 4.

1.5 Objectives of the Dissertation

The overarching aim of this dissertation is to bridge the gap between static academic documents and dynamic, question-driven learning. This work focuses on designing, implementing, and evaluating a modular Retrieval-Augmented Generation (RAG) system tailored for the educational domain. The specific objectives are as follows:

1. Develop a scalable, semantic QA system for multi-format academic documents

Build a platform that can ingest and process PDF and DOCX files while preserving their structure, including multi-column layouts, tables, and formulas, ensuring accurate downstream retrieval and answering.

2. Implement modular LLM orchestration for cloud and local inference

Integrate Gemini, Groq, and Ollama as interchangeable backends, enabling deployment scenarios that balance performance, cost, and privacy needs.

3. Design advanced retrieval strategies for high-fidelity context selection

Use semantic embeddings with context expansion and metadata filtering to ensure relevant, complete, and traceable evidence is retrieved for each question.

4. Ensure trustworthiness and transparency in generated answers

Ground every answer in retrieved context, display provenance metadata, and use linguistic polishing without altering factual accuracy.

5. Enable summarization and prompt-driven Q/A generation

Support concise content summarization and use it to generate focused questions and answers for study or assessment purposes.

6. Integrate user feedback for continuous improvement

Capture binary and comment-based feedback to inform retrieval re-ranking and model tuning over time.

7. Evaluate the system using RAGAS and comparative benchmarking

Measure performance across multiple metrics—faithfulness, context recall, and answer relevance—while benchmarking LLM backends for speed, accuracy, and usability.

Chapter 2

Literature Survey

2.1 Introduction

The landscape of question answering (QA) systems has evolved rapidly with advancements in natural language processing and deep learning. Traditional information retrieval systems—relying on keyword matching, bag-of-words models, or statistical ranking—could rarely capture the rich context and deep semantics inherent in complex academic materials. As educational resources have grown in scale and complexity, students increasingly need tools that can understand context, reason over information, and generate precise, trustworthy answers. Retrieval-Augmented Generation (RAG) addresses this need by combining the power of modern information retrieval with the generation capabilities of large language models (LLMs). This dual approach enables systems to first identify the most relevant parts of vast document collections and then generate high-quality, contextually aware answers that are both accurate and explainable—qualities essential for educational applications where trust and traceability matter ([Liu et al., 2023](#); [Bo Ni et al., 2025](#); [Shah et al., 2024](#)).

2.2 Classic RAG and BERT-Based QA

Classic RAG pipelines represent a significant leap forward from earlier QA architectures. At their core, these systems operate in two tightly integrated stages: retrieval and reading. In the retrieval stage, a user's question is converted into a dense vector representation, often using models such as Sentence-BERT. This vector is then used to search a vector database—commonly implemented with scalable tools like FAISS—to identify and rank the most semantically relevant document chunks ([Liu et al., 2023](#); [Wenqi Fan et al., 2024](#)). The use of dense retrieval allows the system to capture subtle relationships, such as synonyms or rephrased concepts, that keyword search would miss.

Once the top-ranked chunks are retrieved, the reader stage utilizes a powerful pre-trained or fine-tuned BERT model to extract the specific answer span. This model can process entire passages, weighing both the immediate question and the broader context, to find the most likely answer. This two-step process—first narrowing the search space and then drilling down to the answer—delivers both efficiency and accuracy, making it highly effective for academic QA where precision is critical and answers are often embedded in dense, technical text ([Wenqi Fan et al., 2024](#); [Liu et al., 2023](#)).

2.3 Advances in Chunking and Embedding

Chunking and embedding are foundational steps in any RAG system, but their importance is magnified in the educational domain. Academic materials are typically structured around complex, multi-part explanations, hierarchical headings, bulleted lists, and embedded formulas. Basic approaches that divide text by arbitrary length or page breaks risk

fragmenting crucial context, leading to poor retrieval performance. Recent research and practical experiments advocate for **semantic chunking**: grouping together related explanations, definitions, and associated mathematical content so that each chunk represents a coherent knowledge unit (Franklin Lee & Tengfei Ma, 2025; Shah et al., 2024).

These carefully constructed chunks are then embedded into a high-dimensional vector space using advanced models like FastEmbed or domain-adapted Sentence-BERT. Embeddings capture both local and global context, enabling the system to match queries with the most appropriate content, even if it is phrased differently or spread across multiple sections. Effective chunking and embedding ensure that answers are not only relevant but also pedagogically sound—providing students with both factual content and the necessary context for deeper understanding (M. Shah et al., 2024; Bo Ni et al., 2025).

2.4 Modern Modular RAG with LLMs

The newest generation of RAG systems capitalizes on the capabilities of large language models and the flexibility of modular orchestration frameworks. Unlike earlier extractive QA pipelines, these architectures can perform complex reasoning and synthesis—combining information from multiple sources and presenting it in clear, structured, and natural language. Modular tools such as **LangChain** allow developers to seamlessly connect document loaders, semantic chunkers, dense retrievers, and a variety of LLMs (including Gemini, Groq, and locally hosted Ollama) within a single pipeline (Sonal Prabhune & Donald J. Berndt, 2024; Petko Georgiev et al., 2024; Rohan Anil et al., 2023).

Scalable vector databases like **ChromaDB** ensure that even large, multi-document educational datasets can be efficiently indexed and queried. LLMs are not limited to extracting text—they can summarize, paraphrase, and even explain answers, which is particularly valuable in education where explanations often matter as much as facts. The modularity of these systems supports easy experimentation, integration of new models, and adaptation to different academic subjects or user requirements. Studies show that such modular RAG systems outperform traditional pipelines in both answer quality and user engagement, while also making it easier to maintain and upgrade the QA system over time (Sonal Prabhune & Donald J. Berndt, 2024; Franklin Lee & Tengfei Ma, 2025).

2.5 Summary

Overall, the evolution from classic BERT-based QA to advanced, LLM-driven RAG systems enables much richer, more flexible question answering in academic settings. Our project follows this trajectory, combining the best practices identified in recent literature with innovations in modularity and retrieval (Liu et al., 2023; Shah et al., 2024).

Chapter 3

Baseline RAG System (Traditional Approach)

3.1 Introduction

This chapter presents the baseline question answering pipeline developed as the foundation of this project. This approach leverages established retrieval-augmented generation (RAG) techniques, focusing on traditional methods for preprocessing, chunking, semantic retrieval, and answer extraction. By systematically transforming raw academic materials into structured, searchable units and employing proven models for both retrieval and reading, the baseline system establishes the core workflow upon which later advancements are built. The results and observations from this pipeline not only demonstrate the feasibility of retrieval-based QA in educational contexts but also highlight key limitations that motivate the development of a more advanced, LLM-driven solution in subsequent chapters.

3.2 Document Chunking & Pre-processing

A robust question answering system must first convert raw academic materials—whether in PDF or DOCX format—into structured, meaningful segments for downstream analysis. This process, known as pre-processing and chunking, is crucial for ensuring the system can accurately locate and extract relevant answers.

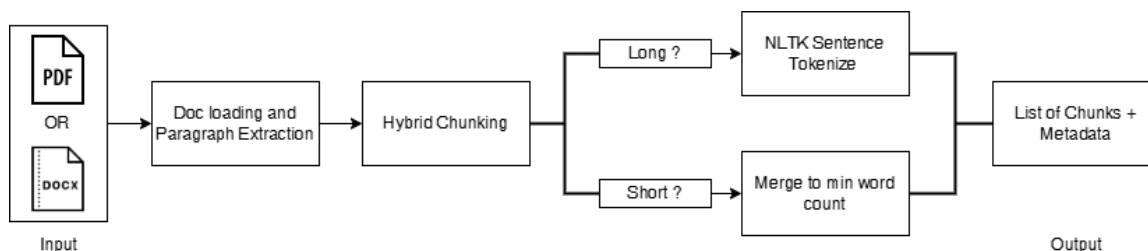


Figure 2: Preprocessing and Hybrid Chunking Workflow

This diagram illustrates the end-to-end workflow from input document (PDF or DOCX) through paragraph extraction and intelligent chunking, resulting in a collection of context-rich text segments ready for retrieval.

In this workflow, input documents are first loaded using appropriate tools (such as python-docx for DOCX files or a PDF parsing library for scanned materials). The system extracts non-empty paragraphs to ensure that only content-bearing text is processed. Each paragraph is then evaluated: unusually long paragraphs are split into sentences and regrouped using a hybrid strategy, while shorter paragraphs are merged to achieve optimal chunk size. This approach leverages both the document's inherent structure and linguistic boundaries, resulting in chunks that balance context with focus.

The importance of this workflow lies in its direct impact on retrieval effectiveness. If chunks are too large, they may contain irrelevant or diluted information; if too small, they can lose

essential context. By dynamically adjusting chunk size and boundaries, the system produces segments that are well-matched to the variety of queries posed by students.

One clear advantage of supporting both PDF and DOCX formats is the flexibility it offers in academic settings, where course materials can appear in a range of digital forms. The use of proven NLP libraries such as NLTK for sentence tokenization further enhances the quality of chunking, ensuring linguistic coherence within each segment.

A practical outcome of this preprocessing and chunking is shown below:

Chunk #	Word Count	Chunk Preview
1	96	Entropy is a measure of uncertainty in machine learning...
2	109	Regularization helps to prevent overfitting by introducing a penalty term...
3	80	The bias-variance tradeoff describes how model complexity affects prediction error...

Table 1: Chunks After Preprocessing and Hybrid Chunking

Through this approach, the system generates a curated list of searchable, informative text segments—laying a strong foundation for subsequent semantic embedding, retrieval, and answer extraction. This thoughtful chunking is essential for both the efficiency and accuracy of the overall QA pipeline, directly influencing the quality of answers provided to students.

3.2 Embedding & Retrieval

Accurate retrieval in question answering systems hinges on two fundamental processes: the conversion of text into meaningful vector representations, and the rapid search of these vectors to find relevant content. This section explains both aspects as implemented in the baseline pipeline.

3.2.1 Semantic Embedding with all-MiniLM-L6-v2

At the core of semantic search lies the **all-MiniLM-L6-v2 model**—a compact yet powerful transformer architecture. This model consists of six transformer encoder layers and roughly 22.7 million parameters. It is specifically optimized to generate fixed-length, 384-dimensional vector embeddings for sentences and short paragraphs, efficiently capturing their semantic content.

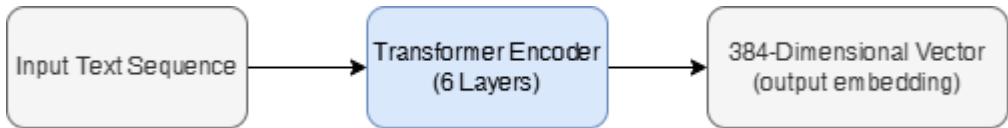


Figure 3: all-MiniLM-L6-v2 Model Architecture

The model is widely recognized for its ability to encode complex meaning into a small vector, making it ideal for large-scale semantic search. The resulting embeddings are robust to variations in wording and can be compared using simple distance metrics.

When a text chunk is processed by all-MiniLM-L6-v2, the output is a dense vector that can be directly compared with embeddings of other chunks or queries. The first few components of such vectors for sample chunks are shown in Table 2:

Chunk #	Chunk Preview	Embedding (First 8 Dims)
1	“Entropy is a measure...”	[0.022, -0.108, 0.176, ...]
2	“Regularization helps...”	[0.073, 0.029, 0.109, ...]

Table 2: Embedding Vectors for Text Chunks

3.2.2 Efficient Retrieval with FAISS

A central challenge in large-scale question answering is finding the most relevant text chunks for any given query—quickly and accurately—among thousands of candidates. This is where **FAISS (Facebook AI Similarity Search)** becomes essential. Developed by Meta AI, FAISS is an open-source library purpose-built for fast similarity search and clustering of dense vectors, making it especially suitable for natural language processing and semantic search applications.

FAISS excels at handling high-dimensional embeddings, such as those produced by neural language models, and can scale to collections containing millions of vectors. It offers a range of indexing strategies, from brute-force (exact) search to highly efficient approximate methods, enabling the system designer to balance accuracy and latency as needed. For academic QA, FAISS is typically used with normalized embeddings and inner product (cosine similarity) search, ensuring that semantically similar text chunks are identified regardless of their original scale or distribution.

Several factors make FAISS the preferred choice for this system:

- **Speed and Scalability:** FAISS supports real-time search, delivering sub-second responses even as the academic corpus expands.
- **Flexibility:** It provides multiple search algorithms suitable for both small and very large datasets.
- **Ease of Integration:** FAISS is compatible with major machine learning environments and is easy to deploy in Python-based pipelines.
- **Reliability:** As a widely adopted open-source project, FAISS is robust and production-ready.

In this pipeline, all chunk embeddings are added to a FAISS index configured for inner product search. When a user question arrives, it is embedded using the same model and submitted to FAISS, which instantly returns the top-k most semantically similar chunks from the entire collection.

The effectiveness of FAISS retrieval is illustrated by the sample queries and their best-matched chunks in Table 3:

Query	Top Chunk (Preview)	Similarity Score
What is entropy in ML?	Entropy is a measure of uncertainty...	0.87
How do we prevent overfitting?	Regularization helps prevent...	0.83

Table 3: FAISS Retrieval Results for Sample Queries

By leveraging FAISS, the retrieval module ensures that user queries are rapidly and accurately matched with the most relevant academic content, providing a robust foundation for high-quality answer extraction downstream.

3.2.3 Implementation Workflow: Embedding and Retrieval

The embedding and retrieval components of the baseline question answering system are designed to operate in a streamlined, reproducible workflow. This architecture ensures that academic materials are consistently processed and indexed, while queries are efficiently matched to relevant content—regardless of scale or document format.

At the implementation level, the workflow begins with the collection of pre-processed text chunks obtained from various academic documents. Each chunk is systematically fed into the all-MiniLM-L6-v2 embedding model, resulting in a dense vector representation that encapsulates its semantic meaning. These vectors are normalized and added to a FAISS index, forming the searchable database.

When a user submits a question, it undergoes the same embedding process, producing a query vector in the same high-dimensional space. This query embedding is then used to search the FAISS index, which rapidly returns the top-k most semantically similar document chunks. These retrieved chunks are passed to the downstream answer extraction stage.

This modular pipeline not only ensures high performance and scalability, but also supports easy extension and debugging—each stage is logically separated and can be independently evaluated.

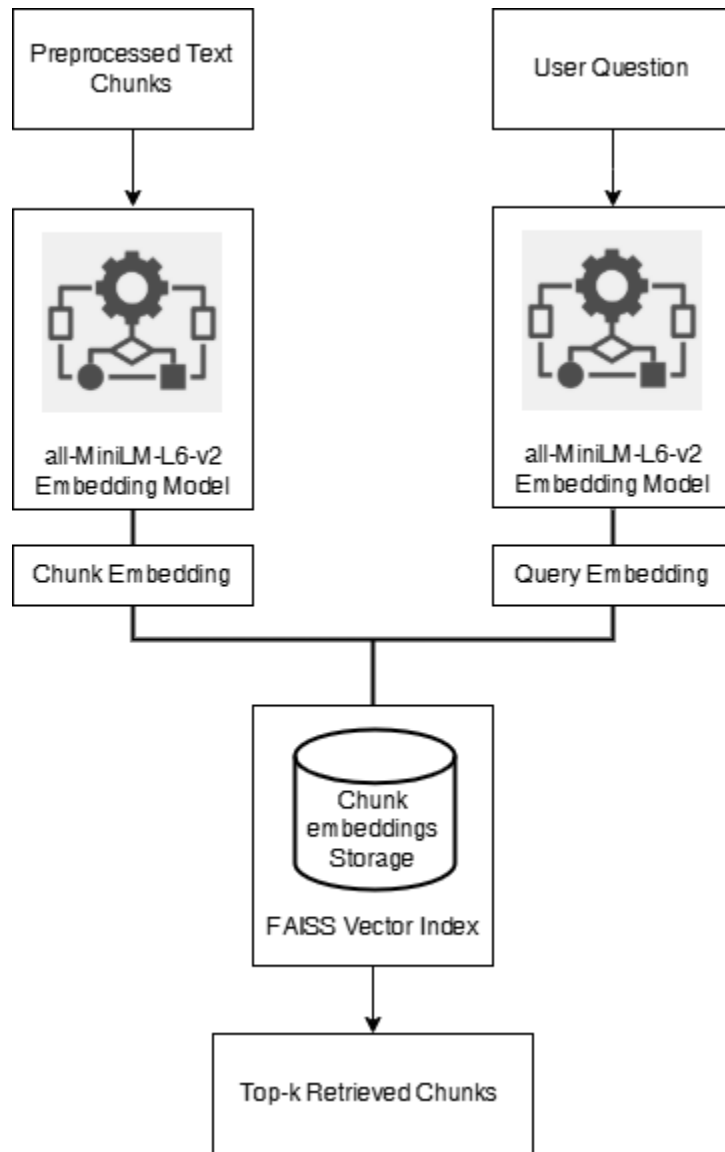


Figure 4: Implementation Flow of Embedding and Retrieval Pipeline

This diagram depicts the system’s operational steps, from input documents through embedding, indexing, and retrieval, with clear indication of data flow between each stage.

3.3 Reader Module

The final step in the baseline pipeline is answer extraction—identifying and presenting the most relevant, concise response to a user’s question. This is accomplished through a two-stage approach: first, a state-of-the-art BERT-based reader is applied to the top retrieved text chunks, and second, the system employs smart keyword-based heuristics and fall-back strategies to maximize answer reliability, even when the direct extractive model is uncertain.

3.3.1 Answer Extraction with BERT QA Model

The reader module in this system is built upon the **bert-large-uncased-whole-word-masking-finetuned-squad** model, one of the most robust and widely adopted architectures for extractive question answering. BERT (Bidirectional Encoder Representations from Transformers) is a deep transformer-based language model designed to understand context in natural language by processing words in both directions—left-to-right and right-to-left—within a sentence.

The "**bert-large-uncased-whole-word-masking-finetuned-squad**" variant is a large-scale model featuring 24 transformer encoder layers and over 340 million parameters. It is pre-trained on a vast English corpus and further fine-tuned on the SQuAD (Stanford Question Answering Dataset), specializing it for reading comprehension and QA tasks. The “whole word masking” technique during training enables BERT to develop a more nuanced sense of word boundaries and contextual relationships, improving answer extraction accuracy.

In this pipeline, when a user submits a question, it is paired with the top retrieved chunk from the document collection. The BERT QA model receives both as input and processes them through its deep stack of transformer layers, leveraging multi-head self-attention mechanisms to model complex contextual dependencies. The model then outputs the most probable start and end positions for the answer span within the context, along with a confidence score for its prediction.

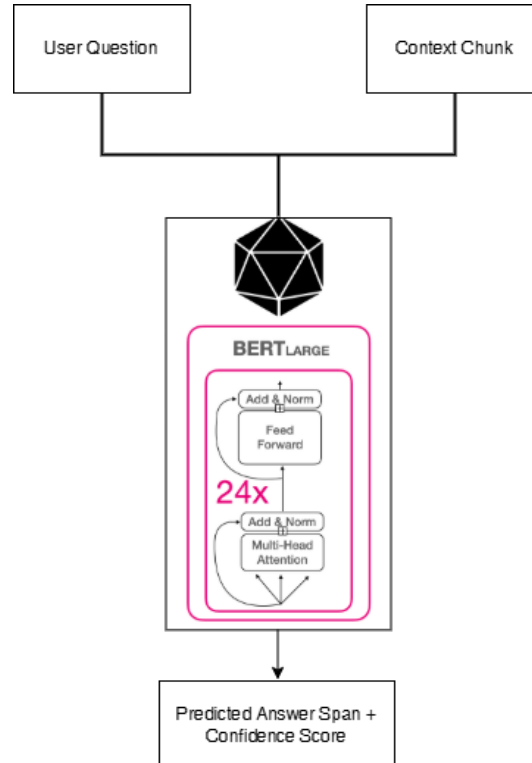


Figure 5: BERT-Large SQuAD QA Pipeline Architecture

The pipeline’s workflow can be summarized as:

- **Input:** User question and top-ranked context chunk.
- **Model:** “bert-large-uncased-whole-word-masking-finetuned-squad” processes through multiple self-attention layers.
- **Output:** Predicted answer span within the context, with an associated confidence score.

This approach ensures that the system can accurately extract answers directly from text, particularly when the answer is explicitly stated in the context and closely matches the phrasing of the question. The result is a high-precision, explainable answer that can be traced to a specific document segment.

3.3.2 Handling Edge Cases: Keyword and Fall-back Heuristics

While transformer-based models like BERT excel at extracting answer spans when the context is explicit and well-matched, real-world academic handouts can present cases where answers are implicit, phrased differently, or distributed across multiple sentences. To ensure robustness, the pipeline incorporates a two-stage backup mechanism: **global keyword matching** and a final **fall-back heuristic**.

In the first backup stage, the system employs a keyword overlap strategy. When the BERT QA model’s output does not meet minimum standards for answer length or confidence, the system scans the Top-K retrieved chunks for sentences sharing key non-stop word terms with the user’s question. By tallying the intersection of content words between the question and each candidate sentence, the method surfaces sentences that, while perhaps not extracted by the neural model, are likely to contain directly relevant information.

If this keyword matching still fails to yield a suitable answer, the pipeline defaults to a straightforward fall-back: presenting the first sentence of the top-ranked chunk. This ensures the user always receives a response grounded in the most relevant context available, even if the answer is only partially matched.

This layered approach—moving from neural extraction, to keyword scoring, to context-based fall-back—ensures answer delivery is both robust and explainable. It also supports transparency for users and evaluators, as each answer can be traced to the logic or source that produced it.

