

AUTOMATED QUESTION BUILDER APPLICATION USING GEN-AI

A PROJECT REPORT

Submitted by

NITHYA SHREE K [211421243113]

ARTHI R [211421243016]

NILA D [211421243111]

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2025

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **“AUTOMATED QUESTION BUILDER APPLICATION USING GEN-AI”** is the bonafide work of **“NITHYA SHREE K [211421243113], ARTHI R[211421243016] and NILA D[211421243111]”** who carried out the project work under my supervision.

SIGNATURE

**Dr. M. S. MAHARAJAN, M.E., Ph.D.,
GUIDE, SUPERVISOR**

ASSOCIATE PROFESSOR
DEPARTMENT OF AI&DS,

PANIMALAR ENGINEERING COLLEGE,
CHENNAI-600123.

SIGNATURE

**Dr. S. MALATHI, M.E., Ph.D.,
HEAD OF THE DEPARTMENT**

PROFESSOR
DEPARTMENT OF AI&DS,

PANIMALAR ENGINEERING COLLEGE,
CHENNAI-600123.

Certified that the above-mentioned students were examined in End Semester Project Work (21AD1811) held on .04.2025

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION BY THE STUDENTS

We, NITHYA SHREE K [211421243113], ARTHI R [211421243016] and NILA D [211421243111], hereby declare that this project report titled “AUTOMATED QUESTION BUILDER APPLICATION USING GEN-AI”, under the guidance of Dr. M.S. MAHARAJAN, M.E., Ph.D., is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr. P. CHINNADURAI, M.A., Ph.D.**, for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our Directors **Tmt. C.VIJAYARAJESWARI, Dr. C. SAKTHIKUMAR, M.E., Ph.D.** and **Dr. SARANYA SREESAKTHI KUMAR B.E., M.B.A., Ph.D.**, for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr. K. MANI, M.E., Ph.D.** who facilitated us in completing the project. We thank the Head of the AI&DS Department, **Dr. S. MALATHI, M.E., Ph.D.**, for the support extended throughout the project.

We would like to thank our supervisor **DR. M. S. MAHARAJAN**, coordinator **Dr. S. CHAKARAVARTHI**, and all the faculty members of the Department of AI&DS for their advice and encouragement for the successful completion of the project.

NITHYA SHREE K (211421243113)

ARTHI R (211421243016)

NILA D (211421243111)

ABSTRACT

The generation of educational content can now be automated because to developments in generative AI, which has important implications for evaluation and personalized learning. In order to generate a variety of questions with different levels of complexity, including multiple choice, coding, and paragraph-based questions, this work provides an automated question generator application. In order to ensure context and relevance, the application dynamically generates questions based on input themes using Groq's Lemma API, large language models (LLMs), and natural language processing (NLP) approaches. The application successfully generates a range of well-organized questions that are in line with input specifications, according to the results. This study shows how generative AI may improve educational technologies by automating content creation, which saves time and money on creating highquality questions.

Keywords—Groq’s Lemma API, Large language model, Natural language processing

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	iv
	LIST OF FIGURES	vii
	LIST OF ABBREVIATIONS	viii
1	INTRODUCTION	1
	1.1 Motivation	3
	1.2 Objective	4
	1.3 Division Related to the Project	6
	1.4 Contribution of the Work	9
2	LITERATUE SURVEY	11
3	SYSTEM REQUIREMENTS	25
4	SYSTEM ARCHITECTURE OF PROPOSED SYSTEM	32
	4.1 Data Flow Diagram	33
	4.2 Entity Relationship Diagram	38
	4.3 Architecture Diagram of Proposed System	44
5	PROPOSED SYSTEM IMPLEMENTATION	48
	5.1 Data Collection	
	5.2 Module 1: Web Page Development	49
	5.3 Module 2: Automatic Question Generator (AQG)	54
	5.4 Module 3: Backend & Database Management	61
		68
6	RESULT AND DISCUSSION	72
7	CONCLUSION AND FUTURE WORKS	88
	7.1 Conclusion	89
	7.2 Future Works	90
8	REFERENCES	91
	PUBLICATION	94
	APPENDIX	95

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE
4.1	0 Level Data Flow Diagram	33
4.2	1 Level Data Flow Diagram	34
4.3	2 Level Data Flow Diagram	36
4.4	Entity Relationship Diagram	38
4.5	Architecture Diagram	44
5.1	Workflow of Automated Question Generation	52
6.1	Paragraph Based Question Model	74
6.2	Output of Paragraph Based Question Model	74
6.3	MCQ Question Based Generator	76
6.4	Output of MCQ Question Based Generator	77
6.5	Coding Based Question Generators	79
6.6	Output of The Coding Based Question Generators	79
6.7	Trainer can Download the Questions	80
6.8	Graph Representation	85

LIST OF ABBREVIATIONS

SERIAL NO	ABBREVIATION	EXPANSION
1	LLM	Large Language Model
2	API	Application Programming Interface
3	NLP	Natural Language Processing
4	AQG	Automatic Question Generator
5	GPT	Generative Pretrained Transformer
6	BERT	Bidirectional Encoder from Transformer
7	RNN	Recurrent Neural Network
8	BART	Bidirectional And Auto-regressive Transformer
9	BLEU	Bilingual Evaluation Under Study Score
10	QG	Question Generation
11	QA	Question Answering
12	RDF	Resource Description Framework
13	LSTM	Long Short-Term Memory
14	MOOC	Massive Open Online Course
15	BASE	Business Information System Engineering
16	QTA	Question-type Awareness
17	QFA	Question-Focus Aware
18	UI	User Interface
19	NLTK	Natural Language Tool Kit
20	NER	Named Entity Recognition

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

In the modern educational landscape, efficiently managing vast amounts of data related to trainers, employees, and students presents a significant challenge. Traditional methods of handling academic and administrative information often rely on manual processes, leading to inefficiencies, increased errors, and difficulties in retrieving essential data. These issues hinder the smooth operation of educational institutions, making it crucial to implement an advanced, automated solution. To address these challenges, we propose the development of a university and school management application designed to streamline data management while enhancing educational processes through automation.

Our application is designed as a centralized platform that effectively stores and manages information related to trainers, employees, and students. It ensures seamless access to academic records, secure storage, and efficient organization of institutional data. By providing a structured system, the platform eliminates redundant administrative efforts and ensures smooth operations. Furthermore, to assist in the learning process, the system incorporates an Automatic Question Generation (AQG) feature powered by Generative AI. This innovative feature enables educators to create multiple-choice, coding, and paragraph-based questions automatically, significantly reducing manual effort while improving the efficiency and quality of assessments.

The backend of the application is developed using Flask and PHP, ensuring a robust and scalable framework that integrates seamlessly with databases. To facilitate efficient database management, we incorporate PHPMyAdmin, allowing structured storage, easy retrieval, and organized handling of institutional data. Additionally, for the AQG feature, we leverage Groq's Lemma API, an advanced AI-based tool that generates diverse and contextually relevant questions. By integrating this API, we provide an intelligent

mechanism for question generation, enhancing the assessment process and supporting educators in delivering high-quality evaluations effortlessly.

Beyond simplifying data management, this system plays a vital role in enhancing the overall educational experience by integrating automation and AI-driven assessments. The application is designed with a user-friendly interface that ensures intuitive navigation, making it accessible to administrators, trainers, and students alike. Secure access protocols guarantee data protection while facilitating smooth interaction with academic records and administrative information. Through automation, structured data management, and AI-powered assessments, the system supports daily academic and administrative activities efficiently, fostering a more intelligent, streamlined, and effective educational environment.

1.1 MOTIVATION

In the modern digital era, educational institutions are responsible for handling vast amounts of data related to students, faculty members, trainers, and administrative staff. This data encompasses academic records, attendance tracking, assessments, fee management, faculty schedules, and various other administrative functions. Traditionally, managing such extensive information has been a manual process, requiring significant time, effort, and resources. Manual data management is not only inefficient but also prone to human errors, misplacements, and redundancies, making it difficult to maintain accurate and up-to-date records. Moreover, the lack of automation in traditional methods poses challenges in organizing academic records, scheduling assessments, generating question papers, and retrieving essential information swiftly when needed.

To address these critical challenges, we propose the development of an advanced university/school management application that enhances the efficiency of academic institutions through automation and streamlined data management. This system aims to integrate key functionalities such as student enrollment, attendance tracking, grade monitoring, faculty management, and other essential administrative processes within a

single, user-friendly interface. A significant feature of this proposed solution is the incorporation of an Automatic Question Generation (AQG) system powered by Generative AI. The AQG feature is designed to automate the creation of question papers, reducing the manual workload for educators while ensuring the generation of diverse, well-structured, and contextually appropriate questions for various subjects and difficulty levels.

By leveraging AI-driven automation, this application will not only optimize administrative workflows but also contribute to improving the overall learning experience for students and educators. The automated question generation system will enable teachers to focus more on delivering quality education rather than spending extensive time on preparing assessments. Additionally, the seamless integration of data management functionalities will facilitate effortless record-keeping, improved accessibility, and real-time updates, ensuring that institutions can operate more efficiently. This project is envisioned as a transformative solution that modernizes traditional educational management practices, making them more reliable, scalable, and effective in meeting the evolving demands of the academic landscape.

1.2 OBJECTIVE

The primary objective of this project is to develop an intelligent and efficient university or school management system that leverages automation and artificial intelligence to streamline administrative and academic processes. This system aims to address key challenges faced by educational institutions and enhance overall functionality, usability, and security. The following objectives highlight the core functionalities and benefits of this project:

Centralizes data management for students, trainers, and employees

Managing vast amounts of data related to students, faculty, and administrative staff is a crucial aspect of educational institutions. Traditional methods often lead to data fragmentation, inconsistencies, and inefficiencies.

By implementing a centralized system, all academic and administrative information will be stored in a structured database, allowing easy access, updates, and organization. This will reduce the risk of data redundancy and ensure that stakeholders can retrieve accurate information without delays.

Automates question generation for academic assessments using Generative AI

One of the most time-consuming tasks for educators is the manual creation of assessments. This system integrates an Automatic Question Generation (AQG) feature powered by Generative AI, enabling the automatic generation of multiple-choice, coding, and paragraph-based questions.

This significantly reduces the workload on educators while maintaining the quality and integrity of evaluations.

Ensures secure and structured data storage with an easy retrieval system

Security and data integrity are paramount in educational institutions, where sensitive information such as student records, faculty details, and administrative data must be protected. This system employs secure storage protocols, including encryption and access control mechanisms, to safeguard institutional data.

Additionally, structured database management ensures that information is systematically stored, allowing for quick and efficient retrieval whenever required. This enhances institutional compliance with data protection regulations and prevents unauthorized access.

Enhances efficiency by reducing manual intervention in administrative and academic processes

Many routine tasks in educational institutions, such as record-keeping, attendance tracking, report generation, and assessment management, require significant manual effort. By integrating automation into these processes, the system minimizes human intervention, reducing errors, improving accuracy, and saving time.

This allows educators and administrators to focus on strategic planning, curriculum development, and student engagement rather than being burdened by repetitive tasks.

Provides a user-friendly interface for seamless interaction among administrators, trainers, and students

A well-designed, intuitive interface is essential to ensure that users of varying technical expertise can navigate the system effortlessly.

This project prioritizes usability by offering a responsive and visually appealing interface that enhances accessibility for all stakeholders, including administrators, trainers, and students.

The interface will provide personalized dashboards, role-based access controls, and seamless communication channels, ensuring that each user can efficiently perform their designated tasks without technical barriers.

1.3 Divisions Related to the Project

The project is organized into several key divisions, each specifically designed to enhance the overall performance, efficiency, and functionality of the system. These divisions focus on different aspects of university and school management, including user interaction, data handling, security, and automation, all of which work together seamlessly within a unified platform.

The **first division**, User Management Module, is integral to ensuring the security and accessibility of the system. It is tasked with managing the registration, authentication, and role-based access control for various users, including students, trainers, and administrators. This module ensures that only authorized individuals can access the appropriate features based on their designated roles. It also supports the creation of user profiles, managing login credentials, enabling password recovery, and implementing advanced security measures such as two-factor authentication (2FA) and CAPTCHA to safeguard against unauthorized access. By enforcing these strict access controls, the

system ensures that sensitive data remains secure while offering a seamless experience for users.

The **second division**, Data Management Module, is focused on the efficient storage, organization, and retrieval of academic and administrative data. This module ensures that the vast amount of institutional data, such as student enrollment records, faculty profiles, course schedules, attendance logs, and administrative reports, is properly structured and categorized for easy access and management. The system indexes and links this data to facilitate smooth navigation, prevent data redundancy, and maintain an organized database. This structured approach to data management ensures that all records are easily accessible and up-to-date, promoting operational efficiency and minimizing data inconsistencies.

The **third division**, Automatic Question Generation (AQG) Module, introduces an innovative AI-powered tool for automatically generating assessment questions. By utilizing Groq's Lemma API, this module can create a wide variety of question types, including multiple-choice, coding, and paragraph-based questions, with minimal manual intervention. The use of Generative AI helps reduce the workload involved in test creation, while ensuring a diverse and contextually relevant set of questions that meet educational standards. This feature enhances both teaching and evaluation by providing a broader range of high-quality, automatically curated assessment materials that can be easily adapted for various courses.

The **fourth division**, Assessment and Evaluation Module, streamlines the process of creating and managing online tests. It integrates directly with the AQG module to automatically generate test questions and supports an automated grading system for multiple-choice and coding questions. Trainers can also review and grade subjective responses manually. This module further offers detailed performance analytics, allowing educators to track student progress, identify learning gaps, and gain insights into areas where students need improvement. By leveraging data-driven insights, the system

enhances learning outcomes and supports continuous improvement in student performance.

The **fifth division**, Database Management Module, ensures the systematic storage, retrieval, and maintenance of large volumes of institutional data. This module employs PHPMyAdmin as the database management system to organize data efficiently, ensuring consistency and integrity across the platform. It supports database queries, indexing, and regular backups to prevent data loss or corruption. Additionally, it facilitates smooth data exchange between other system modules, ensuring that the platform operates as a cohesive and integrated system. The robustness of the database management module ensures that institutional data is always secure and accessible.

The **sixth division**, the Security and Access Control Module addresses the critical need for data protection and privacy. This division ensures that all login mechanisms are secure and that data is encrypted to protect sensitive academic and administrative information. It also implements role-based access controls to restrict unauthorized access, using strong authentication protocols like password hashing and multi-layered encryption. The module maintains audit logs to track user activities, enabling transparency and accountability. By enforcing stringent security measures, this module protects confidential records, such as student data and financial information, ensuring the overall integrity of the system.

Together, these divisions form a robust framework that enhances the reliability, security, and functionality of the educational management system. By integrating these specialized modules, the project provides a comprehensive solution that improves administrative efficiency, enhances the assessment process, and ensures the seamless operation of the institution. Each module plays a vital role in the overall success of the project, contributing to a unified platform that streamlines operations and delivers an effective educational experience.

1.4 Contributions of the Work

This project makes a substantial contribution to the education and technology sectors by merging automation, artificial intelligence, and efficient data management to improve the functioning of educational institutions. It addresses key challenges in managing academic and administrative processes by enhancing efficiency, accessibility, and security.

One of the core contributions is the development of a fully digital, centralized system for managing student records, faculty details, course schedules, and administrative data. Unlike traditional educational systems that rely heavily on manual record-keeping, this project eliminates inefficiencies, data loss, and mismanagement by digitizing these processes. This transition to a digital platform helps educational institutions reduce paperwork, improve organization, and ensure smoother, more efficient data retrieval.

Another significant contribution is the introduction of an AI-driven Automatic Question Generation (AQG) module, which automates the creation of assessments. By leveraging Groq's Lemma API, the system generates diverse, contextually relevant questions in various formats, including quizzes, coding problems, and subjective questions. This innovation greatly reduces the time and effort required by educators to create assessments, allowing them to focus more on teaching while ensuring that a wider variety of assessment materials are available for students, enriching the overall learning experience.

The project also enhances accessibility by utilizing Flask and PHP to create a web-based platform with a well-organized database. The use of PHPMyAdmin ensures that institutional data is stored efficiently and can be easily retrieved. This web-based interface provides administrators, faculty, and students with easy access to information from anywhere, enhancing the platform's usability and enabling smooth management of academic and administrative tasks.

Assessment efficiency is further optimized by the system's ability to automatically generate and categorize questions based on various factors such as subject matter,

difficulty, and format. This categorization ensures that assessments are fair, consistent, and diverse. The automated grading system offers instant feedback on auto-graded questions, helping students track their progress and identify areas that need improvement.

Security is another critical aspect of this project. By implementing advanced security measures, including role-based access control, encryption protocols, and authentication mechanisms, the system ensures that sensitive academic data is protected. Access is granted based on the user's role—whether student, educator, or administrator—thus preventing unauthorized access and ensuring the confidentiality and integrity of institutional records. Regular security audits and encrypted database storage further enhance the platform's reliability.

By integrating these advanced features, this project modernizes educational institutions, creating a more efficient and effective academic environment. It reduces administrative burdens, improves the assessment process, and ensures data security, allowing educators and administrators to focus on delivering high-quality education.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

The text explores Natural Language Processing (NLP) tools, focusing on spaCy, NLTK, and Sense2vec. spaCy provides components like Tokenizer, Tagger, Parser, and Named Entity Recognition (NER) to break text into tokens, assign POS tags, and analyze sentence structure using a dependency parser. NLTK supports tokenization, parsing, lemmatization, and POS tagging, making it widely used for processing human language data in educational contexts. Sense2vec, an extension of Word2Vec, generates context-aware word embeddings, improving word sense disambiguation and integrating with spaCy to enhance syntactic parsing while reducing computational overhead. Using these NLP methods, the paper also explains an Automatic Question Generator (AQG) that generates multiple-choice, fill-in-the-blank, and Boolean questions from textual input. This system helps educators and students by efficiently generating practice assessments, improving learning outcomes. The AQG model prioritizes efficiency, security, and user interaction, making it applicable in academic settings and competitive exam preparation. Future improvements include paraphrased question generation and automated answer validation systems, further enhancing adaptive learning environments.[1]

The development of Automatic Question Generation (AQG) has been greatly influenced by developments in machine learning and natural language processing (NLP). Traditional methods, such as rule-based and template-driven approaches, have been widely employed but often generate rigid, repetitive, and contextually limited questions. The use of deep learning architectures, especially transformer models like BERT and GPT, to improve contextual knowledge and relevance in question generation has become a more prominent focus of recent research. Using natural language processing (NLP) and the K-Nearest Neighbour (KNN) algorithm, one study investigated the creation of short-answer reading comprehension questions, showcasing how machine learning can enhance question diversity and quality. Additionally, advanced techniques such as

semantic pattern recognition, syntactic parsing, and entity tagging have been integrated to further refine and optimize AQG systems. These innovations have resulted in more accurate, diverse, and adaptive question sets, significantly reducing the manual workload for educators while improving the efficiency of automated assessments. Future research directions in AQG focus on enhancing personalized learning experiences, incorporating multimodal question generation (utilizing images and videos), and addressing bias mitigation to ensure greater adaptability, fairness, and inclusivity in AI-generated questions.[2]

The text discusses the architecture and functionalities of various Natural Language Processing (NLP) tools, primarily focusing on spaCy, NLTK, and Sense2vec. spaCy includes components like Tokenizer, Tagger, Parser, and Named Entity Recognition (NER), which break text into tokens and categorize them using POS tags, while also featuring a syntactic dependency parser for analyzing sentence structure. spaCy is extensively used for processing human language data, especially in educational situations, and enables a variety of NLP tasks, including tokenisation, parsing, lemmatisation, and POS tagging. Sense2vec, an extension of Word2Vec, generates vector representations of words based on their context, enhancing disambiguation of word senses and integrating with spaCy to improve syntactic parsing and reduce computational overhead. The AQG emphasizes efficiency, security, and user interaction, making it applicable in educational settings and competitive exam preparation, with potential future enhancements including paraphrased questions and automated answer checking systems. [3]

The BERTSUM model for text summarisation and WordNet for creating distractions are two examples of Natural Language Processing (NLP) approaches used in the paper's automated system for developing Multiple Choice Questions (MCQs). The system addresses the challenges educators face in manually creating MCQs, which is often time-consuming and labor-intensive. Key steps in the proposed system include loading raw text, summarizing it with the BERTSUM model to identify key information, extracting keywords using the Rapid Automatic Keyword Extraction (RAKE) method, generating

distractors based on keyword relationships through WordNet, and producing fill-in-the-blank type questions with keywords and distractors. The system has shown higher accuracy compared to previous models and aims to enhance the efficiency and effectiveness of MCQ generation, benefiting both teachers and students in e-assessments and exam preparation. Future improvements in NLP models like BERT are expected to further enhance the system's accuracy. [4]

The paper discusses the significance of contextual information in training language models for text generation tasks, specifically utilizing Long Short-Term Memory (LSTM) networks. It emphasizes that models trained on domain-specific vector spaces outperform those using general wiki embeddings. The study evaluates the effectiveness of generated text based on its semantic closeness to the context, using cosine similarity as the assessment method. The proposed model integrates context vectors—representations of the semantic meaning of input words—into the training process to enhance semantic coherence, addressing the limitations of traditional LSTMs. Two approaches to context vector extraction are investigated: Word Clustering, which makes use of Word2Vec embeddings, and Word Importance, which makes use of term frequency-inverse document frequency (tf-idf). While the context-based models show improved performance over baseline models, challenges remain in maintaining semantic relevance and syntactic structure in the generated text. The paper suggests potential applications in question answering systems and chatbots, and indicates future work may involve attention-based models like Transformers and different clustering methods for context extraction.[5]

To ensure that the generated questions are relevant and contextually accurate, the question creation process uses a three-step pipeline that includes unconditional response extraction, question generation, and question filtration. The model is trained using a concatenation of context, answer, and question tokens to improve quality and fortify the semantic connection between textual elements. Additionally, the use of large-scale

generative models, reaching 8.3 billion parameters, improves fluency, contextual awareness, and adaptability across different subjects and learning domains. During the answer extraction phase, NLP-based techniques identify key answers from a given text, ensuring that questions are relevant and precise. In the question generation stage, extracted answers are transformed into syntactically sound and meaningful questions using deep learning techniques. The final step, question filtration, applies heuristic and AI-based evaluation methods to eliminate redundant, ambiguous, or low-quality questions, ensuring a refined and high-quality output. By integrating state-of-the-art transformer architectures, this method ensures high-quality, contextually rich question generation with minimal human intervention. Future advancements could further enhance learning experiences by incorporating multimodal question generation, integrating images, videos, and adaptive learning techniques to create more dynamic AI-driven assessments tailored to individual needs. [6]

In order to train Question Answering (QA) models, the study introduces a unique unsupervised method for Question Generation (QG) that generates synthetic question-answer pairs using summarisation data. The suggested approach uses dependency parsing, named entity identification, and semantic role labelling on news summaries in contrast to conventional QG techniques that rely on templates or supervised learning, which might result in a high lexical overlap between generated questions and their source passages. This approach generates diverse questions that are semantically related to the original passages while minimizing lexical similarity.

Even with fewer training examples, the authors show that their approach dramatically improves the performance of QA models trained on synthetic data, yielding state-of-the-art results across a variety of datasets, including SQuAD1.1 and Natural Questions. The study also highlights the impact of factors such as beam size in decoding and question type distribution on QA system performance. Additionally, the method shows good transferability to out-of-domain datasets like NewsQA and BioASQ, indicating its

effectiveness in various contexts. Future work aims to refine QG heuristics and extend the approach to other languages. [7]

The study examines a transformer-based method for Question Generation (QG) that makes use of GPT-2, a single pre-trained language model. This approach simplifies the QG process by avoiding complex architectures and additional mechanisms. In terms of METEOR and ROUGE L scores, the authors show that their model performs better than conventional RNN-based Seq2Seq models, while also performing on par with more intricate models that make use of answer awareness.

The One Question Per Line (OQPL) model is evaluated in the study; it performs well in a number of measures, such as BLEU and ROUGE scores, but it frequently copies terms from the context, especially when it comes to creating identification-type questions because of dataset bias. It also identifies failure modes, such as word repetition and incomplete questions. The optimal context length for generating questions is found to be around 10 sentences, as longer contexts confuse the model. Incorporating answer-awareness did not enhance performance, indicating a need for a more explicit mechanism for its effective use.

Overall, the findings highlight the effectiveness of transformer-based finetuning techniques for robust question generation, while also noting the limitations of the current model's approach. Future work is suggested to explore larger GPT-2 models, evaluate performance on diverse datasets, and incorporate answer-awareness mechanisms, with potential benefits for natural language understanding tasks. [8]

The study introduces QG-Bench, a thorough benchmark that uses both automated metrics and manual review to assess paragraph-level question generating (QG) models. It demonstrates the T5 LARGE model's superiority in question generation, particularly when initialised with SQuAD data and refined on domain-specific datasets. The study critiques traditional metrics like BLEU4 and ROUGE L, advocating for METEOR and MoverScore for better alignment with human judgments on answerability. It emphasizes the need for improved domain adaptability and cross-lingual transfer in QG models,

while also addressing ethical considerations regarding biases in language models. Future work will focus on developing new evaluation metrics and exploring more complex QG scenarios.

Additionally, QG-Bench standardizes the evaluation of QG models by unifying various question answering datasets into a consistent format, covering multiple domains, styles, and eight languages. The authors emphasise the significance of both automatic and manual evaluation techniques and suggest strong QG baselines based on fine-tuning generative models. The paper discusses challenges in evaluating QG models and explores their domain adaptability and multilingual effectiveness. It also includes results from fine-tuning various QG models, comparing their performance across multiple metrics, and emphasizes the significance of context in QG tasks. The findings suggest that optimal parameter tuning is crucial for improving QG model performance, and larger models tend to perform better in generating high-quality questions. [9]

The paper offers a thorough summary of developments in Automatic Question Generation (AQG) systems, which are primarily concerned with producing questions from images or text. The three primary use-cases for AQG techniques are conversational question generation, visual question generation, and standalone question generation. The paper discusses various approaches, including rulebased methods that utilize ontologies and templates, as well as neural networkbased techniques, particularly encoder-decoder architectures using RNNs and transformer models like GPT-2 and BERT.

The use of evaluation metrics such as BLEU, METEOR, ROUGE, and CIDEr to gauge model performance is covered, in addition to human assessments that emphasise relevance and fluency. Future study directions are identified in the paper's conclusion, including the necessity of high-quality datasets and the investigation of multimodal methodologies to improve the relevance and naturalness of generated queries. [10]

The paper discusses advancements in neural question generation (QG) models, addressing limitations in existing models that struggle with capturing sentence-level semantics and effectively utilizing answer position-aware features. The authors propose

a new model that incorporates two key modules: sentence-level semantic matching and answer position inferring, which enhance the model's understanding of global question semantics and the relevance of answer positions. Additionally, an answer-aware gated fusion mechanism is introduced to improve the initial state of the decoder.

The research employs a multi-task learning approach, combining various loss functions to boost performance on datasets like SQuAD and MS MARCO. Experimental results demonstrate that the proposed model outperforms state-of-the-art models in generating semantically relevant and accurate questions, as measured by metrics such as BLEU, METEOR, and ROUGE-L. Human evaluations further indicate that the model produces questions with improved semantic matching, fluency, and syntactic correctness compared to baseline models. Overall, the findings suggest that leveraging sentence-level semantics and answer position awareness significantly enhances question generation tasks.[11]

The paper discusses a transformer-based approach to Question Generation (QG) using a single pre-trained language model, specifically GPT-2. This approach simplifies the QG process by avoiding complex architectures and additional mechanisms. The authors demonstrate that their model outperforms traditional RNN-based Seq2Seq models in terms of METEOR and ROUGE L scores, while achieving comparable performance to more complex models that utilize answer-awareness.

The study evaluates the One Question Per Line (OQPL) model, which excels in various metrics like BLEU and ROUGE scores but tends to copy phrases from the context, particularly generating identification-type questions due to dataset bias. It also identifies failure modes, such as word repetition and incomplete questions. The optimal context length for generating questions is found to be around 10 sentences, as longer contexts confuse the model. Incorporating answer-awareness did not enhance performance, indicating a need for a more explicit mechanism for its effective use.

Overall, the findings highlight the effectiveness of transformer-based finetuning techniques for robust question generation, while also noting the limitations of the current

model's approach. Future work is suggested to explore larger GPT-2 models, evaluate performance on diverse datasets, and incorporate answer-awareness mechanisms, with potential benefits for natural language understanding tasks.[12]

The paper discusses the development of a question summarization model tailored for consumer health questions, addressing challenges related to overly descriptive and complex queries. It proposes an abstractive summarization approach that leverages transformer models, enhanced with knowledge of medical entities and question types. The authors introduce multiple Cloze tasks to improve the model's ability to identify key medical entities and question focuses, resulting in more informative summaries. The proposed model outperformed existing state-of-the-art methods on the MEQSUM benchmark, showing significant improvements in generating concise and relevant question summaries.

The study emphasizes the importance of understanding the intent behind consumer health questions for effective question answering systems. It also presents a multi-task learning approach that integrates question-type knowledge through Explicit and Implicit Question-Type Awareness (QTA). The model uses a shared encoder from the MiniLM architecture to classify questions into seven categories and summarize them, achieving significant improvements in ROUGE scores.

Additionally, the paper describes a Question-Focus Aware Model (QFA) and a Question-Type Aware Model (QTA) that enhance summarization by employing various masking techniques and incorporating question-type knowledge. The results indicate that the Explicit QTA model outperforms others, improving the performance of Information Retrieval-based QA systems. Manual evaluations confirm that the proposed models generate more accurate and relevant summaries compared to baseline models. The study concludes that combining question focus and type awareness can significantly enhance the summarization of complex health-related questions, with future work aimed at further integrating these approaches.[13]

Fine-tuning a model for question generation involves training it further on domain-specific data while optimizing its parameters. One effective approach is utilizing a task prefix, which provides explicit instructions to the model. In this case, the prefix "generate question:" is added at the beginning of the input text, guiding the model to focus on producing contextually relevant questions. For instance, if the input is "generate question: The process of photosynthesis allows plants to convert sunlight into energy," the model may generate "What is the primary function of photosynthesis in plants?" This structured approach helps the model associate the input with the expected output format, enhancing its question-generation capabilities. The fine-tuning process involves adjusting the model's weights and hyperparameters to improve performance. Key steps include dataset preparation, where text passages and corresponding questions are used for training, and optimization techniques, where parameter adjustments help refine question quality. Additionally, different model architectures like BART (BASE and LARGE) can be fine-tuned with optimized parameter settings to enhance accuracy and fluency in generated questions.[14].

ChatGPT offers a powerful tool for educators by enabling the generation of open-ended question prompts that align with learning objectives and success criteria. By leveraging its capabilities, teachers can craft questions that encourage critical thinking and deeper engagement with the subject matter. Additionally, ChatGPT can assist in designing high-quality rubrics that clearly define the specific tasks students must complete to achieve various levels of competence. These rubrics help ensure consistency in assessment and provide students with a structured understanding of expectations. Moreover, generative AI-driven evaluation systems can integrate ongoing feedback into the learning process, fostering a more adaptive and personalized educational experience. By utilizing unique and innovative artifacts, these AI-powered systems support a dynamic and responsive learning environment where students receive continuous insights into their progress.

At the core of ChatGPT lies the Transformer architecture, a type of neural network well-suited for natural language processing (NLP) tasks. This architecture allows the model

to learn from extensive text datasets, including novels, articles, and other written materials, to generate text that mimics human writing. The model predicts the next word based on the input and previously generated words, processing data in a sequential manner. To generate contextually appropriate and coherent responses, ChatGPT employs attention mechanisms that help it focus on the most relevant parts of the input text. These attention mechanisms enable the model to understand nuances in language, maintain coherence, and adapt its responses dynamically based on the given prompt. By doing so, ChatGPT ensures that its outputs remain logical, relevant, and suitable for a wide range of educational and conversational applications.[15]

The automatic question generation process is implemented using a computational model that integrates the k-Nearest Neighbors (k-NN) algorithm and Natural Language Processing (NLP) techniques to transform text into meaningful questions. The methodology involves multiple stages, starting with scraping news articles from reputable sources, followed by tokenization to break down text into smaller units. The text is then converted into a tree structure, allowing for simple sentence extraction to facilitate question formation. Once the sentences are selected, they undergo a question generation process, which involves cleaning question sentences, transforming them into Part-of-Speech (POS) tags, and calculating the distance between test data and training data using numerical representations. Additionally, synonym replacement enhances variation in question wording, while grammar correction techniques improve fluency and accuracy. To refine the output, duplicate question sentences are removed, ensuring a diverse set of generated questions.

The experiment conducted on the automatic questionZ generation system evaluates the quality of generated questions based on grammatical correctness, answer existence, and difficulty index. The study demonstrates a grammatical correctness rate of 76.19%, indicating that the system effectively generates syntactically accurate questions. Furthermore, expert evaluations yield an overall average score of 63.23%, reflecting improvements in question quality and relevance compared to traditional methods. The

primary objective of this research is to streamline the question creation process for educational applications, particularly for English proficiency tests such as IELTS. The study highlights the potential of NLP-driven question generation systems in enhancing learning assessments by automating question creation with improved coherence, linguistic accuracy, and contextual relevance, thus making it a valuable tool for educators and test developers [16]

The proposed system for automatic multiple-choice question (MCQ) generation employs a combination of advanced natural language processing (NLP) techniques to efficiently create high-quality assessment questions. The process begins with text summarization, where the BERT (Bidirectional Encoder Representations from Transformers) algorithm is used to condense lengthy input text while retaining key information. This step ensures that the most important concepts are extracted, which serves as a foundation for generating meaningful questions. Once the text is summarized, keyword extraction is performed using the Rapid Automatic Keyword Extraction (RAKE) method, which identifies the most relevant terms from the text. The RAKE algorithm first eliminates stopwords and applies phrase delimiters to extract key phrases, ensuring that only significant terms are selected for further processing. These keywords act as the focal points around which questions are built, enhancing the contextual relevance of the generated MCQs.[17]

After keyword extraction, the sentence mapping step aligns the identified keywords with corresponding sentences from the summarized text. This ensures that each keyword is placed in its correct contextual framework, which is essential for generating coherent and meaningful questions. To make the MCQs more challenging and effective, distractor generation is carried out using WordNet, a lexical database that provides semantically related words. High-quality distractors significantly influence the difficulty level of MCQs, as they ensure that incorrect options appear plausible while avoiding ambiguity. Finally, the output generation phase involves constructing fill-in-the-blank questions,

where keywords are replaced by blanks, accompanied by a set of answer choices, including the correct answer and well-formed distractors. By integrating these techniques, the system automates the MCQ generation process, providing an efficient and scalable approach to educational assessments while maintaining question diversity and contextual accuracy[17].

The proposed methodologies for automatic question generation incorporate a variety of rule-based and natural language processing (NLP) techniques to ensure that the generated questions are contextually accurate and meaningful. The rule-based methodology plays a crucial role in automating question generation by considering both the syntactic and semantic structure of sentences. This approach utilizes reliance-based rules, Named Entity Recognition (NER)-based methods, and Semantic Role Labeling (SRL) techniques to construct grammatically correct and contextually relevant questions. These methodologies allow the system to identify key entities and concepts, making it possible to create a diverse set of questions from any given text.

To further enhance the accuracy of question generation, NLP techniques such as pre-processing, key phrase extraction, and syntactic/semantic analysis are integrated into the system. Term Frequency-Inverse Document Frequency (TF-IDF) is used for extracting important key phrases, while WordNet assists in generating triplets for forming well-structured questions. Additionally, syntactic and semantic analysis techniques, including Part-of-Speech (POS) tagging and chunking, help analyze sentence structures and extract the most relevant components for question formation. By mapping appropriate 'WH' question words to the processed text, the system ensures the generation of diverse question types, including multiple-choice, fill-in-the-blank, and Boolean-type questions. The use of the Natural Language Toolkit (NLTK) for sentence splitting and Sense2vector for distractor generation further refines the question quality. Ultimately, automatic question generation reduces the time and effort required for educators and learners to create meaningful assessments, making the process more efficient while maintaining high accuracy and relevance in the generated questions[18].

CHAPTER 3

SYSTEM REQUIREMENTS

CHAPTER 3

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS

The hardware requirements of the Employee Attrition Analysis system are specified below for both the server and the client machines. These requirements ensure smooth operation, efficient processing, and reliable performance across various tasks.

SERVER

Processor

A quad-core processor or higher with a minimum clock speed of 2.5 GHz is necessary for handling multiple concurrent requests and data processing tasks. The multi-core processor ensures that the server can efficiently manage the complex computational needs of the application, especially when dealing with large datasets or running machine learning models for employee attrition predictions.

Memory (RAM)

A minimum of 8 GB RAM is required for the server to function efficiently, but 16 GB is recommended for environments where large datasets or high concurrency are anticipated. The additional memory capacity ensures that the system can handle multiple simultaneous user requests, complex queries, and data processing tasks without lag.

Storage

A 500 GB SSD is necessary to ensure quick data access and efficient storage of large datasets, user-generated content, logs, and other essential files. SSDs offer faster read/write speeds compared to traditional hard drives, which is crucial for database queries, processing real-time data, and storing analysis results for rapid retrieval.

Network

A high-speed, stable internet connection is critical for handling API calls, online interactions, and real-time data access. A reliable network ensures that the server can efficiently communicate with external services, handle multiple user requests, and transmit large data files without delays.

CLIENT MACHINE (USER)

Processor

The client machine requires a dual-core processor or higher with a minimum clock speed of 2.0 GHz. This is sufficient for accessing the web interface, browsing, and interacting with the application, ensuring smooth performance for day-to-day usage.

Memory (RAM)

A minimum of 4 GB RAM is recommended for the client machine. This amount of memory allows for efficient multitasking, browsing, and interacting with the Employee Attrition Analysis application, ensuring that the user experience remains fluid without significant delays.

Storage

The client machine should have at least 2 GB of free disk space. This space is used for caching, temporary files, and storing any locally saved data, which can speed up future interactions with the system.

Internet

A stable internet connection is crucial for accessing the system online and interacting with the web application. A strong connection ensures that data is transmitted and received efficiently, providing a seamless user experience.

SOFTWARE REQUIREMENTS

The software components are essential for the functioning, deployment, and optimization of the Employee Attrition Analysis system. These tools and technologies ensure that the system is scalable, secure, and user-friendly.

MySQL

MySQL is an open-source relational database management system (RDBMS) that uses SQL to manage and manipulate data in tables.

Role in AQG: MySQL serves as the central database for storing important data, including user information, preferences, generated questions, and application settings. It enables structured data storage, efficient retrieval, and manipulation of information.

Data Storage: It stores data in relational tables, organizing it into different categories such as employee details, analysis results, and historical records. It supports operations like querying, updating, deleting, and inserting data, ensuring smooth operation of the system.

Relationships: MySQL allows the creation of relationships among different tables, which is essential for structuring data efficiently. For example, relationships between employee records, departments, and attrition data can be easily managed to ensure data integrity.

Scalability: MySQL is designed to scale well, handling large datasets and a high number of concurrent requests. This scalability ensures that as the system grows, it will still function smoothly without performance degradation.

Python

Python is a powerful, high-level programming language known for its simplicity and versatility. It is widely used in data science, AI, and web development.

Flask Framework: Python, with the Flask framework, will handle backend processes, API requests, and communication with external services like Groq's Lemma

API for NLP-based question generation. Flask's minimalistic design makes it ideal for lightweight web applications and microservices.

Data Processing: Python's extensive libraries (e.g., Pandas, NumPy) support data processing, manipulation, and analysis. For employee attrition analysis, Python's tools allow efficient handling of datasets, performing statistical analysis, and creating predictive models.

Ease of Use: Python's readable and clean syntax makes it easy for developers to quickly write and maintain code, reducing development time. This is important for rapid prototyping and iterative improvements in the analysis system.

Extensive Libraries: Python offers a vast ecosystem of libraries, particularly for natural language processing (NLP), artificial intelligence (AI), and machine learning (ML), which support advanced tasks like data processing and employee attrition predictions.

Cross-Platform: Python is compatible with multiple operating systems, which ensures that the application can be deployed on various platforms without significant adjustments.

PHP

PHP is a server-side scripting language commonly used in web development and database management.

User Authentication: PHP is ideal for managing user authentication processes such as registration, login, and role-based access control. It ensures that users can securely access the system and interact with the data based on their roles.

Database Operations: PHP facilitates interaction with the MySQL database, retrieving, storing, and updating data for user sessions, preferences, and application settings. This ensures smooth integration between the frontend and backend of the system.

Database Integration: PHP is commonly used in conjunction with MySQL for managing data, making it a crucial component in handling dynamic content and user interactions in the Employee Attrition Analysis system.

Compatibility: PHP is widely supported by web servers, ensuring that it can be deployed on a variety of hosting platforms without compatibility issues.

Community Support: PHP has a large and active community, providing abundant resources such as documentation, forums, and third-party libraries, which simplifies development and troubleshooting.

HTML, CSS, JavaScript

These are the foundational technologies for building the frontend user interface (UI) of web applications.

HTML: HTML (Hypertext Markup Language) structures the content on the web page. It defines elements like headers, paragraphs, and tables, which organize the data in a meaningful way for the users.

CSS: CSS (Cascading Style Sheets) is used to style HTML elements, controlling the layout, colors, and visual appearance of the web interface. It enhances the user experience by making the application aesthetically pleasing and easier to navigate.

JavaScript: JavaScript enables interactivity and dynamic content updates on web pages. It handles features like form validation, dynamic content loading via AJAX, and event handling, improving user engagement and responsiveness.

Responsive Design: CSS frameworks like Bootstrap, combined with JavaScript, ensure that the AQG system is responsive and works across various screen sizes and devices, offering a seamless experience on desktops, tablets, and smartphones.

Dynamic Content: JavaScript enables real-time data updates and user interaction without requiring page reloads, making the system more interactive and efficient.

Visual Studio Code (VS Code)

VS Code is a lightweight, extensible code editor used for software and web development. It is particularly useful for developers working with multiple programming languages.

Development Environment: VS Code will be the primary integrated development environment (IDE) for writing, testing, and debugging code across Python, PHP, HTML, CSS, and JavaScript, ensuring that the development process remains smooth and organized.

Extensions: The IDE supports a wide range of extensions that improve syntax highlighting, code formatting, and debugging capabilities, making it easier to work on large projects with multiple technologies.

Multi-Language Support: VS Code supports various languages, enabling developers to seamlessly switch between Python, PHP, JavaScript, and other languages within the same editor.

Integrated Terminal: The integrated terminal allows developers to execute commands directly from the IDE, which is useful for running Python scripts, managing databases, or executing server-side PHP operations.

Git Integration: VS Code comes with built-in Git support, enabling developers to easily track changes in their code, collaborate with team members, and manage version control directly within the editor.

XAMPP

XAMPP is a popular open-source web server stack that simplifies local development by providing Apache, MySQL, PHP, and Perl in one package.

Local Server: XAMPP provides a local environment for testing and development, allowing developers to host and run the AQQ application on their local machines without needing an active internet connection. This is useful for initial testing and debugging.

Database Management: XAMPP includes PHPMyAdmin, a web-based interface that simplifies database management tasks, such as creating, modifying, and deleting MySQL databases and tables.

All-in-One Package: By bundling Apache, MySQL, and PHP together, XAMPP streamlines the setup process for a development environment, reducing the complexity of configuring individual components.

Local Testing: XAMPP enables local testing of the AQG system before deployment to a live server. This allows developers to verify functionality and ensure the system operates as expected in a controlled environment.