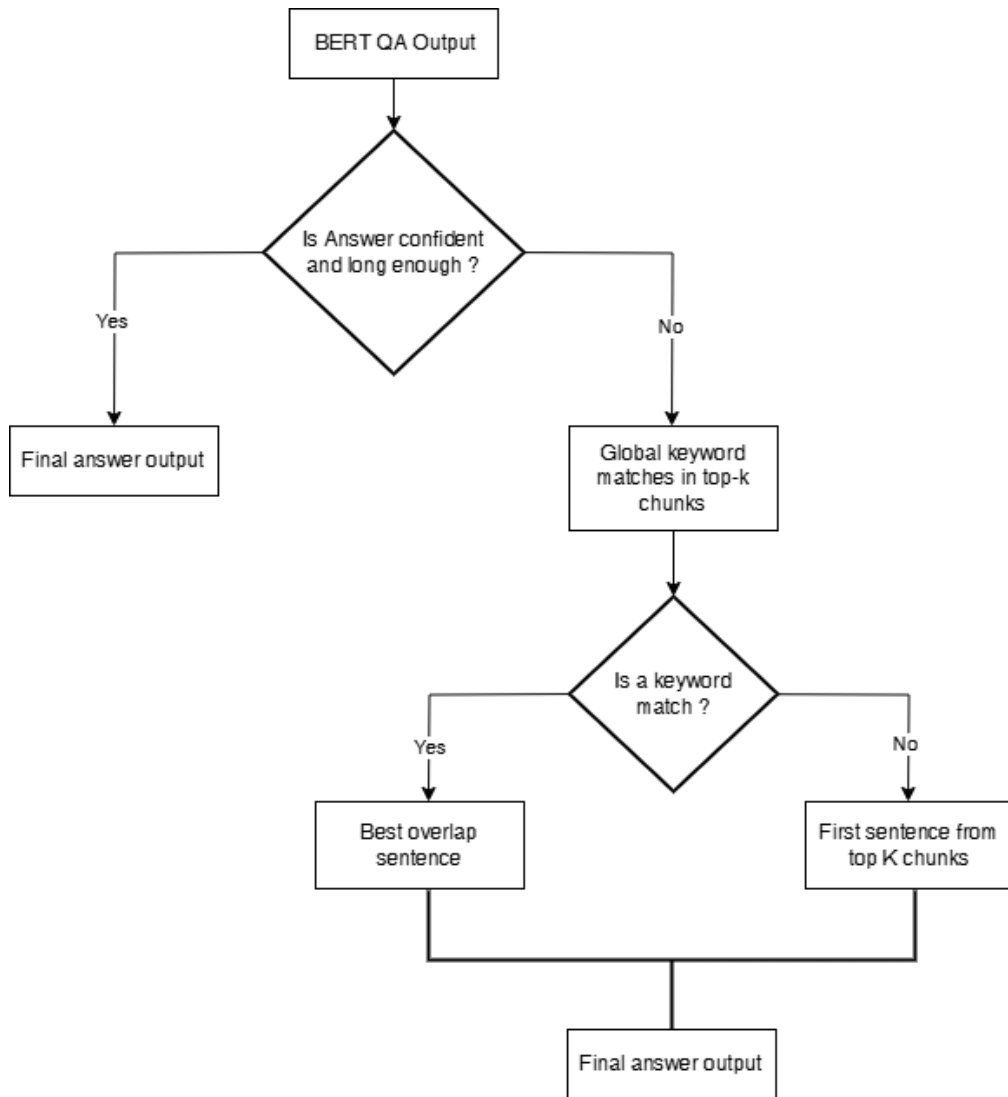


Figure 6: Decision Flow for Reader Module Edge Cases

To illustrate the decision process and variety of answer sources, Table 4 shows example queries with the answer produced, the method used, and any confidence or keyword overlap scores as available.

Query	Answer	Method Used	Score/Type
What is overfitting?	"Overfitting occurs when a model..."	BERT QA	0.93
What are optimization methods?	"Gradient descent is a commonly..."	Keyword Match	Overlap: 3 words
Explain variance in ML.	"Variance is the amount by which..."	Fallback	n/a

Table 4: Example of Answer Sources and Types in Edge Cases

This multi-tiered reader module design ensures high answer reliability, maximizing the chance of providing a relevant, complete response for a broad spectrum of academic questions.

3.4 Observations & Limitations

Evaluation of the baseline retrieval-augmented question answering pipeline highlights several practical strengths and key limitations in handling complex academic documents.

Key Observations:

- The combination of **hybrid chunking** (merging and splitting paragraphs/sentences by length), dense embeddings (all-MiniLM-L6-v2), and FAISS-based retrieval enables the system to surface relevant passages, even when query phrasing differs from the source.
- The BERT-large SQuAD reader shows strong accuracy for fact-based questions where answers are clearly stated within the top retrieved chunk.
- Keyword-based and fall-back heuristics provide robustness, ensuring the system delivers some answer even when the neural model's confidence is low.

Limitations and Challenges:

- **Context Fragmentation:** Only one chunk is used for answer extraction, so questions requiring information across multiple sections or paragraphs are not well-supported.
- **Extractive-only Answers:** The pipeline can only extract literal spans; it cannot synthesize or rephrase for explanatory or multi-part queries.
- **Chunking Strategy Sensitivity:** The effectiveness of retrieval and answer extraction depends heavily on how chunks are formed; improper chunk size or grouping may lead to lost context or irrelevant content.
- **Implicit and Scattered Answers:** When answers are implied or distributed across sentences, the system may fail to retrieve a satisfactory result.
- **Resource Scalability:** As the academic corpus grows, embedding and search operations become more demanding, even with FAISS optimizations.
- **Model Limitations:** Answer quality can be limited by the generalization and vocabulary coverage of the pre-trained models, particularly in specialized subject areas.

Chapter 4

Modern RAG System with LLMs & Ecosystem Tools

Recent advancements in large language models (LLMs) and the open-source NLP ecosystem have greatly enhanced the potential of academic question-answering systems. This chapter presents a fully implemented **modular Retrieval-Augmented Generation (RAG) pipeline**, deployed with an integrated **React frontend** and **Python FastAPI backend**, and supporting multiple LLM backends including **Gemini**, **Groq**, and the locally deployable **Ollama** for on-premises inference.

The system incorporates **advanced document ingestion and parsing with LlamaParse**, **context-aware semantic chunking**, **high-performance embeddings with persistent ChromaDB storage**, and a **feedback-aware retrieval mechanism** that continuously improves answer quality. It has been designed for high scalability, low-latency responses, and transparent provenance, ensuring that every generated answer is both contextually relevant and traceable to its source. This chapter details each stage of the pipeline, connecting user-facing interactions with backend processing logic, and illustrating how the system delivers accurate, flexible, and explainable answers for academic use.

4.1 Motivation for Advancing the Pipeline

While the traditional retrieval-augmented QA pipeline provides a strong foundation for academic search, its limitations become increasingly apparent as the complexity and scale of educational content grow. Three primary challenges drive the motivation for a modernized, LLM-centric architecture:

1. Scalability and Flexibility:

The classic pipeline relies on static chunking and embedding workflows, which become cumbersome as the volume and diversity of course materials increase. Adapting to new document formats, custom chunking logic, or the addition of new data sources often requires manual intervention or pipeline reconfiguration.

2. Limitations of Extractive QA:

BERT-based models, though powerful for extractive tasks, are inherently limited to returning verbatim spans from retrieved chunks. They struggle with open-ended, generative, or multi-hop questions, and cannot synthesize information from across multiple passages.

3. Advances in LLMs and Tooling:

The emergence of highly capable large language models—such as Gemini, Groq, and locally-deployable models via Ollama—enables true generative QA, where answers can be synthesized, summarized, or reworded as needed. At the same time, ecosystem tools like LangChain and ChromaDB offer modular, extensible building blocks for robust retrieval,

prompt chaining, and integration with multiple LLMs, making the system easier to scale and maintain.

These factors collectively motivate the transition to a **modular, LLM-driven RAG architecture**—one designed to overcome the limitations of the traditional baseline, adapt to diverse academic domains, and deliver context-aware, high-quality answers at scale.

4.2 New Architecture Overview

The modern Retrieval-Augmented Generation (RAG) pipeline is composed of tightly integrated, modular components designed to efficiently process academic documents and answer natural-language queries with both precision and traceability. The system brings together cutting-edge open-source tooling, robust vector storage, and multi-LLM orchestration to deliver a responsive and context-aware question-answering experience.

The key components include:

- **Document Ingestion Module:**

Allows users to upload academic handouts in .pdf or .docx format. The uploaded documents are tagged with subject metadata and passed through a preprocessing pipeline for parsing and segmentation.

- **Parsing Engine (Standard + LlamaParse):**

Handles both conventional single-column documents and structurally complex ones, such as multi-column layouts, tables, and mathematical expressions. LlamaParse is selectively applied based on file structure to retain fidelity in academic semantics.

- **Semantic Chunking & Embedding (via BGE + FastEmbed):**

The parsed content is segmented into context-aware chunks, preserving logical boundaries like section headers and formula explanations. Each chunk is embedded using the BAAI/bge-base-en-v1.5 model via FastEmbed to produce dense, semantically rich vectors.

- **Vector Store (ChromaDB):**

All chunk embeddings are stored in a persistent ChromaDB instance, allowing high-throughput semantic search over large and growing document sets. The store supports metadata tagging (subject, page number, etc.) for fine-grained retrieval control.

- **User Interaction Layer (Query Input + LLM Selector):**

Users submit their question through the UI, selecting the academic subject and choosing their preferred LLM backend (Gemini, Groq, or Ollama). This selection dynamically configures the answer generation pipeline.

- **Semantic Retriever (LangChain Orchestrated):**

The user query is embedded and compared against the stored vectors in ChromaDB to fetch the top-k most relevant chunks. Context expansion ensures adjacent content is included to improve grounding.

- **LLM Answer Generator (Gemini / Groq / Ollama):**

The retrieved chunks and question are compiled into a prompt and passed to the selected LLM backend. Gemini and Groq offer cloud-hosted generative capabilities, while Ollama enables local inference—ideal for privacy-conscious deployments.

- **Answer Polishing & Provenance (spaCy):**

The raw LLM output is refined for readability and conciseness using spaCy's NLP tools. Metadata such as document name and page number are attached for transparency.

- **UI Display + Feedback Capture:**

The final answer, along with expandable supporting context and citations, is presented in the frontend. Users may provide binary feedback (Helpful / Not Helpful) and optional comments, which are stored for analysis and pipeline improvement.

4.2.1 System Design and Data Flow

The enhanced system begins with a user submitting a natural-language query through the web interface, selecting the **subject domain** and optionally choosing the **preferred LLM backend (Gemini, Groq, or Ollama)**. The query is **embedded** and processed by the **semantic retrieval module**, which searches a **persistent ChromaDB vector store** populated with **embeddings** generated from ingested academic documents (**PDF/DOCX**). Retrieval incorporates **context expansion** and **metadata filtering** to ensure that relevant surrounding content—such as adjacent paragraphs, equations, or diagrams—is also included.

The most relevant document chunks are then passed into the **LLM orchestration layer**, where the selected backend generates an answer grounded in the retrieved context. The answer undergoes post-processing with **spaCy** for **linguistic polishing** and is annotated with **provenance metadata**, enabling users to trace it back to the original source material. Finally, the **polished answer**, supporting excerpts, and citations are presented in the UI, where the user can provide **binary feedback and optional comments**. This feedback is logged for evaluation and can be leveraged to **improve retrieval ranking** and **overall answer quality** in future interactions.

Please refer to Figure 7 below for a high-level overview of the modern modular RAG pipeline architecture, illustrating the flow of data and modular integration of each component.

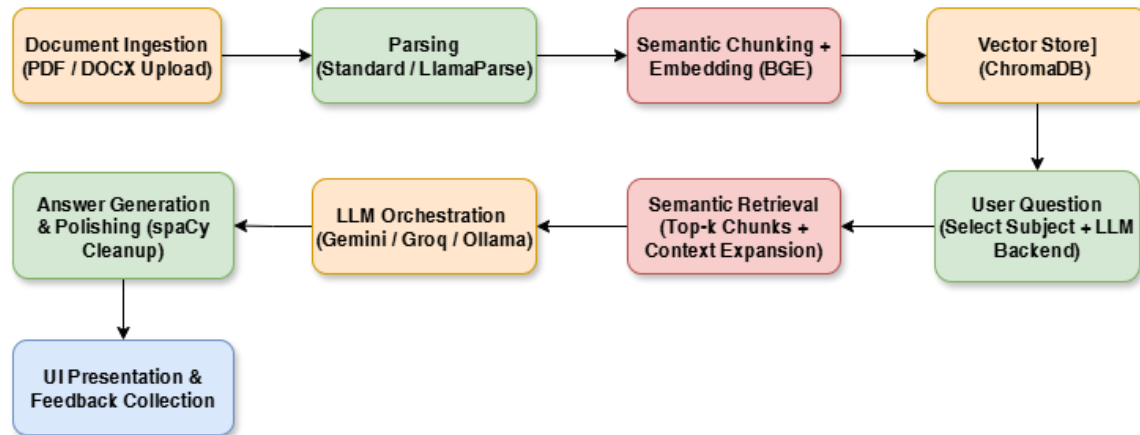


Figure 7: Overview of the modern modular RAG pipeline architecture

Table 5 below summarizes the key differences between the baseline and modern RAG architectures. The modern system offers improved scalability, flexibility, and answer quality through modular orchestration, persistent vector storage, and generative LLM capabilities, while also being future-ready for local LLM integration.

Feature	Baseline System	Modern RAG System
Orchestration	Hardcoded pipeline	LangChain modular flows
Vector Store	FAISS (in-memory)	ChromaDB (persistent)
Reader Model	BERT-large (extractive)	Gemini/Groq (LLM, generative), Ollama (Local)
Document Formats	DOCX, PDF (manual)	Multi-format, extensible
Chunking Strategy	Hybrid size-based	Pluggable/context-aware
Retrieval Flexibility	Top-k only	Multi-hop, hybrid
Scalability	Limited	High
Answer Style	Extractive	Generative/explanatory
Integration Ease	Low (manual code)	High (configurable)

Table 5: comparison of the baseline and modern RAG pipeline architectures

With this modular architecture in place, each stage of the pipeline—from document upload and parsing to semantic chunking, retrieval, LLM-based answer generation, and feedback capture—operates in a cohesive, extensible flow.

The subsequent sections explore each of these components in greater detail, highlighting their design, implementation, and role within the end-to-end academic question-answering system.

4.3 Modular Retrieval Layer: Upload to Semantic Retrieval

The retrieval backbone of the academic question-answering system is designed to transform raw educational documents into semantically searchable units—delivering context-rich content to downstream language models. This layer is responsible for everything from handling multi-format document uploads to generating dense vector embeddings and retrieving top-k relevant content chunks with high precision.

What distinguishes this system from conventional document search workflows is the integration of several specialized modules, including **LlamaParse** for structurally complex files, **semantic chunking** strategies tailored to academic writing, **FastEmbed-powered BGE embeddings**, and **ChromaDB**, a persistent high-performance vector store. These components work together in a tightly orchestrated pipeline that supports subject-specific indexing, advanced metadata tagging, and query-time filtering to maintain both scalability and academic relevance.

In the following subsections, we break down the entire retrieval pipeline into clearly defined stages—starting with how documents are uploaded and parsed, and concluding with how semantically relevant context is retrieved in response to a user query. Each component will be examined in terms of its theory, implementation role, and contributions to answer accuracy.

4.3.1 Document Upload & Subject Mapping

The document upload interface marks the entry point of the academic question-answering system, where users provide their subject-specific learning materials. The interface is designed to be intuitive and flexible, supporting documents in both **PDF** and **DOCX** formats—allowing a wide range of content including lecture notes, handouts, and technical reports.

Each upload begins with the user selecting a **subject domain** (e.g., *Machine Learning*, *Natural Language Processing*) and optionally supplying a brief description. These details are important not just for user organization, but also for how the backend stores and later retrieves the document.

Once a file is selected and submitted, the **frontend sends a structured request** to the backend upload API. This request contains:

- The selected subject
- The uploaded document (as FormData)
- A boolean flag indicating whether to use **LlamaParse**
- An optional description

On the backend, the file is saved inside a central directory named `/uploads/`. To ensure separation by subject, each file is **automatically prefixed with the subject name**, forming paths such as:

`/uploads/NLP_NLP_Sem2_Tables.docx`

`/uploads/MachineLearning_ML_Handout1.pdf`

This filename convention guarantees **subject-level isolation** during future retrieval and indexing, while also making it easy to trace the origin of any document.

Users also have the option to enable **advanced parsing** for structurally complex files. If the **LlamaParse toggle** is selected, this preference is recorded during upload and will later influence how the file is parsed when ingestion is triggered.

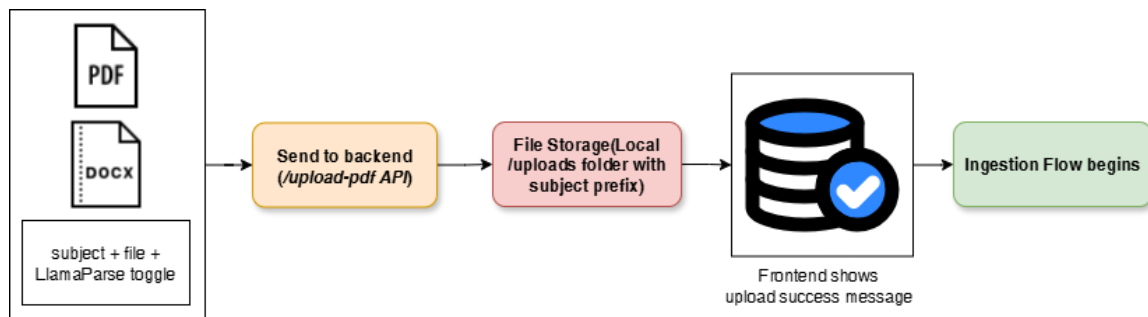


Figure 8: Document upload and ingestion trigger workflow

This flow captures the document ingestion entry point: users upload a file tagged by subject, optionally enabling LlamaParse. Once stored with a subject-prefixed name, the file becomes ready for ingestion through a separate backend trigger.

The **user interface** plays a crucial role in guiding this process. A clean upload panel allows the user to pick a subject, select files, and toggle advanced parsing. Upon successful upload, the UI triggers a backend ingestion pipeline and confirms upload status with appropriate notifications.

Upload Academic Document


Subject *

Machine Learning

Description

Machine Learning Introduction Document

☐ Advanced PDF Parsing (LlamaParse)



ML_course_content_1.pdf

CHOOSE FILE

UPLOAD

Figure 9: Academic document upload panel with optional advanced parsing toggle

Internally, the uploaded documents are indexed with associated metadata such as file name, parser used, and storage location. This information supports both traceability and chunk-level provenance in later stages. An example of this internal structure is outlined below:

Subject	File Name	Stored Path
Machine Learning	ML_course_content_1.pdf	/uploads/Machine Learning/ML_course_content_1.pdf
Natural Language Processing	NLP_Sem2_Tables.docx	/uploads/NLP/NLP_Sem2_Tables.docx

Table 6: Uploaded document storage record

4.3.2 Document Ingestion & Parsing

Once a document is successfully uploaded, the next step in the pipeline is ingestion—a process that transforms the raw file into a format suitable for retrieval-based question answering. This stage focuses on handling relatively clean academic documents (e.g., lecture notes, handouts) that do not require layout-aware parsing. The system extracts meaningful text from each page, associates it with metadata, and prepares it for downstream processing such as semantic chunking and embedding.

- **Triggering the Ingestion Process**

The ingestion process is initiated by the user via the web interface by clicking the “**Process**” or “**Ingest**” button shown after a successful file upload. This action sends a structured POST request to the backend ingestion API.

The request body includes three key fields:

1. **subject:** The subject domain selected by the user during upload (e.g., "Machine Learning")
2. **filename:** The name of the uploaded file (already prefixed with subject to ensure separation)
3. **use_llamaparse:** A boolean flag indicating whether advanced parsing should be used (set to false for standard parsing)

Upon receiving the request, the system first validates that the file exists in the /uploads/ directory. If valid, the system proceeds with standard parsing logic depending on the file type—**PDF** or **DOCX**.

- **File-Type-Specific Parsing Logic**

To support multiple document formats, the system dynamically selects the appropriate parser:

1. **PDFs** are processed using **PyPDFLoader**, which extracts text from each page while preserving academic formatting where possible.
2. **DOCX** files are handled using **Docx2txtLoader**, which flattens Word document content into a plain-text format suitable for further processing.

These loaders abstract away file-specific quirks and return a clean, page-wise representation of the content.

Only pages that contain a meaningful amount of text are retained for the next stages. Blank pages, metadata-only pages, or pages with unreadable formatting are skipped to ensure that downstream modules operate on quality data.

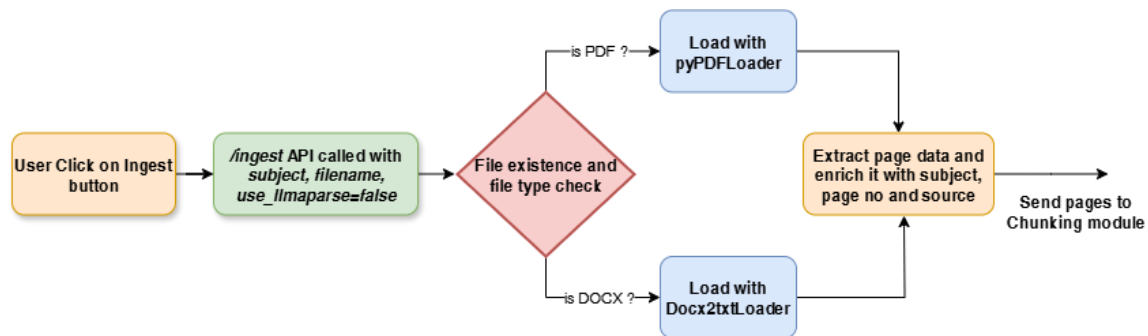


Figure 10: Flowchart for Standard Document Ingestion and Parsing

This diagram illustrates the backend flow triggered by the “Ingest” button on the UI. Based on file type, the system uses either PyPDFLoader or Docx2txtLoader for standard parsing and then prepares the text for semantic chunking.