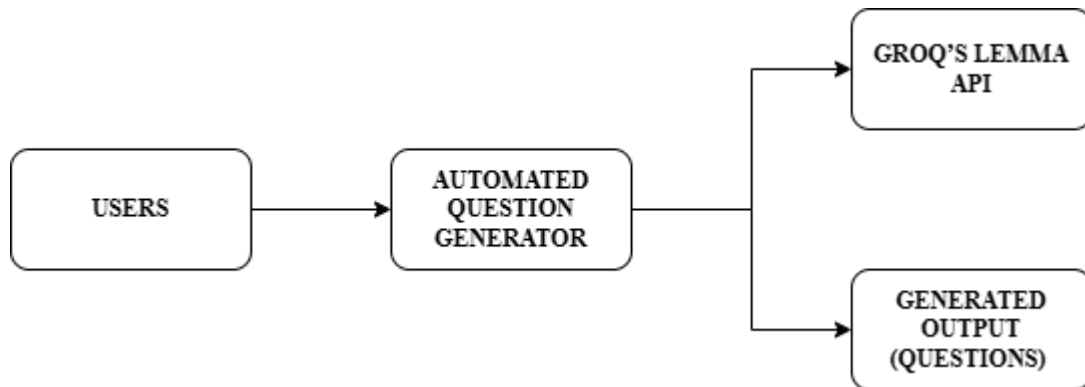
CHAPTER 4

SYSTEM ARCHITECTURE OF PROPOSED SYSTEM

CHAPTER 4

SYSTEM ARCHITECTURE OF PROPOSED SYSTEM

4.1 Data Flow Diagram



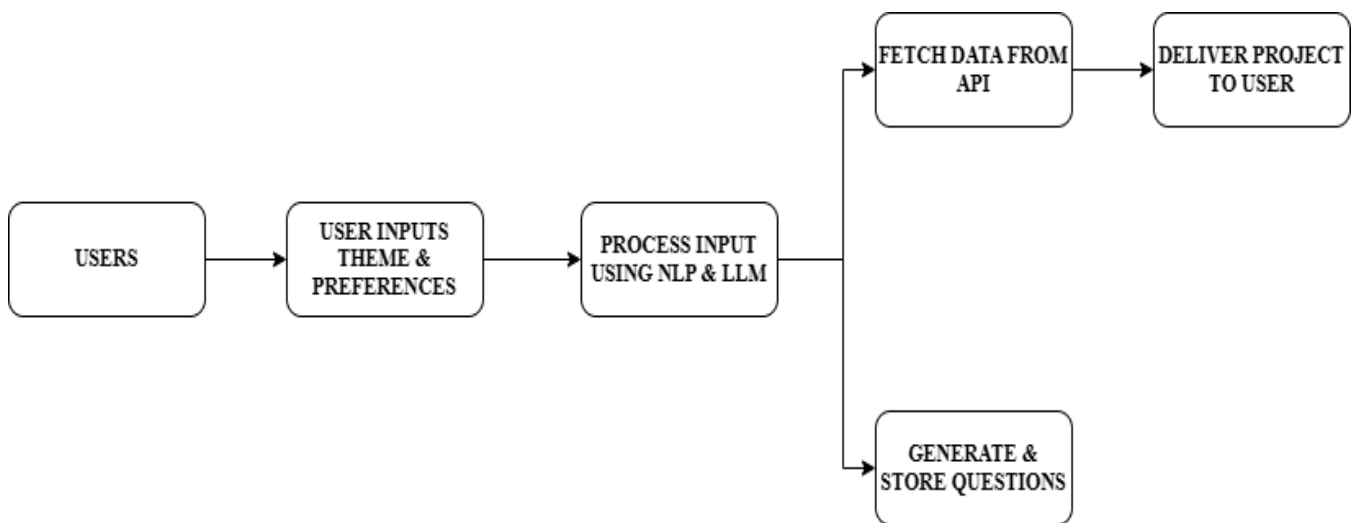
Figure(4.1): 0 level data flow diagram

The Automated Question Generation System streamlines the process of creating questions by leveraging AI-based tools. Users provide input, such as topics or text, which is processed by the Automated Question Generator. This system then communicates with Groq's Lemma API, a powerful linguistic and AI-based service, to analyze and refine the input. The API ensures that the generated questions are contextually accurate and relevant. Once processed, the Generated Output (Questions) is delivered back to the system, making it available for various applications such as quizzes, assessments, and training modules. This automation reduces manual effort and enhances question diversity, ensuring high-quality learning materials.

By integrating AI-driven question generation, this system provides a more efficient and scalable solution for educators, trainers, and organizations. The automation ensures quick adaptation to different subjects and learning objectives, making assessments more dynamic and engaging. Additionally, the use of Groq's Lemma API allows for improved language understanding, generating grammatically sound and conceptually valid questions. This approach not only saves time but also enhances the overall learning experience by

providing well-structured and meaningful questions. Ultimately, the system transforms traditional content creation methods, making them faster, more precise, and highly adaptable to modern educational needs.

Furthermore, the **Automated Question Generation System** can be integrated into various educational platforms, enabling seamless content creation for online learning, corporate training, and academic assessments. The system's ability to generate diverse question types—such as multiple-choice, short answer, and fill-in-the-blanks—enhances its applicability across different domains. Additionally, by leveraging **Groq’s Lemma API**, the system can continuously improve question quality based on user feedback and real-time data analysis. This ensures that the generated questions remain relevant, challenging, and aligned with evolving learning objectives. As a result, institutions and educators can focus more on interactive teaching and personalized learning experiences while relying on AI-powered automation to handle the repetitive task of question creation.



Figure(4.2): 1 level data flow diagram

The Automated Project and Question Generation System allows users to customize content by providing specific themes and preferences. The process begins when Users input their desired project theme and preferences, which serve as the foundation for content creation. This input is then processed using Natural Language Processing (NLP) and Large

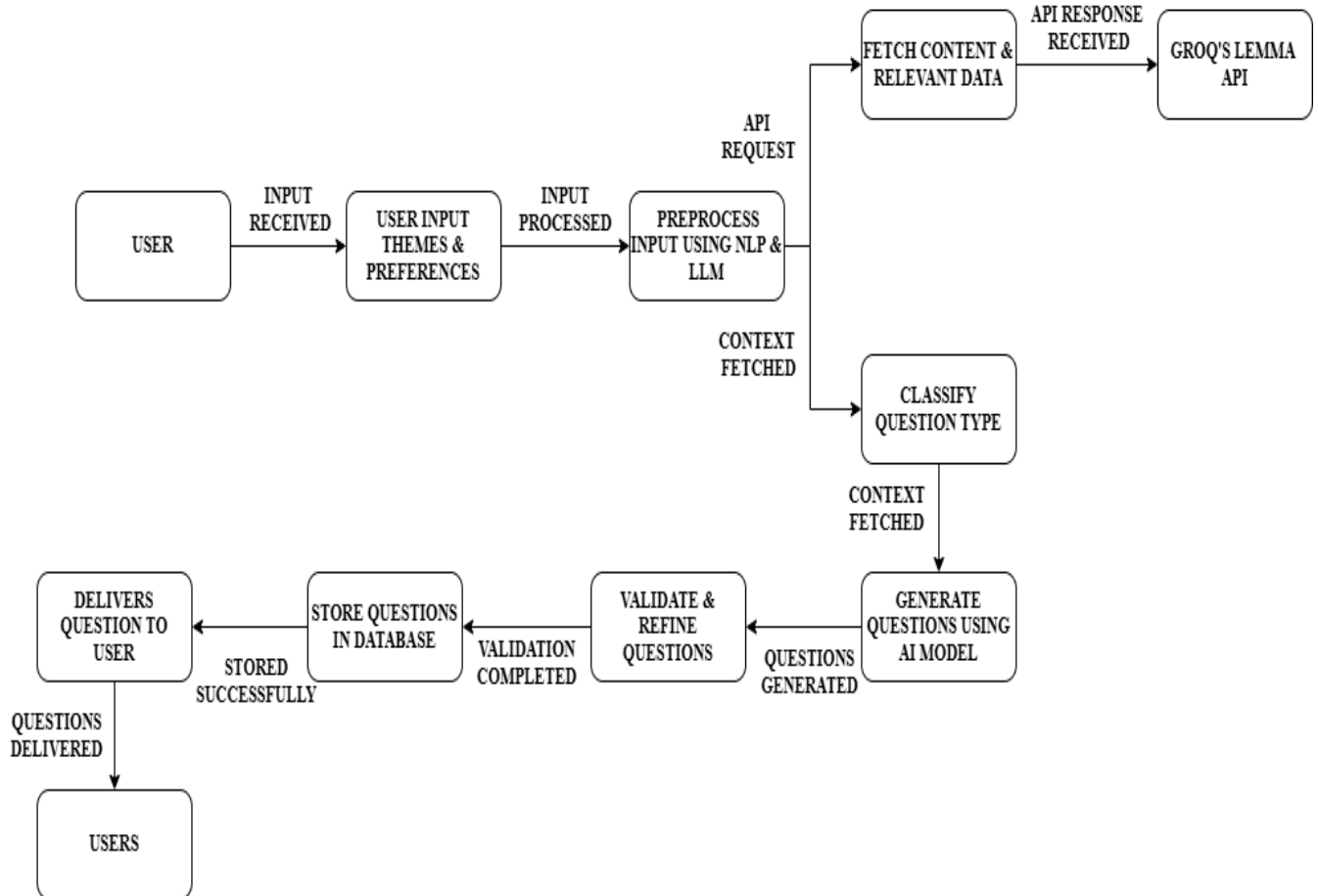
Language Models (LLM) to interpret user intent and structure relevant content. The system ensures accuracy and coherence by analyzing linguistic patterns and contextual meanings. By leveraging advanced AI techniques, the system can efficiently generate meaningful output tailored to the user's needs.

Once the input is processed, the system follows two major workflows: fetching data from APIs and generating questions. In the first workflow, the system retrieves relevant data from external sources through API calls. This ensures that the content used for generating questions is not only comprehensive but also up to date with the latest information. The retrieved data is then refined, filtered, and structured based on user preferences, ensuring that only the most relevant and accurate information is utilized. This process is particularly beneficial in dynamic fields where real-time data is essential, such as technology, science, and current affairs. By leveraging external APIs, the system enhances the quality of generated questions, making them more relevant and aligned with evolving educational needs.

In the second workflow, the system focuses on question generation and storage. Once the input data is processed, the AI-powered system automatically formulates questions based on predefined formats, such as multiple-choice, paragraph-based, and coding problems. These questions are then systematically stored in a structured database, allowing educators to retrieve and organize assessments efficiently. The structured database not only enables seamless access to past questions but also facilitates customization, allowing trainers to modify and adapt quizzes according to specific learning objectives. This feature is particularly valuable in educational environments where personalized quizzes and targeted learning materials are required to enhance student engagement and comprehension. By automating question generation and storage, the system significantly reduces the manual effort involved in assessment creation while ensuring consistency and accuracy in learning evaluations.

Finally, the system delivers the processed content back to the user. The fetched project data from APIs is compiled and structured before being delivered to the user, ensuring a seamless experience. The generated questions are stored in the system, ready for retrieval

and application in quizzes, tests, or learning platforms. This AI-driven approach streamlines content creation, making it easier for users to generate high-quality, customized projects and assessments. By automating these processes, the system enhances efficiency, reduces manual effort, and ensures content relevance, making it an invaluable tool for educators, trainers, and professionals.



Figure(4.3): 2 level data flow diagram

The AI-Powered Question Generation System streamlines the process of creating customized assessments based on user inputs. The process begins when the User provides specific themes and preferences, which are then received and processed. The system utilizes Natural Language Processing (NLP) and Large Language Models (LLM) to analyze and structure the input, ensuring that it aligns with the required question format. Once preprocessed, the system fetches additional context and relevant data by making an API request to Groq’s Lemma API, which provides

external knowledge to enhance the generated questions. This integration ensures that the questions are accurate, diverse, and aligned with the given topic.

After retrieving the necessary content, the system classifies the question type by distinguishing between multiple-choice, short answer, and other formats. This classification process ensures that the generated questions align with the intended assessment style, catering to different learning objectives. The AI model then analyzes the structured data and generates contextually appropriate questions tailored to the specified format. By leveraging advanced natural language processing (NLP) techniques, the system ensures that the generated questions are relevant, engaging, and suitable for the designated topic. This automated classification and generation process significantly reduces the time and effort required for educators to create diverse assessments.

Once the questions are generated, they undergo a validation and refinement process to ensure clarity, coherence, and correctness. This step is crucial in maintaining the quality of the assessments, preventing ambiguities, grammatical errors, or irrelevant content. The AI model cross-checks the generated questions against predefined quality standards, and trainers or administrators can review and modify them if necessary. Finally, the stored questions are delivered to the user in a structured format. The system ensures that the questions are successfully stored before proceeding with delivery, providing the user with high-quality, AI-generated assessments. This automated workflow reduces manual effort while maintaining the reliability and relevance of the generated questions. By leveraging NLP, LLM, and external API data, this system creates a seamless and efficient way to generate, validate, and distribute educational content. As a result, educators and trainers can focus on enhancing learning experiences while relying on AI-driven automation to handle the repetitive task of question generation.

4.2 Entity Relationship Diagram

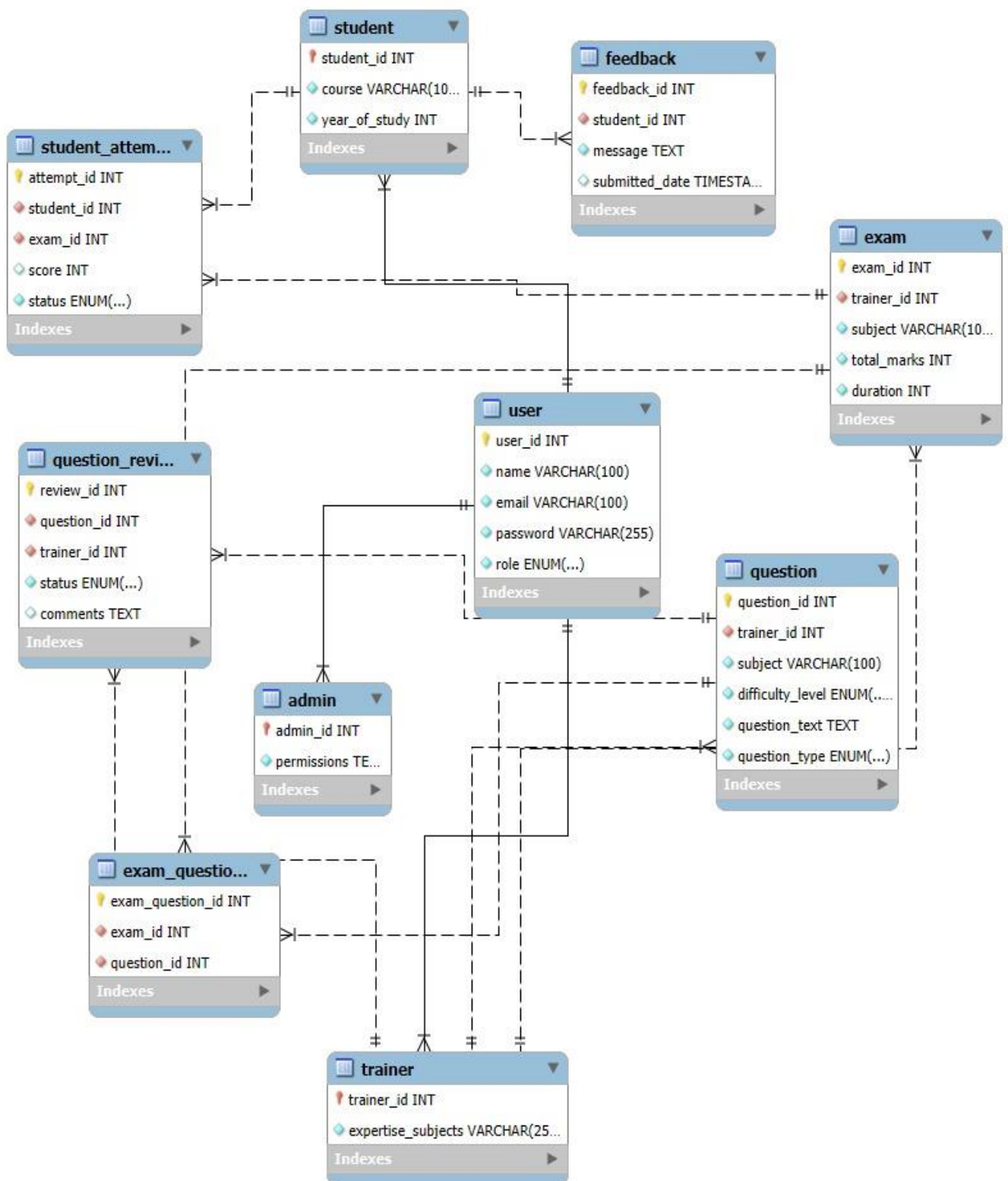


FIG 4.4 Entity-Relationship (ER) Diagram Explanation

The ER diagram represents an AI-powered Question Generator System, where different users interact with the system to manage exams, questions, and feedback. Below is a detailed explanation of each entity and its attributes, structured into subtopics.

1. User :

The USER entity serves as the foundation for managing all individuals in the system, including students, trainers, and admins. Each user has a unique identification number and essential details such as name, email, and password. The role of the user determines their access rights, ensuring that different functionalities are assigned based on whether they are a student, trainer, or admin.

Users are uniquely identified in the system, allowing for smooth interactions between entities like STUDENT, TRAINER, and ADMIN. Their email acts as a unique credential for authentication, and their password is securely stored. Role management ensures that students take exams, trainers manage question generation and reviews, and admins oversee system functionalities.

2. Student :

The STUDENT entity stores academic details of students registered in the system. Since every student is also a user, they are linked to the USER entity. This entity helps in tracking the students' educational journey, their enrolled course, and their year of study.

Each student is assigned a unique ID, ensuring that their progress and exam attempts can be recorded efficiently. The course field stores information about their enrolled program, which helps trainers assign relevant questions. The year of study attribute further categorizes students based on their academic level, ensuring that exams are aligned with their knowledge level.

3. Trainer :

The TRAINER entity holds details of trainers who are responsible for managing questions and reviewing student assessments. Trainers also belong to the USER entity

and are uniquely identified by their trainer ID. They have expertise in specific subjects, which helps them generate and review relevant questions for students.

Each trainer is linked to multiple questions, ensuring that subject matter experts manage the content of the exams. The `expertise_subjects` field stores the subjects they specialize in, making it easier for the system to assign relevant tasks. Trainers also review student answers and provide feedback to improve the learning process.

4. Admin :

The ADMIN entity represents system administrators who oversee user management, question approvals, and exam monitoring. Like students and trainers, admins are also linked to the USER entity but have a different role and set of permissions.

Admins have system-wide access, allowing them to manage users, monitor exam activities, and handle question approvals. Their permissions determine what actions they can take, such as modifying user roles, approving questions, or making system-level changes. This ensures smooth operation and prevents unauthorized modifications.

5. Question :

The QUESTION entity stores all generated questions that students will attempt in exams. Each question is linked to a trainer, ensuring that qualified individuals create and review content before it is used in assessments. Questions have attributes like subject, difficulty level, question type, and text content.

Trainers create and review questions to maintain content quality. The difficulty level ensures that students receive appropriate challenges based on their academic progress. Question types include Multiple-Choice Questions (MCQs), coding problems, and paragraph-based questions, allowing diverse assessment methods.

6. Question Review :

The QUESTION_REVIEW entity is responsible for storing the approval status of generated questions. Since trainers create and review questions, this entity ensures that

only approved questions make it to the exam. It tracks whether a question is approved, rejected, or pending, along with comments from the reviewing trainer.

Each question goes through a review process before being added to an exam. The reviewing trainer provides feedback on rejected or modified questions, ensuring high-quality assessments. This entity helps maintain the credibility of generated questions, preventing errors or irrelevant content from being used in exams.

7. Exam :

The EXAM entity stores details of assessments or quizzes that students take. Each exam is created by a trainer, and it consists of multiple questions chosen based on subject relevance and difficulty level. The exam also has attributes like total marks and duration.

Trainers organize exams by selecting relevant questions from the QUESTION entity. The total marks define the exam's weightage, and the duration ensures students complete the test within a specified time limit. This structure ensures a well-organized assessment process, making it easier to evaluate student performance.

8. Exam Questions :

The EXAM_QUESTIONS entity acts as a bridge between the EXAM and QUESTION entities. Since multiple questions can be part of an exam, this mapping table helps in associating different questions with their respective exams.

Each exam consists of several questions, and this entity ensures proper linking between them. The flexibility of this entity allows different exams to have different combinations of questions, making each assessment unique. It also helps trainers manage question assignments more efficiently.

9. Student Attempt :

The STUDENT_ATTEMPT entity tracks the details of students attempting an exam. It records which student attempted which exam, along with their score and completion status. This entity is crucial for evaluating student performance.

Every student attempt is uniquely recorded, ensuring that results are stored properly for future reference. The status field determines whether a student has completed or left the exam incomplete. The score field records the marks obtained, which helps trainers analyze performance trends.

10. Feedback :

The FEEDBACK entity allows students to submit feedback regarding exams, questions, or the overall system. This helps trainers and admins improve the quality of assessments based on student experiences.

Relationships in the ERD:

- User - Student → One-to-One (Each student is a user, and each user can be a student).
- User - Trainer → One-to-One (Each trainer is a user, and each user can be a trainer).
- User - Admin → One-to-One (Each admin is a user, and each user can be an admin).
- Trainer - Question → One-to-Many (A trainer can create multiple questions, but each question is created by one trainer).
- Question - Question Review → One-to-One (Each question undergoes a review process by a trainer).
- Trainer - Question Review → One-to-Many (A trainer can review multiple questions, but each review is linked to one trainer).
- Exam - Trainer → Many-to-One (Multiple exams can be created by a single trainer).
- Exam - Exam Questions → One-to-Many (An exam consists of multiple questions, but each exam question belongs to one exam).
- Exam Questions - Question → Many-to-One (A question can appear in multiple exams, but each exam question belongs to one question).
- Student - Exam Attempt → One-to-Many (A student can attempt multiple exams, but each attempt is linked to one student).

- Exam - Exam Attempt → One-to-Many (An exam can be attempted by multiple students, but each attempt is for one exam).
- Student - Feedback → One-to-Many (A student can submit multiple feedback entries, but each feedback is linked to one student).

Significance of the ERD:

The relationships in this system ensure efficient data organization and integrity. By maintaining one-to-one relationships between the USER and specific roles (STUDENT, TRAINER, and ADMIN), the system avoids redundant data while ensuring that each user has distinct privileges. The one-to-many relationships between trainers and questions/exams streamline content management, allowing trainers to focus on their specific subject areas.