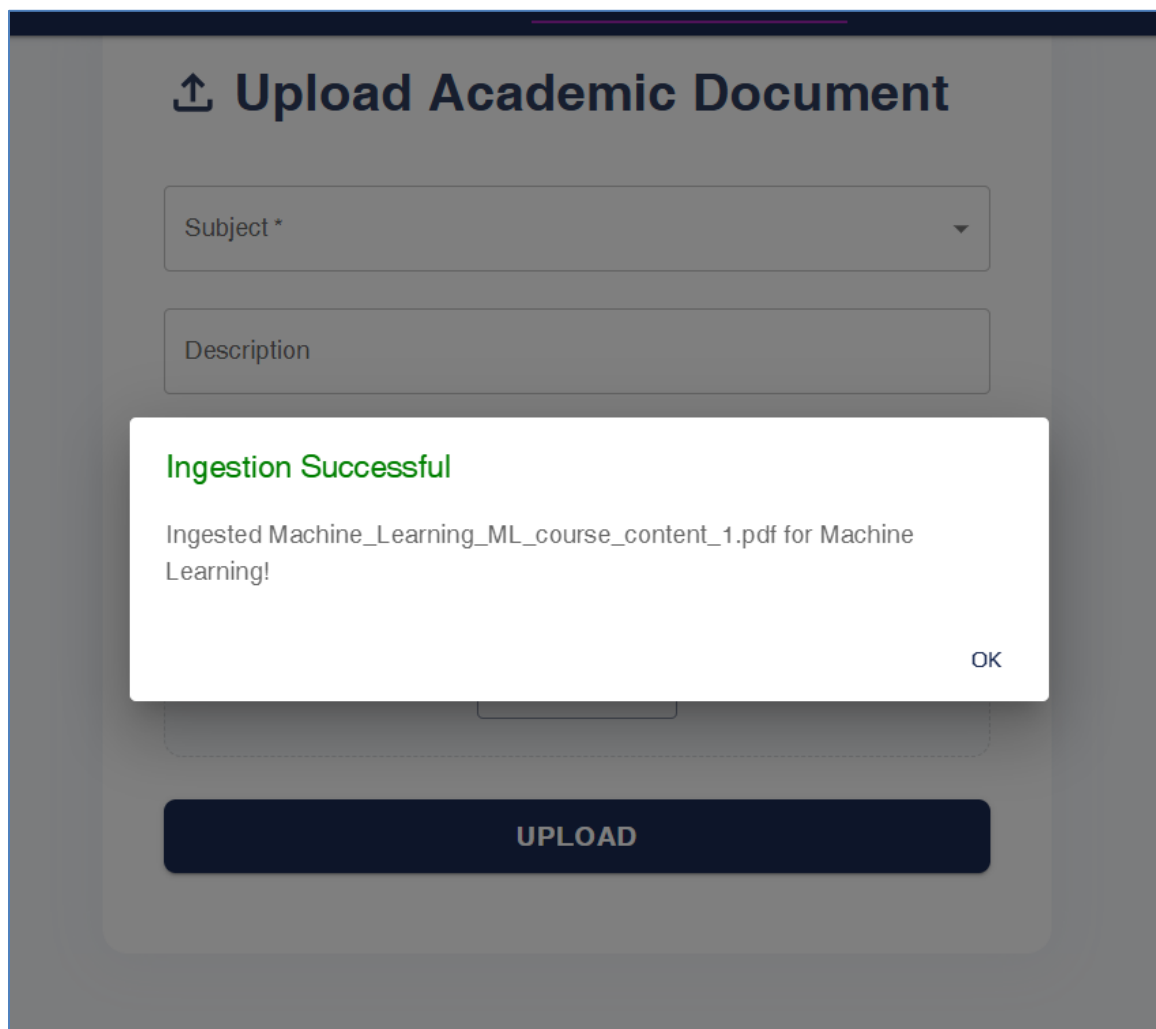


Figure 11: Confirmation message indicating successful document ingestion

The web interface makes this entire flow seamless for the user. After selecting a file and uploading it, a dedicated button allows them to begin processing the document. A backend notification confirms successful ingestion and transitions the system to the chunking phase.

Page No.	File Name	Subject	Parsing Method	Used in Chunking?
1	ML_course_content_1.pdf	Machine Learning	PyPDFLoader	Yes
2	ML_course_content_1.pdf	Machine Learning	PyPDFLoader	No
3	NLP_Sem2_Tables.docx	NLP	Docx2txtLoader	Yes

Table 7: Parsed documents with different Parsing Methods

Each parsed page is enriched with subject metadata and page number, enabling traceability and clean separation across domains. The system is now prepared to perform **semantic chunking**, where these pages are further divided into idea-preserving text units for intelligent indexing.

4.3.2.1 Advanced Parsing with LlamaParse

In conventional academic question-answering systems, PDF parsing is often handled through lightweight tools like PyPDFLoader, which operate on plain text extraction. However, educational documents frequently include complex layouts—multi-column text, embedded tables, mathematical equations, structured headers, and figures—which often break standard parsers. To address this, our system integrates **LlamaParse**, an LLM-powered document parser built for robust, structure-preserving extraction.

What is LlamaParse?

LlamaParse is a state-of-the-art document-parsing engine offered by **LlamaIndex**. It is designed to convert rich-format documents like PDFs, DOCX, PPTX, EPUB, and others into semantically structured Markdown or plain text. Its capabilities are powered by a generative LLM backend, allowing it to follow *custom parsing instructions* and preserve complex layouts such as:

- Tabular data
- Diagrams and figures
- Hierarchical sectioning (headings, subsections)
- Mathematical notations
- Page-level metadata

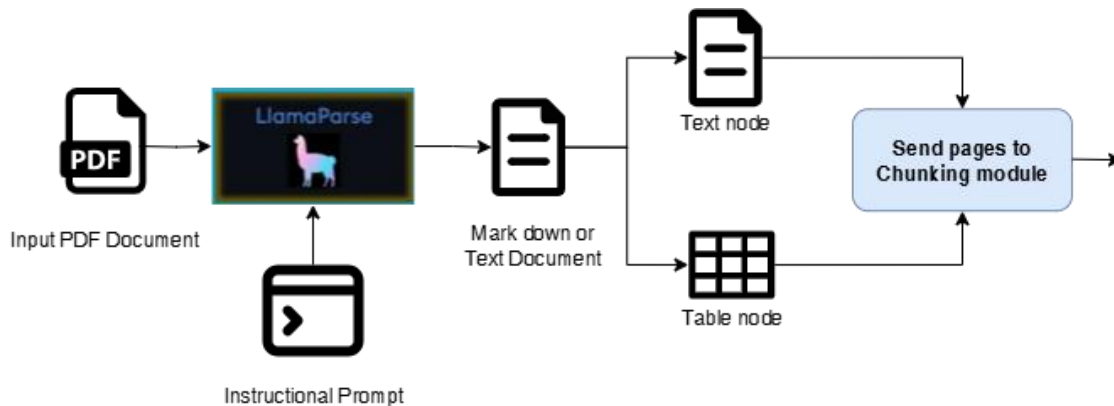


Figure 12: Ingest Complex Documents with LlamaParse

The primary role of **LlamaParse** is to simplify the creation of retrieval-based systems for complex documents such as PDFs. It accomplishes this by extracting the information contained within these documents and converting it into formats that are easier to process, such as markdown or plain text. Once transformed, this data can be embedded and seamlessly integrated into a Retrieval-Augmented Generation (RAG) pipeline.

Key Features of LlamaParse

- **Supported File Formats:** Works with a wide range of file types including PDF, .pptx, .docx, .rtf, .pages, .epub, and more.
- **Output Formats:** Produces outputs in markdown and plain text formats.
- **Extraction Capabilities:** Can handle text, tables, images, charts, comic book layouts, and even mathematical equations.
- **Customizable Parsing Instructions:** Being powered by an LLM, LlamaParse allows you to pass custom instructions, similar to prompting a language model. These instructions can be used to describe the document to improve context understanding, specify desired output formatting, or even request preprocessing during parsing—such as sentiment analysis, translation, summarization, and other NLP tasks.
- **JSON Mode:** Provides a fully structured JSON representation of the document, extracting images with associated size and position metadata, as well as tables in JSON for easier computational analysis. This is highly useful for advanced RAG implementations where document structure and metadata are essential for maximizing information value and enabling citation of the exact location of retrieved content.

The Advantage of Markdown Conversion

One of LlamaParse's notable strengths is its ability to transform PDF documents into markdown format. Markdown inherently preserves the document's structure by tagging key

components like headings, subheadings, tables, and images. While this may seem minor, it enables precise segmentation of a document into smaller, meaningful chunks using specialized LlamaIndex tools such as the *MarkdownElementNodeParser()*. By representing a PDF as markdown, each structural element can be isolated, extracted, and fed directly into the RAG pipeline for improved retrieval accuracy and contextual relevance.

Key Advantage:

Unlike standard extractors, LlamaParse uses the *Markdown format* to retain document hierarchy, which significantly improves downstream chunking, semantic search, and citation accuracy in RAG pipelines.

How We Use It in Our Pipeline

In our architecture, when a user uploads a document and enables the "**Advanced PDF Parsing (LlamaParse)**" toggle, the backend pipeline triggers LlamaParse for that specific file. The logic is configured via a Boolean flag (*use_llamaparse*) sent to the ingestion API.

Here is how the process unfolds:

1. Document Parsing Instruction:

The parser is initialized with a domain-specific *instructional prompt*—e.g., explaining the nature of the uploaded document (like “ML notes” or “research paper”), so it can align extraction with expected content semantics.

2. Structured Output:

LlamaParse converts the PDF into structured nodes (e.g., text, table, image), each enriched with metadata such as page number, chunk type, and position in the hierarchy.

3. Chunking Strategy:

Using LangChain’s **RecursiveCharacterTextSplitter**, we break down each parsed node into smaller semantic sub-chunks. This improves granularity in downstream retrieval.

4. Metadata Enrichment:

Each chunk is tagged with fields like:

- *source_pdf*
- *parsed_by* (marked as "*llamaparse*")
- *parent_chunk_type* (e.g., Table, Text)

- *chunk_uid* (hashed ID for traceability)

This metadata later powers **feedback-based re-ranking**, document traceability, and UI-level provenance display.

Below is a simplified block-style flow representation showing what happens under the hood when advanced parsing is enabled:

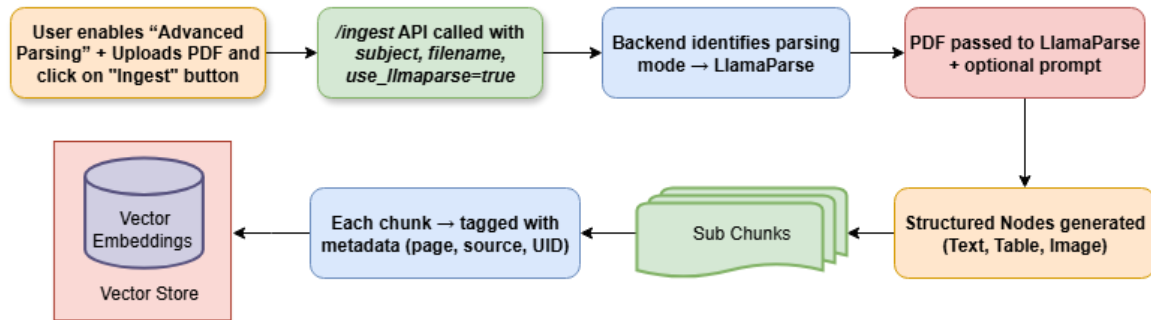


Figure 13: Enhanced ingestion pipeline with LlamaParse

Figure 13 illustrates the enhanced ingestion pipeline when **advanced LlamaParse-based parsing** is enabled. Upon upload, the system routes the document through a parsing mode switch, invoking LlamaParse to extract structured nodes such as text, tables, and images. These nodes are then semantically chunked, enriched with metadata (including page number, document source, and chunk UID), and finally stored as vector embeddings in the persistent vector store (ChromaDB). This structured approach ensures high-fidelity semantic indexing and transparent traceability of all downstream answers.

Sample Instructional Prompt:

"The provided document is a quarterly report filed by Uber Technologies, Inc. with the SEC...
It contains many tables. Try to be precise while answering the questions."

This prompt allows LlamaParse to better interpret the domain, preserve semantic context, and assist in aligning the document layout with our RAG pipeline's goals. Below screenshot of successfully parsed PDF available on llamaIndex Cloud:

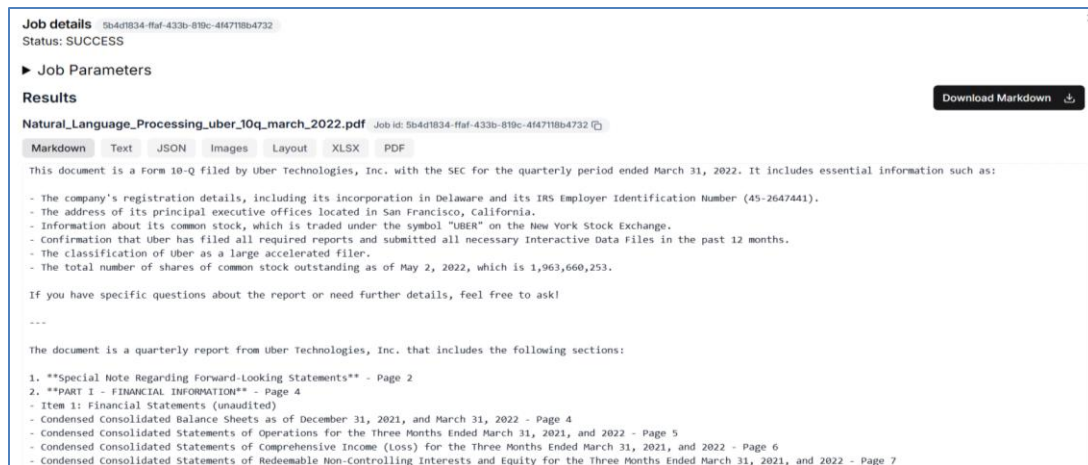


Figure 14: LLmaParsed Document available on cloud

4.3.3 Semantic Chunking of Academic Documents

Semantic chunking is a transformative step that goes beyond basic length-based splitting or hybrid chunking. Rather than merely dividing text by word count or paragraph boundaries, semantic chunking aims to segment documents into coherent, meaning-rich passages that align with natural topic shifts and content boundaries. This ensures that each chunk is not only manageable in size but also maximally relevant for retrieval and answer generation.

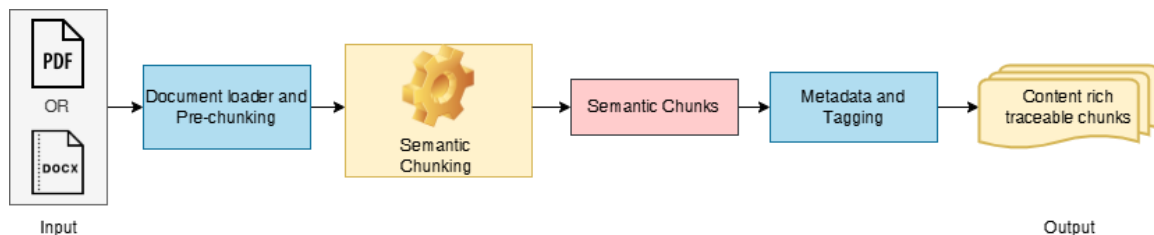


Figure 15: Semantic Chunking Workflow

In practice, the pipeline leverages **LangChain's SemanticChunker**, which utilizes a pre-trained embedding model to identify content boundaries based on semantic coherence rather than superficial markers. Each document is initially pre-chunked—often at the page or paragraph level—to simplify processing. The SemanticChunker then analyses these segments, using statistical or percentile-based thresholds on embedding similarity to determine where one idea ends and another begins.

The goal of semantic chunking is to maintain the structural and logical integrity of academic documents, preserving grouping between **headings and their corresponding explanations**, **formulas with descriptions**, and **tables with associated analysis**. This human-aligned segmentation ensures that each chunk makes sense independently while being optimized for retrieval.

This approach produces chunks that more closely match the way humans naturally navigate and reference academic material, leading to higher retrieval precision. Every chunk created is enriched with essential metadata to support **traceability, explainability, and feedback-aware retrieval**. These include:

Field	Description
source_pdf	The original file name
page	Page number (for both PDF and DOCX)
subject	Subject domain selected by user
chunk_uid	Unique hash of chunk content

Table 8: Metadata for semantically chunked document

A sample output from the semantic chunking stage is illustrated below:

Chunk_uid	Source_pdf	Chunk Text Preview	Page
Machine_Learning_ML_course_content_1.pdf_3_d41ab0c54d	Machine_Learning_ML_course_content_1.pdf	1.5 Challenges in Machine Learning Despite its power, deploying...	2
Machine_Learning_ML_course_content_1.pdf_3_3f1169e00e	Machine_Learning_ML_course_content_1.pdf	Recall (Sensitivity): Out of all actual positives, how many were correctly predicted...	7
Machine_Learning_ML_course_content_2.pdf_1_53c38d0b0f	Machine_Learning_ML_course_content_2.pdf	to the observed data, provided there are enough relevant examples in memory...	1

Table 9: Example of Semantic Chunk Metadata

4.3.4 Advanced Embedding with FastEmbed (BAAI/bge-base-en-v1.5)

A pivotal step in the modern retriever pipeline is converting each semantic chunk into a vector that accurately represents its meaning for semantic search. This process, known as **embedding**, is essential for allowing the system to retrieve content based on conceptual similarity, rather than just matching keywords.

In this work, the **FastEmbed** framework, utilizing the **BAAI/bge-base-en-v1.5** model, is employed for this purpose. This state-of-the-art model produces 768-dimensional embeddings, designed to capture the core semantics of sentences and paragraphs—making it highly effective for academic QA over diverse content.

Before delving deeper, consider the flow illustrated in Figure 9 below:

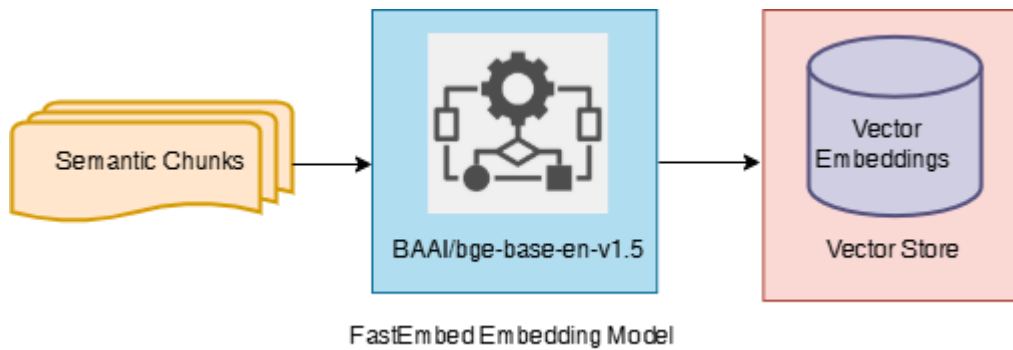


Figure 16: Embedding Workflow for Semantic Chunks

The process begins with the semantic chunks produced in the previous stage. Each chunk enters the embedding model, depicted in the center box. The FastEmbed (BAAI/bge-base-en-v1.5) model applies deep language understanding to encode the full meaning of the text—not just the words used—into a 768-dimensional vector. This is shown in the diagram by the downward arrow from "Semantic Chunks" into the "Embedding Model" box. The resulting output, a dense vector, is directed into the "Vector Embeddings" box, representing a collection of these embeddings for all chunks in the academic corpus.

This visual flow captures the essence of the embedding phase: it bridges raw, meaning-rich text with a mathematical space optimized for rapid, context-sensitive retrieval. By mapping all document chunks in this way, the system can efficiently compare user queries to the most semantically relevant passages, regardless of surface phrasing or document structure.

A sample of the generated embeddings is shown below:

Chunk_uid	Source_pdf	Chunk Preview	Embedding (First 8 Dims)
Machine_Learning_ML_course_content_1.pdf_3_d41ab0c54d	Machine_Learning_ML_course_content_1.pdf	1.5 Challenges in Machine Learning Despite its power, deploying...	0.071, -0.021, 0.122, 0.043, -0.055, 0.108, 0.023, -0.037
Machine_Learning_ML_course_content_1.pdf_3_3f1169e00e	Machine_Learning_ML_course_content_1.pdf	Recall (Sensitivity): Out of all actual positives, how many were correctly predicted...	0.091, -0.014, 0.113, 0.037, -0.051, 0.104, 0.021, -0.029

Table 10: Example Embeddings for Semantic Chunks

4.3.5 Persistent Vector Storage with ChromaDB

Once each semantic chunk is transformed into a dense vector using the FastEmbed framework, the system must store, index, and retrieve these embeddings efficiently. For this, the system employs **ChromaDB**, a lightweight yet production-ready vector database optimized for **fast semantic search** and **persistent storage**.

ChromaDB is designed to handle large-scale academic corpora, supporting rapid insertion, querying, and deletion of high-dimensional vectors. Its persistence feature allows the vector index to be saved and reloaded from disk, ensuring that the system can handle evolving document sets without the need to recompute all embeddings at each startup. This makes the pipeline suitable for real-world, production-scale deployments in educational settings.

Figure 17 below depicts the flow of data and vectors into ChromaDB:

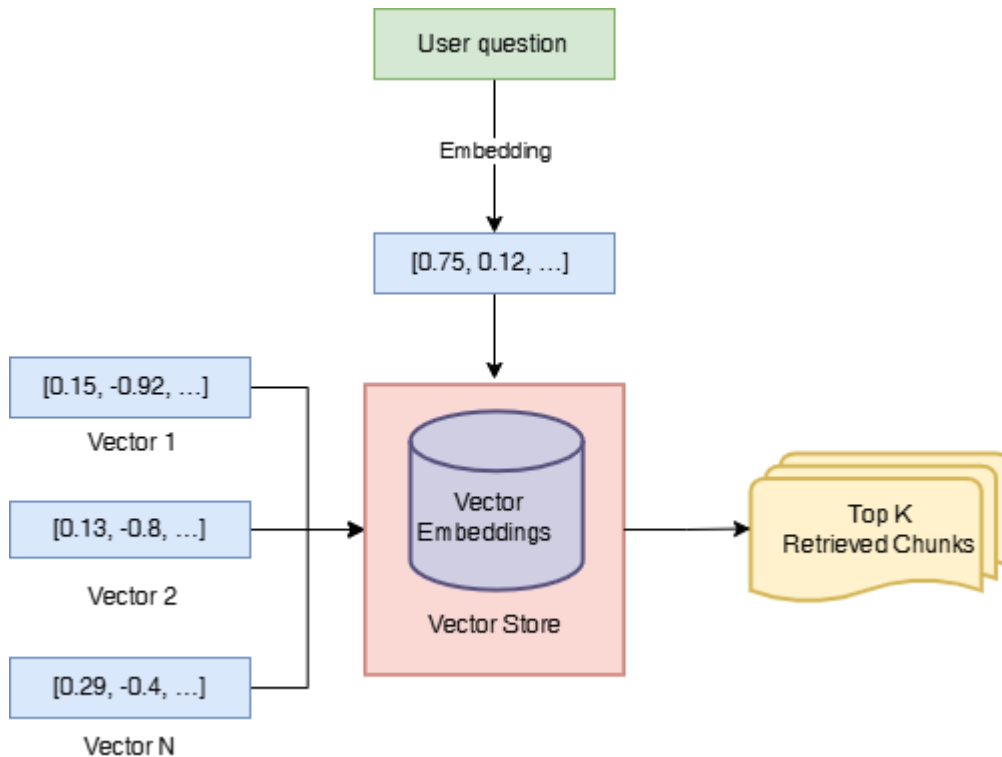


Figure 17: Document and query embeddings into ChromaDB

In this workflow, all chunk embeddings are stored in ChromaDB (center box). When a user submits a question, it is passed through the same embedding model and the resulting query vector is sent to ChromaDB. The vector store rapidly computes similarity scores and returns the top-k most similar document chunks for downstream LLM-based answer generation. This design not only speeds up retrieval, but also ensures persistence—vectors and metadata can be stored across sessions, and the system can scale to tens of thousands of documents.

During ingestion, the system uses **Chroma.from_documents()** to store all vectorized chunks. At retrieval time, **Chroma()** is reloaded using the same directory path. This ensures session persistence and allows incremental document ingestion without loss of prior index data.

Table 11 below provides an example of the indexed data for a few chunks:

Chunk ID	Source Document	Metadata (Page)	Vector Store Path
001	lecture1.pdf	4	/outputs/chroma_machine_learning.
002	lecture2.pdf	7	/outputs/chroma_machine_learning

Table 11: Example Entries in ChromaDB Vector Store

4.3.6 Retriever Object Creation from the Vector Store

Once all semantic chunks are embedded and stored in ChromaDB, the system initializes a standardized retriever object to act as the interface for semantic search. This retriever efficiently surfaces the most relevant passages for search queries or downstream answer generation, ensuring fast and targeted access to stored knowledge.

The retriever connects the persistent vector store of indexed chunks to a top-k similarity search mechanism, enabling precise retrieval. Built on LangChain's abstraction, it offers a modular design that allows easy tuning of retrieval parameters and integration of additional functionality as the pipeline evolves.

This design decouples retrieval logic from vector storage, allowing flexible parameter tuning such as:

- Number of chunks (top_k) to retrieve
- Embedding model variations
- Integration with feedback re-ranking

This process is illustrated below:



Figure 18: Construction of a retriever object from the ChromaDB vector index

By encapsulating the retrieval logic in this way, the modern RAG system enables robust, context-aware selection of document passages assuring that the most relevant academic content is efficiently delivered to the LLM-based answer generation module.

This encapsulation ensures that when a user poses a question, the system efficiently identifies and forwards only the most semantically aligned passages—forming the bridge between storage and reasoning.

Chapter 5

Reader Module and User Interaction Layer

The **Reader Module** is the final stage in the pipeline where user questions meet meaningful answers. It connects retrieved context chunks to large language models (LLMs), performs structured prompt construction, invokes LLMs for response generation, and then presents the answers to the end user in a clean, interactive interface.

This module is not just about generating responses — it is about delivering **grounded**, **context-aware**, and **traceable** answers, leveraging powerful LLM backends and clean UX workflows. The system supports multiple backends, including:

- **Google Gemini (gemini-1.5-flash-latest)** via Gemini API
- **Llama 3 (llama3-8b-8192)** hosted on **Groq** for lightning-fast inference
- **Ollama-compatible local LLMs**

Alongside answer generation, the Reader module also ensures:

- Injection of **retrieved context** into the prompt
- Display of **retrieval metadata** (source document, page, rank)
- **Feedback collection** from users for future ranking improvements

This chapter explains the complete reader pipeline — from query submission to answer presentation — including API orchestration, prompt formatting, LLM execution, and result visualization.

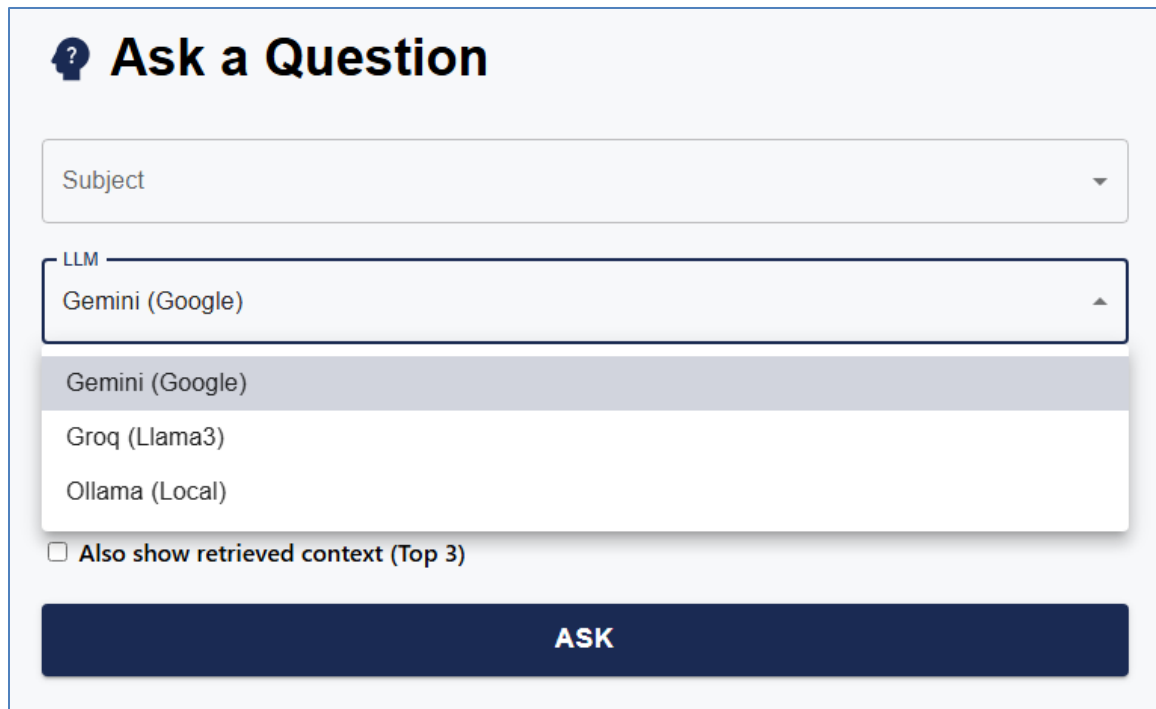
5.1 Multi-LLM Answer Generation and Orchestration

The Reader module is responsible for transforming retrieved chunks into final, polished answers using large language models (LLMs). A key enhancement in this work is the support for **multiple LLM backends**, offering deployment flexibility, performance tuning, and privacy control.

Rather than coupling the system to a single model, the architecture allows seamless switching between **Google Gemini**, **Llama 3 via Groq**, and **locally hosted LLMs via Ollama**. This design empowers users and developers to optimize for latency, cost, and deployment environment — whether cloud-based or offline.

UI Interaction

On the question-answering interface, users can choose their preferred LLM via a dropdown menu. Once a model is selected, the system routes the query through the corresponding backend and returns the generated response along with relevant context.

The image shows a web interface titled "Ask a Question" with a question mark icon. Below the title is a "Subject" dropdown menu. Underneath is an "LLM" selection menu currently showing "Gemini (Google)". A dropdown menu is open, listing "Gemini (Google)", "Groq (Llama3)", and "Ollama (Local)". Below the LLM menu is a checkbox labeled "Also show retrieved context (Top 3)". At the bottom is a large dark blue button labeled "ASK".

Ask a Question

Subject

LLM

Gemini (Google)

Gemini (Google)

Groq (Llama3)

Ollama (Local)

☐ Also show retrieved context (Top 3)

ASK

Figure 19: LLM selection menu on ask question page

5.1.1 Overview of Each LLM

1. Google Gemini (gemini-1.5-flash-latest)

Gemini is Google’s cutting-edge transformer-based LLM known for its **robust context handling** and **scalable prompt processing**. The variant integrated here — **Gemini 1.5 Flash** — is optimized for speed and cost, making it suitable for real-time academic question answering.

- Supports **large context windows** (~1M tokens max, ~32k typical)
- Handles **structured academic prompts** effectively
- Hosted via Google’s official Generative AI API

Gemini excels at answering factual questions by extracting key ideas from the prompt and delivering concise yet informative responses.

2. LLaMA 3 (8B, via Groq)

LLaMA 3 is a state-of-the-art open-weight LLM by Meta, and it's hosted on Groq's **ultra-low-latency inference platform**. This integration offers powerful academic performance with near-instantaneous output.

- Model: llama3-8b-8192 (8B parameters, 8192-token context)
- Hosted via **Groq API**, enabling lightning-fast generation
- Produces well-grounded, structured answers
- Ideal for **chat-like academic experiences**

Groq provides a practical middle-ground between open-source flexibility and production-grade performance.

3. Ollama (Local Inference)

Ollama enables self-hosted LLM execution on a developer's machine or private server. This work supports any LLM served via Ollama, with defaults like llama2 or llama3.

- Ensures **full privacy**: no data leaves the local system
- Supports **offline usage**, ideal for institutional or restricted environments
- Configurable with different model sizes and memory footprints
- Typically used with GPU acceleration for performance

Ollama integration allows this system to run **independently of external APIs**, giving end users full control over data, deployment, and customization.

5.1.2 Unified Backend Workflow for Answer Generation

The /ask-question endpoint acts as the **central API** for the question-answering workflow. Once the user submits a question through the frontend, this route orchestrates the retrieval and LLM generation process based on the selected **subject domain** and **LLM backend** (Gemini, Groq, or Ollama).

Upon receiving the POST request, the backend constructs a **retrieval-augmented chain**, sends the **user query + retrieved chunks** to the selected LLM, and formats the final response. Additionally, when the optional flag add_context is enabled, the endpoint also includes a list of the top-5 retrieved passages, each with source metadata (filename, page, etc.) for provenance tracking and display on the UI.

Parameter / Field	Type	Description
subject	string	Subject selected by user (used to find the right ChromaDB folder)
question	string	7
llm_choice	string	Backend LLM to use: "gemini", "groq", or "ollama"
add_context (optional)	boolean	Whether to return supporting context passages
Response: `answer`	string	Final answer generated by the selected LLM
Response: `qa_session_id`	UUID	Unique identifier for feedback and tracking
Response: `context` (optional)	list[dict]	Top retrieved passages with metadata: `source_pdf`, `page`, `preview`, `rank`

Table 12: API parameters and response fields for question-answering endpoint

LLM Answer Response (Structured View) when add_context is 'false'

```
{
  "answer": "The primary purpose of the sigmoid function in logistic regression is to transform the linear combination of input features (the linear score) into a probability between 0 and 1. This probability represents the likelihood of the input belonging to the \"positive\" class.",
  "qa_session_id": "ba735043-fec7-46ff-8a7e-8cea3be3badd"
}
```

The below interface demonstrates the primary QA interaction screen where a user selects the subject domain, chooses a preferred LLM backend (Gemini in this case), and submits a natural language query. The context toggle (Also show retrieved context) is **unchecked**, so only the **direct answer** from the selected LLM is displayed, without any additional supporting source snippets.

Figure 20: QA interface showing subject selection, LLM choice, and direct answer

5.1.3 In-Memory Caching Mechanism for Optimized RAG Execution

To improve efficiency and reduce latency, the backend introduces a **caching mechanism** via a dictionary object named `rag_chain_cache`. This acts as a **memory-based store** for retrieval-generation chains that are re-used across multiple queries within the same session or subject.

Each cache entry is indexed using a tuple: `(chroma_dir, llm_choice)`. This ensures that for every combination of **subject-specific document collection** and **selected LLM backend**, the system maintains a **ready-to-use pipeline** consisting of the retriever and the LLM interface. If a request comes in and the corresponding chain already exists in memory, the system avoids reloading the embedding model or ChromaDB vector store—saving both time and compute.

This caching logic is seamlessly integrated into the `/ask-question` endpoint. It allows users to interactively submit multiple questions from the same subject without any degradation in performance, even when switching between different LLM backends like **Groq**, **Gemini**, or **Ollama**.

5.1.4 Prompt Construction with Retrieved Context

An essential component of the Reader module is the construction of an effective prompt for the LLM. The goal is to inject both the **user's query** and the **retrieved academic content** into a format that guides the LLM toward accurate, grounded, and focused answer generation.

In this project, we use a **template-based approach** for prompt creation. The LangChain **ChatPromptTemplate** utility is employed to create consistent and well-structured prompts across different LLMs. Each prompt consists of:

- A system instruction block defining behavior
- The **user query** (e.g., "What is Logistic Regression?")
- The **retrieved context** (top-k semantic chunks from ChromaDB)
- Guidelines to avoid hallucinations or poor formatting

This strategy ensures **factual grounding**, **linguistic clarity**, and **reduced redundancy**, especially when used with models like Gemini or Llama 3, which are sensitive to context design.

Prompt Definition (LangChain-compatible)

```
def get_rag_prompt():
    rag_template = """\
Use the following context to answer the user's query. If you cannot answer, please respond
with 'I don't know'.

User's Query:
{question}

Context:
{context}

Instructions:
- Ensure the answer clearly refers to and aligns with the provided context
- Avoid unnecessary line breaks, slashes, or bullet points unless specifically required.
"""
```

This prompt is injected into the LLM pipeline right before invoking the model, ensuring each answer is shaped by the most relevant academic content.

5.1.5 LLM-Based Answer Generation Workflow

After the retriever provides the top-k most relevant document chunks, these passages, along with the user's question, are formatted into a prompt and submitted to the selected LLM. The

initial LLM-generated answer is then passed through a dedicated polishing stage, which utilizes the **spaCy** natural language processing library for refinement. This polishing process may involve:

- Trimming redundant phrases,
- Removing irrelevant content,
- Standardizing terminology,
- Formatting the answer for clarity or length.

By leveraging spaCy, the system can perform advanced linguistic analysis, such as part-of-speech tagging and named entity recognition, to systematically clean and enhance the LLM output before presenting it to the user.

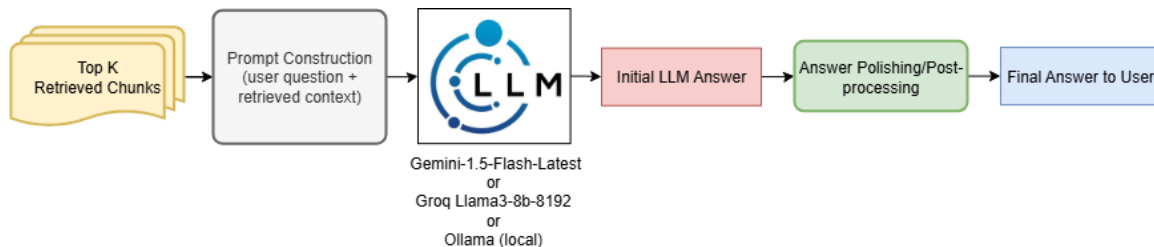


Figure 21: LLM-Based Answer Generation and Polishing Workflow

The final answer is polished and returned to the UI, with optional context and metadata for user review.

5.2 Answer Presentation with Context Preview and Metadata

In any Retrieval-Augmented Generation (RAG) system, presenting a fluent answer alone is not enough—**transparency and traceability** are vital. That’s where this answer presentation layer comes in. Once a user’s query is processed and an answer is generated by the selected LLM, the system complements it with a **contextual snapshot** of the retrieved information that influenced the response.

This phase closes the loop between **retrieval, reasoning, and user interaction**, giving users confidence in the source of the answers they receive. By showing **where the answer came from**, and **how relevant context chunks were selected**, the platform enhances educational usability and avoids black-box behavior.

Core Goals of This Stage

- Present the final **LLM-generated answer** in a readable form.
- Optionally show **retrieved context snippets** that contributed to the answer.

- Reveal supporting **metadata** such as source document name, page number, and retrieval rank.
- Enable structured backend caching (via session ID) to support **feedback collection**.

This metadata-driven approach allows users to *expand and verify* how the LLM interpreted the retrieved data—especially useful for academic use cases where correctness matters.

Below is the visual diagram that captures the flow from question to final answer presentation:

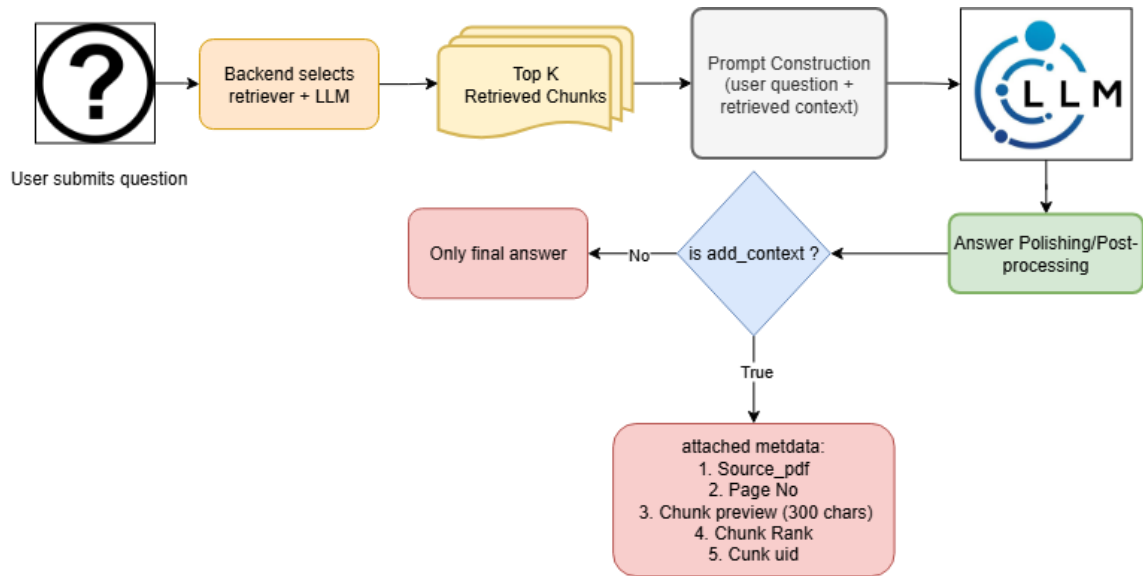


Figure 22: Comprehensive overview of the answer generation and context preview

The above flow diagram provides a **comprehensive overview** of the answer generation and context preview process in the RAG system.

The flow begins with a **user question**, which triggers the backend to select the appropriate **retriever and LLM backend** (Gemini, Groq, or Ollama). The retriever then fetches the **Top-K Retrieved Chunks**, which are used alongside the user's query to perform **Prompt Construction**. This unified prompt is sent to the **LLM for response generation**.

A key decision point follows—**is add_context set to true?** If not, only the **final answer** is returned to the user. However, if context is enabled, the backend includes **additional metadata** in the response, shown in below table:

Field	Description
source_pdf	File from which the chunk was retrieved (e.g., ML_Notes.pdf)
preview	First ~300 characters of the chunk
page	Page number of the document where the chunk exists
chunk_uid	Unique identifier used internally (not shown on UI)
rank	Position in top-k list (used for ordering)

Table 13: Metadata in case of additional context flag enabled

Sample output generated:

```
{
  "answer": "The primary purpose of the sigmoid function in Logistic Regression is to transform the linear combination of input features (the linear score) into a probability between 0 and 1. This probability represents the likelihood of the input belonging to the \"positive\" class.",
  "qa_session_id": "761d38b4-297e-4a35-8ff7-537ba1eef940",
  "context": [
    {
      "source_pdf": "Machine_Learning_ML_course_content_1.pdf",
      "page": 12,
      "rank": 1,
      "preview": "At its core, Logistic Regression is indeed a linear model. It starts by calculating a linear combination of input features, just like the other linear models: Linear Score = (Weight 1 * Feature 1) + ... + (Weight N * Feature N) + Bias However, instead of using this raw linear score directly, Logisti..."
    }
  ]
}
```

This metadata is also cached server-side using a `qa_session_id`, enabling accurate feedback submission in the next phase.