4.3 Architecture Diagram Of Proposed System

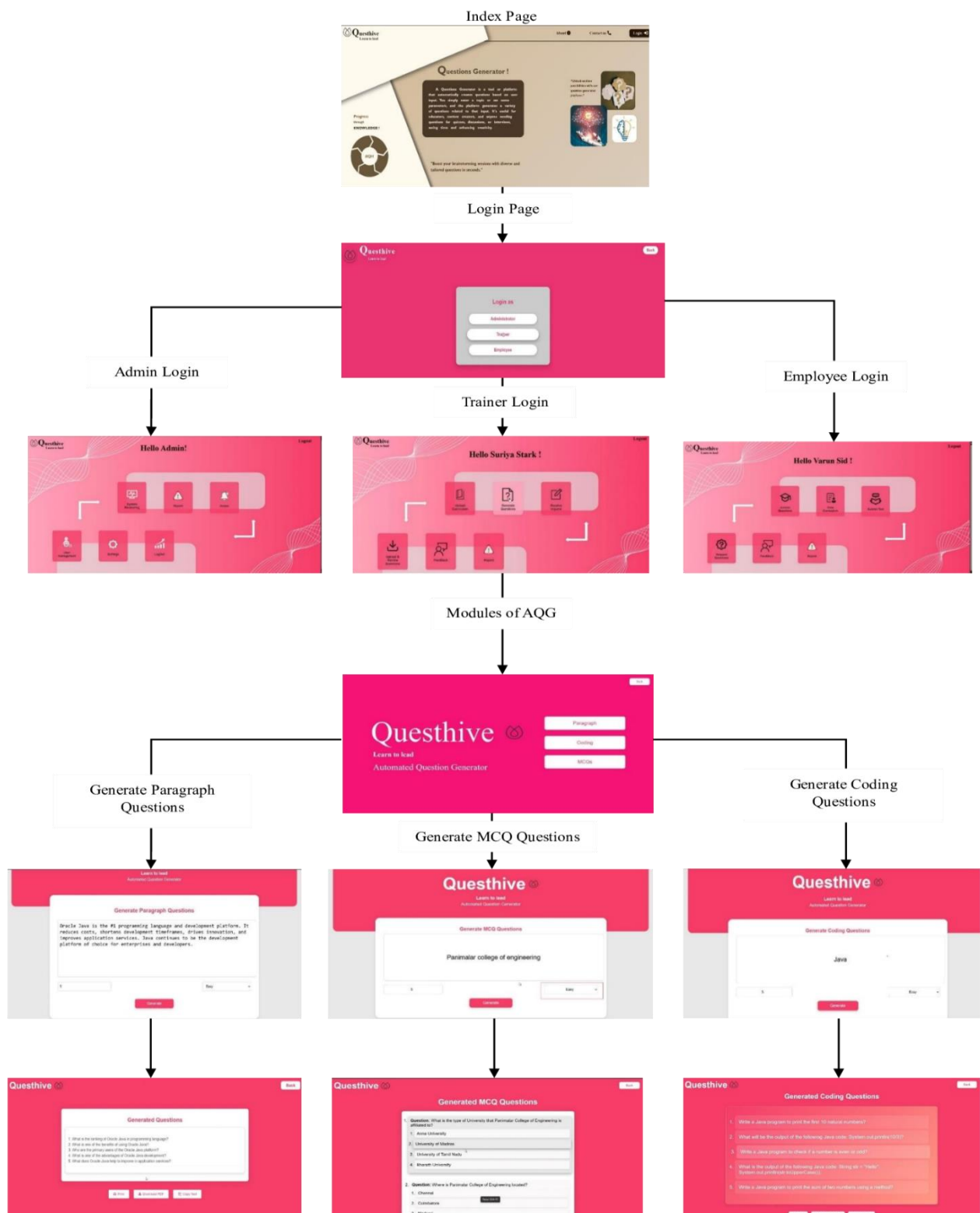


Figure (4.5): Architecture Diagram

One of the first steps in creating a strong question generation system is defining the different kinds of questions, such as descriptive questions, coding challenges, and multiple choice questions (MCQs). Different organising techniques are needed for each category.

- **MCQs:** By making sure the distractions are believable but untrue, the likelihood of guessing should be decreased. Additionally, the MCQs should be categorized based on difficulty levels to cater to learners of varying expertise.
- **Coding Questions:** These should include a problem statement, constraints, input/output formats, and sample test cases. The problem statement should be clear and concise, providing enough details for the user to understand the requirements. Constraints should be well-defined to guide the coding approach while ensuring efficiency and correctness.
- **Descriptive Questions:** Should focus on open-ended responses, essay-type answers, or short explanations. These questions should encourage critical thinking and articulation skills, making them ideal for subjective evaluations. Additionally, a grading rubric should be developed to maintain consistency in evaluation.

Additionally, determining the input sources is crucial. These can be:

- **Predefined Databases:** A structured repository of topics, concepts, and question templates. These databases should be regularly updated to reflect new advancements and evolving educational requirements. Having a well-indexed database ensures efficient retrieval and scalability of question generation.
- **User-Provided Topics:** Dynamic question generation based on custom user input. The system should be designed to interpret and generate relevant questions based on user preferences, enhancing personalization. This feature makes the system adaptable across different learning environments and industries.
- **Web Scraping:** Extracting relevant content from credible sources while ensuring accuracy and relevance. The web scraping process should include validation mechanisms to filter out unreliable information. Additionally, ethical

considerations and compliance with data usage policies should be taken into account to ensure responsible implementation.

The system should be designed for adaptability across multiple domains, including education, recruitment, and corporate training. It should be capable of handling domain-specific terminologies and diverse content structures. A modular architecture will enhance flexibility, making it easier to integrate new functionalities over time.

Data Collection and Preparation

To build a comprehensive question-generation system, it is necessary to gather a diverse dataset comprising:

- **Question Templates:** Various structured templates across subjects, categorized by difficulty levels (easy, medium, hard). Templates should be designed to standardize the question format, ensuring consistency across different topics. Having categorized templates allows for quick adaptation to different learning requirements.
- **Subject-Specific Questions:** Ensuring coverage across multiple domains such as mathematics, science, programming, history, and more. The system should accommodate specialized subject areas, making it versatile for different educational and professional applications. Incorporating domain experts in data collection ensures that the questions remain accurate and relevant.
- **Format Organization:** Storing data in a structured format to facilitate seamless retrieval and usage. Proper data formatting ensures that the questions are easily accessible for processing and generation. Additionally, metadata tagging should be implemented to enhance searchability and categorization.

Preprocessing includes:

- **Data Cleaning:** Removing inconsistencies, duplicates, and formatting errors. This ensures that the generated questions are of high quality, without errors that may confuse learners. Automated cleaning mechanisms should be employed to streamline this process effectively.

- **Standardization:** Ensuring uniformity in phrasing, structuring, and language consistency. Standardization makes it easier to maintain coherence across different question types, improving readability. Linguistic validation techniques should be applied to ensure grammatical correctness.
- **Indexing:** Implementing an efficient indexing mechanism for quick retrieval of relevant question templates. Indexing speeds up search operations, making the system more responsive and efficient. Proper indexing techniques such as keyword- based retrieval should be integrated to enhance usability.

Natural Language Processing (NLP) Implementation

NLP is essential for automating the creation of questions because it ensures language consistency and extracts important topics.

- **Concept Extraction:** Identifying primary topics and related subtopics for question creation. This involves analyzing user input or predefined datasets to determine the core subject matter. Using topic modeling techniques enhances the relevance of generated questions.
- **Synonym Expansion:** Enhancing keyword variations to diversify question phrasing. Synonym analysis ensures that questions are not repetitive, improving engagement for learners. This technique also helps in avoiding rigid question formats by introducing natural linguistic variations.
- **Named Entity Recognition (NER):** Recognizing specific terms, dates, names, and other essential elements to ensure contextual relevance. NER helps in identifying key domain-specific terms that should be included in the questions. This enhances precision and maintains the accuracy of the generated content.
- **Sentence Structuring Techniques:** Applying linguistic models to maintain grammatical correctness and natural flow in generated questions. Proper sentence structuring ensures that the questions are coherent and easy to understand. Machine learning-based syntactic analysis can further refine sentence composition.

CHAPTER 5

PROPOSED SYSTEM IMPLEMENTATION

CHAPTER 5

PROPOSED SYSTEM IMPLEMENTATION

5.1 Data Collection

The data collection phase is a critical foundation of the Automatic Question Generation (AQG) system. This phase ensures that the generated questions are relevant, meaningful, and tailored to the user's learning needs. The data collection process involves user input, topic selection, customization, multi-domain support, and user accessibility. The system aims to create a seamless experience that allows users to generate questions based on their preferences and areas of study.

User Input and Topic Selection

The AQG system allows users to specify topics for question generation based on their preferences. This feature ensures that the questions generated align with specific subject areas, making the learning process more effective. Users can manually enter topics of interest or select from predefined categories to make the process easier. This flexibility ensures that individuals can focus on subjects relevant to their academic or professional needs.

The system supports multiple domains, such as general knowledge, coding, and descriptive topics. This versatility allows users from different backgrounds to generate meaningful questions for learning and assessment. Additionally, the AQG system offers an intuitive interface that simplifies navigation, making it easier for users to find topics and generate questions efficiently.

Topic Selection and Customization

Customization plays a significant role in enhancing the learning experience. The AQG system allows users to either enter specific topics manually or choose from a list of predefined categories. This dual approach ensures that users can align the generated questions with their academic goals or professional needs.

The interface of the system is designed to be user-friendly, allowing users to explore various subject areas effortlessly. By selecting particular topics, users can gain in-depth knowledge on specific subjects while ensuring that their study sessions remain focused and productive. Customization options also include selecting question formats, difficulty levels, and the number of questions, providing a tailored learning experience.

Support for Multiple Domains

The AQG system is built to generate questions across various domains, accommodating different learning styles and subject requirements. These domains include:

- **General Knowledge:** This category covers a broad range of topics, including history, geography, science, and current affairs. It helps learners enhance their overall knowledge by generating questions that encourage critical thinking and awareness of different subjects.
- **Coding & Programming:** This section generates programming-related questions covering different aspects, such as syntax-based questions, debugging, logical reasoning, and problem-solving. It supports multiple programming languages, including Python, Java, C++, and JavaScript. Learners can use this feature to improve their coding skills, test their knowledge, and prepare for coding assessments.
- **Descriptive Topics:** The AQG system also supports generating essay-based or paragraph-style questions in subjects that require detailed explanations, such as literature, philosophy, and theoretical sciences. This feature is particularly useful for students preparing for written examinations or essay-based assessments.
- **Mathematics & Logical Reasoning:** Mathematical problem-solving and logical reasoning are crucial skills in various fields. The system generates arithmetic, algebra, geometry, and problem-solving questions that help learners develop a well-rounded understanding of mathematical concepts.

Tailored Learning Experience

To provide a personalized learning experience, the AQG system allows users to customize their question generation preferences. Users can select the question difficulty level and format, such as multiple-choice, short-answer, or coding problems. The system also

incorporates adaptive learning techniques, which refine question suggestions based on past interactions and performance trends.

Natural Language Processing (NLP) ensures that generated questions are meaningful and contextually relevant, enhancing the overall quality of assessments. By analyzing user input and extracting key concepts, the system generates structured and well-organized questions that meet the learner's needs.

Integration with AI and NLP

The AQQ system leverages AI-driven models, specifically Groq's Lemma API, to generate high-quality questions. NLP techniques are used to analyze user inputs, extract key concepts, and structure the questions appropriately. Additionally, the system employs automatic keyword extraction to refine topic selection, ensuring that generated questions remain coherent and relevant.

By using AI and NLP, the AQQ system continuously improves its accuracy and efficiency. It ensures that the generated questions are not only grammatically correct but also logically structured and aligned with the topic requirements.

User Convenience and Accessibility

A well-designed platform enhances user experience by providing a simple and accessible interface. The AQQ system enables seamless topic selection and quick access to generated questions. Users can store or bookmark topics for future reference, making it easier to revisit subjects and practice regularly.

The system also allows users to filter questions based on difficulty levels, formats, or domains, ensuring that they can focus on specific areas of interest. These features make the AQQ system a powerful tool for students, professionals, and educators.

Question Type Selection

The AQQ system provides users with the flexibility to choose from different question formats. These include multiple-choice questions (MCQs), paragraph-based questions, and coding-related problems. This variety ensures that the system caters to different learning preferences and assessment strategies.

Multiple-choice questions are widely used in various exams and assessments. They help users evaluate their understanding of a topic quickly. Paragraph-based questions, on the

other hand, are beneficial for subjects that require detailed explanations, such as literature and philosophy. Coding-related problems allow programmers to practice and improve their problem-solving skills by working on syntax, logic, and debugging.

By offering multiple formats, the AQG system ensures that learners can engage with different types of questions, enhancing their overall knowledge retention and assessment experience.

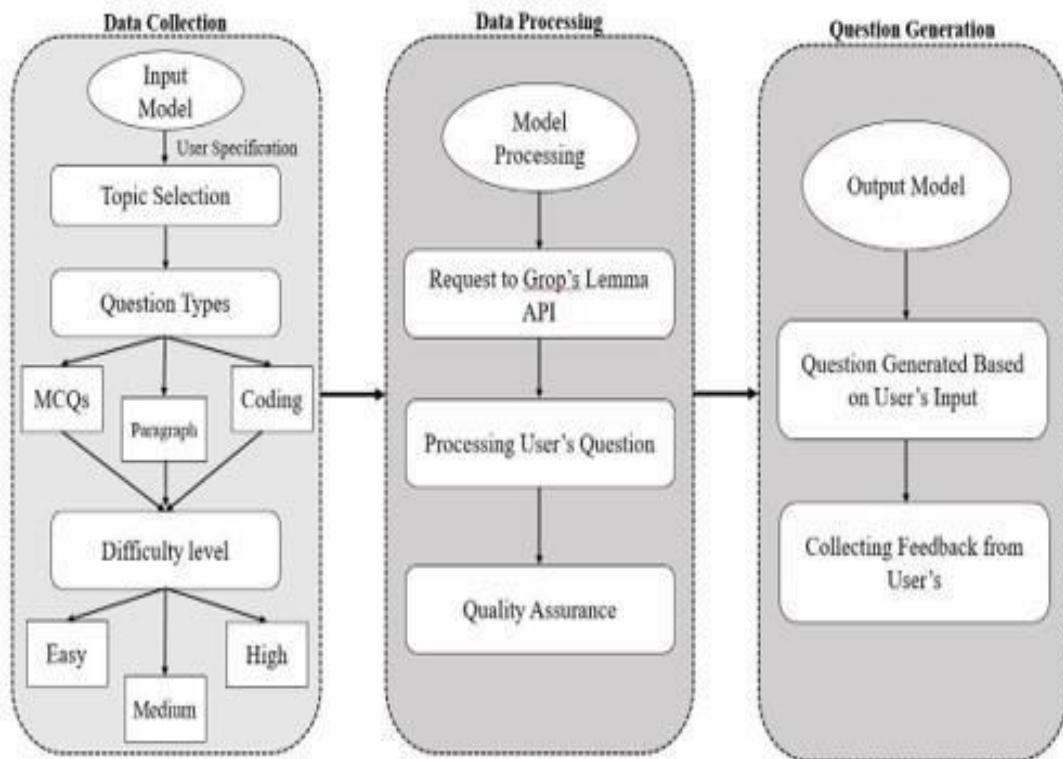


FIG 5.1 Workflow Of Automated Question Generation

Specify the Number of Questions

Users have the flexibility to specify the number of questions they wish to generate. This feature adds versatility to study sessions by allowing learners to control the depth and duration of their assessments. The system offers the following options:

- **Fixed Number Selection:** Users can manually enter the desired number of questions, ensuring they generate an exact amount for their practice sessions.

- **Predefined Question Sets:** The system provides preset options, such as 5, 10, 20, or 50 questions. This helps users quickly select a suitable number of questions without manually entering a value.
- **Adaptive Question Count:** Based on user performance, the system can suggest additional questions to enhance learning. If a user struggles with certain topics, the system may generate more questions in that area.
- **Time-Based Selection:** Users can choose the number of questions based on their available time. For instance, if a user has 30 minutes, the system can estimate and generate an appropriate number of questions to fit that time frame.

These features allow for a personalized and efficient learning process.

Difficulty Level Categorization

The AQG system categorizes questions into three difficulty levels—Easy, Medium, and Hard—to match different proficiency levels.

- **Easy Level:** This category is designed for beginners, covering fundamental concepts and recall-based questions. It helps users build a strong foundation before moving to more complex topics.
- **Medium Level:** Medium-level questions involve moderate problem-solving and conceptual application. They require users to apply their knowledge and understand concepts in more depth.
- **Hard Level:** Hard-level questions target advanced learners, focusing on complex analytical and critical-thinking skills. These questions often require in-depth problem-solving and advanced reasoning.

The system also supports adaptive difficulty selection, dynamically adjusting question difficulty based on user performance. If a user consistently answers questions correctly, the system may suggest more challenging questions to enhance their learning experience.

Data Processing

Once the user selects topics, question types, and difficulty levels, the system processes this data to generate relevant questions.

Model Processing: The AI model extracts key concepts, analyzes user input, and structures questions in a logical and coherent manner. By breaking down the topic into sub-components, the system ensures that the questions are meaningful and align with the user's learning objectives.

Request to Groq's Lemma API: The system refines question content, ensuring grammatical accuracy, contextual relevance, and linguistic clarity. This API helps maintain consistency in the generated questions.

User Input Processing and Quality Assurance: The AQG system validates user inputs to eliminate errors and ensure logical flow. This step helps maintain the integrity of the generated questions and enhances their overall quality.

5.2 Module 1: Web Page Development

The frontend module is meticulously designed to provide an interactive and user-friendly experience for students, trainers, and administrators. It ensures seamless navigation and efficient management of data and assessments by offering a well-structured interface. One of the fundamental aspects of this module is user authentication and role management, which plays a critical role in securing access to the system. The authentication mechanism allows users to log in securely, ensuring that students, trainers, and administrators have appropriate access based on their roles. Students can access quizzes, learning materials, and progress reports, while trainers have the ability to create and manage assessments. Administrators, on the other hand, oversee the overall management of users, records, and reports, ensuring that the system functions smoothly. By implementing role-based access control, the module enhances security and prevents unauthorized access to sensitive information.

To further streamline the user experience, the frontend incorporates an intuitive dashboard and navigation system that enables efficient management of assessments, student records, and performance reports. The dashboard provides a centralized interface where users can easily access relevant features without unnecessary complexity. Students can quickly navigate through available quizzes, track their progress, and review past

assessments. Trainers benefit from an organized layout that simplifies the process of creating, modifying, and evaluating quizzes, making their workflow more efficient. Additionally, administrators have access to tools for managing user roles, monitoring system activity, and generating insightful reports. The entire frontend structure is designed with usability in mind, ensuring that even users with minimal technical expertise can interact with the system effortlessly.

The frontend module is built using a combination of HTML, CSS, and JavaScript, ensuring a well-structured and visually appealing interface. HTML provides the foundational structure, CSS enhances the styling, and JavaScript adds interactivity and functionality. To guarantee responsiveness across different devices, Bootstrap is integrated, allowing users to access the platform seamlessly from desktops, tablets, and mobile phones. JavaScript plays a crucial role in enhancing user experience by enabling form validation, real-time feedback, and dynamic content updates. For example, it ensures that students receive immediate validation when submitting quiz responses, reducing errors and improving engagement. By leveraging modern web technologies, the frontend module not only provides an aesthetically pleasing interface but also ensures efficiency, security, and accessibility for all users.

Algorithm 1: Form Validation & Submission

Step 1: Load the input form containing Name, Email, Password fields, and a Submit button.

Step 2: Assign unique IDs and classes to each input field for validation and styling.

Step 3: Attach an event listener to the Submit button to detect form submission.

Step 4: Initialize `isValid = TRUE` before performing validation checks.

Step 5: Retrieve the value entered in the Name field.

Step 6: If the Name field is empty, display an error message and highlight the field.

Step 7: If the Name field is valid, remove any previous error messages and reset styling.

Step 8: Retrieve the value entered in the Email field.

Step 9: If the Email field is empty, display an error message and highlight the field.

Step 10: If the Email format is incorrect, display an error message and highlight the field.

Step 11: If the Email field is valid, remove any previous error messages and reset styling.

Step 12: Retrieve the value entered in the Password field.

Step 13: If the Password field is empty, display an error message and highlight the field.

Step 14: If the Password is less than 8 characters, display an error message and highlight the field.

Step 15: If the Password meets the criteria, remove any previous error messages and reset styling.

Step 16: If isValid == TRUE, allow form submission and send data to the backend.

Step 17: If isValid == FALSE, prevent form submission and prompt the user to correct errors.

Step 18: End the validation process.

Step 1: Load the input form containing Name, Email, Password fields, and a Submit button.

The user interface displays a structured HTML form that includes three input fields: Name, Email, and Password. A Submit button is added to allow form submission. These fields serve as essential user details required for registration or login.

Step 2: Assign unique IDs and classes to each input field for validation and styling.

Each input field is assigned unique IDs and CSS classes to facilitate JavaScript validation and styling. These IDs help in selecting the elements during validation, while the classes enable dynamic styling, such as changing border colors for error indication.

Step 3: Attach an event listener to the Submit button to detect form submission.

An event listener is linked to the Submit button to trigger validation when the user attempts to submit the form. This prevents the default behavior of immediate form submission, ensuring validation checks are executed before sending data to the backend.

Step 4: Initialize isValid = TRUE before performing validation checks.

A boolean flag isValid is initialized to TRUE at the start of validation. If any field fails the validation criteria, this flag is set to FALSE, indicating that the form contains errors and should not be submitted.

Step 5: Retrieve the value entered in the Name field.

The JavaScript function extracts the user-entered value from the Name field. This step allows further verification to ensure that the field is not left empty. The retrieved data is stored in a variable for processing.

Step 6: If the Name field is empty, display an error message and highlight the field.

If the Name field is left blank, an error message such as "Name is required" is displayed near the field. Additionally, the field's border color is changed (e.g., to red) to visually indicate an issue to the user.

Step 7: If the Name field is valid, remove any previous error messages and reset styling.

If a valid name is entered, any previous error messages are removed from the UI. The input field's border color is reset to its default styling, ensuring a clean and consistent user experience.

Step 8: Retrieve the value entered in the Email field.

Similar to the Name field, the value entered in the Email input box is fetched and stored for validation. This ensures that the entered email follows the correct format before proceeding further.

Step 9: If the Email field is empty, display an error message and highlight the field.

If the Email field is blank, a warning message such as "Email is required" is shown. The border of the input box is also highlighted to visually signal the user to enter a valid email.

Step 10: If the Email format is incorrect, display an error message and highlight the field.

If the email format does not match the standard pattern (e.g., example@domain.com), an error message "Enter a valid email address" is displayed. The field is also visually highlighted to prompt the user for correction.

Step 11: If the Email field is valid, remove any previous error messages and reset styling.

If the user enters a correctly formatted email, any previous error messages are cleared. The field's border styling is reset to normal, ensuring the UI remains clean and user-friendly.

Step 12: Retrieve the value entered in the Password field.

The function retrieves the password entered by the user. This value is stored in a variable and later checked to ensure that it meets the minimum security requirements.

Step 13: If the Password field is empty, display an error message and highlight the field.

If no password is entered, an error message such as "Password is required" is displayed. The input box is also visually highlighted to draw attention to the missing information.

Step 14: If the Password is less than 8 characters, display an error message and highlight the field.

If the entered password is too short (less than 8 characters), an error message appears warning the user. The input field is also highlighted, ensuring the user is aware of the security requirement.

Step 15: If the Password meets the criteria, remove any previous error messages and reset styling.

If the entered password is valid, all previous error messages related to password validation are removed. The field's border color is reset to normal, indicating successful validation.

Step 16: If `isValid == TRUE`, allow form submission and send data to the backend.

If all fields pass validation (`isValid` remains `TRUE`), the form is successfully submitted. The user data is sent to the backend using an HTTP request (such as `fetch` or `AJAX`) for further processing.

Step 17: If `isValid == FALSE`, prevent form submission and prompt the user to correct errors.

If any field fails validation (`isValid` is set to `FALSE`), form submission is blocked. The user is prompted to correct errors before attempting to submit again. This ensures only valid data is sent to the backend.

Step 18: End the validation process.

Once validation checks are completed and the form is either submitted or rejected based on the results, the process concludes. The user either receives error feedback or successfully submits the form.

The frontend module plays a critical role in ensuring data integrity, security, and user experience by validating user input before submission. When users interact with a

web form, they are required to enter information such as Name, Email, and Password, which are essential for authentication and further processing. Without proper validation, users may submit incorrect, incomplete, or malicious data, leading to backend errors, security vulnerabilities, and poor system performance. To prevent such issues, the system implements real-time validation checks that immediately notify users if they enter invalid data, improving both usability and efficiency.

The validation process begins when a user attempts to submit the form. The system first checks the Name field to ensure that it is not left empty. If the field is blank, an error message appears, prompting the user to enter their name before proceeding. This simple check prevents missing data from being stored in the database. Similarly, the Email field undergoes rigorous validation to ensure it follows the correct email format (e.g., "user@example.com"). If the email is invalid or left blank, the user is notified to enter a correct email address, reducing the chances of incorrect or non-functional email addresses being stored. This step is particularly important for user authentication, account recovery, and communication purposes.

In addition to validating the name and email fields, the system enforces strong password policies to enhance security. The Password field is checked to ensure that it meets a minimum length requirement (typically 8 or more characters). If the password is too short, a warning message prompts the user to enter a longer password. Enforcing password strength requirements helps protect user accounts from brute-force attacks and unauthorized access. Some implementations may include additional security measures, such as requiring a combination of uppercase letters, numbers, and special characters to further strengthen passwords. If any validation check fails, the form blocks submission, and users must correct their input before proceeding.

By integrating frontend validation, the system significantly enhances data accuracy, user experience, and backend efficiency. Real-time feedback ensures that users can correct errors instantly, preventing frustration and reducing unnecessary server requests caused by invalid submissions. Moreover, frontend validation reduces the workload on

the backend by ensuring that only properly formatted data reaches the server, preventing unnecessary processing and database inconsistencies. This approach ultimately improves system reliability, enhances security, and ensures a smooth user experience. Additionally, it helps mitigate risks such as SQL injection and cross-site scripting (XSS) attacks, making the application more robust against cyber threats. By enforcing strict validation rules at the frontend level, developers can create a secure, user-friendly, and high-performing web application.

5.3 Module 2: Automatic Question Generator (AQG)

This module is designed to facilitate AI-powered question generation using Groq's Lemma API, allowing for the creation of diverse question formats, including multiple-choice, coding, and paragraph-based questions. By leveraging advanced natural language processing (NLP) techniques, the system ensures that the generated questions are contextually relevant and aligned with the specified subject matter. One of the standout features of this module is its ability to offer customizable question settings, enabling trainers to define parameters such as difficulty levels and subject areas. This ensures that assessments can be tailored to match the proficiency levels and learning objectives of students, making the evaluation process more adaptive and effective.

To enhance usability and efficiency, the module incorporates a robust question storage and management system, ensuring that generated questions can be saved and reused for future assessments. Trainers have the flexibility to review and approve AI-generated questions, ensuring accuracy and relevance before they are finalized for student assessments. Additionally, trainers can edit or modify questions, allowing for manual refinements to improve clarity or align questions more closely with curriculum requirements. This functionality ensures a balance

between automated efficiency and human oversight, ultimately leading to high-quality assessments.

The module is built using PHP for backend API integration, ensuring seamless communication between the system and Groq's Lemma API. PHP scripts are responsible for processing generated questions, handling API requests, and managing data flow within the system. To store and manage question data securely, MySQL/PHPMyAdmin is utilized, offering a structured and scalable approach to database management. By integrating these technologies, the system enhances automated question generation, providing trainers with an efficient, customizable, and reliable tool for creating assessments tailored to various educational needs.

Algorithm 2: Automated Question Generation System

Step 1: Initialize the Web Form Interface.

Step 2: Provide an input field for paragraph-based question generation.

Step 3: Allow users to submit a paragraph via the web form.

Step 4: Provide an input field for coding-based question generation.

Step 5: Allow users to enter a topic, number of questions, and difficulty level.

Step 6: Store user inputs and validate the entered data.

Step 7: If paragraph input is provided, process it for question generation.

Step 8: If coding input is provided, process the topic, number of questions, and difficulty level.

Step 9: Call the Groq API to generate MCQ questions based on the given inputs.

Step 10: Format the API request with specified topic, number of questions, and difficulty.

Step 11: Retrieve the API response and extract generated questions.

Step 12: Parse the JSON output and validate the extracted questions.

Step 13: If JSON extraction fails, display an error message.

Step 14: If JSON extraction succeeds, display the generated MCQs to the user.

Step 15: Process the response for paragraph-based question generation.

Step 16: Convert the given paragraph into relevant questions using NLP techniques.

Step 17: Store generated questions in a structured format for display.

Step 18: Render the generated MCQ and paragraph-based questions on the web interface.

Step 19: If the response is valid, send data to the backend for storage and retrieval.

Step 20: End the process.

Step 1: Initialize the Web Form Interface.

The system loads an HTML form that allows users to input text for generating questions. It includes different sections for paragraph-based, coding-based, and MCQ-based question generation. The user interface is designed for easy navigation and input validation.

Step 2: Provide an input field for paragraph-based question generation.

A `<textarea>` field is created where users can enter a paragraph for processing. The system ensures the field is mandatory and prevents empty submissions. Users can type or paste a paragraph, which will be analyzed for generating questions.

Step 3: Allow users to submit a paragraph via the web form.

Once the paragraph is entered, the form captures and stores the user input. It ensures the paragraph meets the required conditions before submission. Upon clicking submit, the paragraph is sent for further processing using NLP.

Step 4: Provide an input field for coding-based question generation.

The form includes an input box where users can enter a coding topic. An additional field allows specifying the number of questions to be generated. The difficulty level (Easy, Medium, Hard) can be selected from a dropdown menu.

Step 5: Allow users to enter a topic, number of questions, and difficulty level.

The form ensures that all fields—topic, number, and difficulty—are filled correctly. If any input is missing, an error message is displayed to guide the user. The selected difficulty level determines the complexity of generated questions.

Step 6: Store user inputs and validate the entered data.

User inputs are stored in temporary variables before processing. Validation is performed to check for empty fields, incorrect formats, and invalid values. If validation fails, the form prevents submission and prompts the user for corrections.

Step 7: If paragraph input is provided, process it for question generation.

The entered paragraph is sent to the backend for text analysis. Natural Language Processing (NLP) techniques are used to extract meaningful content. Key phrases and important details are identified to form relevant questions.

Step 8: If coding input is provided, process the topic, number of questions, and difficulty level.

The entered coding topic is analyzed to determine relevant question patterns. Based on difficulty selection, the complexity of questions is adjusted. The total number of questions requested is verified and prepared for API processing.

Step 9: Call the Groq API to generate MCQ questions based on the given inputs.

The backend sends a request to Groq's API with topic, difficulty, and question count. The API processes the request and generates multiple-choice questions accordingly. A streaming response is received, containing dynamically generated MCQs.

Step 10: Format the API request with specified topic, number of questions, and difficulty.

The request structure follows the correct API format to ensure accurate responses. Each field is formatted into a structured JSON payload before sending it. This ensures the API understands the request and returns relevant questions.

Step 11: Retrieve the API response and extract generated questions.

The system receives a response from the Groq API containing MCQs. The response is streamed in chunks and stored in a temporary variable. If valid, the JSON data is parsed to extract questions in structured format.

Step 12: Parse the JSON output and validate the extracted questions.

Regular expressions are used to extract the JSON array of questions. If extraction fails, an error message is displayed, and debugging logs are generated. If extraction succeeds, the formatted questions are prepared for display.

Step 13: If JSON extraction fails, display an error message.

The system checks if the response contains a valid JSON structure. If no valid JSON is found, an error is shown to the user. Debugging messages are logged for troubleshooting incorrect API responses.

Step 14: If JSON extraction succeeds, display the generated MCQs to the user.

Extracted multiple-choice questions are formatted and shown in a readable manner. Each question is presented with four options and marked with correct answers. Users can review the MCQs before saving them or using them in assessments.

Step 15: Process the response for paragraph-based question generation.

For paragraphs, an NLP-based question extraction technique is applied. Sentences are analyzed to identify key points that can be turned into questions. The system ensures that extracted questions are grammatically correct and meaningful.

Step 16: Convert the given paragraph into relevant questions using NLP techniques.

AI-powered algorithms analyze sentence structures and extract possible question types. Questions may be generated as direct factual queries or inference-based ones. The goal is to maintain question clarity while ensuring relevance to the given paragraph.

Step 17: Store generated questions in a structured format for display.

All generated MCQs and paragraph-based questions are stored in an array. The data is then structured for easy retrieval and presentation. Proper formatting ensures that questions are aligned and properly displayed.

Step 18: Render the generated MCQ and paragraph-based questions on the web interface.

The frontend dynamically updates to show the generated questions. Users can view MCQs and paragraph-based questions in their respective sections. The interface also provides an option to regenerate or modify questions if needed.

Step 19: If the response is valid, send data to the backend for storage and retrieval.

The generated questions are stored in a MySQL database via PHP scripts. Role-based access ensures that only authorized users can view or modify them. Questions can be retrieved later for assessments, quizzes, or further modifications.

Step 20: End the process.

The question generation process is successfully completed. Users can now either save the questions, modify them, or start a new request. All temporary variables are cleared, and the system is ready for the next input.

The Automated Question Generation System is designed to assist users in dynamically creating multiple-choice questions (MCQs), coding-based questions, and paragraph-based questions. This system combines web-based forms, backend validation, API interactions, and AI-based text processing to generate relevant questions for different topics. Users interact with a frontend web interface where they can enter a paragraph, specify a coding topic, or request a certain number of MCQs. These inputs are validated and processed using Natural Language Processing (NLP) techniques and Groq API calls. The system ensures the generated questions are accurate, well-structured, and formatted

properly before displaying them back to the user. The final questions can be stored in a MySQL database for future use in quizzes, assessments, or educational platforms.

When a paragraph is submitted by the user, it is sent to the backend for text analysis. The system first validates whether the input is meaningful and properly formatted. If valid, the Natural Language Processing (NLP) model processes the text to extract key points, important details, and sentence structures that can be converted into questions. The NLP model analyzes the paragraph to generate different types of questions, such as factual questions, inference-based questions, and fill-in-the-blank questions. The extracted questions are then formatted into a structured list and displayed to the user. This approach helps in automatically converting textual content into meaningful assessment questions, reducing manual effort in quiz creation and ensuring accuracy.

For coding-related questions, users enter a topic (e.g., Java, Python, Data Structures) along with the number of questions they need and their difficulty level (Easy, Medium, or Hard). The system validates the inputs and ensures they meet the required conditions before proceeding. Once verified, a request is sent to the backend, where the system analyzes the topic and difficulty level. Based on these inputs, the Groq API is called to generate programming-related questions. The API dynamically creates multiple-choice, fill-in-the-blank, or descriptive coding questions based on the selected parameters. The generated questions are then formatted and displayed in a user-friendly manner on the web interface. If needed, users can modify or regenerate questions before finalizing them.

The MCQ generation process uses the Groq API, which takes user inputs such as topic, number of questions, and difficulty level to generate multiple-choice questions dynamically. The API processes the request and streams back a response containing well-structured MCQs, including a question, four answer choices, and the correct answer. The system extracts the response, formats the JSON data, and verifies the validity and correctness of the generated questions. If the extraction fails, an error message is

displayed, allowing users to adjust inputs and retry. If successful, the MCQs are displayed on the frontend, where users can review and use them in assessments. The system ensures the questions are aligned with the requested difficulty level and cover relevant topics, making the process efficient and user-friendly.

This Automated Question Generation System streamlines the process of creating assessment questions, saving time for educators, trainers, and students. By leveraging NLP techniques, API integrations, and structured data processing, the system ensures high-quality, accurate, and diverse questions for various educational and training purposes.

5.4 Module 3: Backend & Database Management

This module plays a crucial role in managing data storage, retrieval, and organization, ensuring efficient handling of user data, assessments, and access control. It systematically stores and organizes records of students, trainers, and employees, providing a structured database that enhances seamless data management. By maintaining quiz results and tracking student progress, the system allows educators and administrators to monitor academic performance over time. This feature is essential for identifying learning trends, assessing student development, and making necessary adjustments to training programs for better outcomes.

To enhance security and prevent unauthorized modifications, the module incorporates Role-Based Access Control (RBAC). This mechanism ensures that users only have access to functionalities relevant to their roles, thereby maintaining data integrity and confidentiality. Trainers can create and manage assessments, students can attempt quizzes and review their progress, while administrators can oversee system activities and generate reports. Additionally, the module provides comprehensive performance reports and statistical insights on assessments and user engagement. These analytics help educators make data-driven decisions, improving learning strategies and system effectiveness.

The module is developed using PHP for server-side logic, ensuring smooth execution of data-related operations. MySQL/PHPMyAdmin is utilized for structured data storage, offering a scalable and efficient database management solution. To further enhance security and system reliability, the module implements key protective measures such as role-based access control, session management, and encryption of sensitive information. These security protocols ensure that user data remains protected, reinforcing the system's trustworthiness while providing a reliable and efficient platform for educational management.

Algorithm 3: Backend & Database Management

Step 1: RECEIVE username and password from the login form.

Step 2: SEARCH user in the database.

Step 3: IF user exists AND password matches, FETCH user's role.

Step 4: IF role == "Student", GRANT access to quizzes and learning materials.

Step 5: ELSE IF role == "Trainer", GRANT access to create and evaluate assessments.

Step 6: ELSE IF role == "Admin", GRANT access to manage users and system settings.

Step 7: ELSE, DISPLAY "Unauthorized access".

Step 8: ELSE, DISPLAY "Invalid username or password".

Step 9: END SEARCH.

Step 1: RECEIVE username and password from the login form.

The system receives the user's login credentials entered in the input fields. These details are passed to the backend for authentication and validation.

Step 2: SEARCH user in the database.

The backend searches for the provided username in the database records. If found, it retrieves the corresponding password and user role.

Step 3: IF user exists AND password matches, FETCH user's role.

If the username exists and the entered password matches the stored password, the system fetches the role associated with that user from the database.

Step 4: IF role == "Student", GRANT access to quizzes and learning materials.

If the user is a student, they are granted access to the available quizzes and study materials. They can participate in assessments and track their progress.

Step 5: ELSE IF role == "Trainer", GRANT access to create and evaluate assessments.

If the user is a trainer, they are given permissions to create new assessments, evaluate student performance, and analyze quiz results.

Step 6: ELSE IF role == "Admin", GRANT access to manage users and system settings.

If the user is an admin, they are provided full control over user management, system settings, and security configurations.

Step 7: ELSE, DISPLAY "Unauthorized access".

If the user's role does not match any predefined categories, the system denies access and displays an "Unauthorized access" message.

Step 8: ELSE, DISPLAY "Invalid username or password".

If the username is not found or the password entered is incorrect, the system returns an error message asking the user to enter valid credentials.

Step 9: END SEARCH.

The authentication process concludes. If the login is successful, the user is redirected to their respective dashboard. Otherwise, they are prompted to retry logging in.

The Role-Based Access Control (RBAC) module is a crucial part of the backend system, responsible for managing user authentication and enforcing access restrictions based on predefined roles. The process starts when a user enters their username and password into the login form. The system verifies these credentials by checking the database to determine if the user exists. If the credentials are incorrect, an error message is displayed, preventing unauthorized access. However, if authentication is successful, the system retrieves the user's assigned role from the database and determines the appropriate level of access. This ensures that users can only interact with the functionalities they are permitted to use.

Once the system identifies the user's role, it grants access accordingly. Students are allowed to participate in quizzes and access learning materials, ensuring they can engage with the educational content. Trainers are provided with tools to create, manage, and evaluate assessments, giving them the ability to generate and modify questions. Admins, on the other hand, have the highest level of access, enabling them to manage users, monitor system settings, and oversee overall platform operations. If a user does not have a valid role assigned, the system immediately blocks access and displays an "Unauthorized access" message to prevent security breaches.

This RBAC model enhances security, efficiency, and scalability by ensuring that users can only access the system functionalities relevant to their role. By preventing unauthorized users from viewing or modifying sensitive information, it maintains data integrity and confidentiality. This structured approach not only improves security but also enhances the usability and maintainability of the entire system, ensuring smooth operations across different user levels.

CHAPTER 6

RESULTS AND DISCUSSION

CHAPTER 6

RESULTS AND DISCUSSION

The Automated Question Generator (AQG) system is designed to help users generate paragraph-based questions by analyzing input text and identifying key concepts. It processes the given text using advanced natural language processing techniques to extract essential information. By recognizing important themes, entities, and relationships within the text, the system ensures that the generated questions are relevant and well-structured.

This section explores the detailed workings of the AQG system, focusing on how it interprets user input to create structured questions. It breaks down the question generation process, from text analysis to question formation, ensuring clarity and coherence. By automating this process, the AQG system enhances efficiency in educational assessments and content development, making question generation more systematic and insightful.

Paragraph Based Question Model

The Automated Question Generator (AQG) system provides users with a text input field where they can enter a paragraph, passage, or any topic-specific content. This input mechanism is designed to be user-friendly, allowing manual text entry as well as the option to paste pre-existing content. The system is flexible enough to support multiple Languages, making it accessible to a diverse range of users. Additionally, it is equipped to handle various technical and academic topics, including programming, history, and science. One of the standout features of the AQG system is its ability to adapt to different difficulty levels. Users can select from three levels—Easy, Medium, or Hard—based on their requirements. This ensures that the generated questions match the intended complexity, making the system useful for learners at different stages. The process is

initiated by a "Generate" button, which triggers the algorithm responsible for analyzing the text.

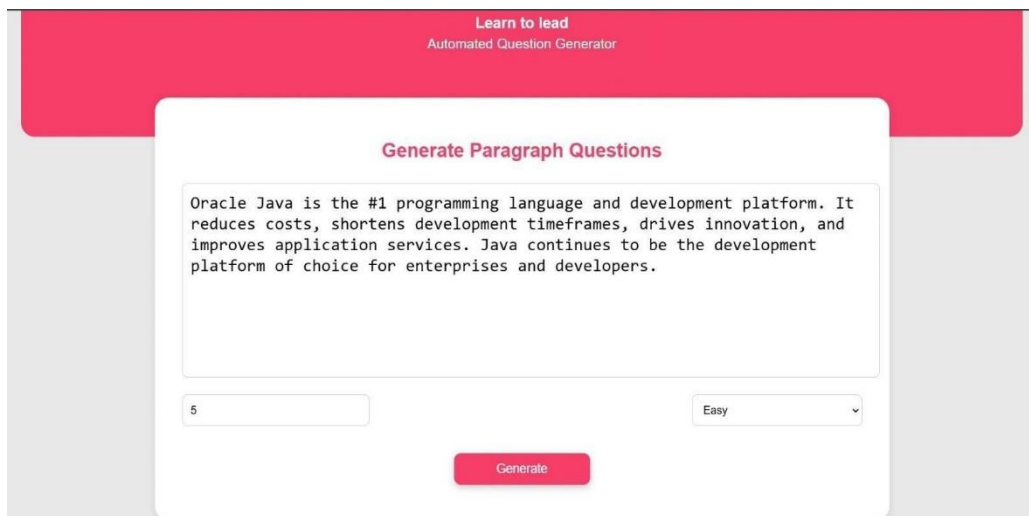


FIG 6.1 Paragraph Based Question Model

By integrating Natural Language Processing (NLP) techniques, the system efficiently extracts relevant information from the input, setting the foundation for meaningful question generation.

Once a paragraph is submitted, the system undergoes a structured, multi-step process to generate questions. The first step is **text tokenization**, where the paragraph is divided into individual sentences and words. This segmentation helps in identifying distinct ideas and information units. Following this, **keyword extraction** takes place, where essential terms, nouns, and key phrases are identified. These extracted elements play a crucial role in shaping the questions, ensuring they are directly relevant to the original text.