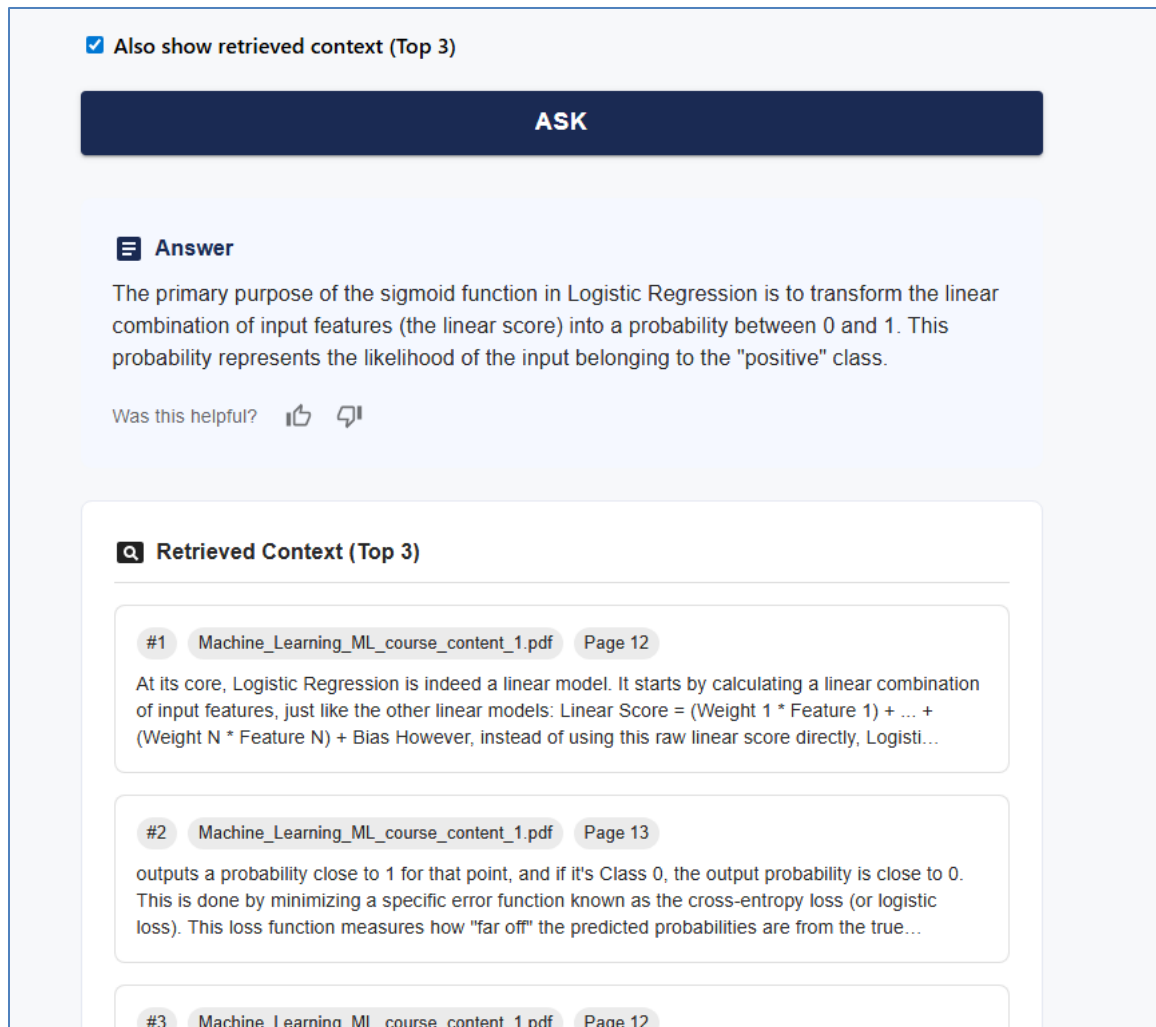


Figure 23: Generated answer with additional context

The image demonstrates how the system displays both the LLM-generated answer and the top 3 retrieved context chunks when the user enables the context preview option. Each snippet includes metadata like the source PDF and page number, enhancing traceability and user trust.

5.3 Feedback and Retrieval Adaptation

A powerful feature of this RAG-based QA system lies in its ability to learn from user interactions and adapt its retrieval process accordingly. Once an answer is generated and displayed to the user, the system provides an intuitive feedback interface—allowing the user to indicate whether the provided answer was **helpful or not**. This binary feedback mechanism is accompanied by an optional text comment for unhelpful responses. The feedback process serves two key roles: (1) collecting user insights on the system's accuracy, and (2) dynamically improving future retrieval quality through **feedback-aware reranking**.

Behind the scenes, a structured session-level feedback architecture is employed. When a question is asked, a unique `qa_session_id` is generated and cached in-memory (QA_CACHE) along with three key artifacts: the original user query, a snapshot of the top retrieved chunks (each tagged with metadata like file name, page, and rank), and a short excerpt of the generated answer. This snapshot is crucial—it allows the system to later associate the user's feedback with specific context chunks, even if the full documents have changed or the session has ended.

Once feedback is submitted, a JSONL-formatted log entry is written to a local file (`feedback.jsonl`). Each line corresponds to a structured feedback object and includes:

- The session ID
- Binary helpful status
- Optional user comment
- Retrieval snapshot (list of chunks returned)
- Timestamp and backend LLM used

Sample feedback.jsonl entry:

```
{
  "qa_session_id": "f6705050-bef6-42c3-825f-04b204cc91b2",
  "helpful": false,
  "comment": "Example feedback provided",
  "llm_backend": "gemini",
  "timestamp": "2025-08-09T16:25:11.830668",
  "query": "What is the primary purpose of the sigmoid function in Logistic Regression?",
  "retrieval_snapshot": [
    {
      "chunk_uid": "Machine_Learning_ML_course_content_1.pdf_12_abc123",
      "source_pdf": "Machine_Learning_ML_course_content_1.pdf",
      "page": 12,
      "rank": 1
    }
  ],
  "answer_excerpt": "The sigmoid function is used to transform the linear score into a probability between 0 and 1..."
}
```

This log not only provides an auditable trail of user feedback, but also powers a **real-time feedback-aware reranking system**. Upon startup (or refresh), the backend invokes

rebuild_reputation_from_log(), which parses the log file and rebuilds a global reputation dictionary (CHUNK_REP). This dictionary maps each chunk's unique ID to a running count of upvotes and downvotes, stored in-memory.

The below diagram illustrates the complete end-to-end **feedback flow** and how it influences **retrieval reranking** in the system.

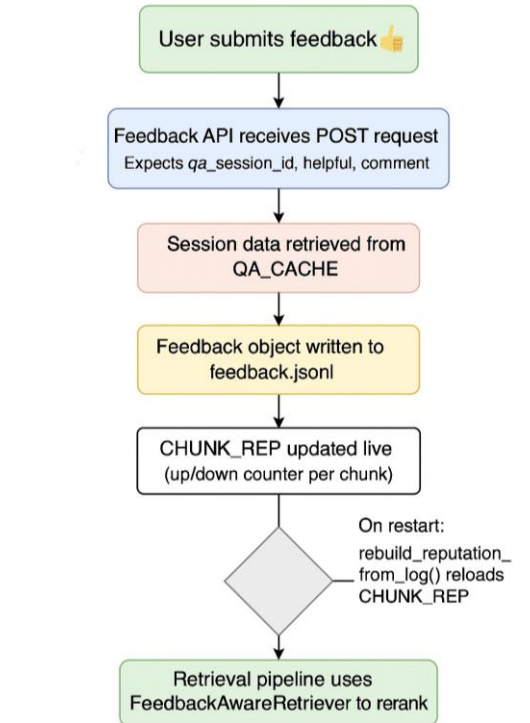


Figure 24: Feedback flow mechanism in QA system

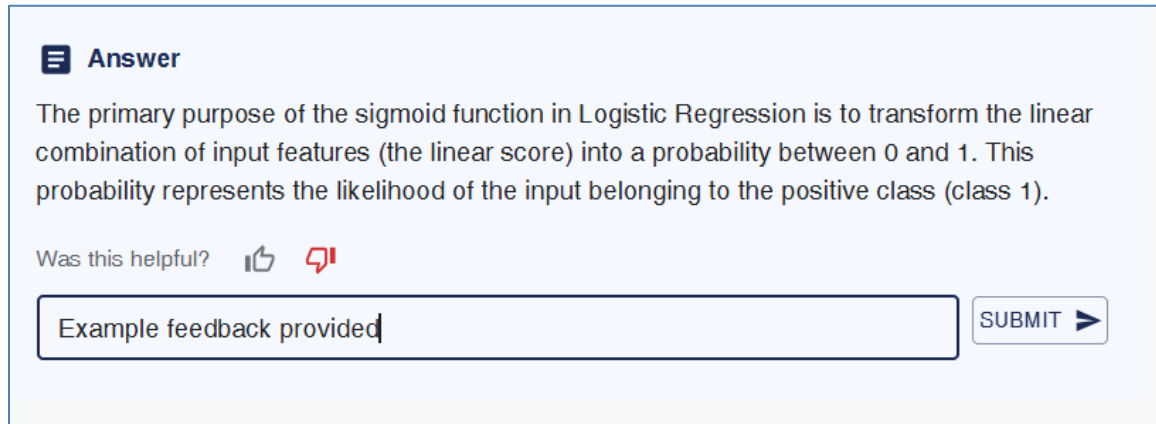
5.3.1 Feedback-Aware Re-ranking Logic

To make practical use of feedback data, the system wraps the default retriever with a custom class called **FeedbackAwareRetriever**. This class inherits from BaseRetriever and reorders the top-k retrieved chunks by applying a small penalty to chunks that have **more downvotes than upvotes**. Specifically:

- Chunks with negative feedback are demoted.
- Those with positive or no feedback retain their ranking.
- The magnitude of penalty is governed by a tunable hyperparameter β (default 0.2).



This ensures that the retrieval step evolves over time, pushing unhelpful chunks lower in the list while promoting context segments that have proven to be valuable to past users. The

reranker respects original ranking for all unknown chunks, thereby maintaining stability for newer or rarely used contexts.



Answer

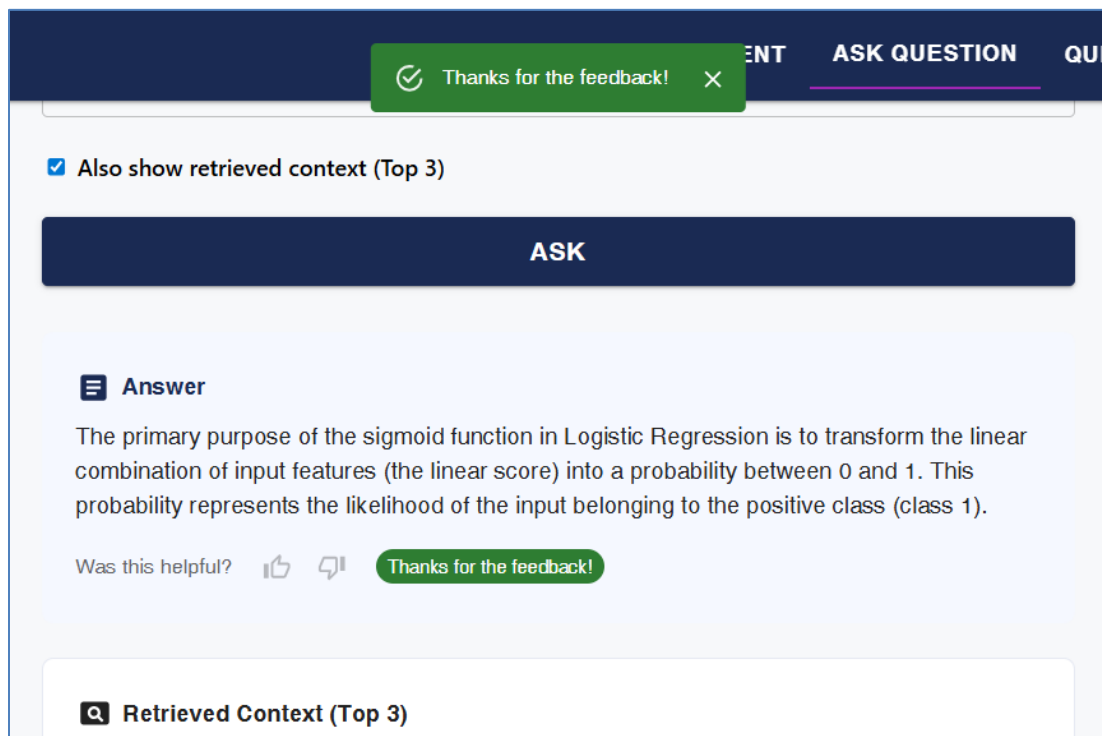
The primary purpose of the sigmoid function in Logistic Regression is to transform the linear combination of input features (the linear score) into a probability between 0 and 1. This probability represents the likelihood of the input belonging to the positive class (class 1).

Was this helpful?  

Example feedback provided

Figure 25: Feedback providing UI option

As shown above, the user is presented with an option to provide binary feedback (thumbs up/down) along with an optional text comment. This helps the system capture qualitative insights when an answer is not helpful.





☒ Also show retrieved context (Top 3)

ASK

Answer

The primary purpose of the sigmoid function in Logistic Regression is to transform the linear combination of input features (the linear score) into a probability between 0 and 1. This probability represents the likelihood of the input belonging to the positive class (class 1).

Was this helpful?  

Retrieved Context (Top 3)

Figure 26: Success notification for user feedback submission

Upon submitting feedback, the system acknowledges the user's input with a success message. This feedback is then logged and contributes to the chunk's reputation for future answer reranking.

Chapter 6

Document Summarization & Prompt-Driven Q/A Generation

This module enables educators to **automatically generate exam-ready question papers** from uploaded academic documents. By leveraging LLM-based summarization and carefully structured prompts, it transforms large textual content into meaningful and balanced sets of questions across various formats (MCQs, one-liners, fill-in-the-blanks, etc.). This pipeline ensures:

- Context-aware content summarization,
- Prompt-driven question generation,
- UI-backed control over document selection, difficulty, and type,
- Exportable output (PDF) for easy use.

6.1 Summarization Pipeline: Reducing Context to Its Core

As educational handouts and academic PDFs grow in size, feeding them directly into an LLM becomes infeasible due to token limits. Not all information in these documents is equally useful either — raw content often contains redundant data, boilerplate text, or sections unrelated to assessment. To address these limitations, this system incorporates an **automated summarization pipeline**, which extracts **key educational insights** and prepares them for downstream tasks such as question generation. The summarization step also ensures that even long-form input can be compressed into a **token-efficient, meaningful summary**, enabling prompt-driven Q/A to operate effectively.

The entire pipeline spans multiple components — from frontend UI where the user selects files, all the way to recursive summarization, filtering, and result caching. Designed for **performance, stability, and reusability**, the process involves recursive LLM summarization with token budget enforcement, multi-level retries, and intelligent caching to avoid repeated computations. Furthermore, it supports various LLM backends (Groq, Gemini, Ollama) and ensures compatibility with token constraints of each. This makes it both robust and LLM-agnostic.

The final output is a **clean, exam-friendly summary**, typically under 1800 words, structured enough to allow educators or the backend engine to formulate meaningful assessments. This summary is passed on to the next stage (detailed in Section 6.2) where prompts are constructed and Q/A generation takes place.

6.2 End-to-End Summarization Pipeline

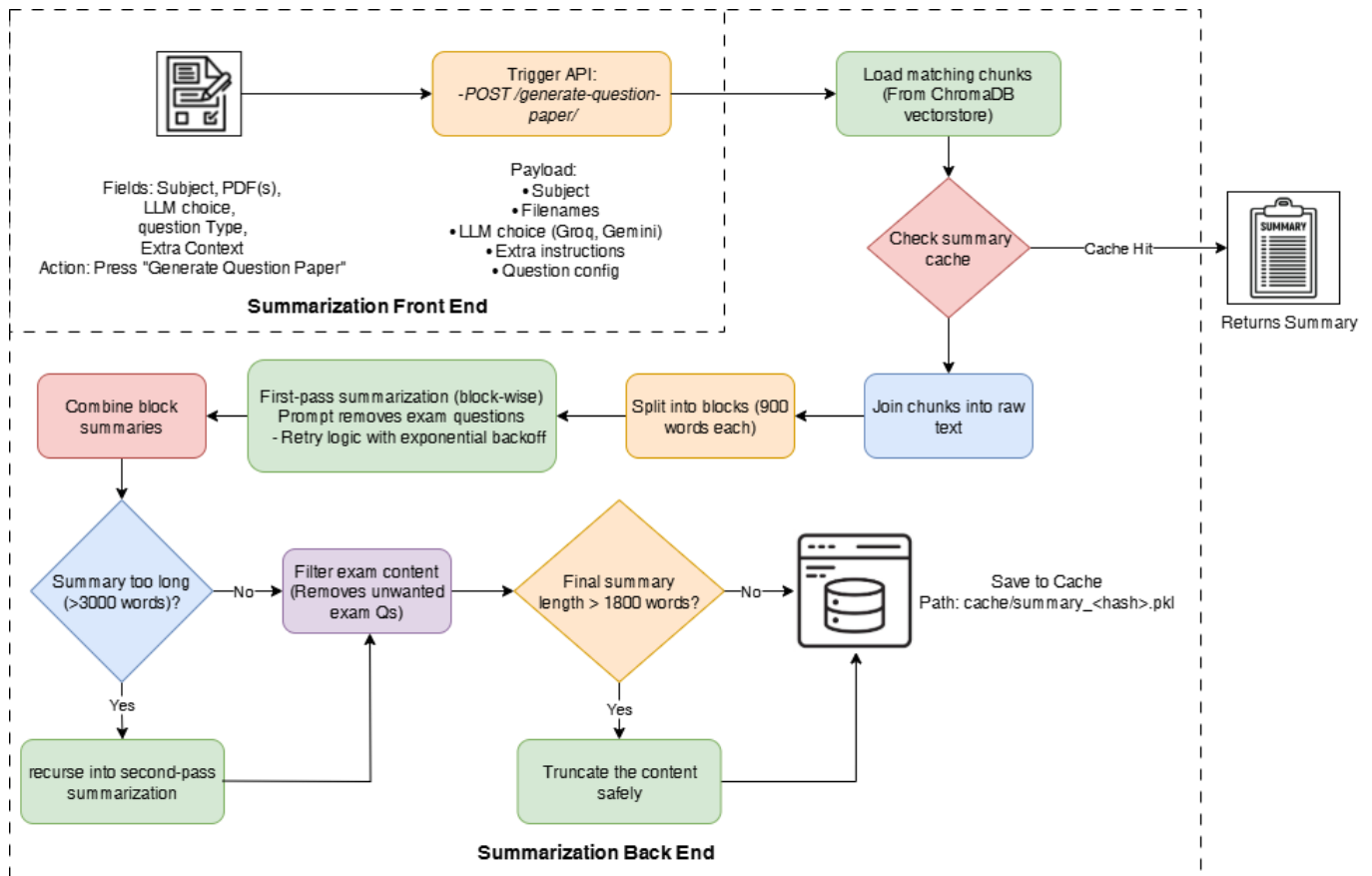


Figure 27: End-to-End document summarization flow in QA generation

Frontend Overview

The summarization pipeline begins when the user interacts with the UI form titled "**Generate Question Paper**". The user is required to select a **Subject**, one or more **PDF files** associated with that subject, choose an **LLM backend** (like Groq or Gemini), and optionally provide any **extra instructions** to guide the summarization and question generation process.

Once selections are made and the "**Generate Question Paper**" button is pressed, a POST request is triggered to the backend API endpoint `/generate-question-paper/`. The payload of this request is rich — it includes the user's selected files, the target LLM, the configuration for how many and what kind of questions to generate, and additional user-defined context. This API call becomes the entry point for the backend pipeline, passing the entire context for processing.

Backend Processing Pipeline

1. Chunk Retrieval and Cache Check

The backend first calls `get_chunks_by_filenames(...)` to extract all relevant chunks from ChromaDB whose metadata matches the user-selected PDFs. These chunks, essentially short segments of the educational content, are stitched together into a single string. Before doing any heavy computation, the system checks if a summary already exists in the cache using `cache_summary_load(...)`. If a match is found (based on a hashed key of subject, files, LLM, and context), the summary is returned immediately, saving time and compute.

2. Recursive Summarization Logic

If no cached summary exists, the system begins a **deep summarization process**. First, the raw text is split into manageable blocks (≈ 900 words each). Each block is then passed through the `llm_summarize_func(...)` function, which communicates with the chosen LLM using a prompt that enforces strict summarization rules (e.g., no exam questions, structured output). This prompt is carefully designed and enforced with **retry logic** in case of transient failures.

After the first-pass block-wise summarization, the outputs are stitched together. If the combined result is still too long (e.g., over 3000 words), a **second recursive summarization pass** is triggered using the same logic, compressing the content even further while preserving its structure.

3. Filtering, Final Compression, and Caching

Once a reasonable-sized summary is generated, it passes through a filtering function `filter_summary(...)` that removes any residual "exam questions" or irrelevant formatting. If the summary still exceeds the 1800-word budget, it's **safely truncated or re-summarized** to fit within token constraints for future LLM calls. Finally, the processed summary is stored using `cache_summary_save(...)` under a uniquely hashed filename. The backend then returns the summary (and later, the generated questions) as a structured JSON response to the frontend.

Example API Usage

API Endpoint: POST `/generate-question-paper/`

Request JSON:

```
{
  "subject": "Machine Learning",
  "filenames": ["ML_Handout_1.pdf"],
  "llm_choice": "groq",
  "question_config": {
    "total_questions": 6,
    "difficulty": "medium",
```

```

    "distribution": {
      "one_liner": 2,
      "true_false": 1,
      "multiple_choice": 2,
      "descriptive": 1
    },
    "extra_context": "Focus more on regression and classification topics."
  }
}

```

6.3 Prompt Construction and Response Generation

This section outlines the backend mechanism that transforms a **finalized summary** into a well-structured **set of academic questions**. Once summarization is complete, the system invokes a dedicated logic to construct a precise **LLM prompt** and send it to the chosen model (e.g., **Groq Llama3**, **Gemini Flash**). The model is instructed to return a **strictly validated JSON** object containing question-answer pairs, formatted according to a predefined schema. This ensures **consistency**, **validity**, and **clean parsing** for further frontend use or export (e.g., to PDF).

Key Features of Prompt Construction Logic

- **System Prompt Template:** Defines the role of the LLM (exam generator) and enforces a strict output format with all required fields (type, question, options, answer).
- **User Prompt:** Dynamically built based on user inputs — including total question count, difficulty, and distribution across question types.
- **Strict Output Schema:** The response is parsed and validated using QuestionPaper Pydantic schema (LangChain for Groq; Gemini schema for Google API).
- **Structured Output Mode:**
 1. **For Groq**, `json_mode` ensures the output strictly follows the schema.
 2. **For Gemini**, `response_schema` validation is used via Google's SDK.

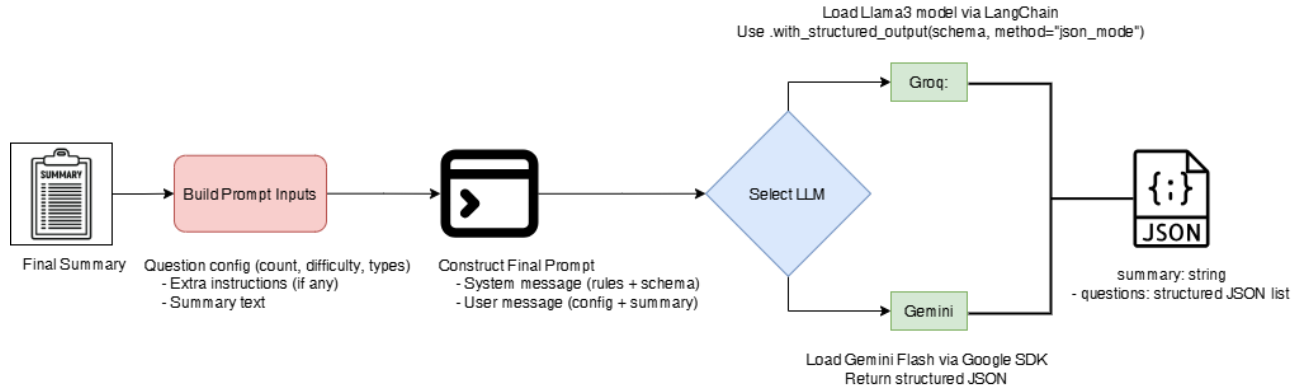


Figure 28: QA pair generation flow using summary and prompt

1. Prompt Formation

The prompt begins with a **system message**, which instructs the LLM to act as a strict exam paper generator and follow a **hardcoded schema**. This avoids unpredictable outputs and ensures that fields like type, question, options, and answer are present and correctly formatted.

A **user message** is then assembled dynamically using:

- The user's selected **question distribution** (e.g., 2 MCQs, 1 descriptive),
- The **difficulty level** (easy/medium/hard),
- Optional **extra context or constraints**, and
- The full **deep summary** from the previous step.

This prompt ensures that the LLM operates within a clear instructional boundary.

2. LLM Invocation

- If **Groq** is selected, the system uses **LangChain's ChatPromptTemplate** and binds the model to a schema using `.with_structured_output()` in `json_mode`. This ensures that the LLM generates **valid Pydantic-compatible JSON** directly — eliminating the need for fragile string parsing.
- If **Gemini** is selected, the backend uses **Google's native SDK** with `generate_content(...)`, supplying a `response_schema` to validate the structure. The model responds with a **strict JSON block**, and any format violations are caught at the SDK level.

3. Structured Output and Return

Once the response is received:

- If the output passes validation, it's returned as part of the final payload to the frontend (i.e., summary + questions).
- If any failure occurs (e.g., schema mismatch), a fallback error message is returned indicating the issue.

This strict mechanism improves **reliability**, **integrity**, and **user confidence** in the generated question papers.

Sample response:

```
{
  "summary": "This chapter introduces fundamental concepts in supervised machine learning, covering algorithms such as linear regression, logistic regression, and decision trees. It explains the process of training models using labeled datasets, evaluating performance through metrics like accuracy and F1-score, and techniques to prevent overfitting, including regularization. The section concludes with a discussion on the importance of feature selection and preprocessing for optimal model performance.....",
  "questions": {
    "questions": [
      {
        "type": "one_liner",
        "question": "What is the primary goal of supervised learning?",
        "options": null,
        "answer": "To learn a mapping from input to output using labeled data."
      },
      {
        "type": "true_false",
        "question": "Decision trees cannot be used for classification tasks.",
        "options": null,
        "answer": "False"
      },
      {
        "type": "fill_blank",
        "question": "_____ is a technique used to prevent overfitting by adding a penalty to the loss function.",
        "options": null,
        "answer": "Regularization"
      },
      {
        "type": "multiple_choice",
```

```

    "question": "Which metric is commonly used to evaluate classification models?",
    "options": ["MSE", "F1-score", "MAE", "R-squared"],
    "answer": "F1-score"
  }
]
}
}

```

6.4 Educator View: Summary and Generated Questions

The below screenshot displays the top portion of the “**Generate Question Paper**” module, where the user configures input parameters for the generation task. The user selects the **Subject, PDF(s)** from the ingested corpus, and the desired **LLM backend**(Groq or Gemini).

In the “Question Setup” section, different types of questions can be toggled along with their desired **count**, and a difficulty level (Easy/Medium/Hard) is chosen.

The screenshot shows the 'Generate Question Paper' interface. It is divided into two main sections: '1. Data Source' and '2. Question Setup'.

1. Data Source: This section contains three dropdown menus. The 'Subject' dropdown is set to 'Machine Learning'. The 'PDF(s)' dropdown is set to 'Machine_Learning_ML_course_content_1.pdf'. The 'LLM' dropdown is set to 'Groq (Llama3)'.

2. Question Setup: This section contains a 'Difficulty' dropdown set to 'Medium'. To the right of this section is a badge that says 'Total Questions: 7'. Below the difficulty dropdown is a list of question types, each with a checkbox and a count input field:

- One Liner:** Checked, count 1
- True/False:** Checked, count 2
- Fill Blank:** Checked, count 1
- Multiple Choice:** Checked, count 2
- Descriptive:** Checked, count 1

Figure 29: Input Configuration & Question Setup for Question paper generation

This section allows the user to provide **extra context or educator instructions**, such as focusing on specific chapters or topics. Once “Generate Question Paper” is clicked, a deep LLM-generated **summary** of the selected documents is displayed in an accordion view.

This summary is factual, structured, and optimized to serve as a foundation for high-quality question generation, ensuring the questions are tightly aligned with the source material.

3. Additional Context or Instructions

Extra Context / Instructions (Optional)

Focus more on regression algorithms, decision tree and evaluation metrics.
Prioritize questions from Chapter 2, 3 and 5 only.
Include at least one conceptual question on decision tree topic.

GENERATE QUESTION PAPER

Deep Summary

****Machine Learning Overview and Fundamentals****
****Introduction**** Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that involves designing algorithms capable of learning from data and improving over time without being explicitly programmed. ML systems can identify and extract patterns from large datasets, allowing them to make predictions, detect anomalies, and generate new content. The applications of ML are vast and varied, including self-driving cars, language translation, recommendation engines, medical image analysis, and fraud detection.

****Taxonomy of Machine Learning**** ML can be broadly classified into four main categories:
* ****Supervised Learning****: Learning from labeled datasets, where each input comes with a corresponding output (e.g., spam detection, house price prediction).
* ****Unsupervised Learning****: Extracting patterns from unlabeled data, such as clustering or dimensionality reduction (e.g., customer segmentation, topic modeling).
* ****Semi-Supervised Learning****: Using both labeled and unlabeled data, common when labeled data is scarce but unlabeled data is plentiful.
* ****Reinforcement Learning****: Learning optimal actions through trial and error and receiving feedback via rewards (e.g., game playing, robotics).

****Design of a Learning System**** A typical machine learning system involves several important steps and components, all of which work together to enable accurate predictions or decisions:
* ****Data Collection & Preprocessing****: Collecting relevant data, which may include text, images, audio, or tabular data, and preprocessing it to transform raw data into a format suitable for analysis.
* ****Feature Selection and Engineering****: Identifying the most predictive features and creating new features based on existing ones (e.g., polynomial features, interaction terms).
* ****Model Selection****: Choosing the right learning algorithm, which may depend on the dataset, problem, and the interpretability required.

Figure 30: Additional Instructions & Summary Preview

Here, the generated set of **Q/A pairs** is displayed according to the configuration provided earlier.

Questions span **multiple types** including one-liners, true/false, fill-in-the-blank, MCQs, and descriptive.

Each question is clearly labeled and optionally includes answer options where applicable. The final step allows the user to **download the complete question paper as a PDF**, including both questions and answers neatly formatted.

Generated Questions:

Q1.

What is the primary goal of machine learning?

Q2.

Supervised learning involves learning from unlabeled data.

Q3.

Regularization techniques can help in selecting appropriate basis functions.

Q4.

What is the primary difference between overfitting and underfitting?

- a. Overfitting occurs when a model is too complex, while underfitting occurs when a model is too simple.
 - b. Overfitting occurs when a model is too simple, while underfitting occurs when a model is too complex.
 - c. Both occur when a model is too complex.
 - d. Both occur when a model is too simple.
-

Q5.

What is the primary advantage of using the Perceptron algorithm?

- a. It can handle non-linearly separable data.
 - b. It is guaranteed to find a perfect solution in a finite number of steps if the data is perfectly linearly separable.
 - c. It is a type of decision tree.
 - d. It is a type of neural network.
-

Q6.

Gradient Descent is an optimization algorithm that iteratively updates the weights to minimize the _____ function.

Q7.

Describe the concept of the bias-variance decomposition in machine learning.

DOWNLOAD AS PDF

Figure 31: Generated Questions Preview & Export

Chapter 7

Evaluation & RAGAS-Based Benchmarking

Evaluation is a vital part of any LLM-based pipeline, especially for educational question-answering systems. To ensure both the correctness of the answer and the faithfulness of its supporting context, this project integrates the **RAGAS framework** (Retrieval-Augmented Generation Assessment Scores) to benchmark LLM outputs using structured metrics.

The evaluation process measures how well the retrieved context supports the final answer, how semantically aligned the generated answer is to the ground truth, and how factually grounded the result is. The RAGAS pipeline allows single-model evaluation as well as multi-model benchmarking, providing detailed metric-wise comparisons across models like Groq (Llama3), Gemini, and Ollama.

OpenAI's GPT-4 (via ChatGPT) is used as a neutral judge by RAGAS to perform these evaluations consistently across all metrics.

RAGAS metrics are categorized into the following sections:

- **Retrieval Metrics**
Evaluate how relevant the retrieved context is with respect to the ground truth.
(*e.g., Context Precision, Context Recall, Faithfulness*)
- **Nvidia Metrics**
Proposed by Nvidia to measure accuracy and context-groundedness in enterprise QA tasks.
(*e.g., Answer Accuracy, Context Relevance, Response Groundedness*)
- **Language Metrics**
Focus on textual similarity and generation quality.
(*e.g., Factual Correctness, Semantic Similarity, ROUGE Score, BLEU Score, etc.*)

Each metric category is explored in a separate section (6.2–6.4), where detailed visualizations and interpretations are provided.

7.1 RAGAS Evaluation Pipeline

This section outlines the entire evaluation setup, including how LLM-generated answers are evaluated against reference ground truths using the RAGAS metric suite. The backend supports both single and multi-model evaluation, returning results in structured format suitable for tables, charts, and further visualization.

Evaluation Modes Supported

- **Single Model Evaluation:**
 1. Generates answers using the selected LLM.
 2. Runs all applicable RAGAS metrics.
 3. Displays per-question results with precision.
- **Multi-Model Benchmarking:**
 1. Runs the same question set across multiple LLMs.
 2. Compares all metrics across LLMs.
 3. Shows results as bar graphs, radar charts, and per-question breakdowns.

Below is the flowchart depicting the complete RAGAS evaluation pipeline — from input to metric scoring and caching:

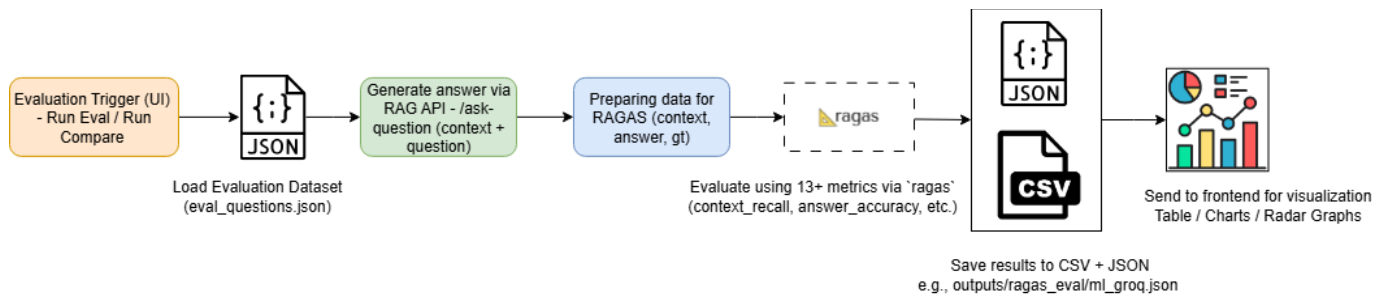


Figure 32: RAGAS evaluation pipeline

Input Format: Evaluation JSON

The input to the RAGAS pipeline is a structured JSON file named (typically) `eval_questions.json`, containing a list of dictionaries — each representing a **question**, the **expected answer (ground truth)**, and optionally pre-defined **retrieved contexts**. This file can be prepared manually or generated via earlier stages of the application.

```
[
  {
    "question": "List four application areas mentioned for ML in the document.",
    "ground_truth": "Examples include self-driving cars, language translation,
recommendation engines, medical image analysis.",
    "contexts": [
      "The applications of ML are vast and varied: self-driving cars, language translation,
recommendation engines, medical image analysis, and fraud detection.",
      "Machine Learning (ML) is a subfield of AI that involves designing algorithms capable
of learning from data and improving over time."
    ]
  },
  ...
]
```

Backend Design Summary

- **Evaluation endpoint:** POST /evaluate-ragas/

Supports both { model_name: "groq" } and { model_names: ["groq", "gemini", "ollama"] }.

- **Result format:**

1. Saved in both .json and .csv formats.
2. Organized by model name, subject, and metric type.

- **Delay mechanism:**

Answer generation and evaluation are spaced using time.sleep() for safer throttling of LLM calls

Output Format: RAGAS Evaluation Results

After LLM-generated answers are evaluated against the ground truth and contexts, a JSON file is produced per model (e.g., ml_groq_results.json) containing per-question metric scores.

Example: Output Sample (ml_groq_results.json)

```
[
  {
    "id": 2,
    "question": "List four application areas mentioned for ML in the document.",
    "contexts": [
```

```

    "The applications of ML are vast and varied: self-driving cars, language translation,
    recommendation engines, medical image analysis, and fraud detection.",
    ...
  ],
  "answer": "Language translation, recommendation engines, and medical image analysis.",
  "ground_truth": "Examples include self-driving cars, language translation,
recommendation engines, medical image analysis.",
  "context_precision": 0.90,
  "context_recall": 0.86,
  "faithfulness": 0.94,
  "nv_accuracy": 0.50,
  "semantic_similarity": 0.88,
  "factual_correctness": 0.82,
  "bleu_score": 0.70,
  "rouge_score": 0.95,
  "exact_match": 0.75,
  ...
},
...
]

```

7.2: Retrieval Metrics

Retrieval metrics evaluate how well the system retrieves relevant context from the knowledge base in response to a user query. These metrics focus on the relevance and coverage of retrieved chunks relative to the ground truth and final answer. RAGAS provides powerful tools to quantify this through three primary metrics:

- **Context Precision**
- **Context Recall**
- **Faithfulness**

These metrics assess whether the retrieved context:

- Matches the information needed to answer the question (**Recall**),
- Avoids irrelevant or noisy context (**Precision**), and
- Is actually used to support the response (**Faithfulness**).

Each score ranges from **0 to 1**, where **1** indicates ideal performance.

1. Context Precision Metric

Definition:

Context Precision quantifies how many of the retrieved context chunks are actually relevant to the generated answer. High precision implies minimal noise in retrieval — only the most pertinent content is selected.

LLM-Based Scoring:

Each chunk in the retrieved context is evaluated by an LLM to determine its relevance to the final answer. Only chunks directly contributing to the answer are considered relevant.

Formula:

$$\text{Context Precision@K} = \frac{\sum_{k=1}^K (\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{(\text{true positives@k} + \text{false positives@k})}$$

Associated Chart:

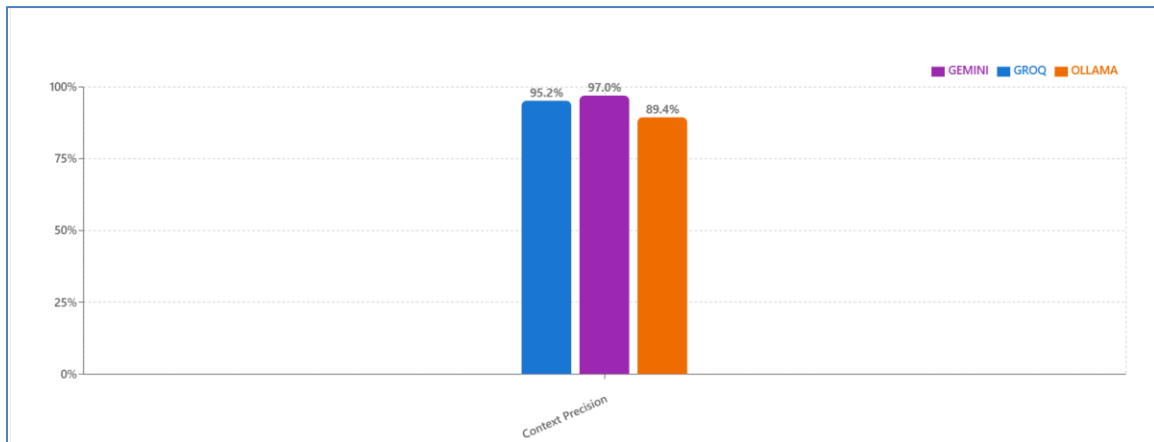


Figure 33: Bar chart for Context Precision scores per model

Interpretation:

- **Gemini 97.0% > Groq 95.2% > Ollama 89.4%**

Gemini retrieves the most on-target chunks; Groq is close behind; Ollama includes slightly more off-topic or less relevant passages.

- **So what:**

High precision means fewer irrelevant chunks enter the LLM, reducing distraction and potential hallucinations. The gap suggests Ollama's answers may occasionally be influenced by extra or noisy context.

2. Context Recall Metrics

Definition:

Context Recall measures how much of the **important information** from the reference answer is present in the retrieved context. It reflects the coverage of necessary facts.

LLM-Based Scoring:

The reference answer is decomposed into factual claims. An LLM checks which of these are supported by the retrieved context.

Formula:

$$\text{Context Recall} = \frac{\text{Number of claims in the reference supported by the retrieved context}}{\text{Total number of claims in the reference}}$$

Associated Chart:

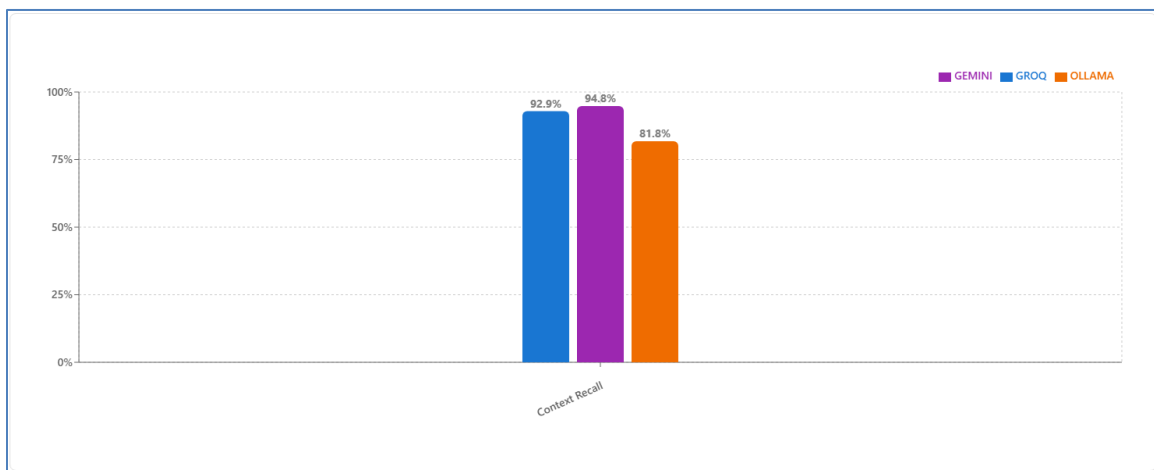


Figure 34: Bar chart for Context Recall scores per model

Interpretation:

- **Gemini 94.8% > Groq 92.9% > Ollama 81.8%**

Gemini retrieves almost all relevant content; Groq is close behind; Ollama misses more supporting details.

- **So what:**

High recall ensures the LLM has all necessary facts to form complete answers. Ollama's lower recall means it's more likely to omit important context, leading to partial or incomplete responses.

3. Faithfulness Metric

Definition:

Faithfulness evaluates whether the generated answer is fully grounded in the retrieved context. This metric detects hallucinations — unsupported claims or fabricated facts.