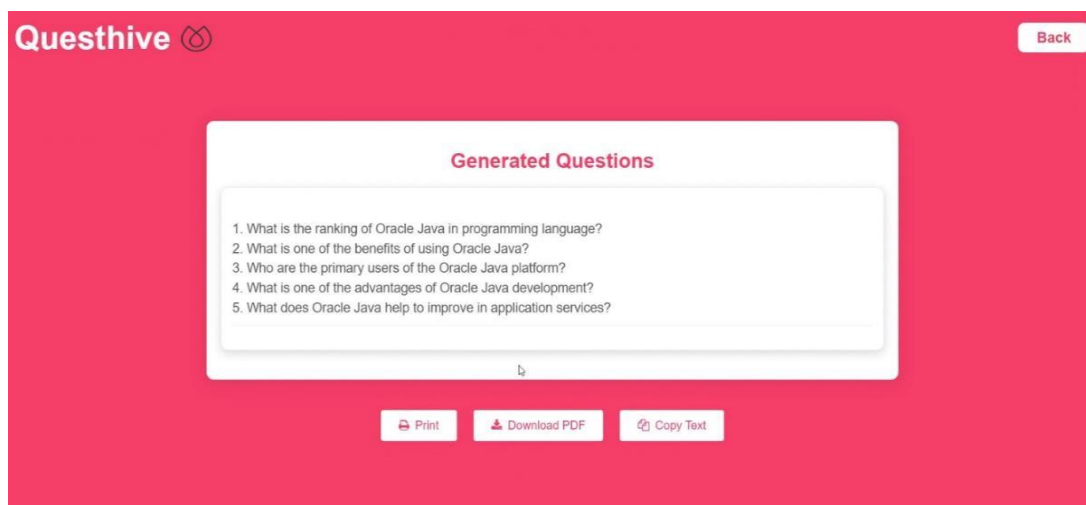


FIG 6.2 Output of the Paragraph Based Question Model

After identifying key terms, the AQG system proceeds to **question formation**, structuring meaningful questions based on the extracted keywords. The generated questions follow standard comprehension formats, ensuring clarity and relevance. If the user opts for multiple-choice questions (MCQs), the system automatically generates possible answer choices. These answer options are designed to be contextually accurate, improving the reliability of the generated assessment material. By incorporating this feature, the AQG system streamlines the process of creating structured quizzes and assessments, making it valuable for both educators and learners.

Sample Output Demonstration

To illustrate the functionality of the AQG system, consider the following input paragraph:

"Python is a popular programming language used for web development, data science, and artificial intelligence. It is known for its readability and versatility."

Based on this input, the system would generate structured comprehension questions, such as:

- What are some common applications of Python?
- Why is Python considered a versatile programming language?
- What makes Python easy to read and understand?

For the multiple-choice question (MCQ) format, the system generates the main question along with relevant answer choices, including the correct option and plausible distractors. The AI ensures that distractors are neither too obvious nor entirely unrelated, maintaining a balanced difficulty level. It intelligently adjusts the complexity of the MCQs to cater to different learning levels.

Automating MCQ generation improves efficiency by allowing large question banks to be created instantly, reducing manual effort. Trainers can review and modify questions to align with curriculum requirements. The system also supports adaptive learning by adjusting difficulty levels based on student performance.

Additionally, it generates scenario-based MCQs that encourage critical thinking and application of knowledge. These questions can be integrated into digital learning platforms for seamless assessments. This automation ensures scalable, reliable, and engaging educational evaluations.

MCQ Based Question Model

The Multiple-Choice Question (MCQ) Generator is a key feature of the Automated Question Generator (AQG) system, designed to streamline the process of creating well-structured MCQs. This automated system enables users to generate multiple-choice questions based on structured input, eliminating the need for manual question formulation. By leveraging Natural Language Processing (NLP) techniques, the AQG system ensures that the generated MCQs are contextually relevant and educationally valuable.

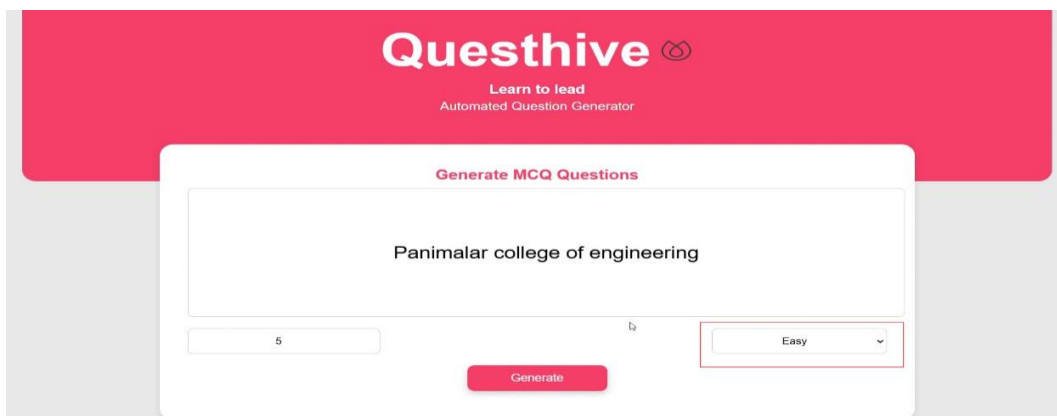


FIG 6.3 MCQ Question Based Generator

To create MCQs, users provide input in the form of a specific topic, a paragraph, or a dataset containing relevant information. Once the input is received, the system analyzes the text and extracts crucial elements such as facts, numerical values, definitions, and key concepts. This step ensures that the generated questions are not only relevant but also diverse in terms of the information they assess.

The AQG system offers several user-friendly features to enhance the MCQ generation process. Users can specify the number of MCQs they want to generate, allowing customization based on assessment needs. The system automatically structures each question in a standardized format, ensuring clarity and consistency. Furthermore, every MCQ includes four answer choices—one correct answer and three distractors—ensuring comprehensive assessment options. This structured approach guarantees that the generated questions effectively evaluate the user's understanding of the topic.

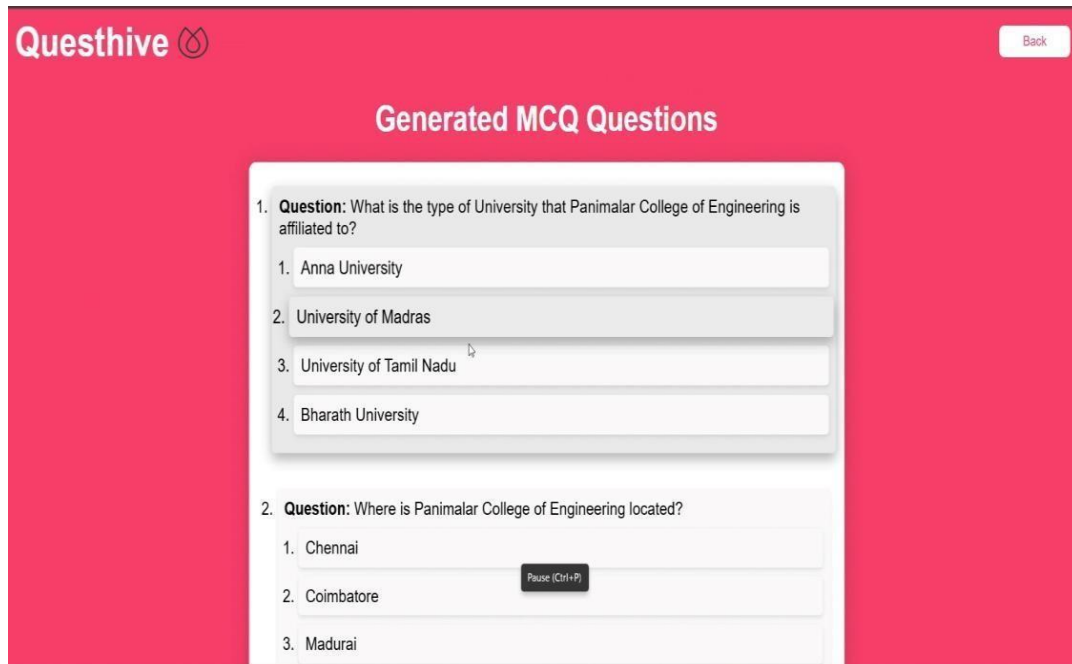


FIG 6.4 Output of MCQ Based Question Generation

The system follows a multi-step approach to ensure the accuracy and quality of the generated MCQs. First, it identifies key concepts from the input text by extracting important terms and phrases. Next, these key concepts are converted into structured question formats, such as, *"What is Python used for?"* Once the question is formed, the system generates four answer choices—one correct answer and three distractors. The distractors are designed to be plausible yet incorrect, making the question challenging and effective for learning. To further enhance the quality of the MCQs, the system applies randomization to ensure that the answer choices do not follow a predictable pattern.

Output and Benefits of Automated MCQ Generation

The AQG system's MCQ Generator significantly improves the efficiency of educational assessments by automating the question creation process. As shown in Figure 8.4, the generated output consists of structured MCQs that align with the given input. This automation benefits educators by saving time, reducing human error, and ensuring question diversity. Additionally, it enhances the learning experience for students by providing well-structured, topic-relevant questions that encourage deeper engagement. By incorporating AI-driven automation, the AQG system transforms how multiple-choice assessments are created, making it a valuable tool in modern education and training environments.

Coding Based Question Model

The **Automatic Code Question Generator** is one of the most advanced features of the Automated Question Generator (AQG) system, enabling users to generate coding-related questions effortlessly. By leveraging AI and Natural Language Processing (NLP) techniques, this feature allows users to create well-structured coding questions tailored to specific programming concepts. It eliminates the need for manual question formulation, making it an essential tool for educators, trainers, and coding assessment platforms.

The system provides a text input field where users can specify a programming topic, concept, or problem statement. Once the input is submitted, the system processes the information using AI-driven text analysis to extract essential details such as programming syntax, logic, and problem requirements. Based on this analysis, it generates structured coding questions that assess various aspects of programming knowledge, such as algorithm implementation, debugging, and code optimization.

Figure 6.5 illustrates how the Coding-Based Question Generator functions, ensuring that the generated questions align with the given topic.

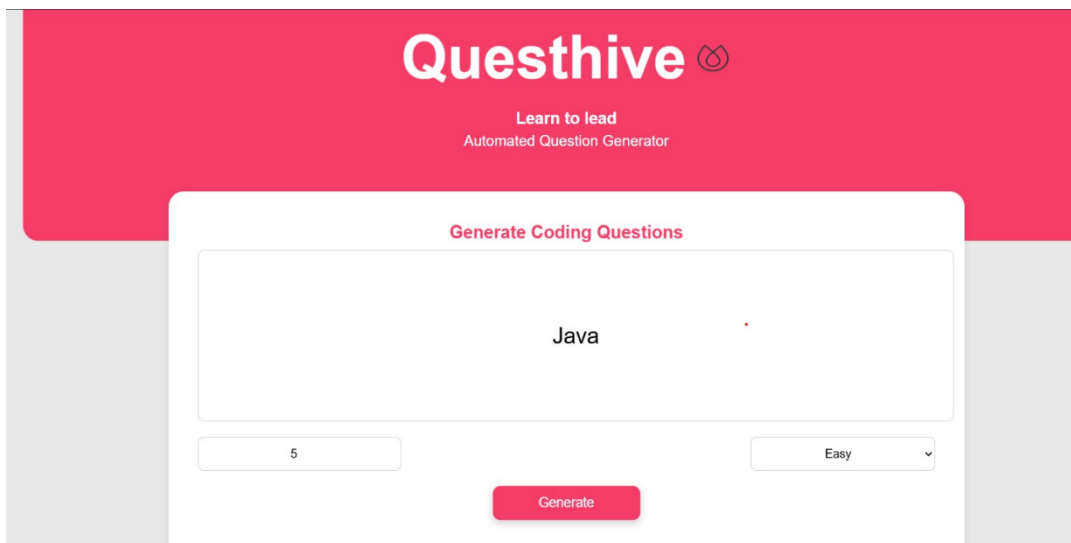
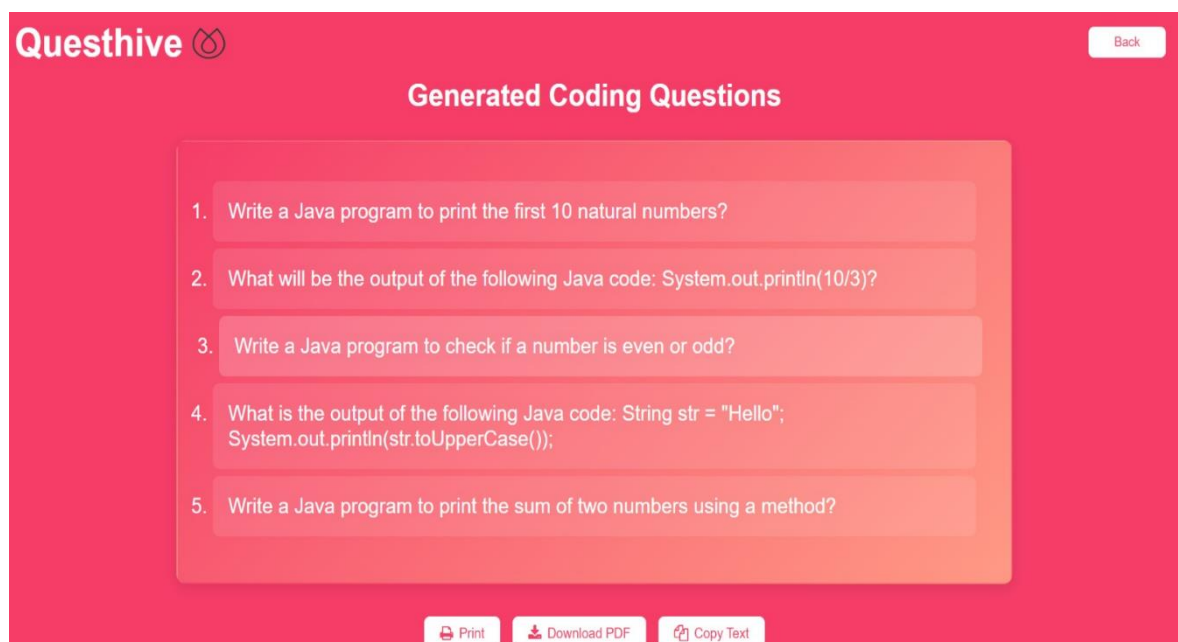
The screenshot shows the Questhive website's 'Generate Coding Questions' interface. At the top, the Questhive logo and tagline 'Learn to lead Automated Question Generator' are visible. The main form has a title 'Generate Coding Questions' and a large text input field containing the word 'Java'. Below this, there is a numeric input field with the value '5' and a dropdown menu set to 'Easy'. A red 'Generate' button is positioned at the bottom of the form.

FIG 6.5 Coding Based Question Generators

After processing the input, the AQQ system generates coding-related questions in different formats, such as coding problems, fill-in-the-blanks with missing code, debugging exercises, and multiple-choice coding questions (MCQs). This versatility

The screenshot displays the 'Generated Coding Questions' page on the Questhive website. It features a list of five coding questions in a light orange box. At the top right of the page is a 'Back' button. At the bottom, there are three buttons: 'Print', 'Download PDF', and 'Copy Text'.

1. Write a Java program to print the first 10 natural numbers?
2. What will be the output of the following Java code: `System.out.println(10/3)?`
3. Write a Java program to check if a number is even or odd?
4. What is the output of the following Java code: `String str = "Hello"; System.out.println(str.toUpperCase());`
5. Write a Java program to print the sum of two numbers using a method?

FIG 6.6 Output of the Coding Based Question Generators.

makes the system highly effective for use in coding assessments, competitive programming practice, and technical interviews. The output is structured to ensure clarity, with proper code snippets, test cases, and expected outputs, making it easy for

learners to understand and solve the problems. The generated output, as shown in Figure 6.6, reflects the system's ability to create relevant and meaningful coding challenges.

To enhance usability, the AQG system provides multiple export options that allow users to save, share, or integrate the generated coding questions into different formats. The Print feature enables users to print the generated questions directly, which is useful for teachers and trainers who need hard copies for classroom assessments. The print layout is optimized for readability, ensuring that printed coding problems retain proper formatting.

The Download feature allows users to save coding questions in formats such as PDF, DOCX, or CSV, facilitating easy integration with Learning Management Systems (LMS), online exam platforms, and coding boot camps. This ensures that educators and examiners can store and reuse generated questions for future assessments. Additionally, the Copy feature allows users to copy the generated questions and paste them into external documents, making it convenient for content creators, trainers, and e-learning platforms.

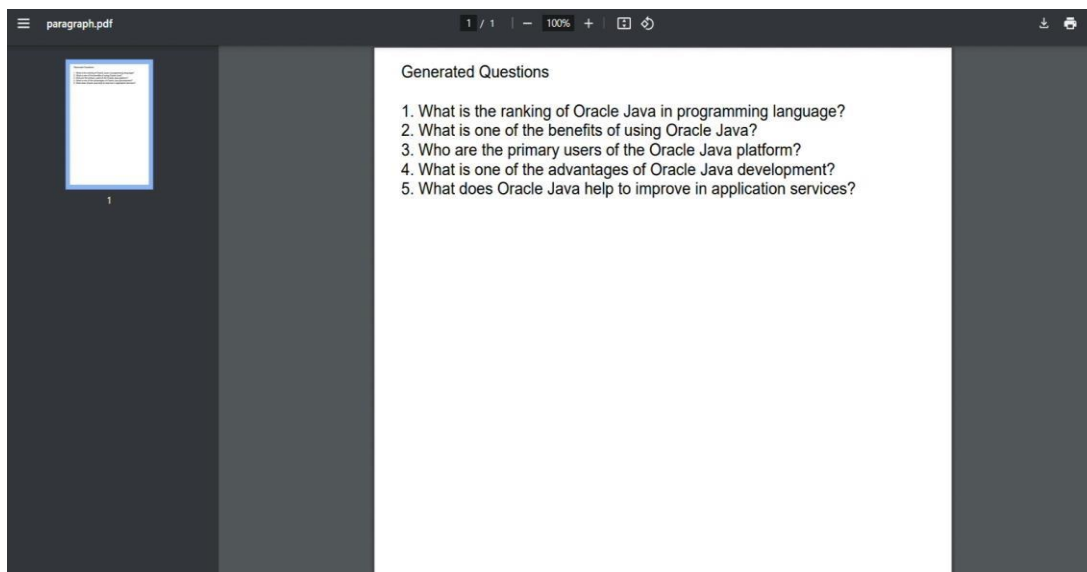


FIG 6.7 Trainer can Download the Questions

For example, a teacher preparing an online coding quiz can quickly generate questions, copy them, and paste them into a Google Form for seamless digital assessments. As

illustrated in Figure 6.7, these export functionalities significantly enhance the efficiency and accessibility of coding-based question generation, making the AQG system a powerful tool for programming education.

Comparison of Manual vs. Automated Question Generation

Question generation is a crucial aspect of education, assessments, and training. Traditionally, educators and examiners manually create questions based on textbooks, research materials, and subject expertise. However, with advancements in Artificial Intelligence (AI) and Natural Language Processing (NLP), Automated Question Generation (AQG) systems have emerged as a powerful alternative. These systems analyze input text and generate structured questions efficiently, reducing the effort required for manual question creation. This comparison highlights the key differences between manual and automated question generation in terms of efficiency, accuracy, scalability, and usability.

1. Efficiency and Time Consumption

Manual question generation is a labor-intensive process that requires educators to carefully analyze educational content, identify key concepts, and formulate questions that align with learning objectives. This process can be particularly challenging when designing comprehensive assessments that cover multiple topics, difficulty levels, or question formats such as multiple-choice, short answer, and coding-based questions. Additionally, educators must ensure that the questions are clear, unbiased, and accurately assess a student's understanding. This level of detail requires significant time and effort, making manual question generation a slow and resource-intensive task, especially in large-scale academic settings.

In contrast, automated question generation systems leverage AI and Natural Language Processing (NLP) to process input text and generate relevant questions instantly. These systems analyze textual content, extract essential concepts, and structure well-formulated questions within seconds, drastically reducing the time required for assessment preparation. This efficiency allows educators to focus more on refining and curating questions rather than spending excessive time drafting them

from scratch. Furthermore, automated systems can generate a diverse set of questions tailored to different difficulty levels, ensuring that assessments remain dynamic and comprehensive while significantly improving the overall productivity of educators.

2. Accuracy and Quality of Questions

The accuracy and quality of manually created questions largely depend on the educator's expertise, experience, and attention to detail. Skilled educators can craft well-structured and meaningful questions that effectively assess students' understanding. However, human error is inevitable, and issues such as grammatical mistakes, ambiguity, or unintended bias can affect the clarity and fairness of the questions. Additionally, maintaining consistency in question difficulty and ensuring coverage of all key concepts across multiple assessments can be challenging, particularly when large question banks need to be developed. These limitations make manual question generation a meticulous and sometimes inconsistent process.

Automated question generation systems leverage AI and Natural Language Processing (NLP) to ensure consistent and high-quality question formation. By analyzing large datasets, these systems can extract key terms, identify essential concepts, and generate grammatically correct and structured questions with minimal errors. This eliminates common human mistakes and ensures a standardized question format across assessments. However, while AI-generated questions are highly efficient and accurate, they may sometimes lack contextual depth or nuanced phrasing that a human educator would naturally incorporate. As a result, minimal human review and refinement are often necessary to optimize question quality, ensuring that they align with specific learning objectives and assessment standards.

3. Scalability and Adaptability

Manual question generation poses significant challenges in terms of scalability, as it requires educators to create a diverse set of questions across multiple subjects, difficulty levels, and assessment formats. Crafting questions for various exams—ranging from multiple-choice and descriptive questions to coding-based assessments—demands a considerable investment of time and effort. When dealing with large-scale assessments, such as standardized tests or university exams,

manually generating a sufficient number of quality questions becomes increasingly difficult. Furthermore, ensuring uniform difficulty levels and comprehensive subject coverage adds to the complexity, making manual question generation a time-intensive and resource-heavy task.

In contrast, automated question generation systems excel in scalability by rapidly producing large question banks across multiple topics and difficulty levels with minimal effort. These AI-driven systems can process vast amounts of educational content, extract key concepts, and generate a diverse range of questions tailored to specific learning objectives. Additionally, modern Automated Question Generation (AQG) systems are designed to adapt to different subjects, languages, and technical domains such as programming, mathematics, science, and humanities. This adaptability makes them highly versatile for academic institutions, corporate training, and professional certification exams. By leveraging automation, educational institutions and training organizations can ensure a steady supply of well-structured questions while significantly reducing the time and effort required for manual content creation.