LLM-Based Scoring:

The answer is split into individual claims. An LLM checks whether each claim is directly verifiable from the retrieved documents.

Formula:

$$\text{Faithfulness Score} = \frac{\text{Number of claims in the response supported by the retrieved context}}{\text{Total number of claims in the response}}$$

Associated Chart:

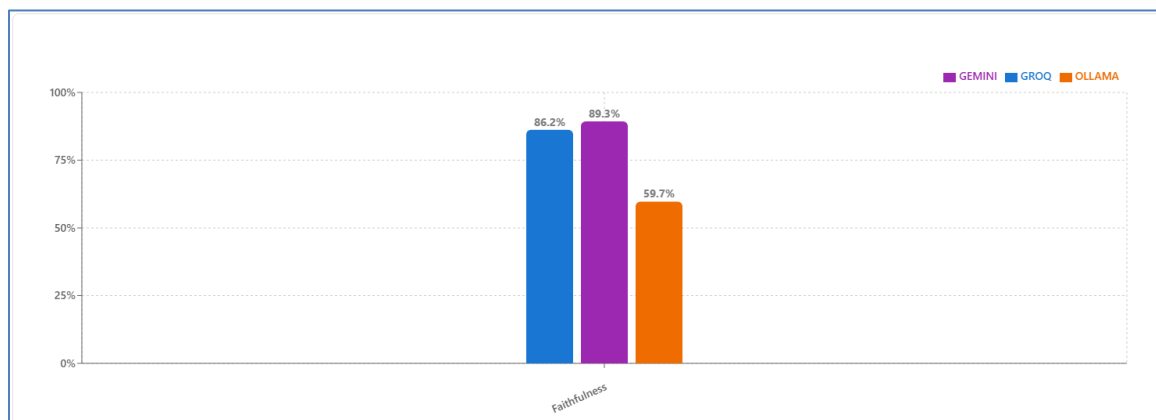


Figure 35: Bar chart for Faithfulness scores per model

Interpretation:

- **Gemini 89.3% > Groq 86.2% > Ollama 59.7%**

Gemini and Groq produce responses that stay closely aligned with retrieved content; Ollama shows a much higher tendency to introduce unsupported information.

- **So what:**

Faithfulness reflects how well an answer is grounded in retrieved facts. Ollama's lower score indicates a greater risk of hallucinations, which could mislead users in factual or academic contexts.

Model Comparison — Retrieval Metrics

We evaluated the retrieval quality of **Gemini**, **Groq**, and **Ollama** across the above three metrics using both **bar charts**, **radar plots**, and an **aggregated table view**.

Radar Chart Insight

To provide an at-a-glance view of retrieval behavior across all three metrics, we plotted individual radar charts for each model.

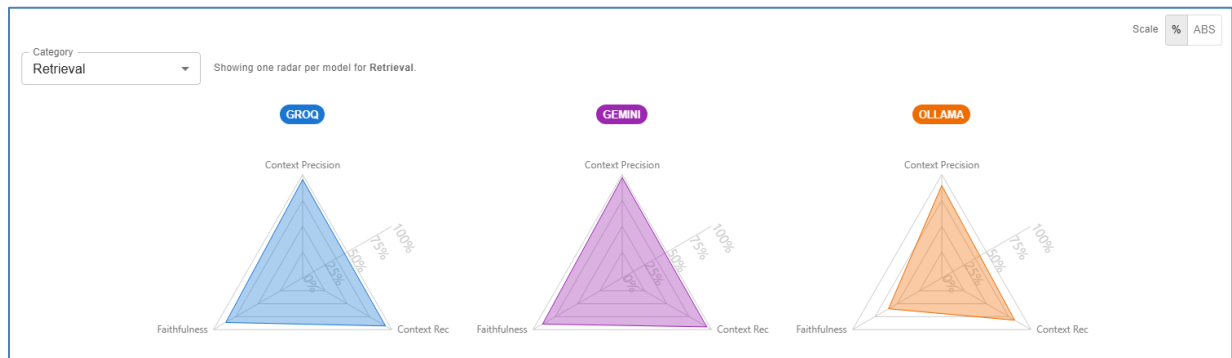


Figure 36: Radar chart for Retrieval Metrics

Radar Highlights:

- **Gemini** nearly touches the outer edges on all axes, forming a near-equilateral triangle — a sign of **balanced and strong** retrieval.
- **Groq** shows a similar shape, just slightly smaller — indicating reliability.
- **Ollama**'s triangle is noticeably compressed, especially along the Faithfulness axis, matching its lower bar scores.

Aggregated Metric Table

Metric	GROQ	GEMINI	OLLAMA
Context Precision	95.2%	97%	89.4%
Context Recall	92.9%	94.8%	81.8%
Faithfulness	86.2%	89.3%	59.7%

Table 14: Aggregated Metric Table for Retrieval Category

Summary of Retrieval Metrics

In retrieval tasks:

- **Gemini** stands out as the most **precise, comprehensive, and grounded** model.
- **Groq** is a close runner-up with similarly reliable retrieval performance.
- **Ollama** lags in both **recall** and **faithfulness**, suggesting a need for refinement in grounding and information coverage.

These metrics are crucial for assessing RAG system quality before downstream answer generation.

7.3 Nvidia Metrics

This section focuses on evaluating model performance through the lens of Nvidia-provided quality metrics. These metrics reflect not just retrieval strength, but how well the model-generated answers align with ground truth and remain faithful to retrieved context. The three primary metrics used are:

1. NV Answer Accuracy

Definition:

NV Answer Accuracy measures how closely a model's generated response matches the reference answer. It reflects factual correctness, determined through a pair of "LLM-as-a-judge" templates, each returning a rating (0, 2, or 4), which are then normalized and averaged.

- **0:** Completely inaccurate or unrelated answer
- **2:** Partially aligned with the reference
- **4:** Fully aligned with the reference

LLM-Based Scoring:

Two prompts are used to cross-check the answer:

- In the first, the model's response is compared to the reference.
- In the second, roles are swapped.

The two scores are normalized to a [0,1] scale and averaged.

Associated Bar Chart:

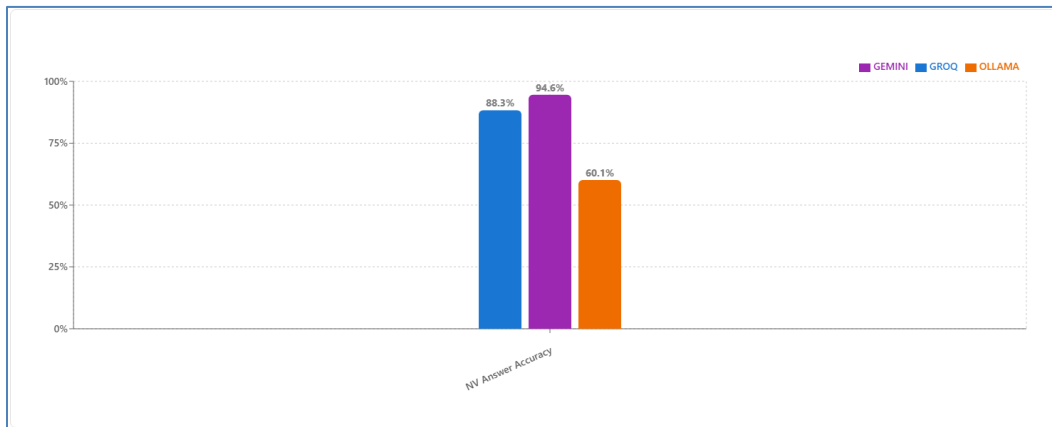


Figure 37: Bar chart for NV Answer Accuracy

Interpretation:

- **Gemini 94.6% > Groq 88.3% > Ollama 60.1%**

Gemini delivers the most factually correct answers, closely matching reference responses; Groq is slightly behind but still strong. Ollama's much lower accuracy suggests frequent factual errors or partial responses.

- **So what:**

High NV Answer Accuracy means the model can produce outputs that are both relevant and factually consistent with the expected answer. Ollama's gap here signals the need for additional fact-checking or post-processing before deployment in critical use cases.

2. NV Context Relevance

Definition:

This metric evaluates how relevant the retrieved context is with respect to the user's query. It ensures that retrieval isn't just accurate but topically aligned.

- **0:** Irrelevant
- **1:** Partially relevant
- **2:** Fully relevant

LLM-Based Scoring:

The model is queried with two relevance-based templates to evaluate if the retrieved content supports the original question. Scores are normalized by dividing by 2.

Associated Bar Chart:

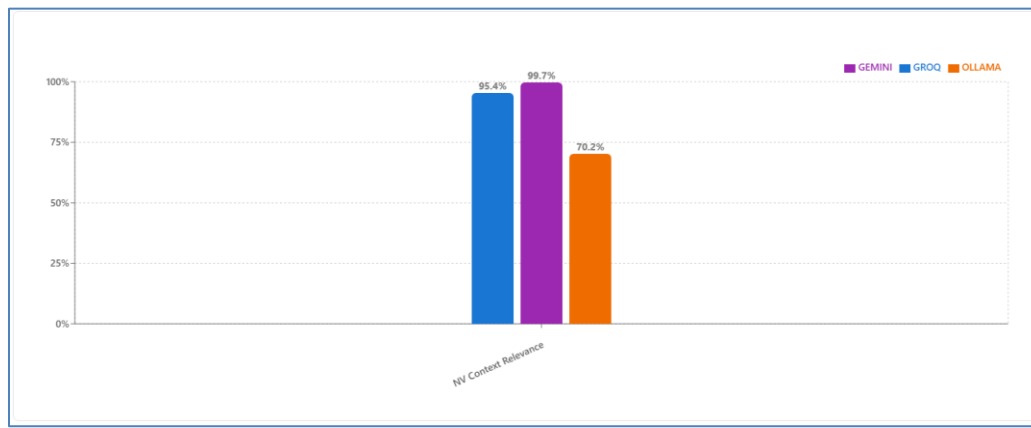


Figure 38: Bar Chart for NV Context Relevance

Interpretation:

- **Gemini 99.7% > Groq 95.4% > Ollama 70.2%**

Gemini retrieves context that is almost perfectly aligned with the query, while Groq remains very close in relevance. Ollama's lower score suggests that a notable portion of retrieved content is only partially related or off-topic.

- **So what:**

High context relevance ensures that answers are grounded in the most appropriate information. Ollama's gap here could lead to weaker grounding and more off-target responses in the RAG workflow.

3. NV Response Groundedness

Definition:

This metric assesses how well the generated response is supported (or "grounded") by the retrieved contexts. It penalizes hallucinations or unsupported claims.

- **0:** Not grounded
- **1:** Partially grounded
- **2:** Fully grounded

LLM-Based Scoring:

Two prompts ask the LLM to verify if each claim in the response is present in the retrieved contexts. Scores are again normalized (e.g., 2 becomes 1.0) and averaged.

Associated Bar Chart:

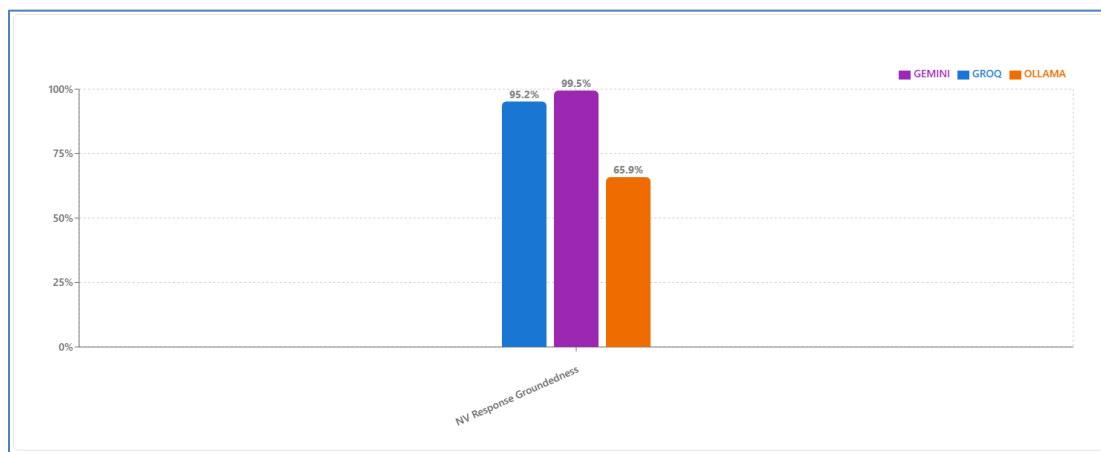


Figure 39: Bar Chart for NV Response Groundedness

Interpretation:

- **Gemini 99.5% > Groq 95.2% > Ollama 65.9%**

Gemini and Groq demonstrate very strong factual grounding, with most of their responses fully supported by retrieved evidence. Ollama's much lower score indicates a higher tendency toward unsupported or hallucinated content.

- **So what:**

Strong grounding reduces the risk of factual errors and boosts user trust. Ollama's gap here highlights the need for tighter retrieval-generation integration or post-processing checks to validate claims.

Radar Chart Comparison

To visually capture the distribution and consistency of performance across all three Nvidia metrics, we use a triangular radar chart representation per model.

Radar Chart — NV Metrics:

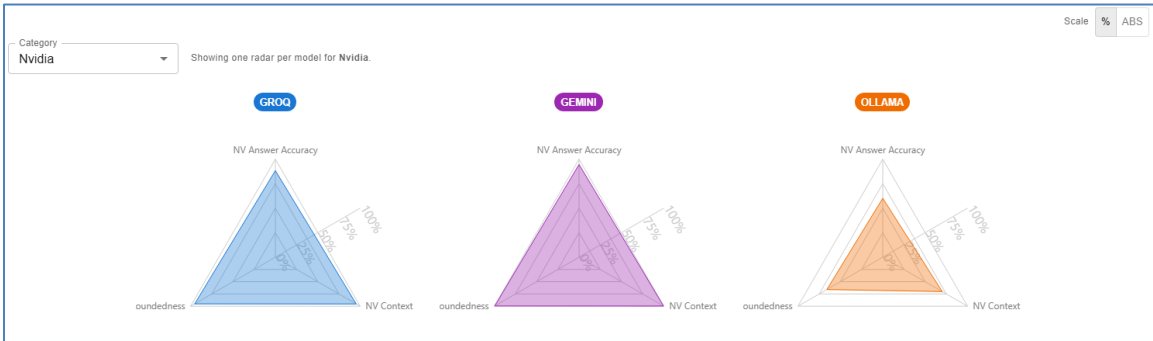


Figure 40: Radar chart for Nvidia metrics Category

Insights:

- **Gemini** nearly maxes out all three axes, showing both accuracy and contextual grounding.
- **Groq** forms a balanced triangle, albeit slightly lower in NV Answer Accuracy.
- **Ollama’s** triangle is notably compressed—especially along the accuracy axis—signaling room for improvement.

Metric Summary Table

Here is the detailed numerical comparison:

Metric	GROQ	GEMINI	OLLAMA
NV Answer Accuracy	88.3%	94.6%	60.1%
NV Context Relevance	95.4%	99.7%	70.2%
NV Response Groundedness	95.2%	99.5%	65.7%

Table 15: Aggregated Metric Table for Nvidia Category

Summary of Nvidia Metrics

Gemini emerges as the most consistent performer, achieving near-perfect scores across all three Nvidia evaluation axes. Groq follows closely, proving itself a highly reliable model. Ollama, while functional, struggles with accuracy and grounding, and may require post-processing strategies to enhance output reliability.

7.4 Natural Language Comparison Metrics

This section evaluates model outputs using a diverse set of **language-focused metrics** that assess factual alignment, semantic similarity, lexical overlap, and exactness of match with the

reference answers. Together, these metrics give a holistic picture of how well a model communicates accurate, relevant, and precisely worded responses.

The metrics span both **LLM-based** and **non-LLM-based** approaches:

1. Factual Correctness (F1)

Definition:

Factual Correctness measures the alignment between the claims in the generated response and the claims in the ground truth. Both response and reference are broken down into atomic claims by an LLM, followed by a **Natural Language Inference (NLI)** check to determine factual overlap.

Performance is quantified using **Precision**, **Recall**, and **F1 score**.

LLM-Based Scoring:

- **True Positive (TP)**: Claims in response present in reference.
- **False Positive (FP)**: Claims in response absent in reference.
- **False Negative (FN)**: Claims in reference absent in response.

Formula:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Associated Bar Chart:

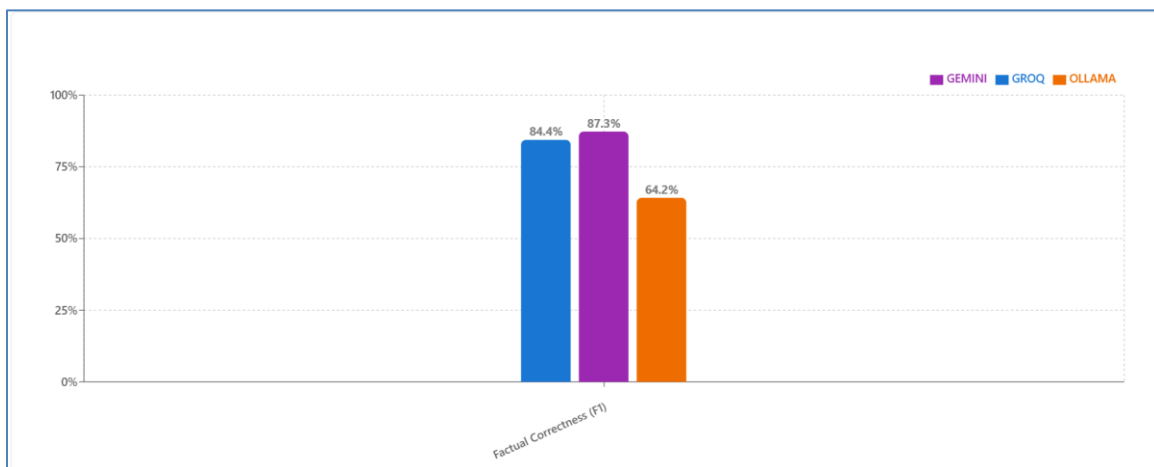


Figure 41: Bar Chart for Factual Correctness

Interpretation:

- **Gemini 87.3% > Groq 84.4% > Ollama 64.2%**

Gemini and Groq both exhibit strong claim-level accuracy, with Gemini slightly ahead. Ollama's notably lower score points to more frequent factual omissions or inaccuracies in its generated responses.

- **So what:**

High factual correctness ensures that generated answers not only address the query but also remain truthful. Ollama's performance gap suggests it may require additional fact-checking or refinement in claim extraction and validation.

2. Semantic Similarity

Definition:

This measures the semantic closeness between the generated answer and the ground truth, beyond exact word matches. The higher the score, the more meaning-equivalent the responses are.

Method:

A **bi-encoder model** computes sentence embeddings for both generated and reference answers, followed by cosine similarity.

Associated Bar Chart:

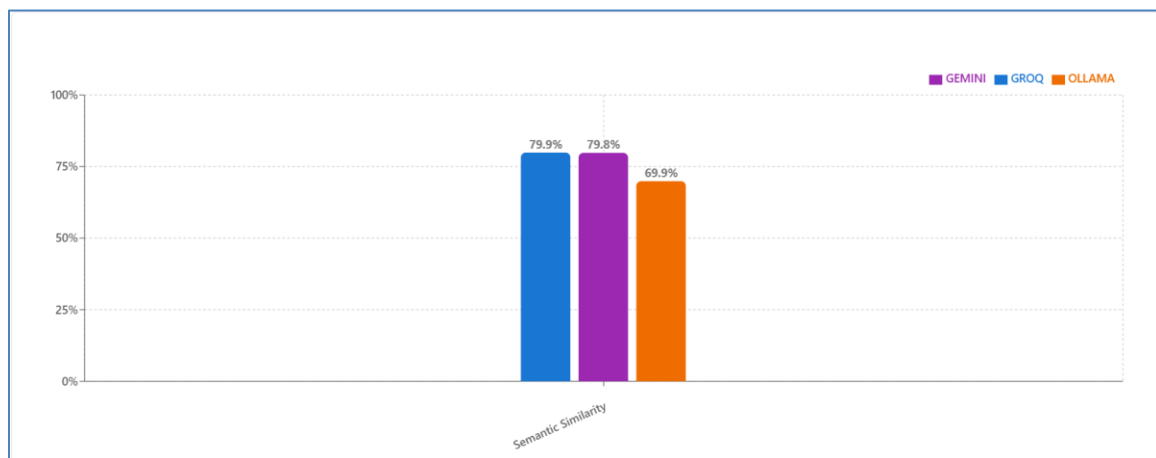


Figure 42: Bar Chart for Semantic Similarity

Interpretation:

- **Groq 79.9% \approx Gemini 79.8% » Ollama 69.9%.**

Groq and Gemini are consistently producing answers that *mean* the same thing as the references, even when they paraphrase. Ollama's ~10-point gap suggests more meaning drift (e.g., partial lists, hedging, or adding unrelated qualifiers).

- **Caveat:**

High semantic similarity \neq guaranteed factual accuracy; fluent hallucinations can still look semantically "close." That's why you also check Factual Correctness and Groundedness.

3. BLEU Score

Definition:

A non-LLM metric evaluating the **n-gram precision** between generated and reference texts, with a brevity penalty to discourage overly short answers. Originating from machine translation evaluation, it rewards both correctness and fluency.