4. Usability and Integration with Digital Platforms

As education and assessments increasingly shift toward digital platforms, the manual creation of questions presents several challenges in terms of usability and integration. Educators and exam administrators must manually format questions, enter them into Learning Management Systems (LMS), and ensure compatibility with different online assessment tools. This process can be cumbersome, especially when dealing with large question banks or diverse question formats like multiple-choice, coding-based, or comprehension-based exercises. Additionally, manually structuring assessments for digital platforms requires extra effort in organizing question types, assigning difficulty levels, and formatting them for seamless display on different devices. These factors make manual question creation time-consuming and less efficient for modern e-learning environments.

Automated Question Generation (AQG) systems significantly enhance usability by offering direct integration with digital platforms, reducing the need for manual

formatting and data entry. These systems support seamless export options, allowing users to download questions in multiple formats such as PDF, DOCX, and CSV, which can be easily uploaded to LMS, online quizzes, or e-learning portals. Additionally, automated systems can generate interactive question types, including MCQs, coding-based assessments, and comprehension exercises, making them highly suitable for modern online education, corporate training, and certification programs. While manual question generation allows for more creativity and customization, automated systems provide a faster, scalable, and more accessible alternative, ensuring that assessments remain dynamic, well-structured, and adaptable to various digital learning environments.

User Engagement and Performance Analysis

This section provides graphical representations and statistical evaluations to assess the system's efficiency, effectiveness, and overall performance. By analyzing key metrics, developers can gain valuable insights into user engagement levels, the quality of AI-generated questions, and the system's backend performance. These evaluations help in understanding how well the platform is functioning and whether it meets the intended objectives of automating question generation and improving assessment workflows.

By leveraging these insights, developers can refine the system, address potential limitations, and optimize various functionalities to enhance user experience. Continuous monitoring of performance metrics enables proactive improvements, ensuring that the platform remains efficient, scalable, and user-friendly. This iterative approach helps in maintaining a seamless experience for both educators and learners while fostering greater adoption of AI-driven question generation.

User engagement plays a critical role in evaluating the effectiveness of the platform. This metric measures how actively users interact with the system, including the number of registered users, login frequency, and total assessments completed. The system tracks the number of active users on a daily, weekly, and monthly basis, helping administrators understand engagement trends and peak usage periods. Additionally, the total number of assessments taken by students provides insight into learning activity levels. A high

number of completed quizzes indicates strong user participation and successful adoption of the system for educational and assessment purposes.

Another important engagement metric is session duration and interaction rates, which measure how long users stay on the platform and how frequently they interact with different features. A higher session duration indicates that users find the interface engaging and easy to navigate. Trainer involvement is also monitored by tracking the number of quizzes created by educators, the frequency of AI-generated question reviews, and modifications made to enhance question quality. By analyzing these factors, developers and administrators can refine the platform to improve user retention, enhance the learning experience, and introduce features that encourage greater interaction.

AI Question Generation Performance

The efficiency and reliability of the AI-driven question generator are crucial for ensuring the platform meets user expectations. The system evaluates question diversity by tracking the distribution of different question formats, including multiple-choice questions (MCQs), paragraph-based questions, and coding questions. MCQs are the most frequently generated type, accounting for 350 questions, indicating a strong preference for objective assessments in quizzes and exams. Paragraph-based questions (180 questions) are generated moderately, likely for comprehension-based assessments,

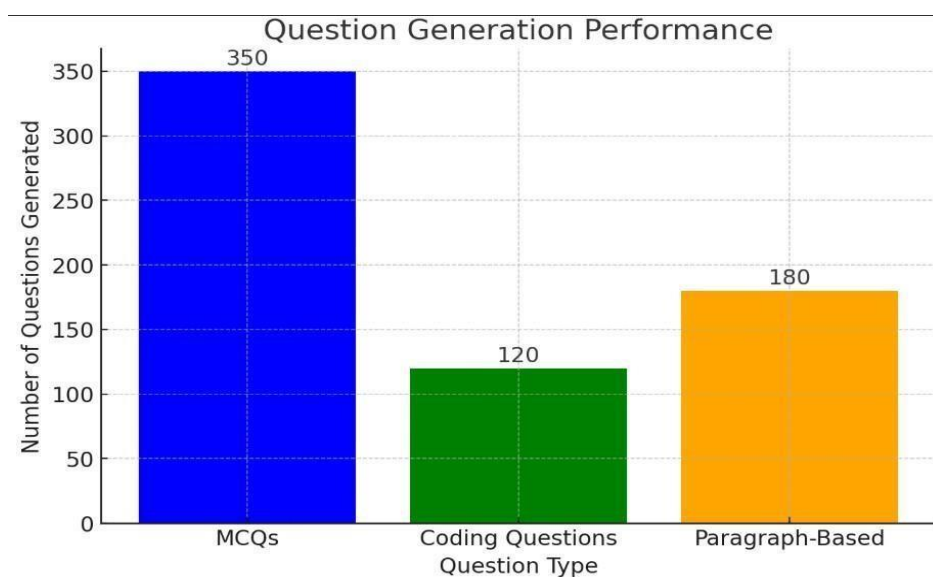


FIG 6.8 Graph Representation

while coding questions (120 questions) are the least generated, suggesting either lower demand or challenges in AI-generated programming questions.

Additionally, the accuracy and relevance of AI-generated questions are evaluated to ensure that they align with the selected topics and difficulty levels. If AI-generated questions appear irrelevant, repetitive, or inconsistent, further refinements to the AI model may be required. The efficiency of question generation is another key metric, as the system must generate questions quickly to meet real-time assessment demands. If any performance issues are detected, developers may need to enhance AI training data, improve question-generation algorithms, or introduce customization options that allow trainers to adjust the balance of different question types. These improvements ensure that the system remains adaptable, efficient, and valuable for both educators and learners.

System Performance Metrics

Evaluating the platform's backend performance is essential to ensure smooth operation, minimal delays, and efficient data handling. One of the key metrics analyzed is response time, which measures how quickly the system generates and displays content after a user action. A faster response time enhances the overall user experience, ensuring seamless interaction with the platform. If delays occur, it may indicate database inefficiencies or server limitations, requiring query optimizations or improvements in server infrastructure. Another critical factor is database query execution speed, which tracks how quickly stored data is retrieved and processed. Efficient query structures and indexing frequently accessed data can significantly improve system performance, reducing lag and ensuring a smooth user experience.

Another crucial metric is load handling and scalability, which determines the platform's ability to support multiple users simultaneously. Stress-testing server capacity helps evaluate whether the system can function smoothly during peak usage times without performance degradation. If bottlenecks are identified, developers can implement caching mechanisms, load balancing strategies, and database optimizations to improve efficiency. The graph below illustrates system performance metrics, showcasing

response time trends, database query execution speed, and system scalability performance over different load conditions. Maintaining low response times, fast query execution, and efficient load handling ensures that the platform remains reliable and scalable for a growing user base.

- **Response Time** – Measures the time taken for the system to generate and display content after a user action. A faster response time enhances the user experience, while delays may indicate the need for database query optimizations or improved server infrastructure.
- **Database Query Execution Speed** – Tracks how quickly the system retrieves and processes stored data. Optimizing query structures and indexing frequently accessed data can improve efficiency.
- **Load Handling & Scalability** – Assesses the system's ability to manage multiple users simultaneously. This includes stress-testing server capacity and ensuring smooth performance during peak usage times.

A well-optimized system should maintain low response times, execute queries efficiently, and handle concurrent users without significant slowdowns. If bottlenecks are identified, developers can implement caching mechanisms, load balancing strategies, and database optimizations to enhance performance.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

CHAPTER 7

CONCLUSION AND FUTURE WORKS

7.1 Conclusion

The development of an automatic question generator using Flask and the OpenAI API establishes a robust framework for generating diverse and dynamic coding questions tailored to different skill levels. By leveraging AI-powered automation, the system can create well-structured coding challenges that cater to beginners, intermediate learners, and advanced programmers. This not only streamlines the process of question generation but also ensures that assessments are fair, engaging, and aligned with the user's proficiency level. The integration of AI also minimizes human effort in curating questions manually, making it a valuable tool for educators, trainers, and online learning platforms.

Beyond its immediate benefits, this initiative significantly enhances the efficiency of content creation for educational and training purposes. Automated question generation reduces the workload for instructors while ensuring a steady supply of high-quality coding problems. The system can dynamically create questions based on specific programming concepts, ensuring that learners receive targeted assessments that reinforce their understanding. As AI-driven learning continues to evolve, the potential for real-time evaluations becomes a critical aspect of personalized education, allowing students to receive instant feedback and refine their coding skills effectively.

Future advancements in this project could include multimedia-rich questions, adaptive learning features, and integration with widely used learning management systems (LMS). By incorporating multimedia elements such as interactive code editors, explanatory videos, and real-time hints, the system can enhance the learning experience. Adaptive learning features could enable the system to modify question difficulty based on a learner's performance, ensuring a customized and engaging journey. Integration with popular LMS platforms like Moodle, Blackboard, or Google Classroom would further expand the reach of this technology, making it accessible to a larger audience.

Additionally, the implementation of security features and AI-driven analytics will play a crucial role in maintaining the reliability and effectiveness of the system. Ensuring secure access to question generation, preventing misuse, and maintaining data integrity will be essential for widespread adoption. Moreover, incorporating gamification and collaborative tools—such as coding challenges, leaderboards, and peer discussions—will enhance user engagement and foster a more interactive learning environment. Ultimately, this project lays the groundwork for a scalable, customizable, and interactive approach to modern learning and assessment methodologies.

7.2 Future Works

The future development of the automatic question generator can focus on enhancing personalization, scalability, and interactivity. One key area for improvement is the integration of adaptive learning techniques, where the system can analyze user performance and dynamically adjust question difficulty to match individual skill levels. This would create a more tailored learning experience, helping learners progress at their own pace. Expanding support for multiple programming languages and frameworks will further increase the system's applicability across different educational and professional training domains.

Another crucial aspect of future work involves improving system security, analytics, and collaborative features. Strengthening security protocols will ensure that question generation remains fair and protected from misuse, while AI-driven analytics can provide valuable insights into user performance, learning patterns, and question effectiveness. Gamification elements, such as coding challenges, leaderboards, and peer-reviewed solutions, can boost engagement and motivation. Furthermore, seamless integration with existing Learning Management Systems (LMS) will allow educational institutions and training organizations to adopt the system effortlessly. By continuously refining its capabilities, the question generator can evolve into a comprehensive, intelligent assessment tool that supports modern education and professional development.

CHAPTER 8

REFERENCES

CHAPTER 8

REFERENCES

REFERENCES

- [1] M. Onat Topal, Anil Bas, Imke van Heerden “Exploring Transformers in Natural Language Generation: GPT, BERT, and XLNet” 2021.
- [2] Raul Puri, Ryan Spring, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro “Training Question Answering Models From Synthetic Data” 22 Feb 2020
- [3] Agnès Mustar, Sylvain Lamprier, Benjamin Piwowarski “Using BERT and BART for Query Suggestion” 5 Oct 2021.
- [4] Shuyang Cao and Lu Wang “Controllable Open-ended Question Generation with A New Question Type Ontology “1 Jul 2021
- [5] Chenyang Lyu, Lifeng Shang, Yvette Graham, Jennifer Foster, Xin Jiang, Qun Liu “Improving Unsupervised Question Answering via Summarization- Informed Question Generation “ 16 Sep 2021 .
- [6] Md Rashad Al Hasan Rony, Uttam Kumar, Roman Teucher, Liubov Kovrigina, and Jens Lehmann “A Generative Approach for SPARQL Query Generation From Natural Language Questions “5 July 2022 .
- [7] Sivasurya Santhanam “context based text-generation using lstm networks “May 4, 2020
- [8] Nikahat Mulla, Prachi Gharpure “Automatic question generation: a review of methodologies, datasets, evaluation metrics, and applications “30 January 2023
- [9] Ghader Kurdi, Jared Leo, Bijan Parsia, Uli Sattler, Salam Al-Emari “A Systematic Review of Automatic Question Generation for Educational Purposes “21 November 2019

- [10] Stefan Feuerriegel, Jochen Hartmann, Christian Janiesch, Patrick Zschech
“Generative AI “: 12 September 2023
- [11] Xiyao Ma, Qile Zhu, Yanlin Zhou, Xiaolin Li² “Improving Question
Generation with Sentence-Level Semantic Matching and Answer Inferring
- [12] Luis Enrico Lopez, Diane Kathryn Cruz, Jan Christian Blaise Cruz,
Charibeth Cheng “Transformer-based End-to-End Question Generation” May 2020.
- [13] Shweta Yadav, Deepak Gupta , Asma Ben Abacha, Dina DemnerFushman”
Question-aware transformer models for consumer health question summarization”
- [14] Asahi Ushio and Fernando Alva-Manchego and Jose Camacho- Collados”
Generative Language Models for Paragraph-Level Question
Generation” 2 Jan 2023.
- [15] David Baidoo-Anu¹ , Leticia Owusu Ansah” Education in the Era of Generative
Artificial Intelligence (AI): Understanding the Potential
Benefits of ChatGPT in Promoting Teaching and Learning” 31.12.2023.
- [15] Kale, Sahil, Gautam Khaire, and Jay Patankar. "FAQ-Gen: An automated system to
generate domain-specific FAQs to aid content comprehension." arXiv preprint
arXiv:2402.05812 (2024).
- [16] Riza, Lala Septem, et al. "Automatic generation of short-answer questions in
reading comprehension using NLP and KNN." Multimedia Tools and Applications 82.27
(2023): 41913-41940.
- [17] Mehta, Pritam Kumar, et al. "Automated MCQ generator using natural language
processing." Proceedings of the Thirteenth Workshop on Innovative Use of NLP for
Building Educational Applications. 2021.
- [18] Thotad Puneeth, Shanta Kallur, and Sukanya Amminabhavi. "Automatic question
generator using natural language processing." Journal of Pharmaceutical Negative
Results 13 (2022): 2759-2764

PUBLICATION



Inderscience Publishers: IJAISC-258191 - Submission Acknowledgement

1 message

Inderscience Submissions <submissions@inderscience.com>
To: arthirbsa22@gmail.com

Mon, 17 Mar 2025 at 11:14 am



Dear Ms. Arthi R, Ms. Nila D, Ms. Nithya Shree K,

Thank you for submitting your paper entitled 'AUTOMATED QUESTION BUILDER APPLICATION USING GEN-AI' to the journal: Int. J. of Artificial Intelligence and Soft Computing

Your submission code is: IJAISC-258191.

This paper will now be screened to filter out incomplete or unsuitable content (like author identifying details etc.).

You can track progress by logging in to the Inderscience Submissions system at
<https://indersciencesubmissions.com/>

You can get username and password reminders on the log in page.

Please note that there are no charges for publishing with Inderscience, unless you are submitting to an Open Access only journal or you want your article to be Open Access (OA).

If you receive an email requesting payment in relation to your article (for example for editing or reviewing services), then you should ignore and delete the email – it is not a legitimate Inderscience email. If you are unsure, you can check with us at: submissions@inderscience.com
If you are considering publishing an Open Access article with us, remember that we will never request payment before your paper has been accepted and payment will be only be organised and handled by the Inderscience Editorial Office.

Thank you for your interest in publishing with Inderscience.

Kind regards,
The Inderscience Submissions Team
Inderscience Publishers
submissions@inderscience.com

APPENDIX

CODE

```
import sys
import json
import re
from groq import Groq

def generate_mcqs(num_questions, topic, difficulty):

    client = Groq(api_key="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")

    completion = client.chat.completions.create(
        model="llama3-8b-8192",
        messages=[
            {
                "role": "user",
                "content": f"Generate me {num_questions} MCQ questions on the topic of {topic} with a difficulty level of {difficulty}. Return the output in JSON format."
            }
        ],
        temperature=1,
        max_tokens=1024,
        top_p=1,
        stream=True,
        stop=None,
    )

    response_text = ""
```

```

for chunk in completion:
    response_text += chunk.choices[0].delta.content or ""

json_match = re.search(r'\[s*\{.*\}\s*\]', response_text, re.DOTALL)

if not json_match:
    return json.dumps({"error": "Failed to extract JSON"})

json_data = json_match.group(0)

try:
    questions_list = json.loads(json_data)
    return json.dumps(questions_list)
except json.JSONDecodeError:
    return json.dumps({"error": "Failed to decode JSON"})

if __name__ == "__main__":
    num_questions = sys.argv[1]
    topic = sys.argv[2]
    difficulty = sys.argv[3]

    questions_json = generate_mcqs(num_questions, topic, difficulty)
    print(questions_json)

```

Coding part :

```
<div id="questhive">
```

```
    Questhive
```

```
        
```



```
</div>
<h4>Learn to lead</h4>
<p>Automated Question Generator</p>
</div>
```

```
<div id="form-container">
  <div class="by-topic" >Generate Coding Questions</div>
  <form action="coding_result.php" method="POST">
    <input type="text" id="topic" name="topic" placeholder="Enter the topic (Ex:
JAVA)" required>
    <div class="input-container">
      <input type="number" id="number" placeholder="Number of Questions"
name="number" required>
      <select id="difficulty" name="difficulty" required>
        <option value="">Select Difficulty</option>
        <option value="easy">Easy</option>
        <option value="medium">Medium</option>
        <option value="hard">Hard</option>
      </select>
    </div>
  </div>
```

Paragraph generation :

```
<div id="form1">
  <h1>Generate Paragraph Questions</h1>
  <form method="post" action="paragraph_results.php">
    <textarea name="itext" id="itext" placeholder="Enter the Paragraph"
required></textarea><br/>
```

```
<div class="input-container">
  <input type="number" id="noq" placeholder="Number of Questions" name="noq"
required>
  <select id="noq" name="difficulty" required>
    <option value="">Select Difficulty</option>
    <option value="easy">Easy</option>
    <option value="medium">Medium</option>
    <option value="hard">Hard</option>
  </select>
</div>
```

Drive Link:

<https://drive.google.com/drive/folders/1JCjvqxmExACHRBLR-pEaBhLBU8XgwCG2?usp=sharing>

Github Link:

<https://github.com/NilaDhanagopal/automated-question-generator.git>

AUTOMATED QUESTION BUILDER APPLICATION USING GEN-AI

Arthi R¹

Department of Artificial Intelligence and Data Science
Panimalar Engineering College Chennai, India.
arthirameshkumar22@gmail.com

Nithya Shree K³

Department of Artificial Intelligence and Data Science
Panimalar Engineering College Chennai, India.
nithya.shree12304@gmail.com

Nila D²

Department of Artificial Intelligence and Data Science
Panimalar Engineering College Chennai, India.
niladhanagopal2003@gmail.com

Dr. M.S.Maharajan ⁴

Associate Professor
Department of Artificial Intelligence and Data Science
Panimalar Engineering College Chennai, India.
maha84rajan@gmail.com

Abstract — *The generation of educational content can now be automated because to developments in generative AI, which has important implications for evaluation and personalized learning. In order to generate a variety of questions with different levels of complexity, including multiple-choice, coding, and paragraphbased questions, this work provides an automated question generator application. In order to ensure context and relevance, the application dynamically generates questions based on input themes using Groq's Lemma API, large language models (LLMs), and natural language processing (NLP) approaches. The application successfully generates a range of well-organized questions that are in line with input specifications, according to the results. This study shows how generative AI may improve educational technologies by automating content creation, which saves time and money on creating high-quality questions.*

Keywords—Groq's Lemma API, Large language model, Natural language processing.

I. INTRODUCTION:

Automatic Question Generation (AQG) has emerged as a crucial area in Natural Language Processing (NLP) as a result of the need to enhance conversational agents and instructional materials. New developments in AQG systems demonstrate a number of techniques, including as rule-based strategies and neural network structures like Transformers, which have transformed the creation of logical and contextually appropriate queries. There is a noticeable emphasis on a variety of applications in the literature, from visual and conversational contexts to independent question production. In order to overcome the difficulties of preserving quality and relevance, these developments show how models such as GPT, BERT, and XLNet may create questions that nearly resemble content created by humans.

Using Groq's Lemma API, we create an Automated Question Generator Application in this project that effectively creates questions from text inputs. Our application uses the insights from related efforts to improve the quality of question generation by incorporating cutting-edge generative AI techniques. This emphasis on using huge language models is consistent with recent research showing that synthetic data is a powerful tool for training question-answering systems.

Additionally, by filling in gaps in the creation of excellent, pertinent questions that can greatly improve user engagement and learning outcomes, this initiative aims to add to the current conversation in AQG.

A key component of enhancing user engagement and interactive learning in a range of applications, including chatbots and educational systems, is Automatic Question Generation (AQG). The caliber and pertinence of generated questions have been greatly enhanced by the latest developments in AQG techniques, especially those that employ neural network designs like as Transformers. Although template-based and supervised question generation are examples of classic approaches that have established the foundation, their dependence on particular datasets and templates typically limits their flexibility and diversity.

Recent research on the development of open-ended questions that call for multiple-sentence answers has suggested a new question type ontology that more accurately captures the complex nature of questions than conventional question words. This study introduces a labeled dataset with 4,959 questions to demonstrate the significance of developing datasets that more accurately capture the intricacies of natural language

II. RELATED WORKS:

Advanced technologies like as BERT and BART, which are based on transformer topologies, are used in question generating when query suggestion is being used. In order to anticipate the subsequent inquiry based on the order of prior queries, these models are trained on enormous datasets of user sessions. These models can provide inquiries that are pertinent to the context by efficiently capturing the connections and dependencies between queries through the use of attention techniques. By optimizing the probability of noticing the subsequent question based on the preceding ones, the training process allows the models to understand user intent and behavior during intricate search tasks.

Transformer-based models can therefore generate a variety of well-reasoned query recommendations that meet user

requirements, surpassing conventional RNN-based methods in managing long-range dependencies and data noise [3].

Transformer-based models like GPT, BERT, and XLNet can be utilised to effectively create questions in Natural Language Generation (NLG). Based on the input text, these models use attention mechanisms to comprehend context and produce logical queries. With its enormous parameter size, GPT-3, for example, may produce questions regardless of the task, exhibiting good performance even in the absence of task-specific fine-tuning. BERT, on the other hand, uses bidirectional context to improve comprehension, which can be used to make pertinent queries from a text. Overall, by successfully evaluating and comprehending the underlying content, these models' improvements make it easier to create high-quality queries [1].