Associated Bar Chart:

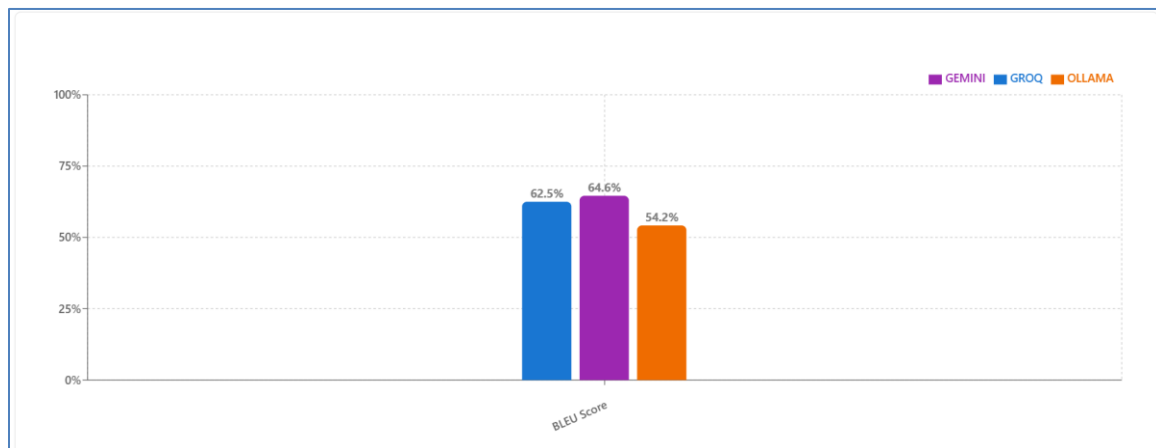


Figure 43: Bar Chart for BLEU Score

Interpretation:

- **Gemini 64.6% > Groq 62.5% > Ollama 54.2%**

Gemini most closely mirrors the reference wording. Groq is close, indicating mild paraphrasing. Ollama's lower BLEU suggests heavier paraphrase or added/omitted tokens that don't appear in the reference.

- **Why it matters:**

Higher BLEU is useful in tasks that expect *standardized phrasing* (definitions, canonical names). Lower BLEU isn't always "wrong," but combined with weaker String Presence and Exact Match, it signals both lexical variance *and* occasional omission.

4. ROUGE Score (F-measure)

Definition:

A non-LLM metric measuring **n-gram recall, precision, and F1 overlap** between generated and reference text. Useful for capturing coverage of reference content.

Associated Bar Chart:

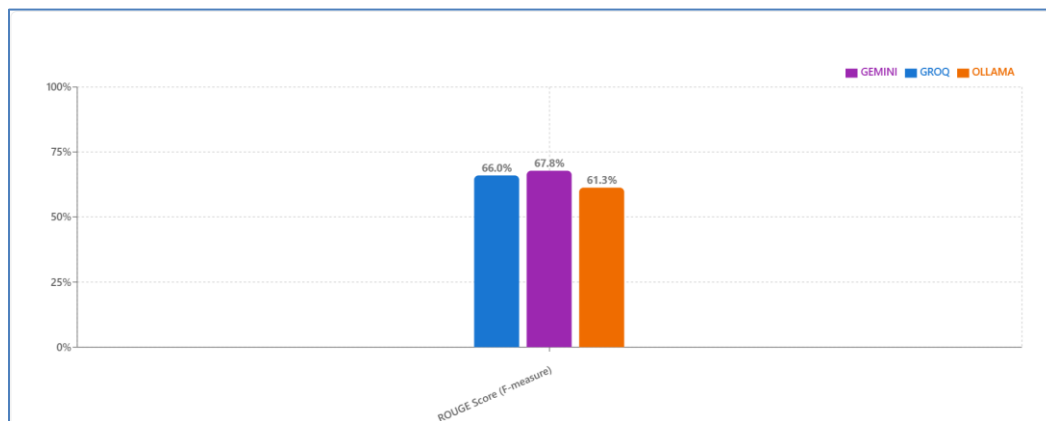


Figure 44: Bar Chart for ROUGE score

Interpretation:

- **Gemini 67.8% > Groq 66.0% > Ollama 61.3%**

Gemini captures slightly more of the reference content verbatim; Groq is close; Ollama drops more reference tokens, which often shows up when lists or multi-part answers are incompletely reproduced.

- **So what:**

ROUGE is a good "did we cover *all the pieces*?" signal. Here gap suggests Ollama more often misses one or two required items in list-style answers.

5. String Presence

Definition:

Checks whether the generated response contains the reference string exactly. Returns 1 if present, 0 otherwise, averaged over all test cases.

Associated Bar Chart:

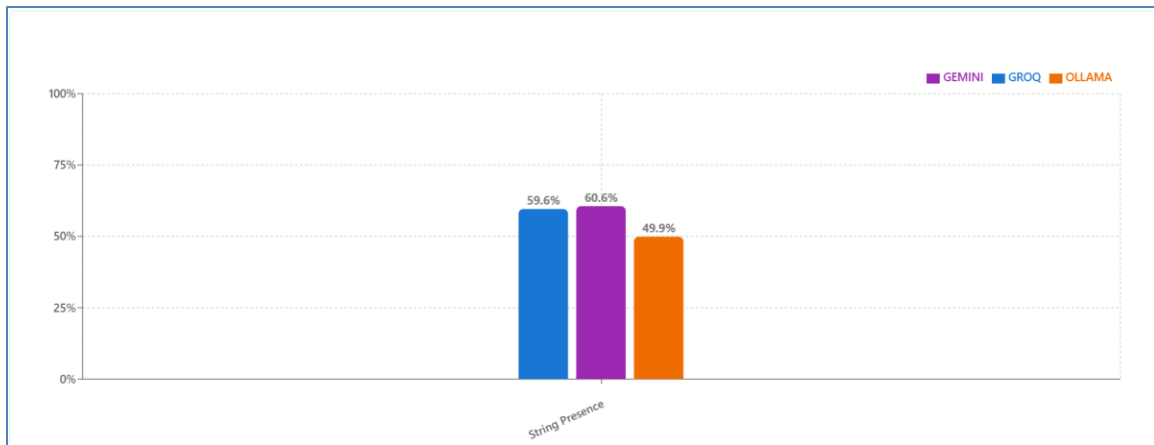


Figure 45: Barc Chart for String Presence

Interpretation & what this means for your results:

- **Gemini 60.6% ≈ Groq 59.6% » Ollama 49.9%**

Gemini/Groq include the exact key phrases in ~6/10 cases; Ollama only ~1/2. This lines up with their lower BLEU/ROUGE: Ollama paraphrases more and drops exact terms more often.

- **So what:**

If downstream logic key-matches phrases (entities, statutes, SKUs), Ollama will fail those checks more frequently.

6. Exact Match

Definition:

Strict equality check between the generated response and the reference text (word-for-word match). Returns 1 if identical, else 0.

Associated Bar Chart:

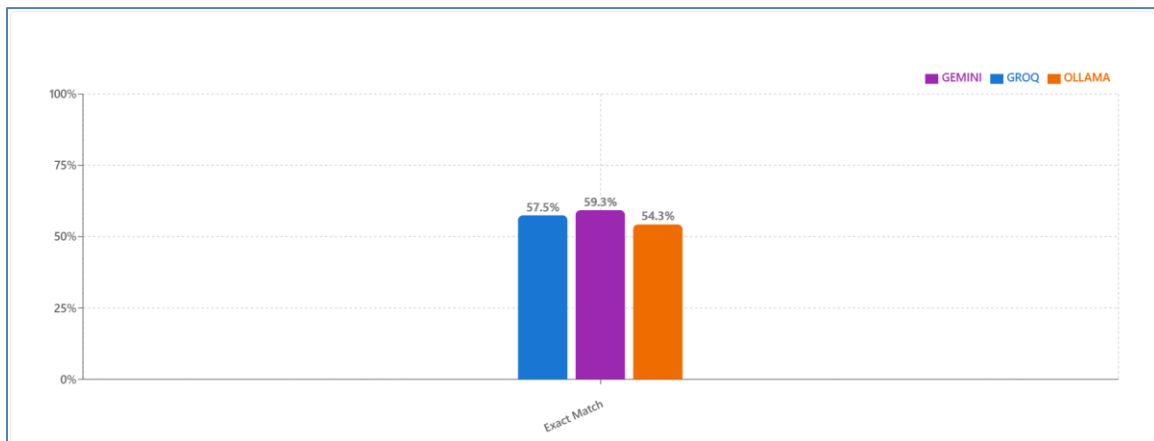


Figure 46: Bar chart for Exact Match

Interpretation:

- **Gemini 59.3% > Groq 57.5% > Ollama 54.3%**

These are *high* for free-form QA, implying many references are short/templated (definitions, lists, names) and Gemini is the most deterministic at reproducing them exactly. Ollama's lower EM mirrors its tendency to paraphrase or add filler.

- **So what:**

When exactness is mission-critical, Gemini is safest out-of-the-box; Groq is close. Ollama will need stricter formatting constraints.

Radar Chart — Language Metrics

The radar plots illustrate each model's profile across all six language metrics.

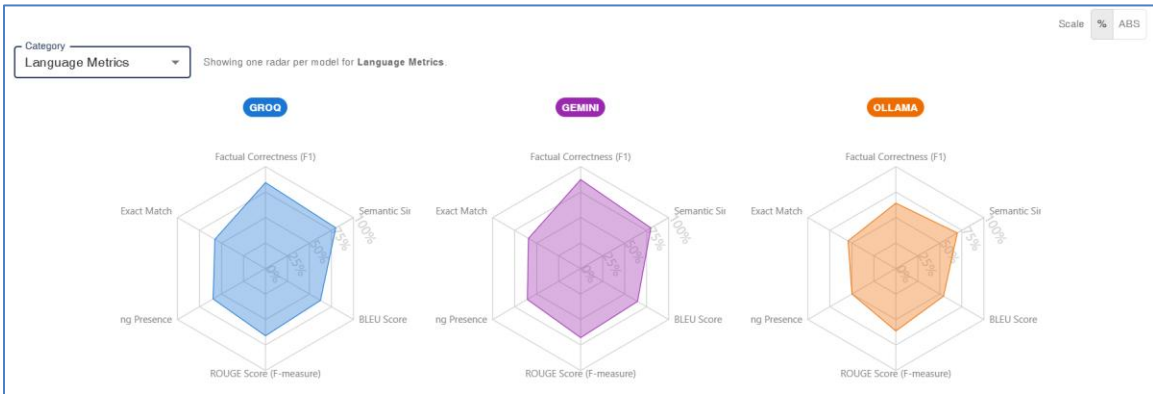


Figure 47: Radar chart for Nvidia metrics Category

- **Gemini** forms the most balanced, large-area polygon, showing consistent strength across all metrics.
- **Groq** is competitive in most metrics, with near-identical performance in semantic similarity and high factual correctness.
- **Ollama's** shape is compressed, particularly in **Exact Match** and **String Presence**, suggesting areas for improvement in precision wording.

Metric Summary Table

Metric	GROQ	GEMINI	OLLAMA
Factual Correctness	84.4%	87.3%	64.2%
Semantic Similarity	79.9%	79.8%	69.9%
BLEU Score	62.5%	64.6%	54.2%
ROUGE Score	66%	67.8%	61.3%
String Presence	59.6%	60.6%	49.9%
Exact Match	57.5%	59.3%	54.3%

Table 16: Aggregated Metric Table for Language Category

Summary of Language Metrics

Gemini consistently ranks first across most metrics, showing superior factual accuracy, lexical overlap, and exact matches. Groq is competitive and occasionally matches or nearly matches Gemini in semantic similarity. Ollama lags behind, particularly in metrics requiring precise reproduction of reference content, highlighting opportunities for refinement in both lexical fidelity and factual grounding.

Chapter 8

Conclusions and Recommendations for Future Work

8.1 Conclusions

This dissertation successfully demonstrated the design and implementation of a **Retrieval-Augmented Generation (RAG) based Question Answering system** tailored for educational content. The system combined **LlamaParse** for structured document parsing, a **semantic retrieval pipeline** using Sentence-BERT embeddings stored in a FAISS/ChromaDB index, and **multiple LLM backends** (Gemini, Groq, Ollama) to produce accurate, context-grounded answers. LlamaParse's ability to extract clean, hierarchically structured text from course PDFs significantly improved retrieval precision, allowing the system to capture both fine-grained details and high-level concepts. The retrieval-reader architecture ensured that the selected knowledge chunks were relevant, concise, and contextually aligned, while the answer generation process consistently adhered to the source material. Furthermore, the integration of **LLM-driven question generation** and **prompt-based summarization** expanded the system's scope beyond basic Q&A, enabling the creation of self-assessment materials and concise learning summaries.

The system's evaluation through **RAGAS**, with GPT-4 serving as a neutral judge, provided a robust and unbiased measurement of performance across factual correctness, context relevance, groundedness, and linguistic quality. Results showed Gemini as the most accurate and consistent model, Groq as a high-speed alternative with competitive quality, and Ollama as a functional choice for offline deployments. These evaluations confirmed that the platform met its primary objectives — delivering a reliable, scalable, and adaptable academic assistant capable of addressing the needs of both learners and educators.

Overall, the project not only fulfilled its technical goals but also demonstrated the potential of combining structured parsing, semantic retrieval, and LLM-based generation to create impactful educational tools.

8.2 Recommendations and Future Work

While the current system performs strongly, there are several promising directions for further improvement and broader applicability:

1. **Domain-Specific Fine-Tuning**

Adapting LLMs to the academic domain by fine-tuning on curated datasets from course handouts, past examinations, and instructor-prepared materials can significantly improve answer style, terminology alignment, and factual accuracy. This would reduce hallucinations and improve the system's authority in specialized subject areas.

2. **Enhanced Multi-hop Reasoning and Context Linking**

Extending retrieval beyond single-chunk answers to a **multi-hop approach**, where the system gathers and connects multiple supporting facts across different document sections, would enable it to tackle complex, reasoning-heavy queries. Leveraging LlamaParse’s hierarchical structure could also allow retrieval of broader context, such as including both a subsection and its parent section.

3. **Citation-Enriched Responses**

Incorporating inline citations with direct references to the source document’s page or section would enhance user trust and transparency. This would be especially valuable in academic settings, where verification of claims is essential, and could be combined with clickable references in the UI.

4. **Adaptive Question Generation**

Expanding the question generation module to personalize difficulty levels, question types (MCQ, descriptive, fill-in-the-blank), and focus areas based on user learning profiles can transform the system into a tailored tutoring tool, fostering active recall and deeper engagement.

5. **Support for Visual and Mathematical Content**

Many academic materials include diagrams, graphs, and equations that are integral to understanding concepts. Integrating OCR for diagrams and MathML/LaTeX parsing for equations would enable the system to answer visual or mathematical queries, such as explaining a figure or deriving a formula, thereby widening its applicability.

Chapter 9

Bibliography / References

- Binita Saha, Utsha Saha et al. (Jan 2025) “QuIM-RAG: Advancing Retrieval-Augmented Generation with Inverted Question Matching for Enhanced QA Performance”, arXiv: 2501.02702
- M. Shah, P. Zhou, et al. (Oct. 2024), “A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions” (*ResearchGate*)
- Prabhan Narak (Apr. 2024), “RAG on Complex PDF using LlamaParse, Langchain and Groq”, Medim.com blog
- Liu, Jintao; Ding, Ruixue; Zhang, Linhao; Xie, Pengjun; Huang, Fei (2024), “CoFE-RAG: A Comprehensive Full-chain Evaluation Framework for Retrieval-Augmented Generation with Enhanced Data Diversity”, arXiv: 2410.12248
- Rohan Anil, Sebastian Borgeaud, et al. (Dec 2023) “Gemini: A Family of Highly Capable Multimodal Models”, arXiv: 2312.11805
- Sonal Prabhune and Donald J. Berndt (Nov 2024), “Deploying Large Language Models with Retrieval Augmented Generation”, arXiv: 2411.11895v1
- Y. Liu, Y. Wang, et al. (Dec. 2023), “Retrieval-Augmented Generation for Large Language Models” arXiv: 2312.10997
- Yu, Hao; Gan, Aoran; Zhang, Kai; Tong, Shiwei; Liu, Qi; Liu, Zhaofeng (2024), “Evaluation of Retrieval-Augmented Generation: A Unified Survey”, arXiv: 2405.07437
- Bo Ni, Zheyuan Liu, et al. (Feb 2025), “Towards Trustworthy Retrieval Augmented Generation for Large Language Models: A Survey”, arXiv: 2502.06872
- Boci Peng, Yun Zhu et al. (Aug. 2024), “Graph Retrieval-Augmented Generation: A Survey”, arXiv: 2408.08921
- Es, Shahul; James, Jithin; Espinosa-Anke, Luis; Schockaert, Steven (2024), “RAGAs: Automated Evaluation of Retrieval Augmented Generation, in Proceedings of EACL Demonstrations”, arXiv: 2309.15217, pages 150–158.
- Alec M. (Mar. 2024), “Retrieval Augmented Generation with Groq API”, Groq blog
- Franklin Lee, Tengfei Ma (June 2025), “The Budget AI Researcher and the Power of RAG Chains”, arXiv: 2506.12317v1
- Jinhyuk Lee, Feiyang Chen et al. (Mar 2025), “Generalizable Embeddings from Gemini”, Xiv: 2503.07891

- Ryan Siegler (Apr 2025), “From documents to insights: Advanced PDF parsing for RAG”, KX blog
- Petko Georgiev, Ving Ian Lei, et al. (Mar 2024), “Gemini 1.5: Unlocking multimodal understanding across millions of tokens”, arXiv: 2403.05530
- Saad-Falcon, Jon; Khattab, Omar; Potts, Christopher; Zaharia, Matei (2023), “ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems”, arXiv: 2311.09476
- Wenqi Fan, Yajuan Ding et al. (Aug. 2024) “A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models” ACM digital library: doi/10.1145/3637528.3671470
- Monica Riedler, Stefan Langer (Oct. 2024), “Optimizing RAG with Multimodal Inputs for Industrial Applications”, arXiv: 2410.21943v1

Chapter 10

Appendix

Appendix A: Llama Ecosystem

A.1 LlamaParse – Advanced Document Parsing Capabilities

Feature	Description
Multi-modal Extraction	Extracts not just text but embedded images, diagrams, mathematical notation, and structured tables from complex PDFs.
Layout-Aware Parsing	Preserves document hierarchy (headings, subheadings, bullet lists) to maintain logical flow for RAG ingestion.
Hybrid Chunking Support	Output can be pre-chunked by semantic boundaries or structural markers (tables, sections) before embedding.
Cloud + Local Modes	Supports both cloud-hosted parsing via LlamaCloud API and self-hosted deployments for privacy-sensitive workflows.
Instruction-Tuned Parsing	Allows custom natural-language parsing prompts to influence how specific sections or elements are extracted.
Metadata-Rich Output	Each extracted element can be annotated with position, page number, and source type for traceability.

A.2 LlamaCloud Parsing Modes Overview

Mode	Description
Fast	Designed for speed and affordability—rapid parsing of simple documents.
Balanced	Provides a middle ground between parsing speed, cost, and output accuracy.
Premium	Highest accuracy, best for parsing complex layouts—powerful but latency/cost may be higher.
Parse Without AI	Outputs raw unstructured text (OCR supported), preserving original layout—minimal processing.
Parse with LLM	Applies initial parsing then uses an LLM to reconstruct structure and correct errors.
Parse with LVM	Captures screenshots and processes them via vision models (e.g., OpenAI, Anthropic), supports custom user models.
Agent Mode	Hybrid pipeline combining LLM and visual models using an agent-based workflow for maximum accuracy.

Appendix B: Extended RAGAS Metrics and Per-Question Drill-Down View

B.1 RAGAS Per-Question Drilldown Analysis View

This mode details the evaluation on a **question-by-question basis**, displaying:

- **Question ID and Text** – To identify the evaluation item.
- **LLM's Answer vs Ground Truth** – To compare generated and expected responses.
- **Metric Scores per Question** – Showing how each metric behaved for that specific query.

Such granular visibility highlights where a model excelled or underperformed, helping in targeted improvement efforts.



Figure 48: RAGAS per question drill down view

B.2 Additional RAGAS Evaluation Metrics

Metric Name	Description
Context Entities Recall	Measures the fraction of named entities from the reference that are successfully recalled in the retrieved context. Especially useful in fact-heavy use cases like tourism or historical QA, ensuring that key entities are preserved in retrieval.
Noise Sensitivity	Evaluates how often the system produces incorrect answers when using relevant or irrelevant retrieved documents. Lower scores indicate better resilience against noise in the context.
Response Relevancy	Determines how well the generated response aligns with the user's input. Penalizes incomplete or unnecessarily verbose answers.
MultiModal Faithfulness	Assesses factual consistency of the answer with both visual and textual context. A high score means all claims are supported by either modality.

MultiModal Relevance	Measures the relevance of the answer with respect to both visual and textual context. Ensures alignment of response content with provided materials.
Topic Adherence	Evaluates whether the AI stays within predefined domain boundaries, crucial for domain-specific conversational systems.
Tool Call Accuracy	Measures the accuracy of the AI in identifying and invoking the correct tools required to complete a given task, based on reference tool calls.
Agent Goal Accuracy	Checks if the AI achieved the user's intended goal. A binary metric (1 = achieved, 0 = not achieved) that can also be computed against a reference ideal outcome.
Non-LLM String Similarity	Uses traditional string distance measures (Levenshtein, Hamming, Jaro) to compare the reference and the response without relying on LLM evaluation.

Check list of items for the Final report

Is the Cover page in proper format?	Y
Is the Title page in proper format?	Y
Is the Certificate from the Supervisor in proper format? Has it been signed?	Y
Is Abstract included in the Report? Is it properly written?	Y
Does the Table of Contents page include chapter page numbers?	Y
Does the Report contain a summary of the literature survey?	Y
Are the Pages numbered properly?	Y
Are the Figures numbered properly?	Y
Are the Tables numbered properly?	Y
Are the Captions for the Figures and Tables proper?	Y
Are the Appendices numbered?	Y
Does the Report have Conclusion / Recommendations of the work?	Y
Are References/Bibliography given in the Report?	Y
Have the References been cited in the Report?	Y
Is the citation of References / Bibliography in proper format?	Y

Verified and Signed by Supervisor



Signed by Student