Generative AI technologies are used to generate questions by using textual descriptions of problems as a basis. According to recent empirical studies, generative AI can reliably create conceptual models from these descriptions, including ER, BPMN, and UML. This feature implies that by analyzing and converting textual inputs into structured queries or prompts, generative AI can be used to generate pertinent questions, improving the efficacy of teaching and evaluation resources [10].

Rule-based approaches apply transformation and pattern matching techniques to declarative statements, simplifying the text first, and then using particular patterns to generate questions. Conversely, neural network-based methods use encoder-decoder architectures specifically, RNNs and transformer models to produce queries from input text. To discover the connections between queries and their accompanying answers, these models are trained on huge datasets like WikiAnswers. In order to develop questions that are both syntactically and semantically legitimate, it is imperative to use Natural Language Processing (NLP) techniques to parse the input text and extract relevant semantic information [8].

This approach generates the questions and anticipates their main points simultaneously. With two model variants—one that utilises pre-identified exemplars and the other that uses generated templates to assist in question writing—the research also used templates to enhance controllability and diversity in the generated questions. A new dataset labeled according to a defined question type ontology supports the method, which distinguishes the subtle nature of questions more well than commonly used question phrases [4].

Employing a methodology that presents QG as a process of summarization and inquiry. This entails using publicly accessible summary data and annotating the summaries with semantic roles, dependency parsing, and named entity recognition. The parsed summaries are utilized to produce to produce questions using a set of heuristics. The generated questions are used in conjunction with the source news articles to train an end-to-end neural QG model [5].

One prominent technique is the sequence-to-sequence system created by Soru et al., which generates SPARQL templates using bi-directional LSTM. Nevertheless, this approach has

drawbacks, including the incapacity to handle terms that are not part of the lexicon and a lack of comprehension of the inquiry, which results in improper graph patterns in the queries that are generated. Using large-scale language models to encode linguistic information from natural language queries, SGPT is an additional method that blends end-to-end and modular systems. To produce precise SPARQL searches, our system learns intricate inquiry patterns and adds graph-specific information to the language model's parameters. Furthermore, SGPT adapts common assessment criteria and presents training methods to gauge performance in SPARQL query creation.

Networks of Long Short-Term Memory (LSTM) are used to produce natural language. The model is trained on a dataset utilising input words and context vectors, which encapsulate the semantic meaning of the sentences, in order to predict the next word in a sequence. To improve the model's comprehension of the context and produce more intelligible text, context vectors are used in conjunction with input-output instances during the training phase [7].

A three-step modeling pipeline comprising question generation, question filtration, and unconditional answer extraction from text is the main method used to generate questions in this context. A next-token-prediction language modeling aim is used in conjunction with pretrained models, namely a GPT-2 decoder model. Concatenating context, answer, and question tokens during training enhances the caliber of questions produced by the model. Large generative transformer models, which can have up to 8.3 billion parameters, are also used to improve the caliber of the questions that are produced [2].

Fine-tuning techniques based on transformers that employ a single pre-trained language model without the need for additional features, answer information, or other procedures [1]. One of the most widely used Deep Learning-based techniques for QG is Sequence-to-Sequence (Seq2Seq) models, which use LSTM-based neural networks to encode the source context paragraph and decode it to generate a generated question [12].

Transformers-based fine-tuning approaches leverage a single pre-trained language model without incorporating extra features, answer annotations, or supplementary steps. Among the most prevalent deep learning methods for question generation (QG) are Sequence-to-Sequence (Seq2Seq) models. These models utilize LSTM-based neural networks to process the given context, encoding the source paragraph and decoding it into a generated question.

By utilizing ChatGPT's features, educators can generate open-ended question prompts that correspond with the learning objectives and success standards of the lesson. Furthermore, ChatGPT may be used to create high-quality rubrics that succinctly and clearly outline the precise tasks that students must complete in order to pass the various competence levels. Accordingly, by employing unique and unusual artifacts, generative AI-powered evaluation systems may facilitate the incorporation of ongoing input into learning procedures [15].

III .EXISTING SYSTEM:

Technology	Function in AQG	Strength	Limitation
LSTM	<ol style="list-style-type: none"> 1. Retains long-term dependencies in sequential data. 2. Generates coherent, contextually relevant questions. 	Handles long sequences and maintains context well.	Limited vocabulary handling; struggles with out-of-vocabulary terms.
Seq2Seq Model	<ol style="list-style-type: none"> 1. Encodes input context and decodes relevant questions. 2. Suitable for translation, summarization, and AQG tasks. 	Captures context effectively; useful in conversational AI.	Struggles with complex dependencies and out-of-vocabulary words.
BERT	<ol style="list-style-type: none"> 1. Generates contextually coherent queries by understanding word relationships. 2. Captures user intent effectively. 	Captures nuanced word meanings, handles long-range relationships well.	High computational requirements; less effective for strict sequential dependencies.
Transformer Model	<ol style="list-style-type: none"> 1. Uses attention to produce contextually relevant questions. 2. Efficient at generating coherent text responses. 	Efficient processing with attention, captures complex dependencies.	High resource demands; may struggle with strictly sequential data.
BART	<ol style="list-style-type: none"> 1. Generates accurate, contextually varied questions from input. 2. Combines bidirectional and autoregressive models. 	Adaptable for diverse tasks, produces coherent and accurate questions.	Requires tuning for complex formats; high computational needs.
RNN	<ol style="list-style-type: none"> 1. Maintains hidden state for sequential question generation. 2. Adapts to user context over a session. 	Effective for short dependencies and session-based tasks.	Struggles with long sequences (vanishing gradient); needs LSTM or GRU variants for better long-range dependency handling.

LSTM:

An LSTM (Long Short-Term Memory) network is a form of recurrent neural network (RNN) designed especially to process sequential data by retaining long-term dependencies.. They are made up of input, forget, and output gates that regulate information flow, enabling the network to remove extraneous information while retaining relevant features across long sequencesAs a result, LSTMs are suitable for a variety of natural language processing applications, including text production and questionanswering systems, since they can preserve the history of lengthy sentences. In educational or conversational AI applications, in particular, LSTMs may formulate queries that are both coherent and contextually accurate by processing text word by word while preserving context across phrases.

LSTM functions by preserving long-term information through its distinct design, which consists of several gates: cell states, input, forget, and output gates. These gates establish what historical data should be kept and what should be thrown away. Because of this, LSTMs can retain the history of lengthy sentences, which makes them appropriate for a range of natural language processing uses the history of lengthy sentences, which makes them appropriate for a range of natural language processing uses, such as text production and question-answering systems. In order to generate more coherent text, the model uses context vectors to better comprehend the relationships between the input words and is trained to predict the next word in a sequence depending on the input words [7].

In order to generate SPARQL templates, Soru et al. created a sequence-to-sequence system that incorporates LSTM. Although it has drawbacks, like the inability to handle terms that are not in the lexicon and a lack of comprehension of the query, this system uses bi-directional LSTM to manage the generation process, which may result in improper graph patterns in the queries that are generated [6].

SEQUENCE TO SEQUENCE:

A neural network design known as a sequence-to-sequence (Seq2Seq) model converts an input sequence into an output sequence; it is frequently employed for tasks including question creation, summarization, and translation. It consists of two primary parts: an encoder that compresses the input text (such a phrase or paragraph) into a context vector and then uses that context vector to create a logical output sequence, word by word. Seq2Seq models are trained to generate pertinent questions from an input passage in automatic question generation (AQG). The decoder formulates queries that are appropriate for the context based on the encoder's knowledge of the input text. This strategy is especially helpful with conversational AI and educational systems, where creating accurate and meaningful questions can improve learning and engagement.

Sequence-to-Sequence (Seq2Seq) models employ LSTM in question creation. Where an LSTM-based neural network encodes the source context paragraph and another LSTM-based network decodes the embedded data to provide a generated question [12].

Sequence-to-sequence (seq-to-seq) models use an encoder-decoder architecture to transform an input sequence into an output sequence. Once the input sequence has been processed, the encoder creates a fixed-size context vector that contains the input's information. After receiving this context vector, the decoder creates the output sequence, usually one element at a time. Attention mechanisms are frequently added to this method to enable the model to concentrate on various input sequence segments while producing each output element, enhancing the output's quality and relevance [11].

BERT :

In query recommendation, BERT (Bidirectional Encoder Representations from Transformers) provides contextual embeddings for user queries. It helps the model better comprehend user intent by capturing the connections and dependencies between words in a query. BERT is optimized for particular tasks, including anticipating the subsequent inquiry in a series based on earlier inquiries in a user session, and has been pre-trained on huge datasets. Because of this feature, BERT can produce contextually appropriate and coherent query suggestions, surpassing more conventional models like RNNs, particularly when it comes to managing long-range relationships and data noise [3].

BERT (Bidirectional Encoder Representations from Transformers) is used to generate enquiries by leveraging its bidirectional context awareness. This enables BERT to perform a thorough analysis of the input text and produce pertinent

queries depending on the information. Without requiring a great deal of task-specific fine-tuning, its architecture allows it to capture the subtleties of language, making it useful for tasks like question generation [1].

TRANSFORMER MODEL:

GPT is based on the transformer architecture, a type of neural network that performs well on tasks involving natural language processing. After being exposed to a large text dataset, which includes books and articles, it learns to generate material that is comparable to the text it was trained on. In response to a prompt or context, the model evaluates the data and generates a response one word at a time, predicting the next word based on the input and the words it has previously created. The model uses attention processes to focus on the most relevant parts of the input in order to generate responses that make sense and are appropriate for the situation [15].

To fine-tune the model for a specific task, T5 is applied by using a task prefix. The task prefix "generate question:" is used at the beginning of the input text to create questions. Parameter optimisation is used during the finetuning phase to determine the model's optimal configuration [14].

BART:

BART functions as a denoising sequence-to-sequence model that has been trained for tasks involving comprehension, translation, and natural language creation. It efficiently generates text based on the input it gets by combining bidirectional and autoregressive transformers. The model is optimized for certain tasks, such question creation, where it may generate questions based on pre-provided answers or circumstances [2].

There are several ways to use BART to generate questions, including using alternative model sizes (BASE and LARGE) and specific parameter settings for optimisation. [14].

An encoder and a decoder make up the entire architecture of the bidirectional and auto-regressive transformer utilised in query suggestion, which makes it particularly helpful for tasks involving text synthesis. It is trained on a variety of tasks, including phrase permutation, text infilling, and token masking, which improves its capacity to produce logical and contextually appropriate inquiries based on prior user interactions. BART uses a gradual unfreezing strategy during training, which enables the model to efficiently adjust its parameters. This method enables BART to produce a variety of query suggestions that are pertinent to the user's search context, outperforming other models like BERT [3].

RECURRENT NEURAL NETWORK:

The ability of recurrent neural networks (RNNs) to evaluate sequential input in Natural Language Generation (NLG) makes them valuable for language modelling and text generated. By keeping a concealed state to preserve data from prior time steps, they are able to produce text one word at a time while taking earlier words' context into account. However, as input length

increases, RNNs face difficulties such as the vanishing gradient problem that make it more difficult for them to understand long-range relationships in sequences. Because of this constraint, sophisticated systems like Transformers have been created to solve these problems [1].

By analysing user query sequences and keeping a hidden state that records information from prior inputs, recurrent neural networks (RNNs) are used in query suggestion. By changing their representation with every query, RNNs are able to comprehend context and intent and model user behaviour over time. RNNs are appropriate for user sessions with varying query counts since they can handle variable-length sequences. Their ability to completely capture user session context may be hampered by their inability to handle long-range dependencies because to vanishing gradient problems. Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) are two variations that have been developed to better regulate information flow across longer sequences [3].

Recurrent neural networks (RNNs) are used in an encoderdecoder architecture for Automatic Question Generation (AQG) The encoder uses several recurrent neural network layers to handle an input sequence that contains a passage and its matching response. The encoded form of the response is then used by the decoder to create a question. A bi-directional RNN is used to enhance the quality of the produced questions by improving context collection from both directions of the input sequence. When compared to baseline techniques, the RNN model has shown higher BLEU ratings, demonstrating its efficacy in generating pertinent queries [8].

IV. ARCHITECTURE DIAGRAM:

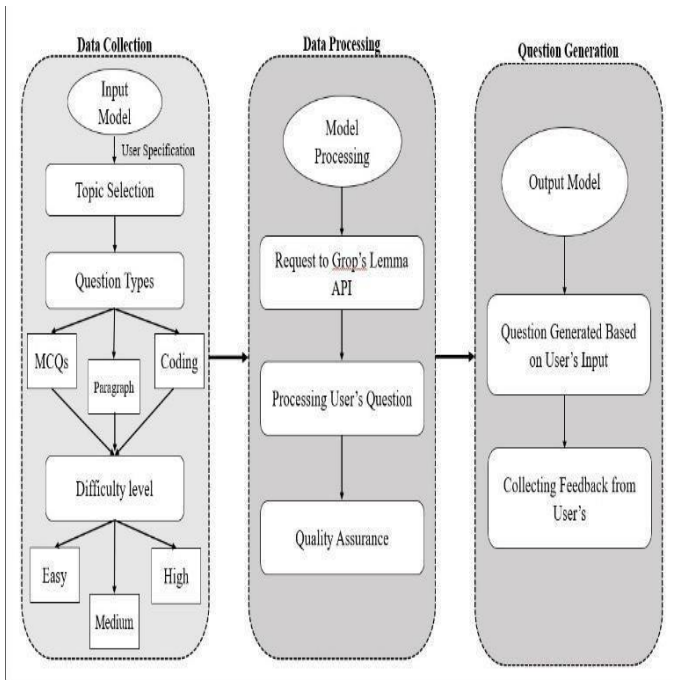


FIG 1: ARCHITECTURE DIAGRAM

With three primary stages Data Collection, Data Processing, and Question Generation—the graphic shows the workflow of

a Automatic Question Generator (AQG). Users can set a number of characteristics, such as the topic, question type, and degree of difficulty, during the Data Collection phase, when the system first gets input from the user. Multiple- choice questions (MCQs), code questions, and paragraph- based questions are among the question categories the user can select from. They can also choose an easy, medium, or high difficulty level to match the amount of complexity they want.

In order to generate questions depending on the chosen topic and format, the AQG system first prepares the input data for question creation by sending a request to Groq's Lemma API, which manages natural language processing. Questions are processed using the API.

V. PROPOSED SYSTEM:

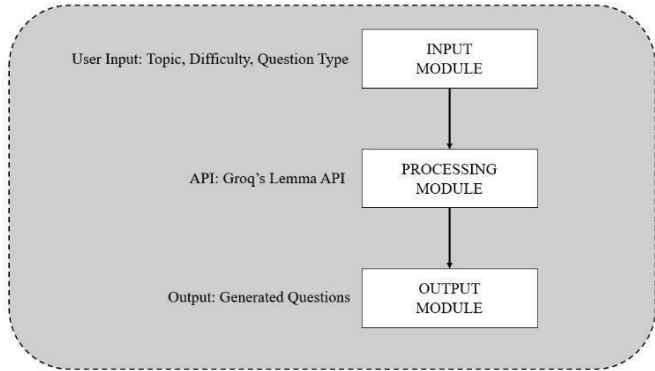


FIG 2: WORKFLOW DIAGRAM

The automated question generator application's suggested system architecture uses a modular approach to expedite the question creation process. Three main parts make up the architecture: the input module, processing module, and output module. Because of the clear data and interaction flow guaranteed by this design, users may quickly create queries based on their needs.

Diagram concept:

- Using a flowchart or diagram, illustrate how the three primary parts are related in order.
- The data flow from the input module to the processing module and then to the output module can be shown using arrows.
- Provide icons or representations for every module, such as a webpage for outputs, a cloud for the API, and text boxes for inputs.

COMPONENT:

1. The input module

The Input Module functions as the user interface via which users, instructors, or trainers can define the specifications needed to generate the questions. This module's user-friendly and intuitive design allows it to record important characteristics.

User field input:

- **Selecting a Topic:** Users can enter a certain subject or topic to have questions generated for it. Auto-suggestion tools can be added to this to assist users identify pertinent topics more quickly.
- **Levels of Difficulty:** Users can choose from a list of predetermined possibilities for the questions' difficulty (e.g., Easy, Medium, Hard). You can use a dropdown menu or radio buttons to accomplish this.
- **Question Types:** Multiple Choice Questions (MCQs), coding questions, and descriptive questions are among the various question styles from which users can select. This can be done using a multi-select dropdown or checkboxes.

Additional features:

- **Preview Section:** Before creating questions, users can verify their selections when a preview section displays an overview of the inputs they have chosen.
- After all inputs have been filled out, the Submit button will start the question creation process.

2. Processing module

- The system's central component, the Processing Module, is in charge of creating queries using the data that the Input Module provides. This module generates high-quality questions by utilizing a variety of NLP approaches and Groq's Lemma API.
- **Data Flow:** The module sends a request to Groq's Lemma API with the chosen topic, degree of difficulty, and question type as parameters after receiving the inputs.

Procedure for Creating Questions:

- **API Interaction:** Using the specified criteria, the Processing Module creates an API call to Groq's Lemma, which yields a JSON response with questions created. The use of AI and NLP models by the API guarantees that the queries are suitably difficult and contextually relevant.
- **Post-processing:** After the questions are received, they could go through extra steps to ensure proper formatting or to eliminate any irrelevant or duplicate results. This action could entail:
- **Natural Language Processing (NLP):** To improve the caliber of questions produced, methods like relevance score, keyword extraction, and text normalization can be used.
- Using validation methods to verify the coherence and clarity of the generated questions is known as quality assurance.

3. The Output Module:

The created questions must be shown to users or sent by the output module. This module makes sure that the created material is easy for users to read, engage with, and use.

Display mechanisms:

- **Question List:** The generated questions are presented in an organized manner, perhaps as a table or list view, with options for each question (if applicable for multiple-choice questions).
- **Interactive Components:** Add buttons that allow users to:
- **Copy:** To make sharing or using questions easier, copy them to the clipboard.
- **Print:** For offline use, print the questions directly.
- **Download:** For convenience, offer the ability to download the questions in a number of different formats (such as Word and PDF).

User feedback:

- A feedback form embedded into the Output Module allows users to comment on the questions' quality and relevancy once they have viewed the generated questions. The question creation algorithms can be gradually improved with the help of this input.

VI. FUTURE WORK:

When developing your automatic question generator in the future, think about adding adaptive learning paths and user progress tracking to improve user customisation. Include project-based assignments, brief responses, and multimedia materials such as code excerpts and video explanations in your question types. Including gamification elements like challenges, leaderboards, and badges can increase user involvement. Include an automated response explanation system, a feedback system to improve the quality of the questions, and an editor for changing the questions. For wider use, integrate the product with learning management systems and develop APIs. Additional enhancements to the site will include multidisciplinary question support, collaborative learning opportunities, and real-time coding exams.

VII. CONCLUSION:

A strong foundation for producing varied and dynamic coding questions suited to different skill levels is provided by the creation of an automatic question generator with Flask and the OpenAI API. In addition to increasing the effectiveness of content production for training and education, this initiative opens the door for major future growth. Real-time evaluations, multimedia-rich questions, adaptive learning features, and integration with well-known learning systems are examples of possible developments. While security features and AI-driven analytics will ensure dependability and ongoing improvement, gamification and collaborative tools will further increase user engagement. All things considered, this project lays the foundation for a more scalable, customized, and interactive approach to contemporary learning and evaluation tools.

VII. REFERENCES

- [1] M. Onat Topal, Anil Bas, Imke van Heerden “Exploring Transformers in Natural Language Generation: GPT, BERT, and XLNet” 2021.
- [2] Raul Puri, Ryan Spring, Mostofa Patwary ,Mohammad Shoeybi, Bryan Catanzaro “Training Question Answering Models From Synthetic Data” 22 Feb 2020
- [3] Agnès Mustar, Sylvain Lamprier, Benjamin Piwowarski “Using BERT and BART for Query Suggestion” 5 Oct 2021.
- [4] Shuyang Cao and Lu Wang “Controllable Open-ended Question Generation with A New Question Type Ontology “1 Jul 2021
- [5] Chenyang Lyu, Lifeng Shang, Yvette Graham, Jennifer Foster, Xin Jiang, Qun Liu “Improving Unsupervised Question Answering via Summarization-Informed Question Generation “ 16 Sep 2021 .
- [6] md rashad al hasan rony,uttam kumar,roman teucher, liubov kovriguinal , and jens lehmann “A Generative Approach for SPARQL Query Generation From Natural Language Questions “5 July 2022 .
- [7] Sivasurya Santhanam “context based text-generation using lstm networks “May 4, 2020
- Nikahat Mulla1,Prachi Gharpure “Automatic question generation: a review of methodologies, datasets, evaluation metrics, and applications “30 January 2023
- [8] Ghader Kurdi1,Jared Leo1, Bijan Parsia1,Uli Sattler,Salam Al-Emari “A Systematic Review of Automatic Question Generation for Educational Purposes “21 November 2019
- [9] Stefan Feuerriegel,Jochen Hartmann,Christian Janiesch, Patrick Zschech “Generative AI “: 12 September 2023
- [10] Xiyao Ma,Qile Zhu,Yanlin Zhou, Xiaolin Li2 “Improving Question Generation with Sentence-Level Semantic Matching and Answer Position Inferring “
- [11] Luis Enrico Lopez, Diane Kathryn Cruz, Jan Christian Blaise Cruz, Charibeth Cheng “Transformer-based End-to-End Question Generation” 3 May 2020.
- [12] Shweta Yadav, Deepak Gupta , Asma Ben Abacha , Dina Demner-Fushman” Question-aware transformer models for consumer health question summarization”
- [14] Asahi Ushio and Fernando Alva-Manchego and Jose Camacho-Collados” Generative Language Models for Paragraph-Level Question Generation” 2 Jan 2023.
- [15] David Baidoo-Anu1 , Leticia Owusu Ansah” Education in the Era of Generative Artificial Intelligence (AI): Understanding the Potential Benefits of ChatGPT in Promoting Teaching and Learning” 31.12.2023